ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н.П. ОГАРЁВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ
Зав. кафедрой
канд. физ.-мат. наук, доцент
_____ А.Г. Смольянов
(подпись)
«18» июня 2020 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ И РАЗВЕРТЫВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ

Автор магистерской диссертации (подпись) 08.06.2020 С. И. Абрамов

Обозначение магистерской диссертации МД–02069964–02.04.02–01–20

Направление 02.04.02 Фундаментальная информатика и информационные технологии

Руководитель работы
канд. физ.-мат. наук (подпись) 08.06.2020 А. В. Попов

Нормоконтролер
доцент, канд. тех. наук (подпись) 10.06.2020 С. В. Гарина

Рецензент
доцент, канд. физ.-мат. наук (подпись) 22.06.2020 В. И. Сафонов

Саранск

2020

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н.П. ОГАРЁВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ
Зав. кафедрой
канд. физ.-мат. наук, доцент
_____ А.Г. Смольянов
(подпись)
«25» декабря 2019 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

(в форме магистерской диссертации)

Студент Абрамов Сергей Игоревич

1 Тема Современные технологии разработки и развертывания веб-приложений

Утверждена приказом № 10281-с от 12.12.2018

2 Срок представления работы к защите 10.06.2020

3 Исходные данные для научного исследования литература и электронные источники по вопросам разработки клиент-серверных приложений

4 Содержание выпускной квалификационной работы

4.1 Повышение качества PHP-приложений для перехода к групповой разработке

4.2 Настройка среды разработки и выполнения веб-приложений

4.3 Создание и настройка окружения для финансового веб-сервиса

4.4 Рефакторинг и доработка финансового веб-сервиса

5 Приложения

5.1 ПРИЛОЖЕНИЕ А (обязательное) Конфигурация окружения

5.2 ПРИЛОЖЕНИЕ Б (обязательное) Функции для финансового анализа и оценки вероятности одобрения кредита

Руководитель работы
канд. физ.-мат. наук (подпись)(дата) А. В. Попов

Задание принял к исполнению (подпись)(дата) С. И. Абрамов

РЕФЕРАТ

Магистерская диссертация содержит 111 страниц, 33 рисунка, 1 таблицу, 1 формулу, 31 использованный источник, 2 приложения.

ВЕБ-ПРИЛОЖЕНИЕ, LARAVEL, DOCKER, PHP, РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЙ, КОМАНДНАЯ РАЗРАБОТКА, КРЕДИТЫ, РЕФАКТОРИНГ, КОНТЕЙНЕРИЗАЦИЯ, ФРЕЙМВОРК, ФИНАНСОВЫЙ СЕРВИС.

Объектом исследования является жизненный цикл современных веб-приложений, в частности групповая разработка и ввод в промышленную эксплуатацию подобных сервисов.

Цель работы – разработка окружения для командной работы над веб-приложением, рефакторинг существующего проекта финансового веб-сервиса для улучшения внутренних характеристик качества приложения, расширение функционала данного сервиса.

В ходе исследования был проведен анализ способов повышения качества веб-приложений и организации их окружений для решения проблем командной разработки и быстрого развертывания на сервере. Было создано окружение для получения идентичных конфигураций на компьютерах разработчиков и на веб-серверах. Разработанный ранее веб-сервис был дополнен новыми режимами финансового анализа и переведен на MVC-архитектуру фреймворка Laravel.

Степень внедрения – веб-приложение развернуто на боевом сервере, внедрено в работу организации и доступно для использования любым пользователям, подключенным к сети Интернет.

Область применения – наработки по настройке окружения для командной разработки и развертывания веб-приложений – в других проектах, веб-сервис – в работе организаций, пользующихся кредитными услугами.

Эффективность – повышение продуктивности командной разработки веб-приложений, снижение трудозатрат при расчете графиков погашения и оценке вероятности одобрения кредитов, повышение эффективности управления организацией за счет автоматизации финансового анализа.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 7

1 Повышение качества PHP-приложений для перехода к групповой разработке 10

1.1 Выбор архитектуры приложения 12

1.2 Использование PHP-фреймворков для стандартизации решений 16

1.2.1 Symfony 19

1.2.2 YII 19

1.2.3 Laravel 21

1.2.4 Выбор оптимального фреймворка 23

2 Настройка среды разработки и выполнения веб-приложений 26

2.1 Варианты размещения веб-приложений 26

2.1.1 Аренда выделенного сервера 27

2.1.2 Виртуальные частные сервера 28

2.2 Технологии настройки окружения веб-приложения 30

2.2.1 Виртуализация серверов 31

2.2.2 Технология контейнеризации приложений 33

2.2.2.1 Docker 36

3 Формирование окружения для разработки сервиса 42

3.1 Установка Docker 42

3.2 Установка Docker Compose 44

3.3 Настройка конфигурации окружения 46

3.3.1 HTTP-сервер и менеджер процессов PHP 48

3.3.1.1 Настройка контейнера web 49

3.3.1.2 Настройка контейнера app 52

3.3.1.3 Конфигурация HTTP-сервера 55
3.3.2 Настройка контейнера СУБД MySQL 56
3.3.3 Настройка контейнера phpMyAdmin 57
3.3.4 Настройка контейнера Composer 58
3.3.5 Сертификация веб-приложения 58
3.4 Запуск окружения 62
4 Рефакторинг и доработка финансового веб-сервиса 65
4.1 Перевод проекта на Laravel 65
4.2 Рефакторинг компонента авторизации пользователей 68
4.2.1 Настройка авторизации через социальные сети 74
4.3 Рефакторинг модуля расчета графиков погашения кредита 81
4.4 Доработка функционала сервиса 82
4.4.1 Методы финансового анализа 84
4.4.2 Реализация методик финансового анализа 87
4.4.2.1 Расчет финансовых коэффициентов 91
4.4.2.2 Реализация расчета индекса Альтмана 95
4.4.2.3 Оценка вероятности получения кредита 97
4.5 Развертывание проекта на VPS 100
ЗАКЛЮЧЕНИЕ 103
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 106
ПРИЛОЖЕНИЕ А (обязательное) Конфигурация окружения 109
ПРИЛОЖЕНИЕ Б (обязательное) Функции для финансового анализа и оценки вероятности одобрения кредита 114

ВВЕДЕНИЕ

В настоящее время большинство сфер жизни уже частично или полностью перешли в Интернет, количество пользователей глобальной сети стремительно растет. Интернет становится доступным и необходимым для разных социальных групп и возрастных категорий населения.

В современном мире для любой организации необходимо иметь хотя бы минимальное представительство в сети. Такой ресурс информирует клиентов об услугах, рекламирует организацию, повышает ее имидж и создает индивидуальный стиль. А во многих сферах большой спектр услуг можно и нужно предоставлять в онлайн-режим – это оказывается удобнее и для клиентов, и для владельцев бизнеса. Услугами, предоставляемыми онлайн, могут пользоваться не только люди, проживающие территориально рядом с организацией, но и потенциальные клиенты из других городов, регионов и даже стран, что создает отличную основу для коммерческой деятельности компаний.

Подобное направление развития характерно не только для частного бизнеса. В России уже законодательно все государственные органы должны максимально предоставлять свои услуги с помощью электронных дистанционных сервисов, таких как «Госуслуги». Все образовательные организации, университеты, школы обязаны иметь собственные или государственные электронные ресурсы, которые должны соответствовать строгим требованиям.

Люди, привыкая к удобству, уже не хотят лично обращаться в организации для решения задач, которые можно было бы выполнить, не выходя из дома. Если компания не может этого предоставить, она теряет клиентов, которые уходят к конкурентам.

Все это приводит к постоянному росту количества и сложности веб-приложений. В одном веб-сервисе может использоваться множество технологий разного уровня сложности, справиться с которыми одному

разработчику становится практически невозможно. Поэтому создание сложных продуктов становится совместной задачей для команд разработчиков.

Командная разработка позволяет значительно ускорить и улучшить процесс создания веб-приложений, однако это возможно лишь при правильной организации процесса разработки вообще и каждого отдельного проекта в частности. Один разработчик способен вести свой собственный проект, он знает его структуру, понимает, какие изменения кода повлекут за собой изменения других областей приложения и т.д. Но когда в работу вступает второй разработчик или команда разработчиков, сразу возникают проблемы:

– неправильно выбранная архитектура приложения (например, смешение слоев данных и их представления) или реализованные ранее нестандартные решения усложняют понимание проекта новыми разработчиками и затрудняют слияние в проекте изменений, сделанных разными программистами;

– отсутствие заранее подготовленных тестов увеличивает вероятность пропустить ошибку после изменений в коде разными разработчиками;

– в силу несогласованности окружения у разработчиков и на боевых серверах могут появляться ошибок в работе приложения после его развертывания на сервере.

Подобные коллизии способны свести на нет большинство плюсов командной разработки, а в некоторых ситуациях и сделать ее невозможной.

Целями магистерской диссертации являются:

– доработка функционала и рефакторинг разработанного ранее веб-приложения по расчету финансовых показателей компаний;

– формирование окружения для групповой разработки и развертывания проекта на отдельном веб-сервере.

В соответствии с целями были сформулированы следующие задачи:

– проанализировать и выбрать технологии для решения проблем, связанных с переходом к групповой разработке веб-приложений;

– выполнить рефакторинг разработанного ранее веб-приложения и его рабочего окружения для улучшения внутренних характеристик качества;

– доработать функционал разрабатываемого сервиса для расчета финансовых показателей;

– развернуть проект в работу на отдельном веб-сервере.

Магистерская диссертация состоит из введения, четырех разделов и заключения.

В первом разделе были проанализированы проблемы, возникающие при переходе к групповой разработке, рассмотрены способы построения проектов, позволяющие решить эти проблемы и повысить эффективность разработки веб-приложений в команде.

Во втором разделе изучены методы размещения веб-сервисов с целью выбора решения для построения окружения разрабатываемого проекта.

В третьем разделе было спроектировано и реализовано Docker-окружение для PHP-приложений, позволяющее запускать проект на компьютерах всех разработчиков, а также непосредственно на сервере, в полностью идентичной среде.

В четвертом разделе был произведен полный рефакторинг веб-сервиса, разработанного в рамках бакалаврской выпускной квалификационной работы, на базу фреймворка Laravel, реализованы методики проведения финансового анализа деятельности компаний, а также создан механизм для оценки вероятности одобрения кредита.

1 Повышение качества PHP-приложений для перехода к групповой разработке

Ранее в рамках работы над бакалаврской выпускной квалификационной работой мы разработали веб-приложение на языке PHP для хранения и расчета финансовых показателей компаний. Работа велась командой из двух человек, и даже в такой небольшой группе уже появлялись некоторые проблемы.

Первая проблема возникла, когда в проект была добавлена функция для расчета дат, на которые необходимо заполнить баланс компаниям, использующим сервис. Выдержка из кода данной функции:

```
$Date = new DateTime($Date3);
$Balance_Dates[]=($Date->modify("-6 month"))->format("Y-m-d");
$Date = new DateTime($Date3);
$Balance_Dates[]=($Date->modify("-3 month"))->format("Y-m-d");
$Balance_Dates[]=$Date3;
```

Функция корректно работала у разработчика, добавившего данный код в проект. Однако после клонирования репозитория с проектом на компьютер второго разработчика, код начал работать совсем не так, как ожидалось. Функция срабатывала, не возвращала никаких ошибок, но при этом дат в массиве `$Balance_Dates` не было. На исправление данной ошибки ушла неделя, так как не было понятно, где именно и что пошло не так.

Как выяснилось, на компьютере одного разработчика был установлен PHP версии 5.3, а у второго – версии 5.2. Метод `modify()` у объекта `DateTime` существует в обеих версиях PHP [30], он срабатывал и не возвращал никаких ошибок. Однако в версии 5.3 `modify()` в случае успешного завершения возвращал измененный объект `DateTime`, который можно было сразу же использовать, а в версии 5.2 этот метод также изменял объект, но в случае успешного завершения возвращал `Null`, поэтому даты в массиве не сохранялись. Таким образом, данная проблема возникла из-за неконсистентности программного окружения у разработчиков, что довольно часто встречается при групповой разработке.

Вторая большая проблема возникла, когда параллельно над проектом велась работа сразу в двух больших направлениях. В репозитории проекта было создано две ветки: в одной из них разрабатывались методы ввода и сохранения баланса компаний, а во второй работали над системой авторизации и регистрации пользователей. По окончанию работы возникла необходимость в слиянии изменений из обеих веток в основную ветку проекта, однако даже с учетом того, что изменения логически не пересекались между собой, слияние в автоматическом режиме средствами системы контроля версий Git выполнить не удалось. Пришлось объединять ветки вручную, на что дополнительно ушло много времени, хотя теоретически эта работа могла быть выполнена за секунды в автоматическом режиме. Данная проблема возникла из-за недостаточно продуманной структуры приложения и нестандартных решений, используемых в проекте.

И подобных проблем при переходе от индивидуальной к групповой разработке может возникнуть масса. Разрабатывая проект в одиночку, можно обойтись текстовым редактором или IDE для написания кода, изобрести собственную структуру проекта, понятную только себе [24], писать название переменных и функций в произвольном стиле и т.п. Но в командной разработке приходится придерживать значительно более жестких требований к архитектуре и оформлению проекта.

Дальнейшее развитие разработанного нами веб-сервиса для расчета финансовых показателей компания приведет к расширению команды разработчиков. Поэтому было принято решение помимо реализации нового функционала провести рефакторинг проекта для повышения качества программного продукта и упрощения перехода на полноценную групповую разработку.

Рефакторинг — это процесс изменения программной системы, в ходе которого внешнее поведение кода остается неизменным при усовершенствовании его внутренней структуры [23]. По сути, рефакторинг

кода представляет собой улучшение проекта уже после того, как этот код написан.

Как известно качество программного обеспечения формируется из двух характеристик:

– внешние характеристики качества ПО – это свойства, с которыми работает пользователь программы;

– внутренние характеристики – это свойства проекта, которые направлены на улучшение качества кода и облегчение процесса разработки.

Рассмотрим технологии и инструменты, позволяющие улучшить внутренние характеристики качества разрабатываемого веб-сервиса и снизить вероятность возникновения проблем при командной разработке. Здесь можно выделить два направления работы:

– структура проекта и его стандартизация;

– изменение окружения веб-приложения.

Рассмотрим в этой главе подходы к стандартизации проекта и организации его структуры. Начнем с самого базового момента – выбора архитектуры приложения.

1.1 Выбор архитектуры приложения

Область разработки программного обеспечения с самого момента своего появления столкнулась с стремительно растущей сложностью разрабатываемых систем и возникающими из-за этого проблемами. Изначально их пытались решить путем разработки более оптимальных алгоритмов, переработки структур данных, а также применение концепции разграничения полномочий. На первых порах это помогло решить возникшие проблемы, но со временем данные решения уперлись в максимум своих возможностей. Поэтому сообществу разработчиков пришлось заняться проработкой новой сферы – архитектуры программного обеспечения.

В основе любого веб-приложения основополагающим звеном является архитектура [15]. Это набор решений о том, как предстоит организовать

программную систему. Первые попытки создания подобных архитектур были неудачны, системы страдали от своей неорганизованности, в них было много неточностей. Довольно часто разработанные архитектуры представляли собой просто набор блок-схем, соединенных линиями, которые было трудно применить на практике.

В настоящее время существует много шаблонов и стилей проектирования, различной формальной логики и множество языков описания, которые лишены недостатков своих предшественников. Такие архитектуры стали пользоваться большой популярностью у разработчиков, так как именно они первыми смогли стандартизировать проекты. Разработчик, знакомый с какой-либо архитектурой программного обеспечения, в будущем без особых проблем сможет разобраться в структуре чужого проекта, построенного на той же или подобной архитектуре.

Наибольшей популярностью сейчас пользуются три архитектурных шаблона. Первый из них – **многоуровневый шаблон** (Layered pattern), когда система разбивается на множество уровней, которые располагаются строго друг над другом. Каждый уровень такой системы может обращаться только к одному уровню, который находится на одну ступень ниже него. Это позволяет вести разработку каждого уровня отдельно, что повышает модифицируемость системы и эффективность командной работы над проектом. Однако при таком подходе система становится еще более сложной, чем была. Связи между слоями по ступеням также могут заметно снижать производительность приложения.

Второй популярный паттерн – **шаблон посредника** (Broker pattern). Он часто применяется в системах, имеющих очень большое количество модулей. В таких условиях их прямое взаимодействие становится слишком сложным и поэтому в систему вводится «посредник». Каждый модуль осуществляет взаимодействие с любым другим модулем только через него. Примером такого посредника может быть шина данных. Это позволяет значительно повысить в проекте совместимость различных модулей и упростить его. Однако такой посредник становится «узким горлышком» всей системы, замедляя ее работу.

Кроме того, повышается риск атаки на посредника, а следствие нарушения его работы сделает нерабочей всю систему.

Последний рассматриваемый паттерн – **шаблон MVC** (Model-View-Controller, «Модель-Представление-Контроллер») (рисунок 1.1). Как показывает практика, в программном обеспечении требования к интерфейсу меняются гораздо чаще, чем к другим компонентам системы. Постоянно возникает необходимость модифицировать пользовательский интерфейс, сохраняя при этом корректное взаимодействие с данными. Именно эта особенность лежала в основе при проектировании шаблона MVC. Здесь интерфейс системы полностью отделен от данных и работы с ними. Это позволяет максимально быстро менять интерфейсы и создавать новые, не касаясь при этом данных.

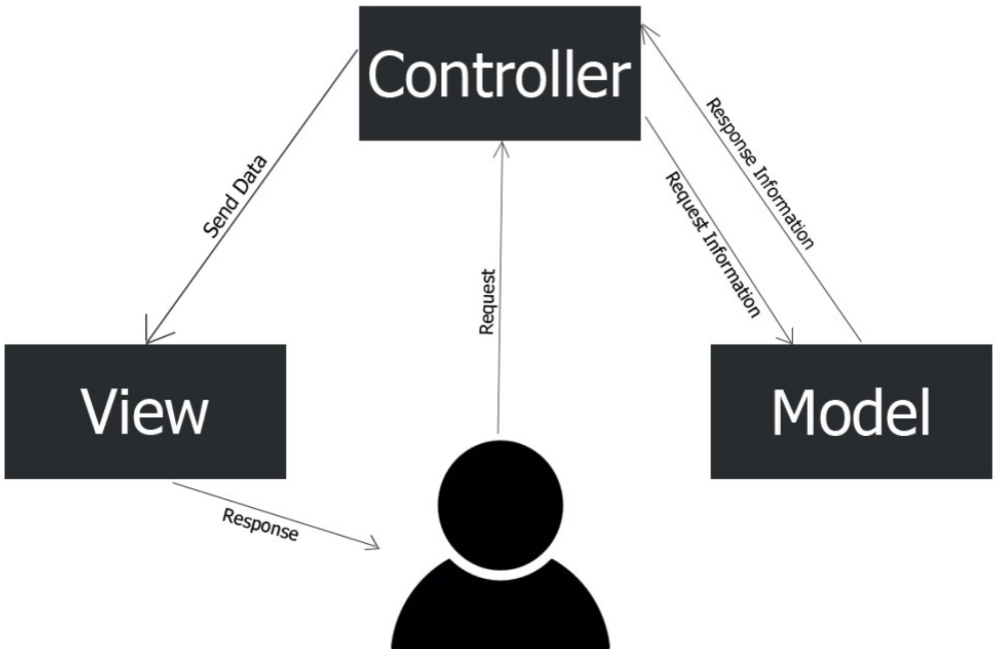


Рисунок Повышение качества PHP-приложений для перехода к групповой разработке.1 – Архитектурная модель MVC

Как следует из самого названия, в MVC система разделена на:

– **Модель**, которая непосредственно хранит данные.

– **Представление**, являющееся интерфейсом, отображающим данные и позволяющим конечному пользователю работать с ними.

– **Контроллер**, играющий роль связующего звена между моделью и представлением.

Некоторым недостатком концепции MVC можно считать небольшое замедление скорости работы, возникающее в силу усложнения взаимодействия между частями системы.

Из-за отделения интерфейса от данных данная архитектура стала сейчас наиболее популярной в сфере веб-разработки. Удобство работы с проектом, построенным на такой архитектуре настолько высоко, что незначительным понижением скорости можно пренебречь, без замеров скорости такое малое падение никто не заметит.

На наш взгляд, оптимальным выбором для разрабатываемого сервиса является именно шаблон MVC. Большая часть веб-приложений на PHP пишется именно на основе этой архитектуры, можно сказать, что она уже стала негласным стандартом. На данный момент MVC уже получил свои собственные направления развития, например MVP, MVVC, которые чуть лучше подходят для определенных типов проектов [20]. При разработке данного веб-сервиса оптимальным будет использование стандартной MVC-модели.

Рефакторинг проекта на данную архитектуру позволит повысить скорость вовлечения в работу новых разработчиков, так как заметно облегчится понимание структуры кода. Будут исключены проблемы при одновременной работе нескольких разработчиков и дальнейшем слиянии изменений в основную ветку проекта.

Изначально наш проект был написан на чистом PHP без применения каких-либо специфических архитектурных решений. Пользователь обращается к веб-серверу, запрос перенаправляется интерпретатору PHP с указанием файла, который нужно обработать. Интерпретатор выполняет скрипт в файле и возвращает представление веб-серверу, который отправляет его пользователю. Рассмотрим, как работает проект построенный на MVC, чтобы понять в каком направлении необходимо двигаться при рефакторинге проекта.

Работу MVC-приложения можно представить как последовательность следующих действий:

1. При посещении пользователем веб-ресурса создается экземпляр приложения. Он запускается и возвращает пользователю представление, например главную страницу веб-приложения.

2. Приложение получает HTTP-запрос от пользователя и сопоставляет его с необходимым контроллером и методом.

3. Создается экземпляр контроллера и запускается необходимый метод. В этом методе могут содержаться вызовы модели, которые считывают данные из базы данных.

4. Формируется представление с данными, которые были получены из модели. Представление выводится непосредственно пользователю.

Таким образом, в основе MVC приложения должны присутствовать следующие компоненты:

– родительский класс, на основе которого будут строиться модели;

– родительский класс для контроллеров;

– маршрутизатор-контроллер, принимающий все запросы и направляющий их в необходимые контроллеры;

– механизм для построения представлений.

Широкая популярность данной архитектуры привела к появлению множества PHP-фреймворков, построенных на MVC [10]. Фреймворки по умолчанию уже имеют все необходимые компоненты и предоставляют удобные методы работы с ними. И это далеко не единственные плюсы использования фреймворков, которые мы рассмотрим более подробно.

1.2 Использование PHP-фреймворков для стандартизации решений

В каждом своем проекте разработчик довольно часто решает однотипные задачи: постоянно приходится создавать механизмы регистрации и авторизации, поддерживать профили пользователей и т.д. Для облегчения своей работы, разработчики постоянно пытаются мигрировать какие-то решение из

старых проектов в новые. Со временем набор стандартных решений вырастает в собственную библиотеку. Но у данного подхода есть большой недостаток – код практически не развивается. Уязвимости мигрируют из проекта в проект, код не переписывается для работы с более новыми версиями программного обеспечения. Поэтому со временем начали появляться различные решения для более быстрой и простой разработки веб-приложений, уже включающие в себя все стандартные механизмы. Подобные решения постоянно развиваются и обновляются командой разработчиков. Они получили название **фреймворков**.

Фреймворки – это программные продукты, которые облегчают разработку и поддержку технически сложных или сильно загруженных проектов. Фреймворки обычно включают в себя базовые программные модули, а все специфические решения для конкретного проекта реализуются на их основе. Благодаря этому достигается не только высокая скорость разработки, но и отличная производительность и надежность систем.

Одним из основных преимуществ при использовании фреймворков является то, что веб-приложения часто используют стандартизированную структуру для организации своих компонентов. Создание структуры объектно-ориентированного приложения во время разработки с использованием фреймворка становится намного проще.

По своей сути, фреймворк – это множество конкретных и абстрактных классов, а также определенные способы взаимодействия с ними. Конкретные классы обычно реализуют взаимодействие между классами, а абстрактные классы являются точками расширения (например, части приложения могут наследоваться от базовых классов фреймворка).

Фреймворки позволяют легко решать даже нестандартные задачи, делая это различными способами. В проектах возможно использование готовых пакетов и библиотек.

Последнее преимущество, которое можно выделить, несколько субъективно. Разработчики единодушно утверждают, что работать с

фреймворками гораздо интереснее. А чем больше разработчику нравится проект, тем более высокий уровень исполнения можно ждать.

Конечно, у фреймворка относительно использования чистого языка есть и недостатки. Во-первых, для начала работы с фреймворками разработчик должен обладать более высокой квалификацией. Хоть фреймворки и используют стандартизированную структуру для организации компонентов, не зная основ этой структуры в ней будет довольно сложно разобраться. Во-вторых, каждый фреймворк по сути является отдельной полноценной средой, которой как-то нужно управлять. Это ведет к тому, что при начальном изучении нового фреймворка приходится тратить достаточно много времени на изучение его собственной документации. Но это время не будет потрачено впустую, так как поможет в будущем значительно сократить время разработки.

На данный момент приходится выбирать между множеством существующих PHP-фреймворков, каждый из которых обладает разными возможностями и предназначен для решения различных задач. Рассмотрим только самые популярные фреймворки (рисунок 1.2), которые используются для разработки веб-приложений, и сравним их возможности.

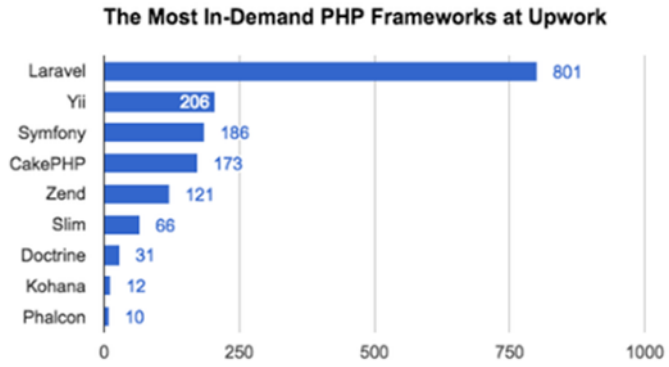


Рисунок Повышение качества PHP-приложений для перехода к групповой разработке.2 – Статистика популярности PHP-фреймворков.

Как видно из статистики, лидерами сегодня являются фреймворки Laravel, Yii и Symfony.

1.2.1 Symfony

Неполный список самых известных проектов, основанных на фреймворке Symfony [13]:

– фреймворк Sylius eCommerce;

– сервис социальных закладок Delicious;

– API Platform – специализированный фреймворк для RESTful API;

– Laravel – доступный, RAD-ориентированный фреймворк;

– платформа Oro и связанная с ней экосистема: Oro CRM, Akeneo, OroCommerce и Marello.

Можно заметить, что в данном списке присутствуют другие фреймворки, и это не случайно. Особенность Symfony в том, что это низкоуровневый фреймворк, который чаще всего используется как основа для написания собственных фреймворков. Это очень мощный и сложный продукт, он позволяет без особых усилий построить систему практически под любые потребности. Но при этом в нем практически нет компонентов, которые можно было бы использовать в собственных проектах в качестве готовых решений.

Отметим, что Symfony в принципе некорректно сравнивать с Laravel и Yii2, так как последние используют компоненты Symfony в своих реализациях. Использовать Symfony имеет смысл в больших высоконагруженных промышленных проектах.

1.2.2 YII

Yii – высокопроизводительный PHP фреймворк для быстрой разработки современных веб-приложений (логотип фреймворка представлен на рисунке 1.3). Фреймворк основан на архитектурном шаблоне MVC, его возможности позволяют реализовывать крупномасштабные проекты, например различные форумы, порталы, CMS, RESTful веб-сервисы и пр.



Рисунок Повышение качества PHP-приложений для перехода к групповой разработке.3 – Логотип Yii

Одно из ключевых преимуществ фреймворка Yii – это низкий порог входа. Данный фреймворк сравнительно легко изучается. Начинающие разработчики иногда недостаточно хорошо умеют пользоваться консолью, на этот случай в Yii присутствует графический интерфейс Gii для выбора различных генераций. В этой области присутствует множество инструментов, самыми популярными из которых является генерация моделей и CRUD.

CRUD генерирует готовые интерфейсы и контроллеры к ним, которые позволяют управлять данными в указанной модели [21]. Эту возможность очень удобно использовать для быстрого и простого создания административных панелей и дашбордов. Одновременно с этим автоматически генерируется готовый интерфейс, который можно отредактировать под собственные потребности, обеспечить необходимую защиту маршрутам – и все, интерфейс готов к использованию. По моему мнению, это самый главный плюс Yii2 в сравнении с другими фреймворками.

К сожалению, у Yii есть и недостатки. На данный момент проект недостаточно быстро развивается, в результате задерживаются исправление уязвимостей и обновление инструментов и технологий, используемых Yii. В проектах, построенных на данном фреймворке, также слишком сильно связаны друг с другом клиентская и серверная части приложения, что влечет за собой некоторые сложности при его поддержке.

Есть и более существенный недостаток: фреймворк Yii не позволяет пользоваться сборщиками типа WebPack, Gulp или другими. Точнее говоря, пользоваться можно, но тогда необходимо отказаться от основного

преимущества – GUI-области. Невозможно будет пользоваться генераторами и готовыми решениями для интерфейса, не удастся использовать классы, позволяющие собирать скрипты и CSS и т.п. А если отказаться от всего этого, то Yii теряет все свои основные преимущества.

В современной веб-разработке ведется активная работа по отделению клиентской и серверной части, и в этом плане данный фреймворк заметно устаревает.

1.2.3 Laravel

Laravel – это фреймворк для создания веб-приложений с выразительным и элегантным синтаксисом, основанный на архитектурном шаблоне MVC. Он позволяет решить множество задач, которые возникают из проекта в проект. Laravel позволяет создавать аутентификацию и регистрацию буквально в одну команду, имеет встроенную поддержку сессий и уже работающее и настроенное кеширование. Это попытка собрать все лучшее из других PHP-фреймворков в один. Логотип фреймворка представлен на рисунке 1.4.



Рисунок Повышение качества PHP-приложений для перехода к групповой разработке.4 – Логотип Laravel

У данного фреймворка много преимуществ, рассмотрим основные из них.

Laravel имеет встроенный сборщик Laravel Mix, основанный на WebPack [27]. На самом деле это надстройка над Webpack, позволяющая легко начать работу с ним и собирать фронтенд, не изучая сложные руководства по WebPack. Благодаря этому инструменту в проекте возможна сборка SCSS и

однофайловых компонентов Vue сразу «из коробки». В последних версиях фреймворка Vue.JS был полноценно встроен в систему по умолчанию.

Второй внушительный плюс Laravel – это его шаблонизатор, получивший название Blade [2]. Он напоминает популярный шаблонизатор Twig, но синтаксис и принцип работы в нем немного другой. Единственный минус Blade – это то, что Blade доступен и известен только разработчикам Laravel, а Twig известен многим и применяется в большом количестве других фреймворков.

Laravel обладает механизмом очень гибкого формирования маршрутов. В целом он довольно сильно похож на маршрутизатор Symfony, но удобнее в использовании. Вместо аннотаций в фреймворке Laravel используется отдельный файл и фасадный класс Route с набором статических методов.

В маршрутах Laravel можно использовать так называемые middleware. Это различные дополнительные проверки, которые необходимо выполнить при отправке запроса контроллеру, например проверка того, что пользователь авторизован в системе. При этом данные проверки можно использовать как в контроллерах, так и при объявлении маршрута, что очень удобно.

Laravel на сегодня является одним из самых динамично развивающихся фреймворков. Постоянно выходят новые версии, исправляющие ошибки и недоработки предыдущих, быстро повышая эффективность фреймворка.

Но несмотря на большое количество плюсов, даже тут есть недостаток. В Laravel нет встроенных генераторов интерфейсов. Это достаточно большой недостаток, особенно для разработчиков переходящих на Laravel с Yii, но в контексте этого фреймворка это можно даже считать плюсом. Хотя здесь и не получится нажатием трех кнопок сгенерировать интерфейс для работы с данными, но зато можно легко отделить фронтенд от бэкенда, что требует современный мир веб-разработки. Благодаря этому и возможно использование сборщиков для фронтенда.

Генераторы в Laravel вообще далеки от Yii, но некоторые основные задачи с их помощью решить можно. Есть генераторы консольных команд, дающие готовый каркас для работы с консолью, генераторы моделей,

контроллеров и другие. Однако, в отличие от генераторов Yii, сгенерированная модель будет пустой. Разработчик сам должен указать, какое поле будет первичным, к какой таблице его надо отнести, с какими другими моделями и как она будет связана. Конечно, это добавляет гибкости, но в Yii можно без проблем написать свою генерацию, удалив стандартную. В итоге, по возможностям и удобству автоматической генерации компонентов системы Yii однозначно является лидером.

1.2.4 Выбор оптимального фреймворка

Использование того или иного фреймворка зависит как от решаемой задачи, так и от индивидуальных предпочтений разработчика. Но все же можно проанализировать интерес современных разработчиков к описанным фреймворкам, и на основе этого выбрать наиболее оптимальный инструмент.

В этой задаче может помочь Google Trends. Этот инструмент позволяет сравнить интерес пользователей к различным продуктам. Используя сервис, был построен график сравнения популярности YII и Laravel. Эта статистика с середины 2008 года по май 2020 года приведена на рисунке 1.5.

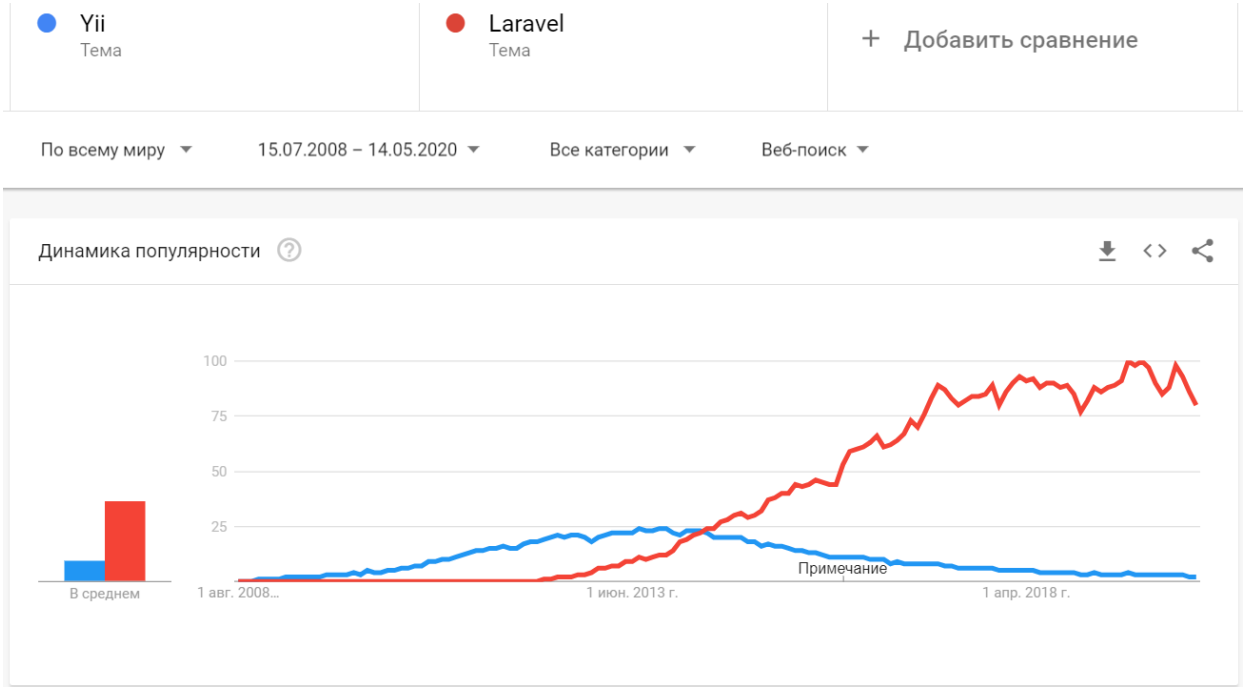


Рисунок Повышение качества PHP-приложений для перехода к групповой разработке.5 – Статистика интереса пользователей к Yii и Laravel

Как видно из статистики, Yii с самого момента выхода быстро получил большую популярность и долгое время оставался лидером. Но выход Laravel очень серьезно пошатнул интерес к Yii. С тех самых пор как вышел Laravel, интерес к нему все время только растет, а к Yii довольно быстро падает. Это говорит о том, что все больше и больше разработчиков начинают использовать для разработки Laravel, и постепенно забывают отстающий в развитии Yii.

На сегодня можно разделить эти фреймворки на две сферы:

– Yii можно использовать в тех случаях, когда нет особо больших требований к административной части фронтенда, а проект нужно сделать красиво и в сжатые сроки.

– Laravel хорошо подойдет, когда есть особые требования к фронтенду и чуть побольше времени на разработку интерфейсов. Также с помощью Laravel можно полностью разделить клиентскую и серверную части веб-приложения.

Большинство современных разрабатываемых проектов требуют не просто красивого интерфейса, а максимально продуманной логики приложений. Это

требование можно применить и к нашему сервису для расчета финансовых показателей компаний.

Перевод проекта на фреймворк Laravel позволит в будущем сократить время разработки и упростить ее процесс. В разрабатываемом проекте присутствуют системы авторизации и регистрации пользователей, авторизации через социальные сети, что в данном фреймворке реализуется очень просто с помощью встроенных компонентов. Структура и организация проекта на основе архитектуры MVC благоприятно скажется на возможностях командной работы с проектом. Имеющиеся в Laravel механизмы миграций позволят исключить проблемы при совместной работе нескольких разработчиков над базой данных, так как в них соблюдается строгая очередность выполняемых манипуляция с базами данных.

Учитывая все эти аспекты, для рефакторинга разрабатываемого сервиса мы выбрали именно фреймворк Laravel.

2 Настройка среды разработки и выполнения веб-приложений

В командной разработке могут возникать проблемы из-за неправильно организованного окружения веб-приложения. Разрабатываемые проекты начинают обрастать десятками различных зависимостей, инструментов и т.д. Для каждого инструмента необходимо отслеживать выходы новых версий, чтобы своевременно их обновлять. В результате даже при индивидуальной разработке простая задача переноса проекта на другой компьютер или сервер начинает отнимать очень много времени на удовлетворение всех зависимостей и установку инструментов. А если разработчиков несколько, то подобную работу необходимо выполнять на каждом компьютере на регулярной основе, ведь у всех разработчиков должен быть полностью идентичный набор окружения и одни и те же инструменты. В такой ситуации приходится задумываться о том, как правильно построить окружение приложения, чтобы упростить управление им, перенос с одного сервера на другой и при этом поддерживать однородность окружения на всех компьютерах и серверах.

Построенное окружение должно соответствовать ряду критериев. Во-первых, оно должно быть быстрым и надежным и для работы, и для доступа к хранимых данных. Во-вторых, оно должно быть достаточно гибким, чтобы у разработчика была возможность при необходимости изменять часть системы, при этом не влияя на остальные части. И наконец, окружение должно оставаться экономичным в потреблении ресурсов сервера.

2.1 Варианты размещения веб-приложений

При создании окружения для веб-приложения необходимо отталкиваться от того, где разрабатываемое приложение будет размещено в будущем. На данный момент наш веб-сервис для расчета финансовых приложений размещен на хостинге, так как это наиболее простой быстрый способ развертывания. Однако, в ходе управления проектом, размещенным на хостинге, довольно часто возникали проблемы. То из-за квоты на работу с жестким диском

невозможно было своевременно обновить проект, для исправления недостатков, то из-за ограничения прав доступа было невозможным более гибко сконфигурировать PHP для более быстрой работы проекта. Связано это с тем, что компания-хостер никогда не предоставляет для пользователя возможность для полноценного управления сервером. Как правило, весь функционал ограничивается местом для файлов проекта и иногда управлением базой данных. Ни о каком гибком управлении в данном случае говорить не приходится, поэтому было принято решение пересмотреть вопрос размещения веб-приложения.

Аналогом хостингу при размещении веб-приложения может быть аренда своего выделенного сервера.

2.1.1 Аренда выделенного сервера

При аренде выделенного сервера разработчик получает полный контроль над сервером. Фактически в этом случае клиент получает доступ к самому обычному «железному» серверу, находящемуся в любой точке мира. Благодаря этому ему становится доступно большое множество способов для построения окружения. Самый простой и прямой путь – это размещение на одном арендованном сервере всего необходимого для разрабатываемого проекта: сервера баз данных, HTTP-сервера, интерпретатора php и самого веб-приложения. Однако, как говорит одна старая пословица: «Не клади все яйца в одну корзину». И это хорошо применимо и в этом случае. Многие задачи принято разносить по разным серверам, это очень важно и для безопасности веб-сервера, и для повышения его отказоустойчивости.

Связано это с тем, что даже одно вышедшее из строя звено может обрушить весь сервер, на котором уже находятся данные. А если там была, например, база данных, содержащая данные предприятия за последние 15 лет – то это просто смерти подобно. Не говоря уже о недоступности сервиса, который может обернуться серьезными финансовыми убытками. Все эти

неприятности возникают из-за допущенной ошибки при проектировании окружения.

Итак, разделение задач по разным серверам – это скорее необходимость в мире современных сложных систем, чем роскошь [18]. Но тут возникает проблема финансовых ресурсов. Каждый отдельный сервер потребляет электроэнергию и занимает место на столе или в стойке. Поэтому аренда даже самого слабого сервера на 2020 год стоит более 10 000 рублей, что крайне дорого для небольшого проекта. А арендовать 3-4 таких сервера финансово не предоставляется возможным.

Частично снизить стоимость данного решения можно, арендуя не полноценные сервера, а VPS – виртуальные частные сервера.

2.1.2 Виртуальные частные сервера

Виртуальный частный сервер – это технология создания нескольких изолированных серверов в рамках одного физического сервера. Такой сервер использует технологии виртуализации для обеспечения пользователя выделенными (приватными) ресурсами на сервере с несколькими пользователями. Виртуальные сервера обычно обозначают аббревиатурами VDS (Virtual Dedicated Server) и VPS (Virtual Private Server), которые по сути являются синонимами [31].

Когда проекту становится мало ресурсов обычного хостинга, а аренда собственного выделенного сервера остается избыточным и слишком дорогим решением, то VPS в данной ситуации является золотой серединой.

По функционалу для разработчика выделенные частные сервера нисколько не уступают аренде собственного сервера, при этом имея все его преимущества перед хостингом.

Во-первых, арендатор такого сервера получает полный доступ к серверу на правах администратора. Разработчик имеет возможность гибко настроить под себя все программное обеспечение на сервере, используя полный SSH-доступ.

Во-вторых, в таком случае клиент получает в свое использование гарантированные выделенные ресурсы. И хоть ресурсы одного физического сервера и используют одновременно множество клиентов, но объем жесткого диска, оперативная память и процессорная мощность строго разделены между пользователями, чтобы каждый владелец VPS мог использовать купленные ресурсы в полном объеме. Кратко говоря, разработчику принадлежит свое собственное виртуальное пространство, при этом он совершенно не зависит от «соседей» по физическому серверу.

В дополнение к этому за счет того, что ресурсы просто выделяются пользователю, VPS отлично масштабируемы. В случае роста проекта и нехватки производственных мощностей всегда можно докупить ресурсов, в которые «упирается» производительность. Чаще всего, в таком случае, ресурсы будут добавлены даже без перезагрузки системы. В случае же с арендованным физическим сервером, как правило нет возможности повысить производительность. Можно только арендовать другой сервер и соответственно перенести туда все вручную.

В-третьих, арендатор VPS получает в использование собственный IP-адрес (на традиционном хостинге все клиенты находятся на одном IP-адресе).

И последнее преимущество состоит в том, что на VPS нет никаких ограничений по количеству сайтов, баз данных, подключений и т. п. Разработчик ограничен только суммарной требовательностью проектов относительно выделенных ресурсов сервера.

В итоге при полностью идентичном функционале выделенного физического сервера за счет того, что ресурсы разделены на множество клиентов, стоимость VDS в сотни раз ниже аренды физического сервера. На 2020 год стоимость VDS начинается от 80 рублей в месяц, за минимальное количество предоставляемых ресурсов. При этом уже такой сервер способен обеспечить обслуживание 200-300 одновременных пользователей. А решения от 200 рублей в месяц уже достаточно производительны, чтобы обеспечивать работу нескольких приложений с тысячами одновременных клиентов.

За счет всех своих преимуществ данное решение отлично подходит для разрабатываемого веб-приложения. Загрузка использования веб-сервиса для расчета финансовых показателей не будет значительной, клиенты находятся на сайте не постоянно, а лишь периодически обращаются для обновления хранимой информации и проведения расчетов. В данном случае производительность физического сервера будет слишком избыточна для проекта и его аренда не будет оправдана финансово. Производительность VDS за 200-300 рублей в месяц удовлетворяет всем требованиям проекта.

Однако необходимо вспомнить о разделении задач по разным серверам. В связи с этим придется арендовать несколько VDS, что также приведет к увеличению финансовых расходов на проект. К тому же проблемы командной разработки и развертывания не только не решаются, но и усугубляются. Синхронизировать окружение и обновлять программное обеспечение придется на множестве серверов. Рассмотрим, как правильно организовать программное окружение веб-приложения, чтобы сэкономить на аренде виртуальных частных серверов и при этом решить проблемы, возникающие при командной разработке.

2.2 Технологии настройки окружения веб-приложения

Каждое приложение, входящее в окружение веб-проекта, потребляет разное количество системных ресурсов. Например, сервер работы с базой данных обычно довольно требователен к предоставляемым ресурсам. А proxy-сервера или менеджеры процессов PHP, напротив, практически никогда не оказывают большой нагрузки на процессор. Использование отдельного сервера под каждую такую задачу выглядит крайне нерациональным решением. А совместить все на одном сервере, как было установлено ранее, неправильно с точки зрения безопасности.

И тут можно вспомнить о том, как работают VDS на физических серверах. Они используют технологии виртуализации, которые могут прийти на помощь и в этой ситуации.

2.2.1 Виртуализация серверов

Виртуализация серверов – это архитектура программного обеспечения, позволяющая обеспечить одновременную работу нескольких серверов в качестве гостя на родительском хосте. Благодаря этому все программное обеспечение становится абстрагированным от физического устройства, такой сервер становится «виртуальной машиной», которая совсем не связана с оборудованием. Сама система гостевого сервера в таком случае «думает», что у нее есть эксклюзивный доступ к вычислительным ресурсам и памяти, хотя фактически это всего лишь виртуальной имитация серверного оборудования.

Выбрав данную архитектуру, разработчик получает следующие преимущества:

– повышение доступности серверов;

– повышение производительности приложений;

– упрощение сред и сдерживание роста числа серверов;

– эффективность инфраструктуры значительно повышается;

– ускоренное развертывание рабочих нагрузок;

– снижение эксплуатационных расходов.

Виртуализация серверов, благодаря консолидации, позволяет значительно сократить количество физических серверов. Там, где раньше требовалось несколько серверов, можно использовать только один сервер с необходимым количеством виртуальных систем. Это позволяет снизить финансовые затраты на аренду или приобретение серверов, а в случае с физическими серверами и снизить энергопотребление и затраты на системы охлаждения.

Простой пример: разработчику необходимо разместить два приложения, которым для работы требуется 512 Мб оперативной памяти. При этом их нельзя устанавливать на один сервер. У разработчика в этом случае есть два варианта:

1. Купить два сервера с 512 Мб RAM.

2. Купить один сервер с 1024 Мб RAM и запустить оба приложения в отдельных виртуальных машинах.

Очевидно, что во втором случае гораздо более рационально используются ресурсы сервера (в частности, процессорное время). При этом стоимость одного сервера с большим количеством RAM намного дешевле, чем стоимость двух серверов, хоть и с меньшим объемом оперативной памяти.

Однако, может показаться, что развертывание виртуальных серверов на одном сервере – это и есть «класть все яйца в одной корзине», но это верно лишь отчасти. Да, при такой организации окружения все же есть единая точка отказа в виде хостовой машины, но тем не менее, в силу самой идеологии виртуализации, все виртуальные машины на физическом уровне изолированы как друг от друга, так и от родительского хоста. Поэтому при поражении одной из виртуальных машин, например, вирусом, остальные машины не будут затронуты деструктивными действиями. Если бы задачи были размещены на одном сервере без виртуализации, то этого было бы невозможно избежать.

Этой единой точки отказа можно избежать, арендовав, например, второй сервер и развернув кластер. В этом случае сразу два сервера будут действовать как единая платформа, для виртуализации. В случае отказа одного из серверов виртуальные операционные системы продолжат работу на оборудовании второго сервера и пользователи приложения этого, скорее всего, не заметят. В крайнем случае работа прервется на несколько секунд, что не так критично, как отказ на несколько часов. И даже если на этом кластере будет запущено пять виртуальных машин, это все равно гораздо выгоднее аренды пяти серверов.

Однако, в аппаратной виртуализации в таком виде есть и недостатки, связанные с самим принципом работы виртуальных гостевых машин. Каждая виртуальная машина включает в себя операционную систему, а значит их размер, как правило, составляет несколько гигабайт. Также недостатком можно назвать то, что для загрузки виртуальной операционной системы и запуска приложения, размещенного в ней, требуется относительно много времени. Да и сама работа виртуальных машин, как уже было упомянуто ранее, потребляет много ресурсов хостовой машины, так как у каждой гостевой операционной системы есть свое ядро и множество внутренних процессов и сервисов

Для решения этих проблем совсем недавно была разработана технология контейнеризации.

2.2.2 Технология контейнеризации приложений

Контейнеризация – это легковесная виртуализация и изоляция ресурсов, на уровне операционной системы. Данная технология позволяет запускать приложение и минимальный набор системных библиотек в полностью стандартизированном контейнере, который с помощью определенных интерфейсов соединяется с хостом [1]. При этом контейнер, как и виртуальные машины, полностью не зависит от ресурсов и архитектуры хоста, на котором он запущен.

Не смотря на похожую идеологию с виртуальными машинами у технологии контейнеризации есть ряд существенных отличий.

Как видно из структуры виртуальной машины, представленной на рисунке 2.1, над хостовой операционной системой присутствует гипервизор, он и занимается управлением операционными системами, а непосредственно в них уже и работают необходимые приложения. В случае с технологией контейнеризации роль гипервизора выполняет движок конкретной используемой контейнерной технологии, но управляет он уже непосредственно приложениями.

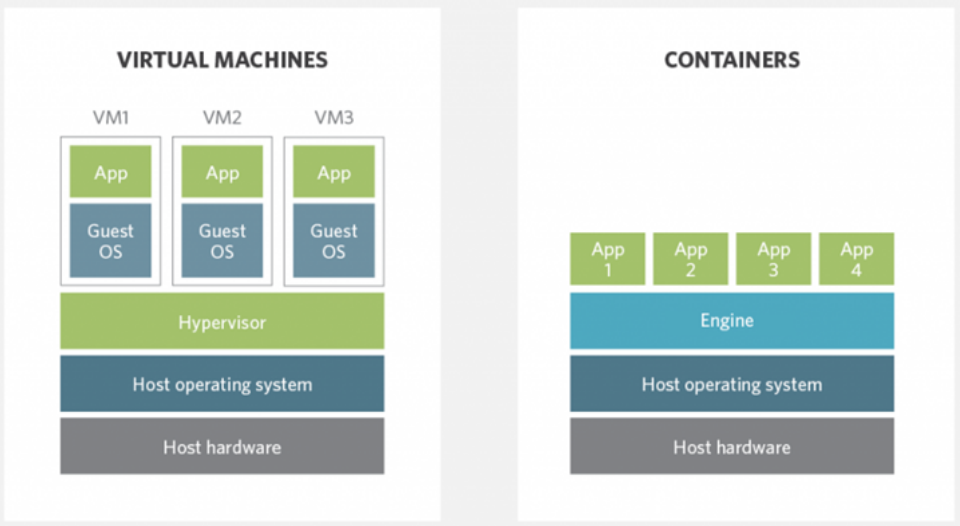


Рисунок Настройка среды разработки и выполнения веб-приложений .6 – Сравнение структур виртуальной машины и контейнеров

Понять идеологию работы с контейнерами можно взглянув на обычные контейнеры, используемые компаниями для транспортировки. Как они позволяют перевозить грузы на корабле, поезде или грузовике, так и программные контейнеры выполняют роль стандартных модулей, позволяющих развернуть программное обеспечение. При этом каждый контейнер имеет различный код и зависимости, не влияя на соседние контейнеры.

Не смотря на упрощенную структуру, контейнеры, как и виртуальные машины, полностью изолируют приложения друг от друга в хостовой операционной системе. Контейнерные приложения выполняются на основе узла контейнеров, а он в свою очередь работает в операционной системе. Именно поэтому контейнеры потребляют гораздо меньше ресурсов, чем построенная для тех же целей виртуальная машина.

Естественно, за счет экономии ресурсов создавать контейнеры гораздо быстрее, чем аналогичные виртуальные машины. Их маленький размер обеспечивает меньше накладных расходов, с точки зрения использования ресурсов сервера, а значит при переходе на данную технологии скорее всего не нужно будет обновлять текущее оборудование, так как запросы ресурсов практически не возрастут.

Технология контейнеризации становится все популярнее из-за удобства при командной разработке и развертывании приложения на сервере, так как она позволяет разворачивать все окружение на разном оборудовании без каких-либо изменений. Это позволяет повысить производительность разработчиков за счет устранения межсетевых зависимостей. Каждый контейнер можно рассматривать как отдельный микросервис, а значит его можно без проблем обновлять и улучшать, не прикасаясь к другим контейнерам. Достигается это также за счет удобного управления версиями, позволяющего отслеживать обновления контейнеров, следить за различиями между ними и т.д.

Как правило, контейнер включает в себя не только одно единственное приложение, а также и все его необходимые для нормальной работы зависимости. Это очень облегчает процесс переноса контейнера из одной среды в другую. И тут даже неважно, насколько различаются эти среды, так как большинство контейнерных технологий основаны на открытых стандартах, а значит могут работать во всех основных дистрибутивах операционных систем Linux, Microsoft Windows и Mac OS.

Такое большое количество преимуществ делает технологию контейнеризацию очень интересной не только в мире веб-разработки, но и во всех сферах разработки программного обеспечения и анализа данных. Используя эти технологии, разрабатываемые проекты становятся более безопасными, облегчается их развертывание и масштабирование. Рост и развитие технологий контейнеризации последние года по праву считается одним из самых важнейших трендов.

Сейчас уже есть достаточно много различных реализаций контейнерной технологии, однако особой популярностью пользуются два вида реализаций:

– ориентированные на создание практически полноценных экземпляров операционных систем;

– варианты, сфокусированные на изоляции отдельных сервисов с минимальным жизнеспособным операционным окружением.

В первом случае контейнер очень похож на виртуальную операционную систему. Такие контейнеры хоть и используют ядро хоста, но запускают полную систему Init. Это позволяет в одном контейнере выполнять сразу несколько процессов. Примером могут быть контейнеры LXC и LXD.

Второй вариант – контейнеры отдельных приложений. На данный момент это самый современный и распространенный вид реализации контейнеров. Такие контейнеры используют ядро хоста, но включают в себя лишь минимальное окружение. Это позволяет выполнять только один процесс, но при этом затрачивая гораздо меньше ресурсов. Идеология «один контейнер – одно приложение» позволяет повысить масштабируемость проекта, что очень важно в мире современных сложных систем. Отчасти благодаря этому данная технология широко распространилась в настоящее время.

Итак, становится очевидным преимущество подобных систем при развертывании приложений, управлении приложениями и групповой разработке. Как следствие этого признания, стало появляться специализированное программное обеспечение для автоматизации развертывания, масштабирования и управления контейнеризованными приложениями. Таким инструментом стал Docker. И в целом, в последнее время слово «Docker» стало чем-то вроде синонима слова «контейнеризация».

2.2.2.1 Docker

Docker – программное обеспечение с открытым кодом, предназначенное для автоматизации процесса развертывания контейнеров и управления приложениями в средах, использующих технологии контейнеризации (логотип Docker представлен на рисунке 2.2). С помощью этого инструмента можно «упаковать» все приложение и его окружение в контейнер, который в дальнейшем можно развернуть на любой системе, поддерживающей контейнеризацию и технологию cgroups в ядре операционной системы [4]. После запуска контейнеров Docker также предоставляет среду для дальнейшего управления ими.



Рисунок Настройка среды разработки и выполнения веб-приложений .7 – Логотип docker

Сам Docker изначально был разработан для более быстрой развертки приложений. Он позволяет отделить приложение от его инфраструктуры и обращаться с инфраструктурой, как с управляемым приложением. С помощью легковесной платформы контейнерной виртуализации Docker позволяет разработчикам быстрее выкладывать код и тестировать его, развертывать приложения. В результате значительно уменьшается время между написанием кода и его запуском.

Docker позволяет в своем ядре запускать практически любое приложение, безопасно изолируя его в контейнере. Подобная безопасная изоляция позволяет запускать на одном хосте множество контейнеров одновременно.

Docker состоит из трех частей (рисунок 2.3):

1. Docker daemon – сердце Docker’a. Это сервис, который работает на хост-машине и позволяет скачивать и заливать образы, запускать на их основе контейнеры, следить и управлять запущенными контейнерами, настраивать сеть между ними и собирать логи. Именно эта служба создает образы контейнеров, хотя с первого взгляда может показаться, что этим занимается docker-client.

2. Docker client – это консольная утилита для быстрого и простого управления службой Docker. Утилита работает как непосредственно с хоста, так и с помощью HTTP-запросов. При сборке образа клиент практически не принимает участия, лишь архивируя локальную папку в tar.gz и передавая ее по сети службе Docker, которая и делает всю работу. Именно из-за такой организации передачи данных, тяжелые образы лучше собирать локально.

3. Последний элемент экосистемы – Docker Hub. Это веб-ресурс, на который все разработчики могут загружать свои образы, а другие разработчики могут их использовать. Для типизированных и популярных задач Docker выпускает собственные официальные образы, которые зачастую поддерживаются лучше, чем образы от сторонних разработчиков.

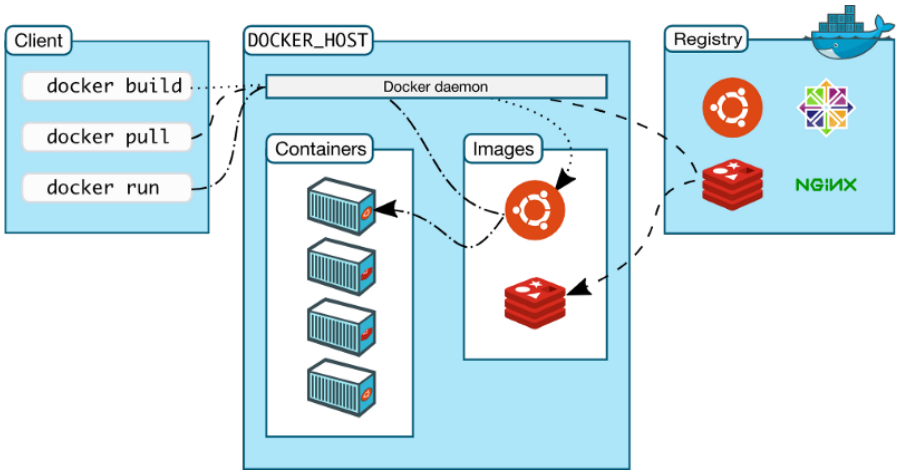


Рисунок Настройка среды разработки и выполнения веб-приложений .8 – Клиент Docker и другие механизмы экосистемы

Два важных понятия в Docker – это образ и контейнер.

Docker-образ – это шаблон, доступный только для чтения. На его основе в конечном итоге создается контейнер. Каждый образ состоит из нескольких уровней, а Docker использует Union File System для объединения этих уровней в один образ. Это позволяет файлам и директориям из разных файловых систем, а точнее разным ветвям, прозрачно накладываться, создавать когерентную файловую систему.

Одна из причин компактности и легковесности Docker – это использование таких уровней. Каждый раз, когда разработчик изменяет образ, например, обновляет приложение на актуальную версию, то в образе обновляется только один уровень. При этом не происходит замены всего образа или его повторной пересборки, как было бы в случае с виртуальной машиной. Как следствие этому – нет необходимости рассылать всем полную новую

версию образа, раздается только обновление. Это позволяет распространять образы гораздо проще и быстрее.

В основе каждого образа находится базовый образ операционной системы, которая выбирается на основании требований приложения, запускаемого в данном контейнере. Чаще всего контейнеры строятся на различных дистрибутивах Linux, но многие специфические приложения построены и на других операционных системах, в том числе и Microsoft Windows. Docker позволяет также построить новый образ на основе уже существующего образа, например, можно использовать образ Nginx как базовый образ, для своих веб-приложений. Среди популярных базовых образов можно выделить python, alpine и ubuntu.

Шаги описания для создания этих образов называются инструкциями. Каждая инструкция создает новый образ или уровень. В данном контексте инструкциями будут следующие действия:

– создание переменной окружения;

– добавление файла или директории;

– указания, что нужно запускать, когда запускается контейнер, основанный на этом образе;

– запуск любой команды.

Например, если создается приложение для решения задач машинного обучения, то, используя инструкции, можно включить в образ библиотеки NumPy, Pandas и Scikit-learn.

Эти инструкции хранятся в Dockerfile. Когда происходит сборка контейнера, Docker считывает этот файл, выполняет инструкции и возвращает готовый конечный образ.

И, наконец, обычно в образе поверх всех остальных содержится еще один тонкий слой, позволяющий изменять хранящиеся в нем данные. Это довольно небольшой по объему слой, но именно в нем содержится программа, которую нужно будет запускать в контейнере.

В этом и состоит суть преобразования образа в контейнер. При построении Docker-контейнера на основе образа происходит добавление верхнего слоя, для которого разрешена запись. Все файлы, полученные в результате работы приложения, пишутся именно в этом слое.

Выполняя построение контейнера, Docker выполняет последовательно следующие шаги:

1. Проверяет наличие образа на локальной машине.

2. Если образа нет, то Docker скачивает его с Docker Hub.

3. Когда образ получен, Docker использует его для создания контейнера.

4. Инициализирует файловую систему и монтирует read-only уровень.

5. Создает сетевой интерфейс, который позволяет Docker’у общаться с хост машиной.

6. Находит и задает IP-адрес.

7. Запускает приложение.

8. Подключается и логирует стандартный вход, вывод и поток ошибок приложения, чтобы можно было отслеживать как работает приложение.

Docker-контейнеры можно сравнить с обычными программами. В конце концов, каждый контейнер действительно является программой, так как представляет собой набор инструкций, который выполняется на некотором процессоре, обрабатывая при этом какие-то данные.

Во время работы контейнера, в нем происходит запуск определенной программы. Благодаря использованию данного инструмента, на одном хосте можно запустить одновременно множество контейнеров. Как и все прочие программы, контейнеры можно запускать, останавливать и удалять.

Рассмотрев все преимущества данного решения, мы приняли решение организовать окружение разрабатываемого приложения на контейнерной технологии виртуализации, используя Docker. Подобное решение позволит в будущем без каких-либо проблем переносить приложение между различными серверами и компьютерами разработчиков, а также разворачивать его за считанные минуты.

При такой построении окружения будут решены все проблемы, связанные с несоответствием окружения веб-приложения у разных разработчиков. Каждый член команды сможет запустить Docker-приложение у себя, получив идентичное окружение тому, которое будет сформировано на сервере и на компьютерах других разработчиков.

Разбивка окружения по разным контейнерам позволит соблюсти требование к разделению задач по разным серверам. При этом проект финансово будет менее затратен, так как нет необходимости арендовать несколько виртуальных частных серверов – все контейнеры можно без проблем запустить и на одном VPS.

Построение окружения веб-приложения на данной технологии также не повлечет значительного увеличения требований к ресурсам арендуемого сервера. Так как контейнеры запускаются без дополнительной нагрузки гипервизора, то от аппаратной составляющей сервера получится добиться гораздо большего, при меньших затратах.

3 Формирование окружения для разработки сервиса

Любая разработка всегда начинается с создания окружения для веб-приложения, так как без него невозможно ни запустить приложение, ни протестировать его работоспособность. В нашем случае окружение строится на базе контейнеров Docker, поэтому мы начинаем с установки этого инструмента.

3.1 Установка Docker

Docker официально поставляется в двух редакциях:

– Community Edition (CE) – полностью бесплатная версия, которая поддерживается сообществом разработчиков;

– Enterprise Edition (EE) – обладает дополнительными сертификациями для некоторых платформ, дополнительно поддерживается компанией Docker, но при этом ее использования платное.

В создаваемом веб-приложении отсутствует необходимость в дополнительных сертификациях, поэтому для формирования окружения будет использована бесплатная версия Community Edition.

Так как Docker является кроссплатформенным решением, то на его сайте доступны ссылки для скачивания под все популярные платформы. В данный момент разработка проекта ведется на операционных системах Microsoft Windows и Linux. В случае с Windows, установка не вызывает вопросов, так как осуществляется стандартным способом для Win-приложений, через диалог со скаченным установщиком. В случае с платформой Linux установку удобнее всего произвести из репозитория.

Docker по умолчанию входит в репозитории многих Linux-систем, однако версии в них могут быть неактуальными. Поэтому разработчики Docker ведут собственный репозиторий и рекомендуют выполнять установку из него.

Для добавления репозитория в источник приложений Linux необходимо выполнить следующий набор команд:

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg
sudo apt-key add –
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository “deb [arch=amd64]
https://download.docker.com/linux/ubuntu lsb_release -cs) stable"
```

В результате репозиторий был успешно добавлен и после получения новейших версий пакетов командой `apt update` можно установить Docker и сопутствующее программное обеспечение из добавленного репозитория с помощью стандартной команды:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Чтобы убедиться, что Docker установлен и запущен, попробуем вывести его версию. Сделать это можно при помощи команды `docker version`. Результат выполнения команды представлен на рисунке 3.1.

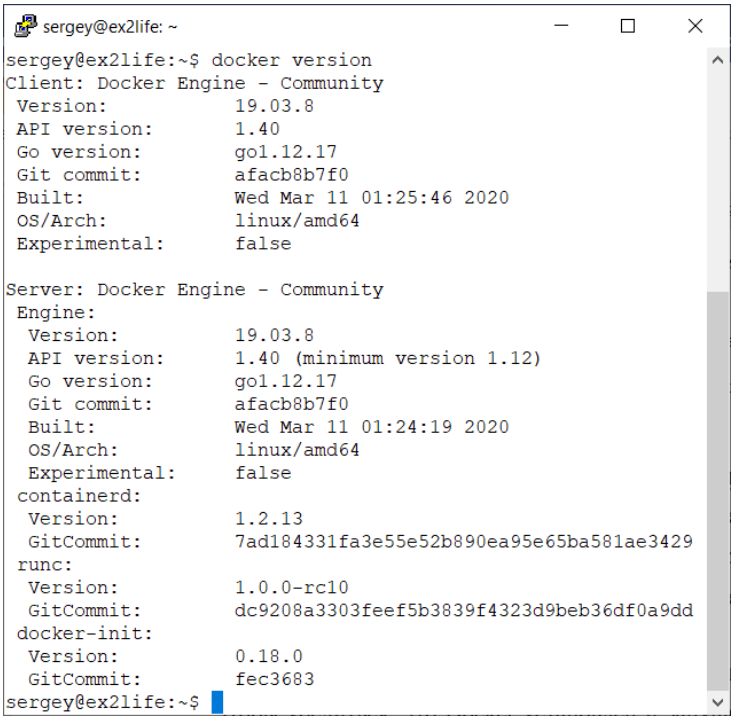


Рисунок Формирование окружения для разработки сервиса.9 –Версия установленного Docker

Начальная конфигурация Docker уже позволяет запускать образы контейнеров из Docker Hub, а также создать собственные образы и использовать их. Но тут есть одна сложность, из-за которой время разработки может несколько увеличиться.

Docker-образ фактически является просто набором файлов, которые упакованы и готовы к изолированному запуску. Можно сказать, что образ – это абстракция. И, как в случае с абстракцией в программировании, здесь для получения конкретной реализации также требуются дополнительные данные. В случае веб-приложений необходимо «пробросить порты», т.е. указать соответствие внутренних портов контейнера с портами хостовой машины, а также настроить связь файловых систем для контейнера и хоста. В некоторых случаях можно указать конкретную команду, которую необходимо выполнить либо во время создания контейнера, либо после его запуска. И это лишь малая часть того, что нужно выполнять для каждого контейнера.

Как правило, большинство веб-приложений требует нескольких контейнеров, причем связанных между собой. Подобное безусловно позволяет сделать один только Docker, все дополнительные параметры передаются как аргументы командной строки. Команду можно сохранить для дальнейшего использования, или даже написать несколько скриптов для запуска всех контейнеров с необходимыми параметрами или их остановки. Но этот вариант сложен и неудобен. Для автоматизации подобных задач разработчики Docker предлагают инструмент Docker Compose.

3.2 Установка Docker Compose

Docker в стандартной конфигурации позволяет управлять отдельно взятым контейнером. Инструмент docker-compose позволяет управлять целым набором контейнеров, из которых состоит распределенное окружение. При этом управление осуществляется не только контейнерами, но и связями файловых систем, сетью, т.е. всеми связанными с контейнерами настройками.

В качестве примера можно рассмотреть простейшее веб-приложение, состоящее всего из двух контейнеров (например, «web» с контентом и «db» с данными). Для правильной работы этих контейнеров нужно как минимум:

– создать новую пользовательскую сеть (иначе контейнеры не будут видеть друг друга по именам);

– запустить контейнер с данными, дать ему имя db и подключить к сети;

– запустить контейнер с веб-контентом, назвать его web и также подключить к сети [25].

Как уже было упомянуто, для решения этих задач можно создать bash-скрипт, однако в дальнейшем его постоянно придется переделывать. Будут добавляться новые контейнеры, изменяться порядок запуска существующих. При этом разработчику добавится много головной боли, так как необходимо учитывать, что какие-то контейнеры уже собраны, а какие-то нужно собрать, сеть контейнеров уже может быть создана, а может еще нет. В случае связи контейнеров между собой необходимо также учитывать, что какие-то контейнеры необходимо запустить первыми, а остальные потом и т.д.

Docker-compose – это инструмент для запуска многоконтейнерных Docker-приложений, позволяющий разработчику описать все распределенное приложение целиком, со всеми контейнерами, сетями и зависимостями, Docker-файлами. Для управления распределенным приложением в docker-compose используется специальный конфигурационный файл. Создание и запуск всех сервисов осуществляется с помощью одной простой команды. Таким образом, docker-compose отлично подходит для формирования, тестирования и настройки окружения, а также для непрерывной разработки веб-приложений.

Использование docker-compose обычно разделяется на три этапа:

1. Задание окружения приложения в Docker-файлах. Сделать это можно в любой папке проекта, но традиционно для каждого элемента окружения создается собственная директория, содержащая Dockerfile.

2. Определение с помощью конфигурационного файла docker-compose.yml сервисов, из которых будет состоять распределенное приложение. В последствии они будут совместно запущены в изолированном окружении.

3. Выполнение команды `docker-compose up` для сборки и запуска приложения.

На платформах Windows и Mac OS docker-compose автоматически устанавливается вместе с Docker. На Linux для установки этого инструмента

необходимо вручную скачать утилиту, сделать ее исполняемой и скопировать в директорию исполняемых программ:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Чтобы убедиться, что docker-compose установлен и работает должным образом, попробуем вывести его версию. Команда и результат ее исполнения представлены на рисунке 3.2.

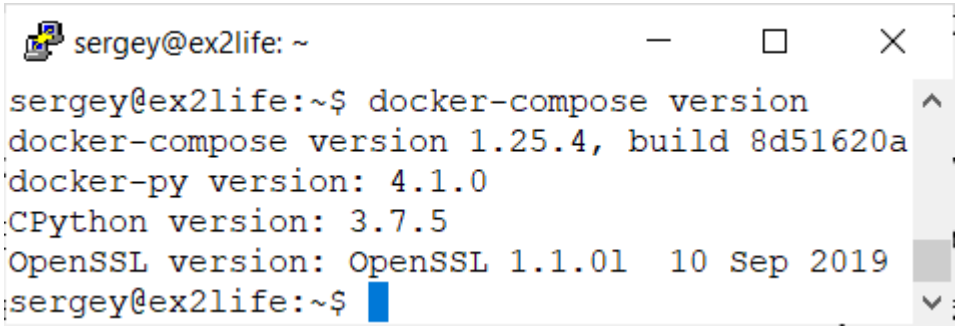


Рисунок Формирование окружения для разработки сервиса.10 – Версия установленного docker-compose

После установки необходимого программного обеспечения можно перейти к созданию окружения.

3.3 Настройка конфигурации окружения

Так как ранее разрабатываемое приложение было размещено на хостинге, то для локальной работы каждый из разработчиков использовал привычное ему минимальное окружение: это веб-сервер Apache с модулем PHP и сервер базы данных. При этом часть проблем окружения решались хостингом, например использованием более безопасного протокола HTTPS или поддержкой утилиты phpMyAdmin для администрирования СУБД MySQL.

Сейчас целью является построение единого окружения для всех разработчиков. При этом после разворачивания такого окружения на чистой

системе с установленными Docker и docker-compose разработчик должен получить полностью готовую среду для запуска, разработки и тестирования веб-приложения, которая будет содержать все необходимое для дальнейшей работы на боевом сервере.

Разрабатываемый нами сервис основывается на PHP-фреймворке Laravel с использованием базы данных MySQL. Минимальное окружение для работы такого приложения состоит из двух элементов:

– HTTP-сервер с возможностью генерации динамических PHP-страниц;

– MySQL-сервер для работы с базой данных.

Этих компонентов достаточно для запуска Laravel-приложения, но мало для полного цикла разработки. Поскольку каждому разработчику необходимо управлять зависимостями проекта и администрировать базу данных, то в конфигурацию добавится еще два элемента:

– система администрирования phpMyAdmin;

– пакетный менеджер Composer.

Эти компоненты позволят обеспечить удобную работу над проектом для каждого разработчика. Осталось позаботиться о размещении приложения на веб-сервере.

На данный момент использование протокола HTTPS является практически обязательным требованием ко всем веб-сайтам. Сайты, загружаемые по HTTP, большинством браузеров индексируются как небезопасные, о чем сообщается клиентам. А значит, есть необходимость в проработке вопроса сертификации.

Традиционно сервер сертификации вручную настраивается на сервере, однако для удобства решено включить его в окружение, чтобы упростить развертывание или перенос приложения на серверах. Именно поэтому, в приложении добавится пятый компонент – сервис сертификации Certbot.

Все эти компоненты необходимо реализовать в создаваемом окружении. Первым делом необходимо создать папку проекта, в которой и будут храниться

инструкции по созданию окружения и в дальнейшем само приложение. В корне проекта необходимо создать два файла:

1. Docker-compose.yml – этот файл по назначению похож на Dockerfile для контейнера, но он позволяет задать конфигурацию для распределенного приложения целиком. В нем будут описаны используемые сервисы, их базовые образы и вся сопутствующая информация.

2. Файл .env, в котором будут храниться необходимые переменные, как это рекомендует официальная документация.

Теперь можно приступать к созданию окружения. Начать лучше с реализации самого важного компонента для любого веб-приложения – HTTP-сервера с возможностью генерации динамических PHP-страниц.

3.3.1 HTTP-сервер и менеджер процессов PHP

Два незаменимых инструмента, без которых не обходится ни одно веб-приложение, – это HTTP-сервер и менеджер процессов PHP. HTTP-сервер обрабатывает запросы от браузеров клиентов, отвечает на них или перенаправляет их туда, где они могут быть обработаны, а менеджер процессов PHP, в свою очередь, занимается генерацией динамических страниц. Два этих инструмента часто идут «бок о бок», например в HTTP-сервере Apache имеется модуль php, подключаемый к серверу. Однако в нашем проекте было принято решение их разделить.

Концепция разделения HTTP-сервера от менеджера процессов PHP может показаться чем-то странным, но так как Docker создает контейнеры, предназначенные для какой-то единственной задачи, то эти инструменты все же стоит разделить. Именно это и позволяет реализовать ту самую гибкость, которой славится Docker. Поэтому данный компонент будет разбит на два: первый контейнер работает с веб-сервером Nginx и обрабатывает веб-запросы, а второй контейнер для менеджера процессов PHP-FPM обрабатывает запросы приложений. Схематическое строение планируемой системы представлено на рисунке 3.3.

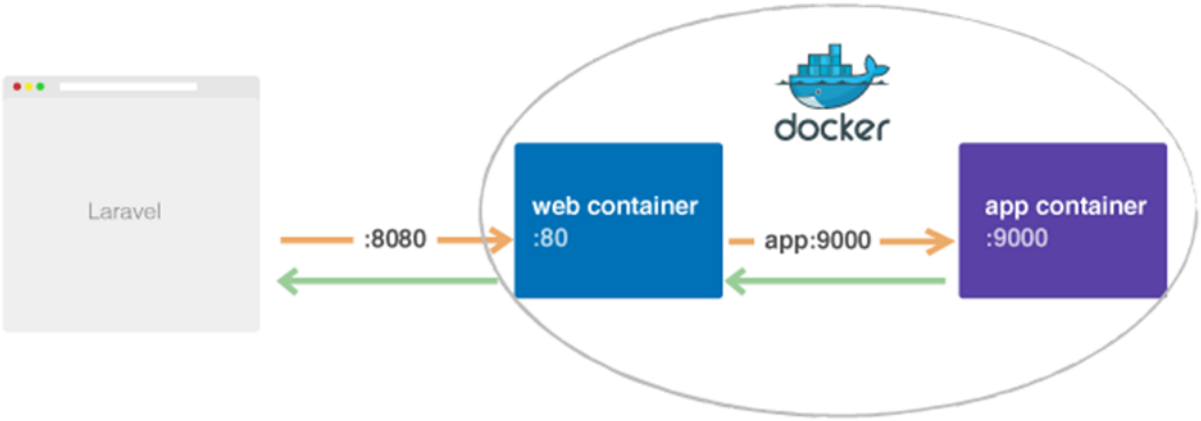


Рисунок Формирование окружения для разработки сервиса.11 Web и App контейнеры

На данной схеме контейнер web обрабатывает все HTTP-запросы, а при необходимости обращается к контейнеру app, который будет обрабатывать запросы приложения и генерировать динамические страницы. Создадим два контейнера для реализации данного компонента.

3.3.1.1 Настройка контейнера web

Поскольку docker-compose постоянно развивается, то его конфигурационный файл всегда начинается с указания версии. На данный момент последняя и рекомендуемая версия 3.x. Эта версия была разработана для обеспечения совместимости между compose и режимом Swarm в Docker Engine.

Необходимо выбрать версию, исходя из установленного docker-engine. Разработчики compose рекомендуют использовать самый свежий вариант, из совместимых с Docker. На данный момент это версия 3.7.

```
version: '3.7'
```

При необходимости можно указать дополнительные параметры распределенного приложения, например, название сети для создаваемого окружения. В случае, если параметры не заданы, то все необходимое сгенерируется автоматически, поэтому, после ключа `services`, можно

начинать перечисление всех сервисов, которые должны быть созданы и запущены в данном проекте. Создадим сервис, ответственный за создание контейнера web, который будет выполнять роль HTTP-сервера.

```
web:
    image: nginx:1.17.9-alpine
    container_name: financier_web
    volumes:
      - ${APP_PATH_HOST}:${APP_PATH_CONTAINER}
      - ${NGINX_CONFIG}:${NGINX_CONFIG_CONTAINER}
    ports:
       - "80:80"
       - "443:443"
    restart: unless-stopped
```

В ключе `image` задается образ, на основе которого собирается контейнер. В разрабатываемом проекте используется официальный образ от Docker – сервер Nginx. Он достаточно оперативно обновляется, к тому же имеет alpine-версию, которая за счет ядра Linux Alpine использует существенно меньше дискового места, чем аналогичные образы, включающие ядра других операционных систем, например Ubuntu. Использование alpine-версий сборок позволяет значительно сэкономить суммарный конечный вес проекта на веб-сервере, что важно при аренде VPS, где предоставляется недостаточно много места на жестком диске.

Для этого образа в проекте принудительно задается желаемая версия с помощью тега `1.17.9-alpine`. В случае, когда выйдут новые образы Nginx, то будет всегда загружаться одна и та же версия, пока разработчик не обновит требования в конфигурационном файле. Это позволяет избежать проблем с неожиданной несовместимостью.

С помощью ключа `container_name` задается имя контейнера, по которому после запуска проекта можно обращаться к конкретным контейнерам, не проверяя какое имя им было сгенерировано. Для этого контейнера было задано имя `financier_web`.

Ключ `ports`, как уже было сказано ранее, осуществляет проброс портов. Для поддержки работы по протоколу HTTP, приложению нужен доступ к 80 порту, а для HTTPS необходим 443 порт. Поскольку пользователь веб-приложения отправляет запросы на хостовую машину, то все запросы с ее 80 и 443 портов необходимо перенаправлять в контейнер web, что и сконфигурировано здесь данным ключом.

С помощью ключа `restart` указывается поведение контейнера при перезагрузке. Значение `unless-stopped` указывает, что при запуске Docker необходимо запустить контейнер, если ранее он был запущен и не был остановлен вручную [28]. По сути, это стандартное поведение для веб-приложений, размещенных на сервере. В случае сбоя хостовой машины она вероятнее всего перезагрузится, и при такой настройке приложение вновь запустится в работу, не требуя дополнительного внимания администратора. Для всех дальнейших контейнеров значение ключа `restart` будет `unless-stopped`.

Последний не рассмотренный ключ – `volumes`. С помощью него задаются связи между файловыми системами хоста и конкретного контейнера [17]. Однако, в этом случае для удобства и гибкой настройки проекта принято решение не указывать в конфигурационном файле пути, а вынести их в созданный ранее файл .env.

```
#app
APP_PATH_CONTAINER=/var/www/html/site
APP_PATH_HOST=./src
#web
NGINX_CONFIG=./web/vhost.conf
NGINX_CONFIG_CONTAINER=/etc/nginx/conf.d/default.conf
```

Параметры вида ${Параметр} будут запрошены инструментом docker-compose при сборке проекта из этого файла в автоматическом режиме. Полный список переменных окружения приведен в приложении А.

В результате такого «проброса» директорий контейнер web будет считать, что по пути `/var/www/html/site` находятся файлы из папки src из

корня проекта. В папке `src` будет размещено разрабатываемое Laravel-приложение, т.е. приложение будет находится на хостовой файловой системе, но сам контейнер будет считать, что файлы находятся непосредственно в нём. Второй маршрут ключа `volumes` позволяет заменить стандартный конфигурационный файл сервера Nginx на написанный нами фай, который находится в папке web. Это позволит более гибко настраивать HTTP-сервер, но начать его конфигурацию будет лучше после создания контейнера app.

3.3.1.2 Настройка контейнера app

Как было сказано ранее, контейнер app будет обрабатывать запросы приложения, полученные от контейнера web, т.е. именно этот контейнер будет выполнять весь PHP-код и генерировать динамические страницы с помощью протокола FastCGI. Это клиент-серверный протокол, позволяющий организовать взаимодействие между веб-сервером и приложением. Именно благодаря этому протоколу и стало возможным вынести приложение в отдельный контейнер, так как в отличии от своего родителя, протокола CGI, приложение общается с веб-сервером через Unix Sockets или по протоколу TCP/IP, что позволяет избежать необходимости работы их на одной машине.

Отделение приложения от веб-сервера открывает большие возможности по масштабированию нагрузки проекта, так как при необходимости позволяет развернуть дополнительные контейнеры, содержащие менеджер задач FastCGI, даже на других серверах и часть запросов перенаправлять им.

Конфигурация сервиса, создающего контейнер app, примет следующий вид:

```
app:
    build: ./app
    container_name: financier_app
    volumes:
      - ${APP_PATH_HOST}:${APP_PATH_CONTAINER}
    working_dir: ${APP_PATH_CONTAINER}
    restart: unless-stopped
    user: www-data
```

Данный контейнер получил название `financier_app`. Как и контейнер web, он основан на официальном образе, но при этом в конфигурации нет ключа `image` с указанием образа, который необходимо собрать.

Связано это с тем, что для правильной работы Laravel необходимо больше PHP-зависимостей, чем предоставляет официальный образ php-frm, реализующий работу менеджера задач FastCGI. Кроме того, нам нужен дополнительный инструмент npm для сборки фронтенда проекта и управления установленными пакетами Node.js, чтобы каждому разработчику не пришлось устанавливать этот инструмент у себя. Поэтому мы строим свой собственный образ на основе официального. В конфигурации для этого задан ключ `build`, который указывает директорию с образом, который необходимо создать.

В этой директории `app`, создается файл `DockerFile`, который будет содержать следующие инструкции для сборки нашего собственного образа:

```
FROM php:7.4-fpm-alpine
RUN apk update && apk add npm
ADD https://raw.githubusercontent.com/mlocati/docker-php-
extension-installer/master/install-php-extensions /usr/local/bin/
RUN chmod uga+x /usr/local/bin/install-php-extensions && sync
&& \ install-php-extensions gd mysqli pdo_mysql zip
RUN apk --no-cache add shadow && usermod -u 1000 www-data &&
groupmod -g 1000 www-data
RUN chown -R www-data:www-data /var/www
```

В этих инструкциях используемый официальный образ php:7.4-fpm-alpine дополняется до необходимого, выполняя следующие шаги:

1. Происходит обновление версий пакетов доступных из репозитория и устанавливается последняя версия инструмента npm.

2. В образ добавляется скрипт install-php-extensions, позволяющий гораздо быстрее выполнять установку php зависимостей.

3. Используя данный скрипт, в образ контейнера включаются все необходимые для работы Laravel php-зависимости.

4. Выполняется настройка прав доступа, необходимая для правильного взаимодействия с файловыми системами хостовых машин, работающих на операционных системах семейства Unix.

В момент первого запуска распределенного окружения будет собран новый образ с помощью инструкций, описанных ранее. В дальнейшем для запуска контейнера будет использоваться скомпилированный собственный образ.

В контейнер app монтируется директория с Laravel-приложением по пути /var/www/html/site, чтобы этот контейнер тоже имел доступ к файлам для обработки скриптов.

Ключ `working_dir` задает рабочий каталог контейнера. В данном случае рабочей директорией устанавливается каталог, содержащий файлы Laravel-приложения. Сделать это необходимо, так как в Laravel много операций осуществляется через командную строку, например генерация моделей, контроллеров, работа с Node.js. Для этих задач необходимы PHP-утилиты для работы с CLI-интерфейсом, artisan и npm для установки, обновления, удаления пакетов, а также сборки фронтенда. Данные инструменты в текущей конфигурации окружения будут работать внутри контейнера app, а значит для их использования необходимо будет войти в контейнер, используя команду `docker exec`. Она используется для запуска команд внутри контейнера или даже позволяет использовать его консоль непосредственно из хостовой машины, используя аргумент -it. Рабочий каталог, заданный ключом `working_dir`, является директорией, в которой эти команды будут выполнены по умолчанию.

Также, для совместимости с родительскими Unix-системами, все эти команды необходимо выполнять от имени пользователя www-data, созданного при сборке контейнера. Эта настройка контейнера задается с помощью ключа user в конфигурационном файле docker-compose.

Контейнер web должен переадресовывать запросы контейнеру app, поэтому он должен знать адрес, по которому необходимо перенаправлять сетевой трафик. Сделать это можно с помощью ключа `links` в конфигурации контейнера web.

```
links:
```

```
      - app
```

Благодаря этому можно, не зная адреса контейнера app, обращаться к нему с HTTP-сервера по имени и номеру порта 9000, используемого для PHP-FPM. Данная особенность будет использована при создании конфигурации HTTP-сервера.

3.3.1.3 Конфигурация HTTP-сервера

Для хранения конфигурации, как было описано ранее, используется директория web. В ней необходимо создать файл vhost.conf, который будет импортирован в контейнер в качестве стандартной конфигурации. Файл конфигурации для разрабатываемого сервиса примет следующий вид:

```
server {
    server_name fin.ex2life.cf;
    listen 80;
    index index.php index.html index.htm;
    root /var/www/html/site/public;
    location / {
        try_files $uri /index.php?$args;
    }
    location ~ \.php$ {
        try_files $uri $uri/ /index.php last;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
    location ~* .php/ { rewrite  (.*.php)/ $1 last; }
}
```

В приложении Laravel нет необходимости выдавать статические страницы, так как все запросы обрабатываются через единую точку входа в виде скрипта index.php, поэтому данный файл конфигурации перенаправляет все запросы непосредственно контейнеру с приложением. Адрес контейнера задается в параметре `fastcgi_pass`. Связь между контейнерами осуществляется внутри их собственной сети, созданной инструментом docker-compose, а значит контейнеру app нет необходимости проброса портов в

хостовую машину, контейнер будет найден даже если находится на другой хостовой машине с сетью, объединённой мостом с текущей.

3.3.2 Настройка контейнера СУБД MySQL

Контейнер, содержащий сервер для управления базой данных, получил название `financier_db`. Первоначально разрабатываемый веб-сервис использовал СУБД MySQL версии 5.7. При рефакторинге приложения на Laravel было принято решение перейти на новую более производительную версию MySQL 8.0.19. В связи с этим конфигурация сервиса database примет следующий вид:

```
database:
    image: mysql:8.0.19
    container_name: financier_db
    restart: unless-stopped
    env_file:
        - .env_mysql
    volumes:
      - ${DB_PATH_HOST}:/var/lib/mysql
      - ${DB_CONFIG}:/etc/mysql/conf.d
```

Важный момент, на которых хотелось бы обратить внимание – это монтирование в контейнер папки из файловой системы хостовой машины, в которой будет храниться база данных.

Конечно, базы данных можно хранить и внутри контейнера. Однако это ненадежный вариант, так как контейнер может быть разрушен и данные будут потеряны. Локальную же базу данных повредить гораздо сложнее, а ее резервное копирование и перенос на другой сервер выполнить проще. Поэтому файлы базы данных будут храниться внутри директории проекта в папке databases.

Для более гибкой настройки сервера базы данных конфигурационный файл также вынесен на хостовую машину в директорию db.

Чтобы обеспечить безопасность данных, для сервера необходимо установить пароль администратора (пользователя root), а также задать имя пользователя, пароль и имя базы данных, которые будет использовать Laravel-

приложение. Для этого в ключе `env_file` указывается файл, содержащий необходимые данные:

```
MYSQL_DATABASE=financier
MYSQL_ROOT_PASSWORD=rootpass
MYSQL_USER=laravel
MYSQL_PASSWORD=laravelpass
```

Важно не забыть зарегистрировать ссылку на контейнер database для контейнера app, так как именно он будет обращаться к базе данных.

```
links:
  - database
```

Это позволит указать в конфигурации Laravel-приложения имя контейнера с сервером базы данных, вместо его полного адреса.

СУБД готова к работе, но для удобства администрирования необходимо добавить в проект инструмент phpMyAdmin.

3.3.3 Настройка контейнера phpMyAdmin

Фактически система phpMyAdmin для администрирования СУБД представляет собой директорию, содержащую классическое веб-приложение. В связи с этим, может показаться довольно странным решение вынесение этого инструмента в отдельный контейнер. Однако, такое решение позволяет поддерживать административную панель в самых новых версиях без участия разработчика. При выходе новых версий образ контейнера и веб-приложение будет обновлено.

Конфигурация данного контейнера примет следующий вид:

```
phpmyadmin:
    image: phpmyadmin/phpmyadmin:fpm-alpine
    container_name: financier_pma
    restart: unless-stopped
    volumes:
      - ${PHPMYADMIN_PATH}:${PHPMYADMIN_PATH_CONTAINER}
    working_dir: ${PHPMYADMIN_PATH_CONTAINER}
    links:
      - database:db
```

В целом конфигурация сервиса довольно похожа на описанные ранее. Для данного образа адресом контейнера с СУБД по умолчанию является db, но

так как наш сервис получил название database, то в ключе `links` происходит переназначение адреса.

Выбранный образ включает в себя собственный менеджер задач FastCGI, поэтому необходимо дополнить конфигурацию HTTP-сервера, чтобы все запросы к субдомену pma перенаправлялись в контейнер phpMyAdmin. Данная конфигурация похожа на написанную ранее, поэтому здесь мы ее опустим.

3.3.4 Настройка контейнера Composer

Контейнер, позволяющий работать с менеджером пакетов Composer, в отличие от остальных контейнеров не должен работать постоянно. Он выполняет установку новых версий пакетов при каждом запуске распределенного приложения, что позволяет поддерживать высокий уровень безопасности. Запустив контейнер вручную, можно при необходимости выполнить установку новых пакетов или обновление текущих. Конфигурация этого сервиса имеет следующий вид:

```
composer:
    image: composer:latest
    volumes:
      - ${APP_PATH_HOST}:${APP_PATH_CONTAINER}
    working_dir: ${APP_PATH_CONTAINER}
    command: composer install --ignore-platform-reqs
```

При запуске контейнера в рабочей директории будет выполнена команда, заданная в ключе `command`. После выполнения команды работа контейнера будет остановлена.

3.3.5 Сертификация веб-приложения

Для работы веб-приложения по протоколу HTTPS необходимо получить сертификат SSL. Сам процесс получения сертификата довольно несложный, но так как он обладает ограниченным временем действия, то его необходимо периодически обновлять. В случае переноса приложения на новый сервер сервис сертификации придется настраивать заново. Для упрощения подобных

действий было принято решение включить инструмент для сертификации в окружение и настроить автоматическое получение и обновление сертификатов.

Как правило, получение сертификатов SSL платное, но существует один сервис, позволяющий сертифицировать веб-ресурс бесплатно, – Let's Encrypt. Единственное ограничение, выставленное данным сервисом, это максимальный срок действия сертификата – 90 дней. Но для разрабатываемого проекта, особенно учитывая, что планируется обновлять сертификаты в автоматическом режиме, этого более чем достаточно, поэтому в проекте будет использован именно этот сервис.

Для удобства получения сертификата компанией Let's Encrypt был разработан инструмент certbot, поэтому создадим контейнер, ответственный за работу этого инструмента. Для него был создан следующий сервис:

```
    certbot:
        build: ./certbot
        container_name: financier_certbot
        restart: unless-stopped
        environment:
          - DOMAINS=${CERTBOT_DOMAIN1} ${CERTBOT_DOMAIN2}
          - EMAIL=${CERTBOT_EMAIL}
        volumes:
          - ${CERTBOT_CERTS_PATH}:
${CERTBOT_CERTS_PATH_CONTAINER}:rw
          - ${CERTBOT_WEBROOT_PATH}:
${CERTBOT_WEBROOT_PATH_CONTAINER}:rw
```

Docker имеет официальный образ для реализации контейнера, однако в своем исходном виде он не полностью подходит для решения задач, поставленных в разрабатываемом проекте. Дело в том, что данный образ предназначен для тех же целей, что и сам инструмент, т.е. получение вручную сертификата для веб-сервиса в диалоговом режиме. Поэтому для получения каждого сертификата необходимо вручную запустить контейнер и ответить на вопросы менеджера сертификатов, что в нашем случае не подходит. Для разрабатываемого веб-сервиса нужно получить сертификаты как минимум для двух сервисов: само приложение и панель администрирования phpMyAdmin, а также отслеживать необходимость получения более свежих сертификатов.

В связи с вышесказанным, принято решение собрать собственный образ certbot на основе официального. Dockerfile содержит следующие инструкции:

```
FROM certbot/certbot
ENV DOMAINS=""
ENV EMAIL=""
VOLUME /etc/letsencrypt
VOLUME /webroot
ENTRYPOINT for domain in ${DOMAINS}; do echo "Processing
$domain..."; certbot -t certonly --webroot --agree-tos --non-
interactive -w /webroot -m ${EMAIL} -d $domain; done; \
  while true; do certbot renew -q; sleep 10h; done
```

Для модифицированного уровня с помощью инструкций были объявлены дополнительные переменные среды:

– `DOMAINS` – массив, содержащий список доменов для которых необходимо получить сертификаты;

– `Email` – адрес электронной почты пользователя, на которого необходимо зарегистрировать полученные сертификаты.

Поскольку в модифицированном образе диалог с пользователем не нужен, то мы переназначим точку входа в контейнер, используя команду ENTRYPOINT. Теперь контейнер при своем запуске будет выполнять команду, сформированную нами: для каждого домена в массиве будет сгенерирован новый сертификат, после чего будет запущен бесконечный цикл, каждые 10 часов проверяющий необходимость в обновлении сертификата. В случае, если certbot порекомендует обновить какой-то сертификат, то это произойдет в автоматическом режиме без непосредственного участия администратора сервера.

Теперь можно переконфигурировать HTTP-сервер для работы с использованием полученных SSL-сертификатов для протокола HTTPS. Поскольку на сегодня всё клиентское программное обеспечение умеет работать по защищенному протоколу, то для повышения уровня безопасности исключим возможность работы по незащищенному протоколу, включив обязательную переадресацию клиентов на HTTPS.

```
server {
  listen      80;
```

```
        server_name .fin.ex2life.cf;

        location ^~ /.well-known/acme-challenge/ {
        auth_basic off;
        allow all;
        root /usr/share/nginx/html;
        try_files $uri =404;
        break;
        }

        # Redirect to HTTPS
        location / { return 301 https://fin. ex2life.cf
$request_uri; }
    }
```

Конфигурация, созданная ранее, практически не изменится. В ней только происходит смена порта 80 на 443, а также включение сертификата и ключа к нему:

```
        ssl_certificate
/etc/letsencrypt/live/fin.ex2life.cf/fullchain.pem;
        ssl_certificate_key
/etc/letsencrypt/live/fin.ex2life.cf/privkey.pem;
```

Предварительно необходимо примонтировать директории с сертификатами и ключами непосредственно в контейнер web. Конфигурация для панели phpMyAdmin аналогична, подробно рассматривать ее не будем. Полная конфигурация сервера Nginx приведена в приложении А.

Фактически окружение готово к работе, однако при автоматическом обновлении сертификатов произойдет неочевидная ошибка. Работающий в контейнере web HTTP-сервер Nginx считывает файл конфигурации при работе один раз – при запуске. Из-за этого, даже если certbot примет решение обновить сертификаты, то Nginx так и продолжит работать со старыми сертификатами, что со временем вызовет ошибки у пользователей. Чтобы избежать этой проблемы, добавим для сервиса web следующий ключ:

```
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx
-s reload; done & nginx -g \"daemon off;\"'"
```

В результате добавления данной команды после запуска контейнера в его внутреннем терминале будет запущен на выполнение бесконечный цикл, осуществляющий перезагрузку HTTP-сервера каждые 6 часов. Эта операция занимает менее секунды, что означает, что никто из пользователей не заметит

этого, но позволит поддерживать актуальную версию SSL-сертификата. Полная конфигурация Docker-окружения представлена в приложении А.

3.4 Запуск окружения

После формирования окружения можно произвести пробный запуск. Для этого в директории с проектом необходимо выполнить команду `docker-compose up`. Данная команда запустит процесс сборки всех контейнеров (рисунок 3.4), создаст сеть и запустит окружение в работу.

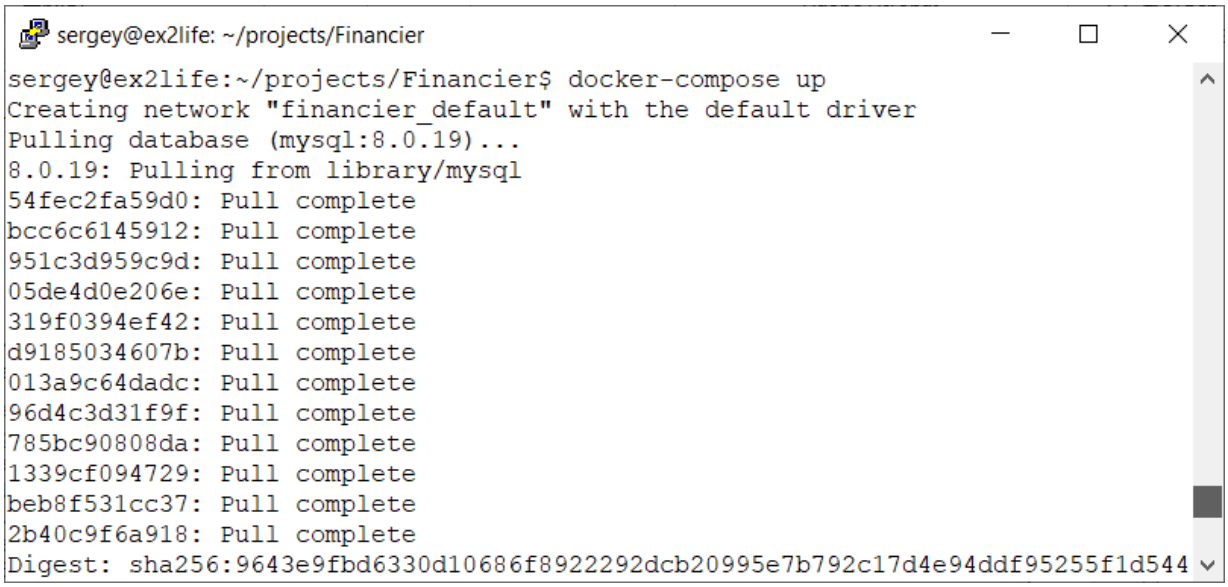


Рисунок Формирование окружения для разработки сервиса.12 – Сборка и запуск docker-приложения

В результате все контейнеры будут запущены. Список запущенных контейнеров можно увидеть с помощью команды `docker ps` (рисунок 3.5).

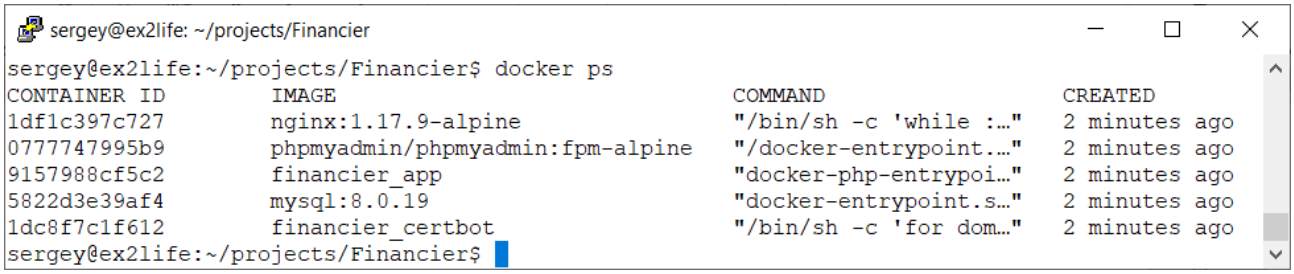
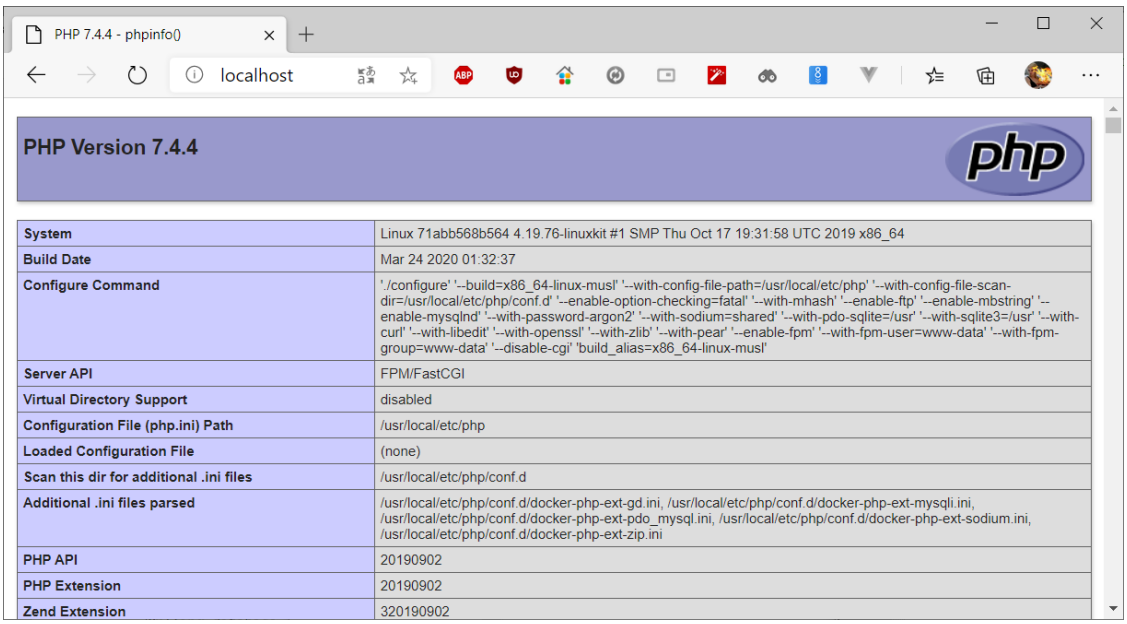


Рисунок Формирование окружения для разработки сервиса.13 – Запущенные контейнеры

Как видно, все контейнеры, кроме завершившего работу Composer, запущены в работу. Убедиться в правильной комплексной работе окружения можно, создав в директории src файл index.php, который будет выводить информацию об установленном PHP.

Итак, HTTP-сервер перенаправил запрос контейнеру app, в котором был выполнен PHP-код, генерирующий HTML-страницу, представленную на рисунке 3.6.



System	Linux 71abb568b564 4.19.76-linuxkit #1 SMP Thu Oct 17 19:31:58 UTC 2019 x86_64
Build Date	Mar 24 2020 01:32:37
Configure Command	'./configure' '--build=x86_64-linux-musl' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-pear' '--enable-fpm' '--with-fpm-user=www-data' '--with-fpm-group=www-data' '--disable-cgi' 'build_alias=x86_64-linux-musl'
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-mysqli.ini, /usr/local/etc/php/conf.d/docker-php-ext-pdo_mysql.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini, /usr/local/etc/php/conf.d/docker-php-ext-zip.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902

Рисунок Формирование окружения для разработки сервиса.14 – Вывод информации о установленном PHP

Построенное нами окружение соответствует требованиям современной веб-разработки и поддерживает удобную и надежную командную работу. Каждый разработчик может клонировать себе проект из GitHub-репозитория и, используя всего лишь одну команду, запустить проект в работу. При этом гарантированно будет получено одинаковое окружение у каждого разработчика.

Также наше окружение соответствует требованиям безопасности и обладает достаточной надежностью. Выход из строя любого из компонентов не повлияет на хранимые данные пользователей. Окружение максимально

эффективно использует ресурсы хостовой машины, что благоприятно скажется при выборе VPS-сервера для аренды.

Имея настроенное рабочее окружение, можно приступать к рефакторингу проекта на фреймворк Laravel и доработке его функционала.

4 Рефакторинг и доработка финансового веб-сервиса

Для повышения качества разрабатываемого веб-сервиса было принято решение произвести его рефакторинг и перейти на архитектуру MVC с использованием фреймворка Laravel.

4.1 Перевод проекта на Laravel

В Laravel управление всеми необходимыми для приложения зависимостями осуществляется с помощью менеджера Composer. Сам каркас приложения также создается с помощью этого инструмента. Для создания проекта достаточно выполнить команду [29]:

```
composer create-project --prefer-dist laravel/laravel src
```

Данная команда выполняется в директории с созданным ранее окружением. В последнего параметре задается директория, в которую будет установлено Laravel-приложение, в нашем случае это папка src.

При выполнении данной команды Composer автоматически скачает Laravel в указанную директорию и запустит процесс установки. После ее завершения в папке будет создан пустой проект, полностью готовый к работе. Так как окружение уже готово и запущено в работу, то убедиться в работе проекта можно уже в веб-браузере. Начальная страница фреймворка имеет вид, представленный на рисунке 4.1.

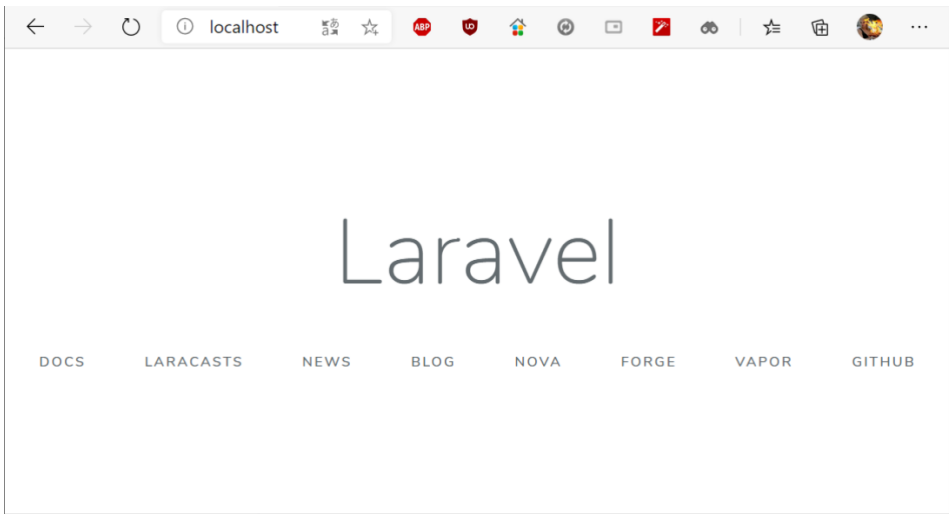


Рисунок Рефакторинг и доработка финансового веб-сервиса.15 – Домашняя страница пустого проекта

После установки Laravel и создания проекта необходимо позаботиться о месте для хранения данных. Ранее был создан контейнер database, содержащий в себе СУБД MySQL и базу данных financier. Теперь необходимо сконфигурировать Laravel-приложение для работы с данной СУБД и созданной базой данных. Все необходимые параметры для подключения к СУБД задаются в конфигурационном файле .env:

```
DB_CONNECTION=mysql
DB_HOST=database
DB_PORT=3306
DB_DATABASE=financier
DB_USERNAME=laravel
DB_PASSWORD=laravelpass
```

Помимо очевидных параметров, таких как название базы данных и имя пользователя с паролем, здесь также задается тип базы данных и ее путь [7]. Задать тип базы данных необходимо, так как Laravel умеет работать не только с СУБД MySQL, но и с другими реляционными СУБД:

– PostgreSQL;

– SQLite;

– Microsoft SQL Server.

Также важно заметить, что путь к базе данных задается не просто в виде IP-адреса, а с помощью ссылки database. Данная ссылка была зарегистрирована в контейнере app при создании окружения:

```
links:
  - database
```

После этого шага взаимодействие с СУБД и базой данных для разработчика завершено. Всю необходимую работу по созданию таблиц и их модификации Laravel будет совершать самостоятельно.

Настроив взаимодействие с базой данных, можно приступить к рефакторингу проекта. Для того, чтобы перенести весь функционал в созданное Laravel-приложение нужно разбить интерфейс и код проекта на три составляющие:

– представления,

– модели,

– контроллеры.

В Laravel определены директории для хранения данных компонентов. Структура проекта представлена на рисунке 4.2.

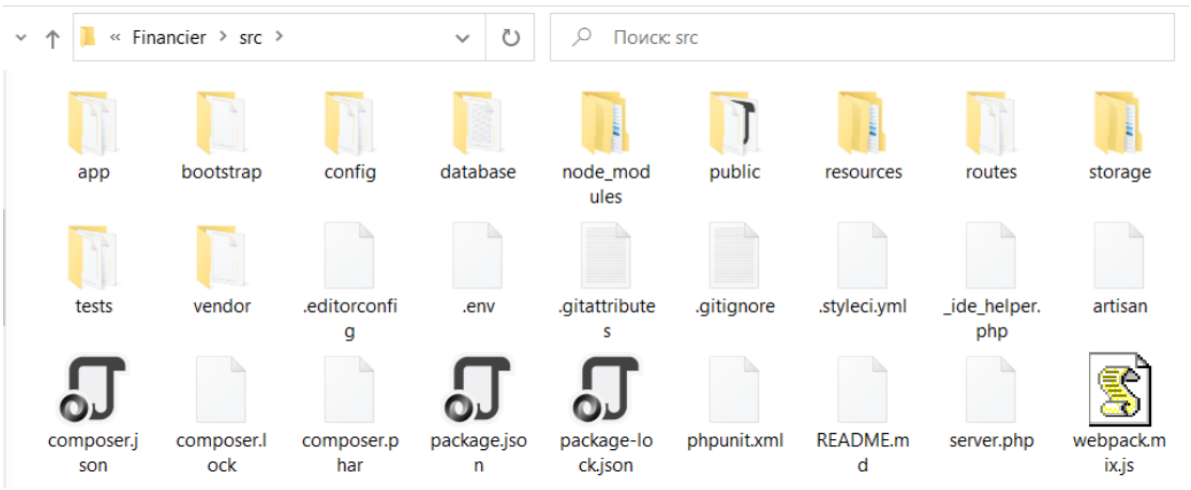


Рисунок Рефакторинг и доработка финансового веб-сервиса.16 – Структура проекта Laravel

Все модели хранятся в директории app, все контроллеры в app\Http\Controllers, а представления в папке \resources\views. В папке resources также хранятся все Sass-стили и сценарии JavaScript.

В веб-сервисе основные модели, такие как компании или группы связанных заемщиков, должны быть изолированы от других пользователей, то есть они должны быть доступны только пользователям, создавшим их. Именно поэтому рефакторинг лучше начать с создания механизма авторизации и регистрации пользователей.

4.2 Рефакторинг компонента авторизации пользователей

Система аутентификации для защиты от неавторизованных пользователей является важной частью веб-приложения. Ранее в разрабатываемом сервисе был разработан собственный модуль авторизации и регистрации пользователей, включающий в себя полный набор функций как для работы с базой данных, так и с данными пользователей. Однако Laravel предоставляет возможность использовать встроенные механизмы авторизации, у которых есть ряд преимуществ. Самое весомое из них – фасад Auth.

Фасады предоставляют «статический» интерфейс к классам, доступным в сервис-контейнере. Laravel поставляется со множеством фасадов, предоставляющих доступ почти ко всем возможностям Laravel.

Фасады Laravel служат «статическими прокси» для классов в сервис-контейнере, поддерживая большую тестируемость и гибкость, чем традиционные статические методы. Фасады обеспечивают лаконичный, запоминающийся синтаксис, который позволяет использовать возможности Laravel, не запоминая длинные имена классов, которые должны внедряться или настраиваться вручную.

Фасад Auth предоставляет удобный доступ ко всем основным функциям, связанным с авторизацией пользователей: проверка авторизации, получение текущего пользователя, выход из учетной записи и другие.

Поэтому в нашем проекте было принято решение обеспечить авторизацию и регистрацию пользователей стандартными средствами Laravel. Еще в шестой версии фреймворка Laravel все необходимое для системы авторизации было непосредственно в проекте, однако в актуальной седьмой версии в этой области произошел ряд изменений.

Как было упомянуто выше, одна из основных тенденций в настоящий момент – отделение бэкенда от фронтенда. Именно в силу этого представления страниц аутентификации были полностью удалены из Laravel 7. Для работы с авторизацией необходимо установить официальный пакет laravel/ui перед стартом, используя менеджер зависимостей Composer:

```
composer require laravel/ui
```

Хотя Laravel и не диктует, какие JavaScript или CSS-процессоры нужно использовать, данный пакет обеспечивает основную отправную точку с помощью Bootstrap, React или Vue, которые могут быть полезны для большинства приложений. В дорабатываемом нами приложении большая часть интерфейса была построена на фреймворке Bootstrap, поэтому здесь мы также будем опираться на него.

После установки пакета пользовательского интерфейса Laravel можно создать систему авторизации. Команда, генерирующая весь необходимый функционал, имеет следующий вид:

```
php artisan ui bootstrap –auth
```

Так как команда выполняется с помощью PHP, то данную команду необходимо выполнять непосредственно в контейнере app. Для исполнения команд внутри контейнера используется команда:

```
docker exec -it financier_app /bin/sh
```

Чтобы каждый раз дополнительно не вводить эту команду, был добавлен атрибут `-it`. Он обеспечивает вход в контейнер в интерактивном режиме, т.е. после выполнения команды выход из контейнера не будет осуществлен. В команде также указывается имя контейнера, в данном случае сконфигурированное нами для удобства – `financier_app`. И последним

атрибутом является команда, которую необходимо выполнить внутри контейнера. В данном случае она запускает обычный терминал, который позволит выполнять все необходимые команды в контейнере без повторного входа. В дальнейшем подразумевается, что все команды, связанные с PHP и самим Laravel-приложением, выполняются не в хостовой операционной системе, а непосредственно в контейнере.

Второе, на что хотелось бы обратить внимание в команде, генерирующей авторизацию пользователей, – artisan. Это интерфейс командной строки для Laravel. Он содержит набор полезных команд, помогающих при разработке приложения: создание миграций, контроллеров, работа с пользовательским интерфейсом.

В результате всех описанных действий будет создан механизм авторизации, однако в проекте он работать пока не будет, так как в базе данных сейчас нет таблицы для хранения пользователей. Создать эту таблицу можно с помощью механизма миграций.

Миграции – это что-то вроде системы контроля версий для базы данных, позволяющей команде разработчиков изменять структуру базы данных, оставаясь в курсе изменений других участников. Миграции обычно используют совместно с построителем структур для более простого обращения с архитектурой базы данных. Фасад Laravel Schema обеспечивает поддержку создания и изменения таблиц независимо от используемой СУБД.

Обычно миграции и модели необходимо создавать вручную, что будет рассмотрено позже, однако в данном случае Laravel уже предоставляет готовую миграцию для создания таблицы пользователей:

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
```

```
        });
    }
```

В миграции фактически описывается то, какие изменения в базе данных необходимо произвести. В данном случае указывается, что необходимо создать таблицу users, содержащую следующие поля: id (первичный ключ), name (строка), email (строка, уникальный элемент), password (строка) и другие технические поля.

В принципе, в таком виде таблица готова для работы с встроенным механизмом авторизации, однако, на наш взгляд, не всегда бывает удобно авторизоваться парой – email и пароль. У многих пользователей адрес электронной почты довольно длинный, поэтому бывает полезно иметь на сервисе собственное короткое имя, которое можно использовать для авторизации вместо email. Именно поэтому, принято решение добавить в стандартную миграцию колонку nickname, в которой и будет храниться короткое уникальное имя пользователя:

```
$table->string('nickname')->unique();
```

Для запуска всех необходимых миграций необходимо использовать artisan-команду `migrate`:

```
php artisan migrate
```

После этого в phpMyAdmin можно убедиться, что таблица users и необходимые технические таблицы были успешно созданы (рисунок 4.3).

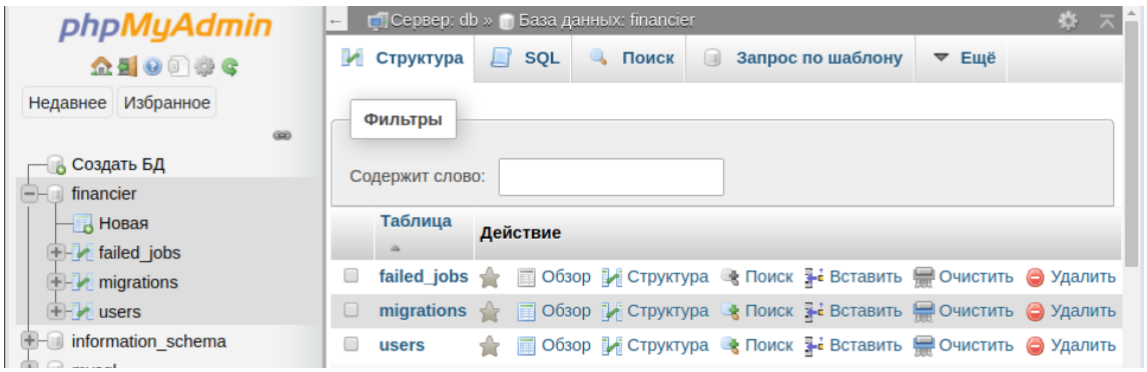


Рисунок Рефакторинг и доработка финансового веб-сервиса.17 – Структура базы данных

Теперь для правильной работы авторизации с добавленным полем nickname необходимо модифицировать стандартные представления и контроллеры. В разрабатываемом проекте предполагается также авторизация и регистрация через социальные сети, поэтому представления нет смысла редактировать, так как в дальнейшем будут созданы собственные варианты.

Редактирование начнем с контроллера регистрации. На самом деле, если взглянуть на механизм регистрации и авторизации, предоставленный фреймворков Laravel, становится понятно, что практически все функции, связанные с этим, находятся непосредственно в ядре фреймворка, поэтому редактировать их крайне нежелательно для сохранения безопасности веб-приложения. Однако две составляющие компонента авторизации были вынесены в контроллер авторизации, доступный для редактирования.

Первая часть – валидатор формы регистрации:

```
protected function validator(array $data){
  return Validator::make($data, [
     'name' => ['required', 'string', 'max:255'],
     'email' => ['required', 'string', 'email', 'max:255',
'unique:users'],
     'password' => ['required', 'string', 'min:8',
'confirmed'],
  ]);}
```

Validator – это очень удобный механизм фреймворка Laravel, позволяющий проверять входящие в веб-приложение данные. При этом осуществлять проверки можно как с помощью уже доступных в фреймворке правил, которых уже насчитывается около 80 штук, так и с помощью правил, создаваемых разработчиком. Validator получает на вход данные для проверки, в данном случае это данные из формы регистрации, проверяет их по описанным правилам и сохраняет возникшие ошибки для их дальнейшей обработки [16].

Как можно заметить в коде валидатора, приведенном ранее, с помощью данного инструмента можно даже проверить email на соответствие стандарту, используя лишь стандартное правило, что позволяет значительно сократить время разработки веб-приложения. Добавим в валидатор данные для проверки поля nickname:

```
    'nickname' => ['required', 'string', 'alpha_dash', 'max:25',
'unique:users'],
```

Здесь было указано, что поле nickname должно быть обязательно заполнено, должно представлять собой строку не длиннее 25 символов, не иметь аналогичных записей в таблице users, а также с помощью правила alpha_dash задается, что поле должно содержать только латинские символы, цифры, знаки подчёркивания и дефисы. Только при соблюдении данных правил пользователь сможет зарегистрироваться в системе, иначе он будет получать сообщения с соответствующим текстом ошибок.

При этом разработчику нет необходимости даже создавать текст для ошибок, все необходимое уже есть в фреймворке. В случае с разработкой под российский рынок необходимо лишь подключить в приложение русский язык, поддерживаемый сообществом разработчиков из репозитория caouecs/Laravel-lang.

В контроллере регистрации также необходимо изменить функцию создания пользователя, добавив сохранение псевдонима пользователя в модели:

```
protected function create(array $data)
{
    $newuser=User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'nickname' => $data['nickname'],
        'password' => Hash::make($data['password']),
    ]);

    return $newuser;
}
```

Теперь поле nickname добавляется в базу данных при регистрации пользователей. Далее необходимо модифицировать контроллер авторизации, чтобы предоставить возможность войти с электронной почтой либо с именем пользователя.

В данном случае процесс чуть сложнее, так как контроллер авторизации не включает в себя функций, которые можно было бы модифицировать. Однако

можно переназначить метод, который использует Laravel для получения имени пользователя [9]:

```
    public function __construct(){
        $this->middleware('guest')->except('logout');
        $this->username = $this->findNickname();}

    public function findNickname(){
        $login = request()->input('login');
        $fieldType = filter_var($login, FILTER_VALIDATE_EMAIL) ?
'email' : 'nickname';
        request()->merge([$fieldType => $login]);
        return $fieldType;}

    public function username(){
       return $this->username;}
```

Здесь было добавлено новое свойство – username, которое будет использоваться для хранения типа входа, т.е. осуществляется вход с помощью электронной почты или никнейма пользователя. Это поле может иметь только два значения: email или nickname. В конструкторе происходит вызов метода `findNickname`. В представлении, ответственном за авторизацию, поле, в которое пользователь будет вводить свой псевдоним или email, будет иметь имя login. Поэтому в этом методе определяется, что находится в поле с именем login (email или nickname), а затем определенный тип данных и введенное значение добавляются во входной запрос с помощью метода слияния – `merge`.

Механизм авторизации и регистрации готов к работе. Добавим возможность авторизации через социальные сети.

4.2.1 Настройка авторизации через социальные сети

Авторизация через социальные сети сейчас является неотъемлемой частью любого сайта, который выполняет проверку подлинности пользователя. Это экономит много времени для клиентов, так как им не нужно будет заполнять всю форму регистрации или авторизации. Они могут просто зарегистрироваться с помощью их учетной записи социальной сети и в следующий раз войти в веб-приложение одним щелчком мыши.

Идентификация пользователя, авторизующегося через социальную сеть, как правило осуществляется через идентификатор социальной сети. Именно поэтому идентификаторы социальных сетей пользователя необходимо хранить в базе данных. Создадим модель SocialIdent и соответствующую миграцию с помощью следующей команды:

```
php artisan make:model SocialIdent --migration
```

Сгенерированной миграцией в базе данных будет создаваться таблица, в которой для каждого пользователя будет храниться строка, содержащая все идентификаторы социальных сетей пользователя. На наш взгляд, в России самыми популярными социальными сетями являются Вконтакте, Google, Facebook и Twitter, поэтому принято решение реализовать авторизацию именно через эти социальные сети. В связи с этим, миграция примет следующий вид:

```
public function up(){
 Schema::create('social_idents', function (Blueprint $table){
        $table->id();
        $table->bigInteger('user_id')->unsigned();
        $table->string('google')->nullable();;
        $table->string('twitter')->nullable();;
        $table->string('facebook')->nullable();;
        $table->string('vkontakte')->nullable();;
        $table->string('telegram')->nullable();;
        $table->timestamps();
        $table-
>foreign('user_id')->references('id')->on('users')->onDelete('cas-
cade');});
```

Помимо полей для непосредственного хранения идентификаторов, было добавлено поле user_id, необходимое для связи строки из этой таблицы с определенным пользователем. Для этого же поля в миграции задан внешний ключ, а именно поле id из таблицы users. Таблицы будут связаны между собой отношением «один к одному». Для того, чтобы каждый раз не «изобретать» запросы к базе данных, свяжем модели между собой.

Поиск дочерней сущности в другой модели осуществляется с помощью метода `hasOne()`, т.е. в указанной модели ищется элемент, где поле с именем (родительская модель + ‘_id’) будет равен первичному ключу родительской

модели. Для связки таблиц отношением «один к одному» необходимо в модель User добавить следующую функцию:

```
public function socialIdent(){
    return $this->hasOne('App\SocialIdent');}
```

Благодаря этой функции для каждого пользователя можно получить список идентификаторов его социальных сетей по ключу socialIdent:

```
$social_ident = $user->socialIdent;
```

Также при входе в систему через социальную сеть необходимо определить, какому пользователю принадлежит тот или иной идентификатор, поэтому необходимо настроить обратную связь. Поиск родительской модели в отношении «один к одному» осуществляется с помощью метода `belongsTo()`, т.е. функция примет следующий вид:

```
public function user(){
    return $this->belongsTo('App\User');}
```

Имея организованную структуру для хранения всех необходимых данных, можно реализовать сам механизм авторизации через социальные сети. С помощью Laravel сделать это довольно просто, так как он предоставляет очень удобный способ для создания способов авторизации через социальные сети.

Совсем недавно в Laravel появился простой и удобный способ аутентификации с помощью поставщиков OAuth – Laravel Socialite [19]. В настоящее время он поддерживает аутентификацию с помощью Facebook, Twitter, LinkedIn, Google, GitHub, GitLab и Bitbucket, а сообществом разработчиков поддерживается еще дополнительно более 30 сервисов. И за счет того, что механизм аутентификации во всех случаях одинаковый, с помощью этого инструмента добавлять новые способы аутентификации можно за считанные минуты (не считая регистрации приложения в социальной сети).

Чтобы воспользоваться данным инструментом, его необходимо установить с помощью менеджера зависимостей Composer. Для каждой социальной сети используется свой собственный провайдер, рассмотрим его конфигурацию на примере авторизации через Google.

Для работы с авторизацией через провайдеры Socialite необходимо указать конфигурационные данные, предоставленные социальной сетью, в файле config/services.php. Однако хранить их там небезопасно, так как данный файл индексируется инструментами контроля версий, в связи с чем данные могут попасть в чужие руки. Поэтому принято решение сами данные указывать в файле переменных среды – .env, а уже оттуда брать их в конфигурацию сервисов:

```
'google' => [
    'client_id' => env('GOOGLE_CLIENT_ID'),
    'client_secret' => env('GOOGLE_CLIENT_SECRET'),
    'redirect' => env('APP_URL') . env('GOOGLE_REDIRECT'),],
```

В файле .env данные переменные выглядят следующим образом:

```
GOOGLE_CLIENT_ID=107944589-h2sf7bd.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=gfj786bGJBj78IGF8
GOOGLE_REDIRECT=/auth/google/callback
```

В параметре `redirect` указывается, куда социальная сеть должна отправить ответ в случае успешной идентификации пользователя.

И на этом необходимые настройки закончены, для каждой социальной сети через которую будет осуществляться авторизация пользователя, необходимо указать такой небольшой набор параметров.

Для авторизации пользователей была разработана форма, представленная на рисунке 4.4. На ней, помимо стандартных полей для ввода логина и пароля, присутствуют кнопки для авторизации через каждую из социальных сетей. Нажав на любую из них, пользователь должен быть перенаправлен в соответствующую социальную сеть для прохождения идентификации.

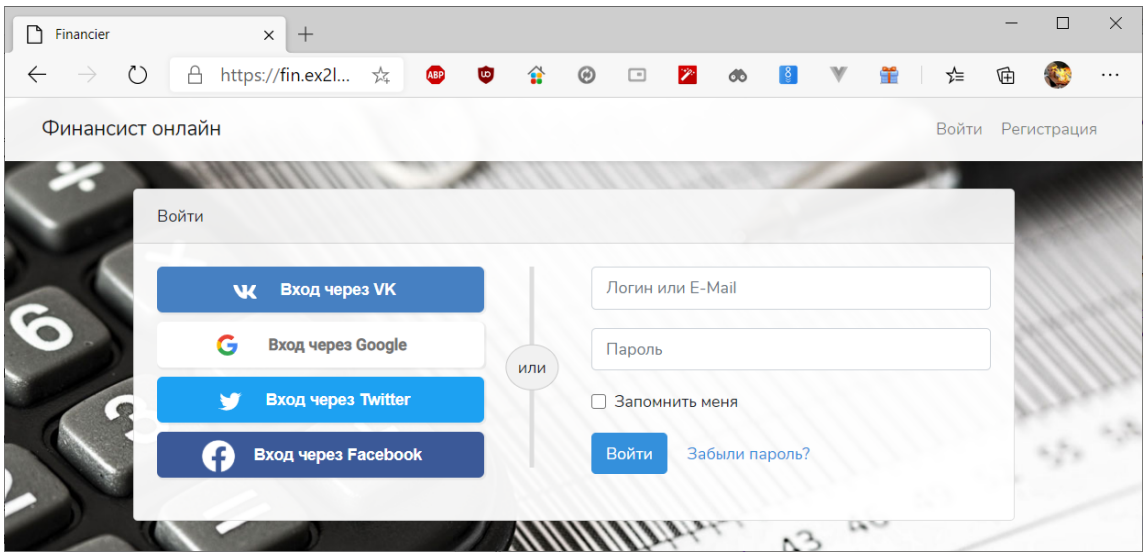


Рисунок Рефакторинг и доработка финансового веб-сервиса.18 – Форма авторизации

Теперь настроим переадресацию пользователя в социальную сеть и обработку полученного ответа в веб-приложении. Для соответствия основным принципам Laravel вынесем все функции, связанные с работой с социальными сетями, в отдельный контроллер. Создать контроллер можно с помощью следующей команды:

```
php artisan make:controller Auth/SocialController
```

Далее необходимо настроить перенаправление всех необходимых запросов, а также ответов от социальных сетей непосредственно в созданный контейнер. Сделать это можно с помощью механизма маршрутизации.

В Laravel простейшие маршруты принимают URI и функцию-замыкание, предоставляя очень простой и выразительный метод определения маршрутов. Все управление маршрутами в Laravel осуществляется в файле routes/web.php. Поскольку каждый провайдер в Socialite работает по одному и тому же принципу, то для удобства дальнейшего добавления социальных сетей все функции контейнера будет универсальны для всех социальных сетей. Маршрутизация для авторизации через социальные сети конфигурируется следующим образом:

```
    Route::get('auth/{provider}',
'Auth\SocialController@redirectToSocial')->name('auth_social');
```

```
Route::get('auth/{provider}/callback',
'Auth\SocialController@handleCallback')->name('auth_social_call-
back');
```

Здесь было создано два маршрута – `auth_social` и `auth_social_callback`. При это область маршрута `{provider}` может быть совершенно любой, а ее значение будет передано непосредственно в функцию. Это позволит узнать от какой социальной сети был получен данный ответ.

Первый маршрут осуществляет перенаправление запроса в функцию `redirectToSocial()` из контроллера SocialController, которая переадресует пользователя в социальную сеть для прохождения авторизации:

```
public function redirectToSocial($provider)
{
  return Socialite::driver($provider)->redirect();
}
```

При этом не нужно даже создавать ссылок для каждой социальной сети, каждый провайдер генерирует ссылку самостоятельно.

Второй маршрут перенаправляет ответ социальной сети в функцию `handleCallback()`. Для того, чтобы уже зарегистрированные пользователи могли добавить себе возможность входа через ту или иную социальную сеть, в этой функции осуществляется проверка – авторизован в данный момент пользователь. Если пользователь уже был авторизован ранее, то новый идентификатор социальной сети добавляется ему. А если пользователь не был авторизован, то осуществляется поиск полученного идентификатора в базе данных. Если идентификатор был найден, то функция авторизует необходимого пользователя:

```
if($findSocialIdent){
  $finduser=$findSocialIdent->user;
  Auth::login($finduser);
  return redirect('/');
}
```

Если же идентификатор не был найден в базе данных, то пользователь перенаправляется на страницу регистрации, где часть данных уже заполнены информацией из социальной сети (рисунок 4.5).

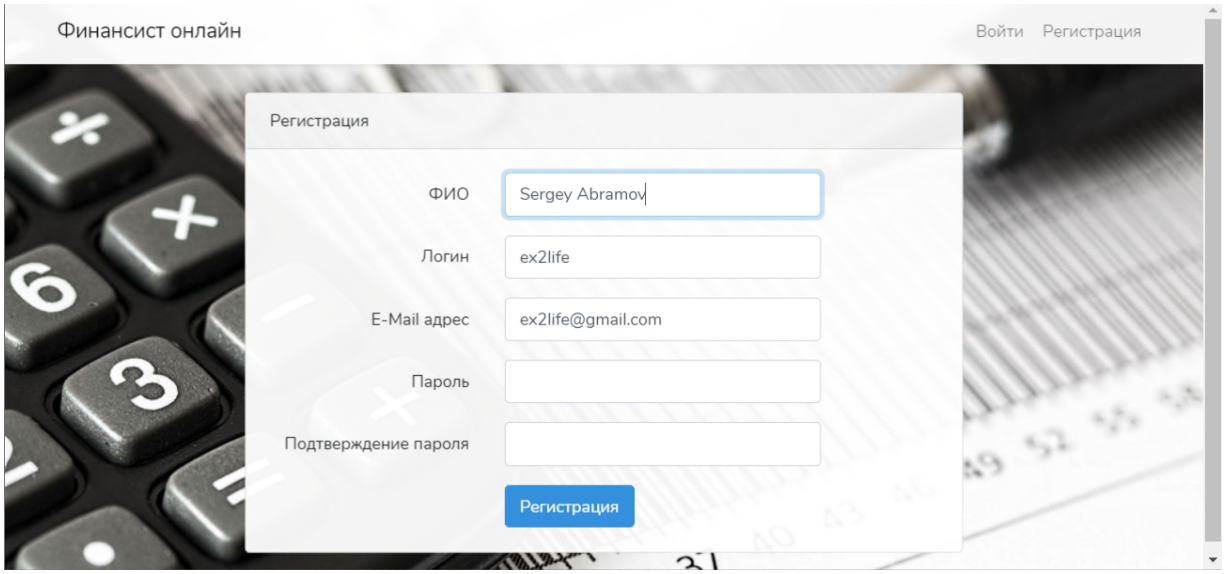


Рисунок Рефакторинг и доработка финансового веб-сервиса.19 – Регистрация

Для удобства использования веб-приложения был разработан раздел, включающий основные данные о пользователе (рисунок 4.6). В нем каждый зарегистрированный пользователь может сменить свои данные, изменить пароль и фотографию, а также подключить или отключить необходимые социальные сети.

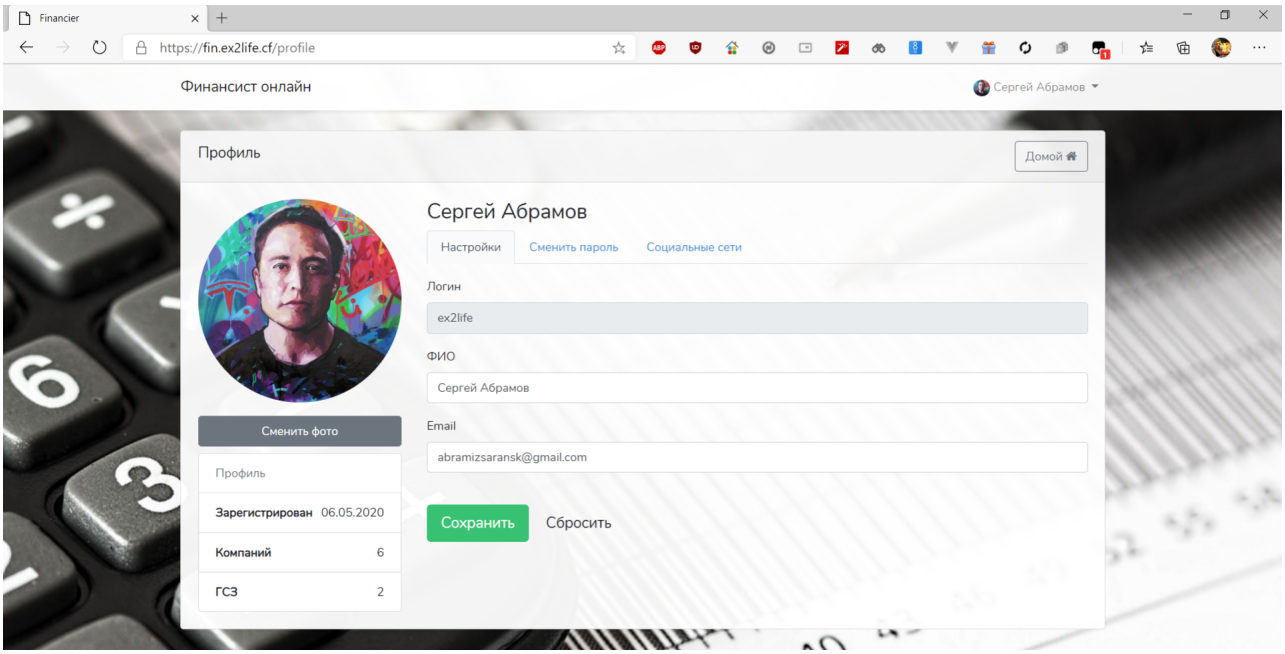


Рисунок Рефакторинг и доработка финансового веб-сервиса.20 – Профиль пользователя

Закончив работу над механизмом аутентификации пользователей, можно приступить к переносу основного функционала веб-приложения.

4.3 Рефакторинг модуля расчета графиков погашения кредита

В процессе написания бакалаврской работы весь функционал для расчета графиков погашения кредита, а также для сохранения полученных графиков был фактически реализован. При рефакторинге данного модуля все необходимые функции были размещены в контроллере CalcController.

Для данного модуля были сформированы следующие маршруты:

```
    //РАСЧЕТ ГРАФИКА КРЕДИТНЫХ ПЛАТЕЖЕЙ
    //Страница для выбора типа платежей
    Route::get('/calc',
'CalcController@calc_list')->name('calc_list');
    //Страница для ввода данных о кредите с ануитетной схемой
платежей
    Route::get('/calc/annuit',
'CalcController@calc_input')->defaults('type',
'annuit')->name('calc_annuit');
    //Страница для ввода данных о кредите с дифференцированной
схемой платежей
    Route::get('/calc/differ',
'CalcController@calc_input')->defaults('type',
'differ')->name('calc_differ');
    //Страница для ввода данных о кредите с гибкой схемой платежей
    Route::get('/calc/flex',
'CalcController@calc_input')->defaults('type',
'flex')->name('calc_flex');
    //Расчет графика платежей в требуемом формате (html для
всплывающего окна, pdf-файл, xls-файл)
    Route::post('/calc/calc_graf',
'CalcController@calc')->name('calc_graf');
```

При рефакторинге модуля на фреймворк Laravel весь функционал был полностью сохранен. Были заменены библиотеки генерации pdf- и xls-документов. Вместо библиотеки tcpdf сейчас используется библиотека tcpdf-laravel, имеющую гораздо лучшую интеграцию с Laravel за счет использования фасадов. Библиотека PHPExcel была заменена на phpspreadsheet, что позволило повысить скорость генерации больших Excel-файлов.

Был переработан пользовательский интерфейс для повышения качества использования на мобильных устройствах. В результате все элементы стали

адаптивными для правильного отображения на устройствах с различными экранам. Например, страница для ввода данных о кредите на мобильном телефоне имеет вид, представленный на рисунке 4.7.

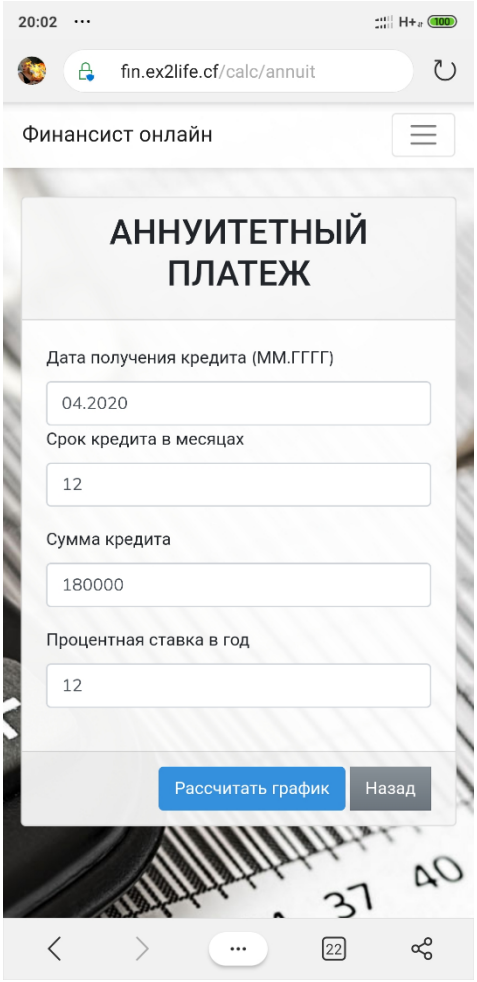


Рисунок Рефакторинг и доработка финансового веб-сервиса.21 – Интерфейс с мобильного устройства

Перейдем к реализации финансового анализа деятельности компании и оценки вероятности одобрения кредита.

4.4 Доработка функционала сервиса

Для практически каждой развивающейся компании рано или поздно возникает необходимость в дополнительном капитале для работы. И, как правило, для получения этого капитала компании обращаются в банки за заемными средствами. За счет них организации могут закупать товары,

расширять производство, модернизировать оборудование или используемые технологии, что в перспективе позволят и окупить кредит и повысить темп развития. В общем, существует немало вещей, на которые необходимы деньги. Однако кредитование очень непростой вопрос как для малого, так и для крупного бизнеса, ведь долги, как известно, необходимо отдавать.

Риски, связанные с кредитованием, существуют не только для заемщиков. Для банка риск порой бывает еще больше, так как всегда есть возможность невозврата денежных средств. Именно поэтому, далеко не каждый банк готов дать свои деньги в использование любой компании, подобное решение должно быть крайне взвешенным и рассмотренным со всех точек зрения.

Однако, для потенциального заемщика довольно трудно узнать заранее будет ли им одобрен тот или иной кредит. При этом отказ банка крайне нежелателен, так как уже с 30 декабря 2004 г. в Российской Федерации действует закон о кредитной истории, по которому для всех как физических, так и юридических лиц хранится информация о исполнении заемщиком кредитных обязательств и всех взаимодействиях с кредитной сферой. В случае отказа в выдаче кредита, данная информация также вероятно попадет в кредитную историю и, как следствие, при обращении за кредитом в будущем у кредитора могут возникнуть вопросы, почему ранее в другом месте компанию заемщика признали ненадежным, что значительно снизит вероятность получения кредита. А в некоторых ситуациях, отказы банков в выдаче кредитов могут совсем привести компанию к банкротству.

В основном, банки используют для оценки кредитоспособности свои собственные системы, работу которых они тщательно скрывают за коммерческой тайной [5]. Однако в их основе обычно лежат общеизвестные алгоритмы оценки, а значит вероятность одобрения кредита для конкретной компании или группы связанных заемщиков всё же возможно оценить, основываясь на ее финансовых показателях.

4.4.1 Методы финансового анализа

Основной целью проведения финансового анализа является получение определенного числа показателей, позволяющих дать объективную оценку финансового состояния организации. Исходной базой для проведения финансового анализа, как правило, служат бухгалтерские отчётности и финансовые результаты, изучение которых помогает воспроизвести все основные аспекты производственной и коммерческой жизни предприятия.

Лишь получив достаточно полную и достоверную картину о деятельности организации в прошлом, о том какие тенденции сложились при его развитии и функционировании можно понять, какие управленческие решения, программы развития и бизнес-планы необходимо выработать на будущие периоды деятельности. Это относится ко всем фирмам и предприятиям независимо от их роли, масштаба, формы собственности или вида деятельности.

Анализ финансового состояния – обязательная составляющая финансового менеджмента любого предприятия [3].

Финансовый анализ играет огромную роль при определении кредитоспособности предприятия, так как он позволяет определить так называемые границы финансовой устойчивости. Недостаточная устойчивость является сигналом того, что компания вероятнее всего в ближайшее время может стать неплатежеспособной, что возникнет нехватка денежных средств для финансирования текущей или инвестиционной деятельности.

Избыточная устойчивость также не является благоприятной ситуацией, так как значительно препятствует развитию, приводя к появлению избыточных резервов и запасов. В перспективе это значительно сокращает прибыль и увеличивает сроки оборота капитала.

Однако сам по себе финансовый анализ довольно сложен. Он связан с обработкой обширной информации, характеризующей самые разнообразные аспекты функционирования предприятия как производственного, финансового, имущественного, социального комплекса. Конечно, часть анализа можно

провести, просмотрев бухгалтерский учет и финансовые результаты, однако для получения более глубокого анализа требуются дополнительные расчеты, за которым компаниям приходится обращаться к дорогостоящим услугам финансовых аналитиков. Однако обычно для расчетов они используют общеизвестные методики подсчета.

На данный момент практикой выработано множество методов финансового анализа, среди которых можно выделить наиболее популярные:

– сравнительный (пространственный) анализ;

– факторный анализ;

– чтение бухгалтерской отчетности;

– вертикальный анализ;

– трендовый анализ;

– горизонтальный анализ;

– метод финансовых коэффициентов.

Часть данных методик стали популярными за счет своей простоты, вертикальный, горизонтальный, трендовый, сравнительный и другие похожие анализы повсеместно применяются бухгалтерами при составлении отчетности, однако плохо позволяют понять причины тех или иных проблем в организации, чтобы понять, где вовремя необходимо приложить усилия для улучшения ситуации. Одним из самых гибких и подробных методик финансового анализа на данный момент является метод финансовых коэффициентов [8].

Этот метод представляет собой расчет отношений показателей отчетности (финансовых коэффициентов) и определение их взаимосвязей. Финансовые коэффициенты характеризуют следующие стороны экономической деятельности предприятия:

– эффективность работы;

– деловая активность;

– рыночные характеристики организации;

– платежеспособность;

– финансовая зависимость или финансовая автономия.

На данный момент в России большинство систем мониторинга деятельности предприятий основывается на коэффициентном анализе. В некоторых федеральных законах Российской Федерации для расчета тех или иных критериев используют коэффициенты из данной методики.

В связи с этим было принято решение реализовать финансовый анализ деятельности организаций на основе именно этой методики.

Однако данная методология позволяет гибко проанализировать отдельно каждую сторону работы предприятия, но не дает итоговой оценки, по которой можно с определенной точностью сказать, насколько финансово стабильна компания. Для этих целей были разработаны специальные модели, позволяющие рассчитать индекс, достаточно точно определяющий риск банкротства предприятия.

На данный момент одной из самых точных таких моделей является индекс Альтмана. Для построения своей модели он использовал 66 американских компаний в период с 1946-1965. 33 компании обанкротились в этот период, а 33 остались финансово устойчивыми [26]. Помимо этого, из 22-х финансовых коэффициентов он выделил всего 5, наиболее полно, по его мнению, отражающих деятельность предприятия. После этого он использовал инструментарий множественного дискриминантного анализа для определения весовых значений у коэффициентов в интегральной модели. В итоге он получил статистическую классификационную модель для определения класса предприятия.

Общий экономический смысл модели Альтмана представляет собой функцию от некоторых показателей, характеризующих экономический потенциал предприятия и результаты его работы за истекший период.

Несмотря на то, что при разработке модели были учтены только компании США, экономисты из множества стран, проверяющие на практике его модель, соглашаются с ее универсальностью и надежностью.

По статистике индекс обладает довольно высокой точностью. На горизонте одного года точность данной модели составляет 95%, на два года – 83%, что, определенно, является ее достоинством.

Одновременно, с расчётом риска банкротства индекс позволяет оценить кредитоспособность компании, что как нельзя лучше подходит для поставленных задач. За счет этого, индекс даже получил свое второе название – индекс кредитоспособности. Именно поэтому, принято решение для финансового анализа, в разрабатываемом проекте, использовать метод расчета финансовых показателей и индекс Альтмана.

4.4.2 Реализация методик финансового анализа

Крайне редко у компании получается удерживать один и тот же стабильный уровень финансовой стабильности. В какие-то периоды складывается более успешная обстановка для ведения бизнеса, в какие-то возникают проблемы. Именно поэтому, для объективности финансового анализа недостаточно проводить расчет по данным, полученным только за последний подотчетный период. Общепринятой практикой является расчет показателей за 3 последних квартала, предшествующих дате проведения финансового анализа, и дальнейший учет среднего из полученных значений.

В настоящее время в Российской Федерации Центробанк значительно ужесточил требования для кредитования компаний, образующих группы связанных заемщиков. Как показывает практика, очень мало компаний осуществляет деятельность, изолировавшись от всего остального мира, большинство компаний финансово довольно сильно связаны с другими компаниями, что крайне важно учитывать при кредитовании и проведении финансового анализа.

Связанные заемщики – юридические и физические лица, связанные между собой таким образом, что ухудшение финансового положения одного делает вероятным ухудшение финансового положения другого (других

заемщиков), что может явиться причиной неисполнения этой группой обязательств перед банком по его кредитным требованиям [14].

Именно поэтому в разрабатываемом сервисе принято решение реализовать расчет финансовых показателей и оценку кредитоспособности с учетом компаний, входящих в группу связанных заемщиков. Для этого был реализован интерфейс для ввода и механизм для хранения групп связанных заемщиков и компаний, входящих в нее (рисунок 4.8).

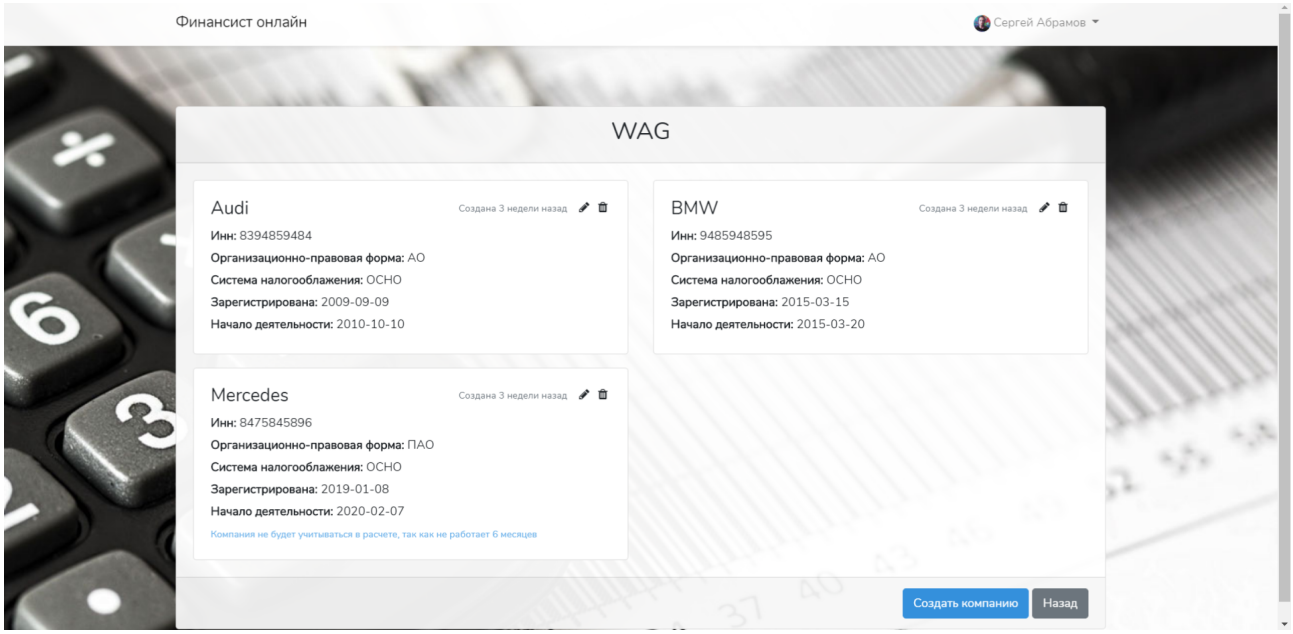


Рисунок Рефакторинг и доработка финансового веб-сервиса.22 – Список компаний, входящих в группу WAG

И для расчета финансовых коэффициентов, и для определения индекса Альтмана необходим бухгалтерский баланс и финансовые результаты деятельности компаний. Поэтому необходимо предусмотреть интерфейс для ввода и хранения этих данных за последние 3 квартала, предшествующих дате расчета. При этом следует учесть, что часто необходимо провести расчет за прошлые периоды, чтобы понять свои ошибки и скорректировать управление организацией, поэтому необходимо предусмотреть возможность смены даты расчета (рисунок 4.9).

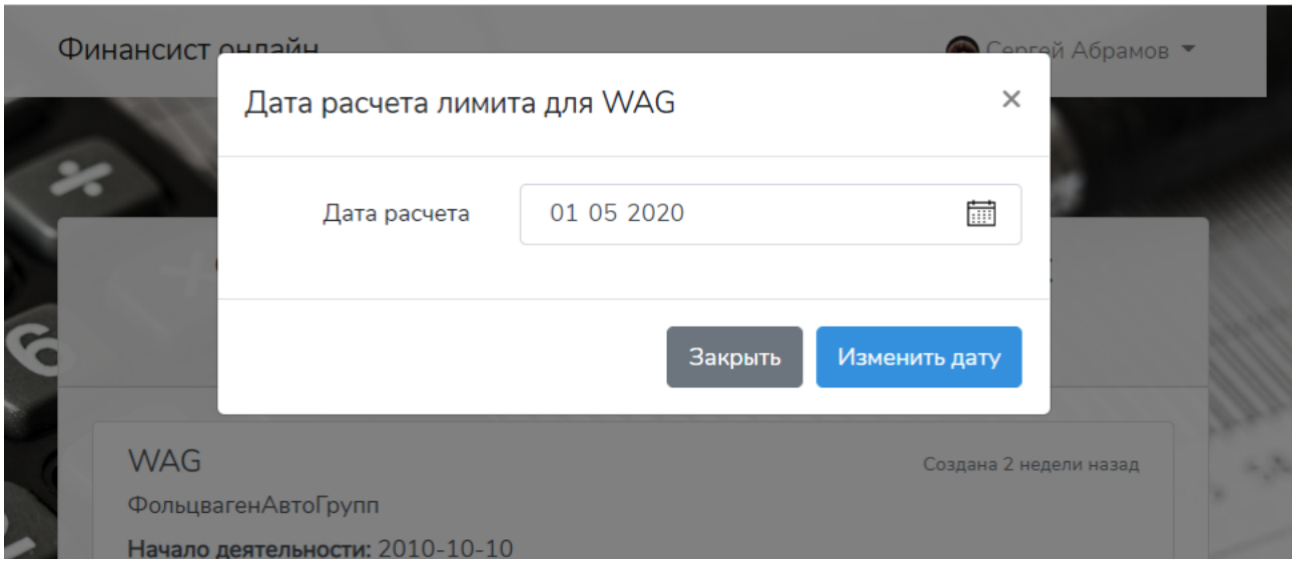


Рисунок Рефакторинг и доработка финансового веб-сервиса.23 – Смена даты расчета

Для каждой новой даты рассчитываются три периода, на которые необходимо заполнить бухгалтерский баланс и финансовые результаты. В связи с тем, что для расчета принято учитывать три последних квартала, то компании, которые на текущий момент не работают шесть месяцев, не будут учитываться в расчетах для групп связанных заемщиков. Спрогнозировать поведение подобных компаний довольно сложно ввиду их малого срока работы.

Для проведения финансового анализа необходимо на каждый из трех кварталов, предшествующих дате расчета, заполнить бухгалтерский баланс и форму, содержащую финансовые результаты деятельности компании [6]. Для этого были разработаны специальные формы, представленные на рисунках 4.10 и 4.11. Эти форма соответствует стандарту, используемому на данный момент бухгалтерами, а значит у пользователей не возникнет проблем при переносе данных в веб-сервис [11].

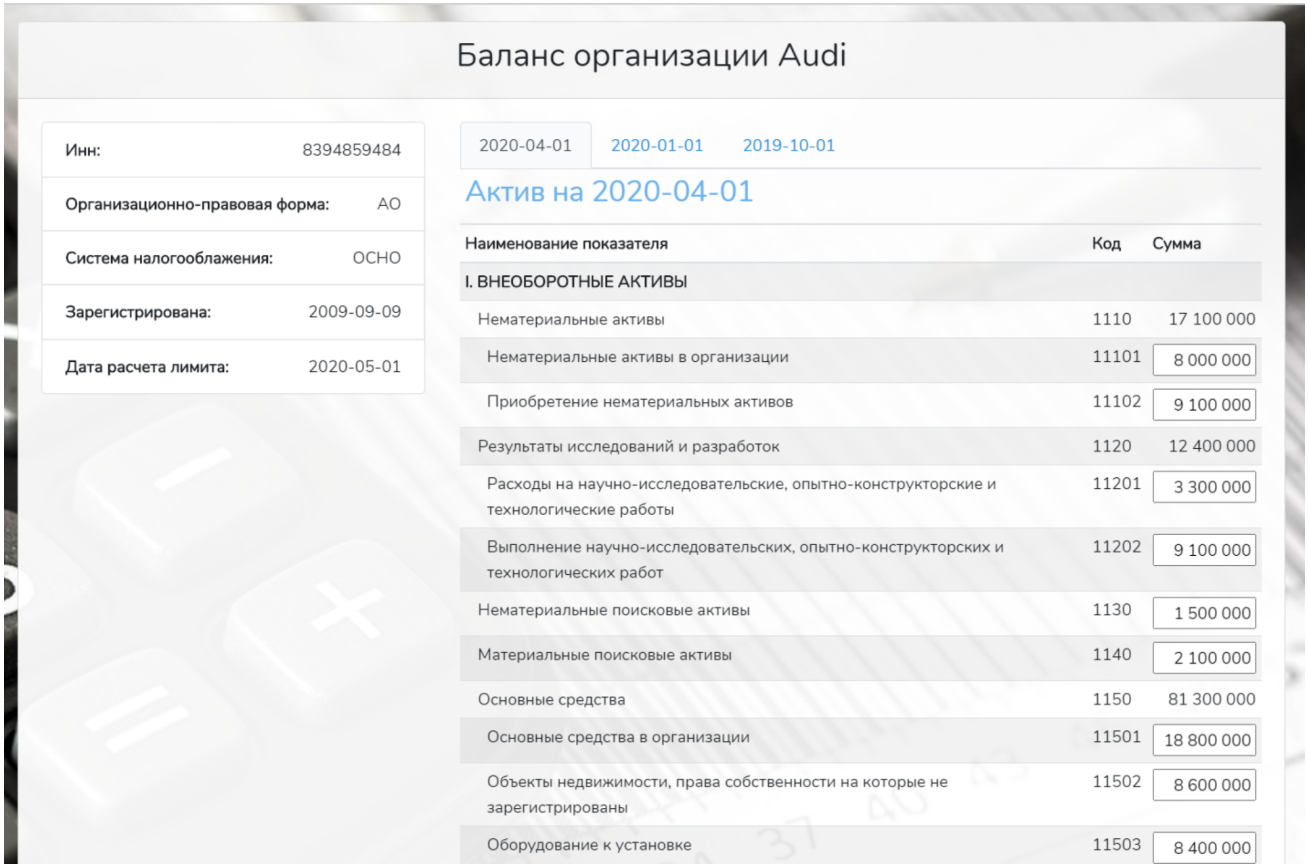


Рисунок Рефакторинг и доработка финансового веб-сервиса.24 – Форма для заполнения бухгалтерской отчётности

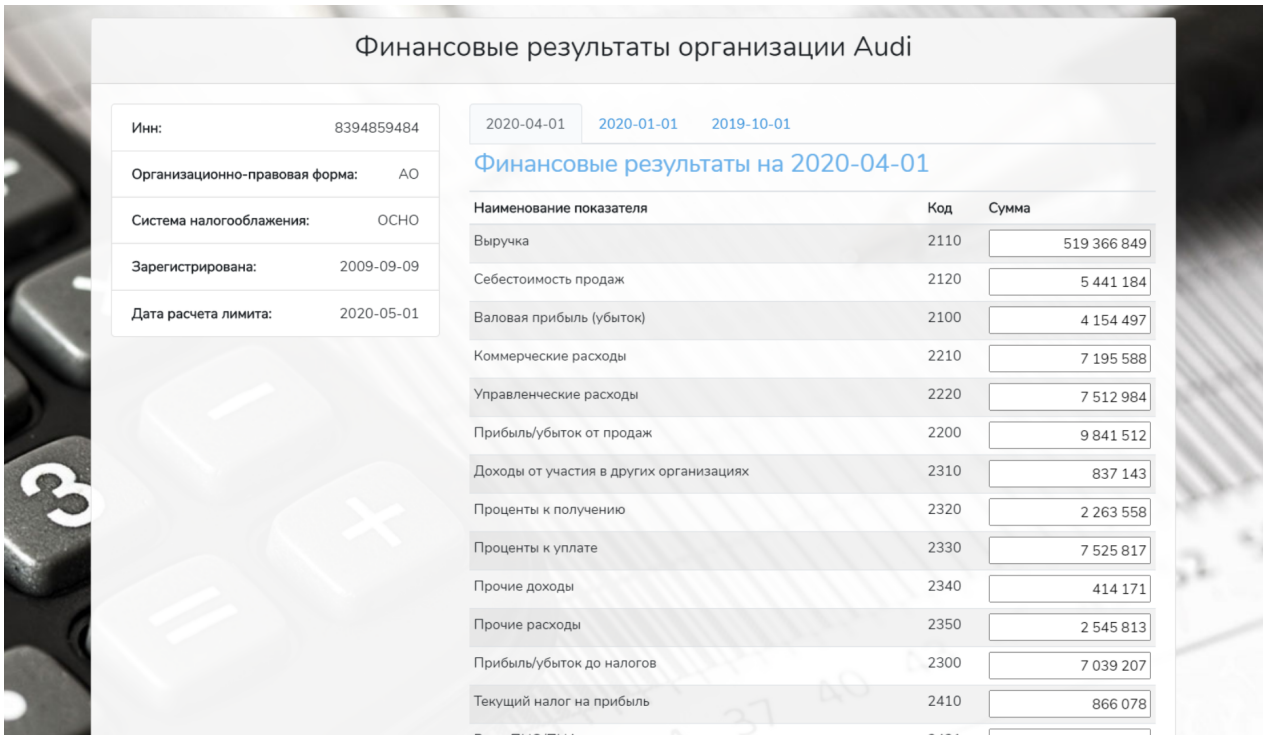


Рисунок Рефакторинг и доработка финансового веб-сервиса.25 – Форма для заполнения финансовых результатов деятельности

На этом этапе фактически полностью готова структура базы данных. Все дальнейшие разрабатываемые параметры будут вычисляться при их запросе. Получившаяся логическая модель данных представлена на рисунке 4.12.

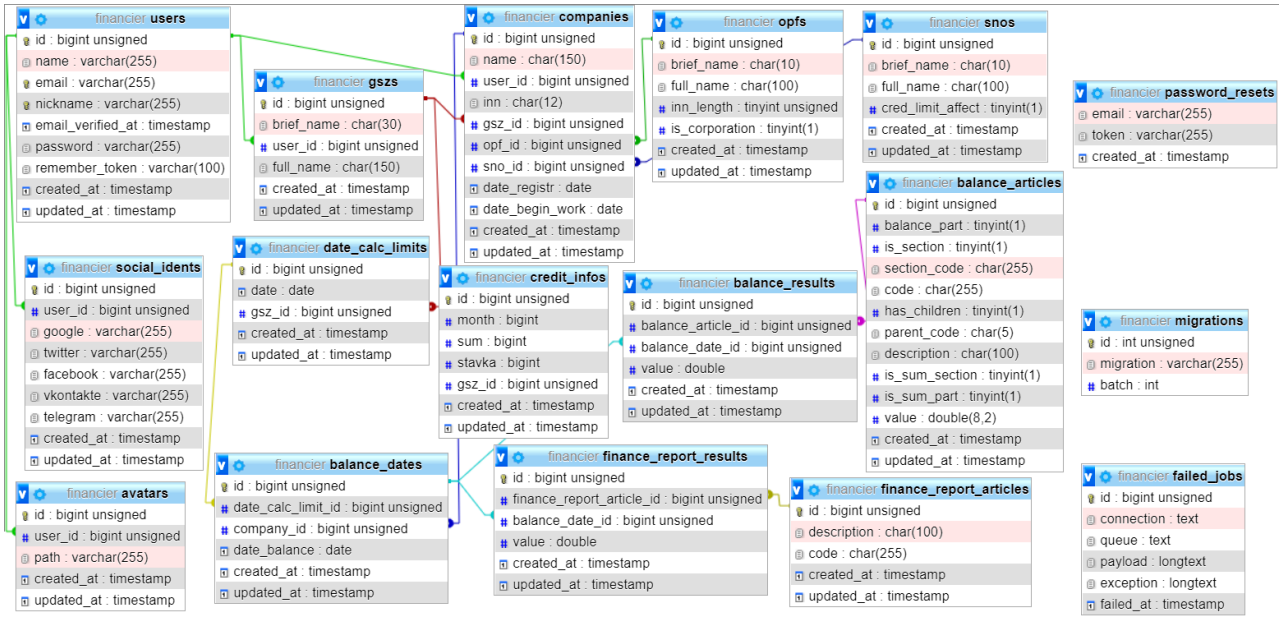


Рисунок Рефакторинг и доработка финансового веб-сервиса.26 – Логическая модель данных

Подготовив интерфейс для ввода данных компаний, можно приступать к реализации финансового анализа.

4.4.2.1 Расчет финансовых коэффициентов

Методы расчёта финансовых коэффициентов постоянно развиваются и соответственно растет количество показателей, однако большая часть из них сугубо индивидуальна и служит строго определенным целям. При разработке данного сервиса ставилась цель создать универсальный инструмент для большинства типов компаний с уклоном в сферу кредитования. В связи с этим, принято решение реализовать расчет самых важных для бизнеса показателей при работе с кредитными средствами:

– коэффициент абсолютной ликвидности;

– промежуточный коэффициент покрытия;

– коэффициент покрытия обязательств;

– коэффициент соотношения собственных и заемных средств;

– рентабельность продукции (рентабельность продаж).

Коэффициент абсолютной ликвидности показывает, какая доля краткосрочных долговых обязательств будет покрыта за счет денежных средств и их эквивалентов в виде рыночных ценных бумаг и депозитов, т.е. абсолютно ликвидными активами. Общая формула расчета коэффициента представлена на рисунке 4.13.

$$К_{ал} = \frac{\text{Денежные средства + Краткосрочные финансовые вложения}}{\text{Текущие обязательства}}$$

Рисунок Рефакторинг и доработка финансового веб-сервиса.27 – Формула расчета коэффициента абсолютной ликвидности

Для расчета данного коэффициента для компаний сформирован следующий метод:

```
    public function k1($credit = false)
        {
            $balance_dates = $this->actual_balance_dates();
            $ds_id = $this->findArticleIdByCode('1250');
            $vcb_id = $this->findArticleIdByCode('1240');
            $k1 = $balance_dates->avg(function ($balance_date) use
($ds_id, $vcb_id, $credit) {
                $ds = $this-
>findResultValueByDateAndArticleId($balance_date, $ds_id);
                $vcb = $this-
>findResultValueByDateAndArticleId($balance_date, $vcb_id);
                return ($ds + $vcb) / ($this->ko($credit,
$balance_date));
            });
            return round($k1, 2, PHP_ROUND_HALF_UP);
        }
```

При вызове данного метода будет осуществлен поиск необходимых строк баланса и расчет по формуле, приведенной ранее. Средний показатель данного коэффициента за три периода отчетности, предшествующих дате расчета, является искомым значением.

Значение коэффициента менее 0.2 указывает, что предприятие не в состоянии оплатить обязательства за счет денежных средств всех видов. При этом коэффициент более 0.5, хоть и не несет больших проблем для предприятия, но означает, что структура капитала распределена нерационально, а именно у компании слишком большая доля неработающих активов в виде наличных денег и средств на счетах. Значение более 0.2, но менее 0.5 является хорошим показателем ликвидности.

Коэффициент промежуточного покрытия – это более жесткая оценка ликвидности предприятия. Этот коэффициент показывает, насколько возможно будет погасить текущие обязательства, если положение станет критическим. При этом исходят из предположения, что у товарно-материальных запасов нет ликвидационной стоимости. Для правильного расчета коэффициента быстрой ликвидности оценивают качество ценных бумаг и дебиторской задолженности. Расчет данного коэффициента осуществляется по формуле, представленной на рисунке 4.14.

$$K_{пп} = \frac{\text{Краткосрочная дебиторская задолженность} + \text{Краткосрочные финансовые вложения} + \text{Денежные средства}}{\text{Текущие обязательства}}$$

Рисунок Рефакторинг и доработка финансового веб-сервиса.28 – Формула расчета коэффициента промежуточного покрытия

Метод, осуществляющий расчет данного коэффициента довольно сильно похож на предыдущий, поэтому отдельно рассмотрен не будет Нормальное значение данного коэффициента попадает в диапазон 0.7-1. Данный показатель довольно сильно учитывается банками при выдаче кредита. Для компаний с коэффициентом промежуточного покрытия ниже 0.7 кредит вероятнее всего будет выдан под больший процент, либо возможен отказ.

Коэффициент покрытия обязательств показывает способность компании погашать текущие (краткосрочные) обязательства за счёт только оборотных активов. Чем больше значение коэффициента, тем лучше платежеспособность предприятия. Рассчитывается данный коэффициент делением текущих активов на краткосрочные обязательства (текущие пассивы).

Нормальным считается значение коэффициента 1.5-2.5. Значение ниже 1 говорит о высоком финансовом риске, – предприятие не в состоянии стабильно оплачивать текущие счета. Значение более 3 свидетельствует о нерациональной структуре капитала.

Коэффициент соотношения заемных и собственных средств показывает, сколько приходится заемных средств на 1 рубль собственных средств. Данный коэффициент позволяет оценить финансовую устойчивость предприятий. Рекомендуемое значение коэффициента меньше 1. Чем ниже значение показателя, тем выше финансовая устойчивость и независимость предприятия от заемного капитала и обязательств. Уровень данного коэффициента выше 1 свидетельствует о потенциальной опасности возникновения недостатка собственных денежных средств, что может служить причиной затруднений в получении новых кредитов.

И последний рассчитываемый коэффициент – рентабельность продаж. Он показывает, какую сумму прибыли получает предприятие с каждого рубля проданной продукции. Формула для вычисления рентабельности продаж представлена на рисунке 4.15.

$$K_{рп} = \frac{\text{Прибыль от продаж}}{\text{Выручка от продаж}} * 100\%$$

Рисунок Рефакторинг и доработка финансового веб-сервиса.29 – Формула расчета рентабельности продаж

Однако в случае с этим коэффициентом важно не само значение, а динамика его изменения за расчетные периоды. В связи с этим, метод примет следующий вид:

```
    public function k5_status(){
       $balance_dates = $this->actual_balance_dates();
        $pp_id =
FinanceReportArticle::wherecode('2200')->first()->id;
        $vp_id =
FinanceReportArticle::wherecode('2110')->first()->id;
        $k1 = $balance_dates->sortBy('date')->reduce(function
($carry, $balance_date) use ($pp_id, $vp_id) {
            $pp =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($pp_id)->first()->value;
            $vp =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($vp_id)->first()->value;
            if ($vp == 0) $vp++;
            $k5 = $pp / $vp;
            if ($k5 > $carry['k']) $carry['status'] = true;
            else $carry['status'] = false;
            $carry['k'] = $k5;
            return $carry;
        }, [
            'status' => false,
            'k' => -999999999,
        ]);
        return $k1['status'];
    }
```

В нём последовательно с помощью функции `reduce` для каждой подотчетной даты осуществляется расчет коэффициента и его сравнение с предыдущим [12]. В результате, если значение коэффициента растет, то методом будет возвращено значение `true`, если падает – `false`.

На этом все требуемые коэффициенты реализованы, функции для их расчета представлены в приложении Б. Можно приступать к расчету индекса Альтмана.

4.4.2.2 Реализация расчета индекса Альтмана

Не смотря на общее название – индекс Альтмана, самим ученым было разработано 5 моделей для расчета этого индекса. Разные модели применимы для разных типов компаний, однако пятифакторная модель стала своего рода

классической, именно её обычно имеют ввиду, говоря о индексе Альтмана. Расчет индекса по данной модели осуществляется в соответствии с формулой (1).

$$ = 6.56 * X1 + 3.26 * X2 + 6.72 * X3 + 1.05 * X4 \quad (1)$$

Коэффициенты, используемые в формуле, рассчитываются с помощью формул, приведенных в таблице 4.1.

Таблица 4.1 – Соответствие коэффициентов Альтмана формулам

Коэффи-циент	**Формула расчета**	**Расчет по РСБУ**
X1	X1 = Оборотный капитал/Активы	(стр.1200-стр.1500)/ стр.1600
X2	X2=Нераспределенная прибыль/ Активы	стр.2400/ стр.1600
X3	X3 = Операционная прибыль/ Активы	стр.2300/ стр.1600
X4	X4 = Собственный капитал/ Обязательства	стр.1300/ (стр.1400+стр.1500)
X5	X5 = Выручка/Активы	стр.2110 /стр.1600

Метод, определяющий класс компании, осуществляет поиск всех необходимых строк баланса и совершает операции на основе формул, приведенных ранее. Полный код данного метода представлен в приложении Б.

Исходя из полученного результата, компания относится к одной из категорий:

– если Z>2,9 – зона финансовой устойчивости;

– если 1,23<Z<2,9 – зона неопределенности;

– если Z<1,23 – зона финансового риска.

На основе выделенной категории можно судить о вероятности банкротства компании. Готовый финансовый анализ одной из тестовых компаний представлен на рисунке 4.16.

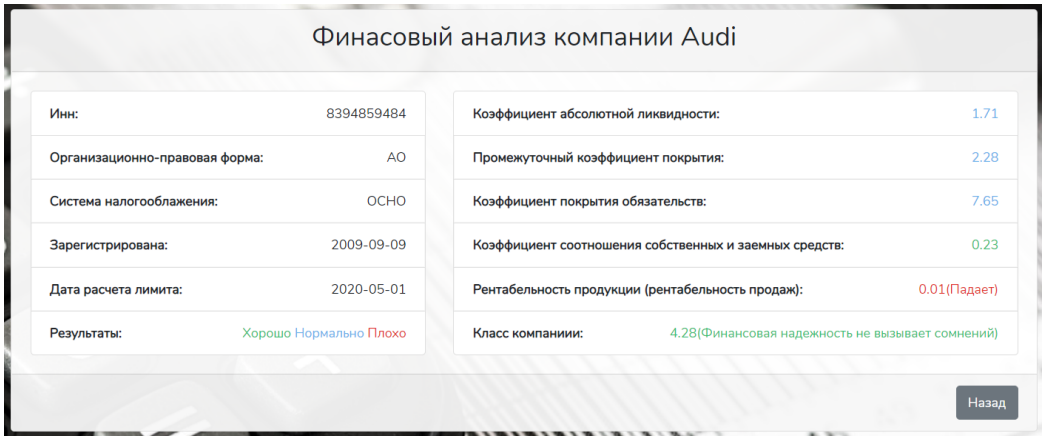


Рисунок Рефакторинг и доработка финансового веб-сервиса.30 – Финансовый анализ

Для удобства каждый параметр обозначен цветом, характеризующим результат. Используя данный анализ, руководители компаний могут понять какие действия стоит предпринять для повышения финансовой надежности.

4.4.2.3 Оценка вероятности получения кредита

Как было сказано ранее, кредит компании обязательно нужно рассматривать с учетом всех компаний, входящих в группу связанных заемщиков. Именно поэтому, оценка вероятности одобрения кредита в разрабатываемом сервисе будет выполняться непосредственно для группы связанных заемщиков.

Для оценки кредита необходимо задать его параметры на специальной странице, представленной на рисунке 4.17. Здесь группа связанных заемщиков

указывает сумму кредита, которую они рассчитывают запросить, срок кредита, а также ставку кредита, предлагаемую выбранным ими банком.

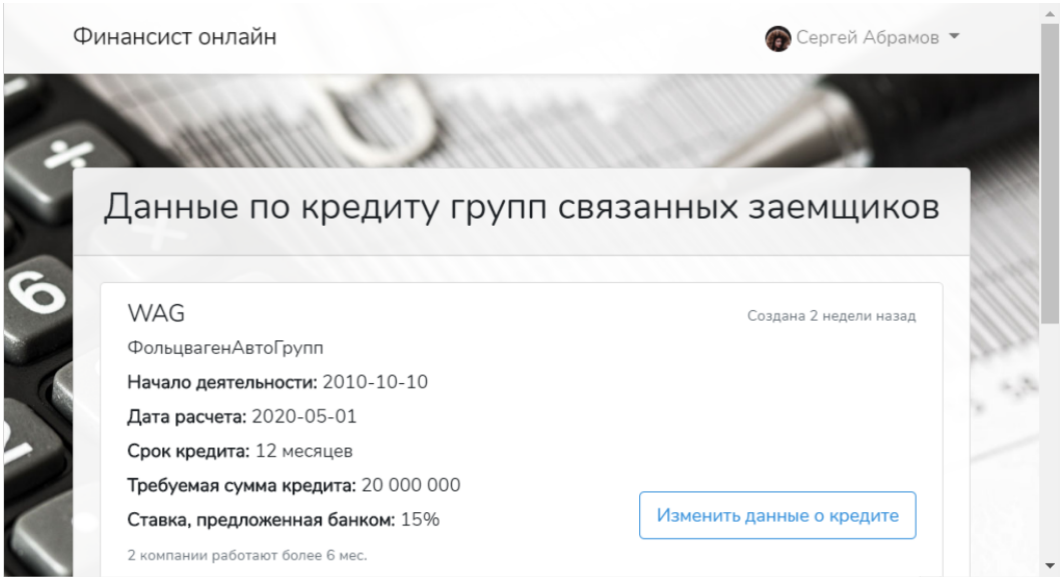


Рисунок Рефакторинг и доработка финансового веб-сервиса.31 – Данные о кредите

Поскольку индекс Альтмана сам по себе является индексом кредитоспособности, то можно использовать его для оценки вероятности одобрения кредита [22]. При оценке необходимо произвести повторный финансовый анализ, но с небольшим допущением. Необходимо узнать, какие финансовые показатели были бы у группы, обладай она данным кредитом в предыдущие отчетные периоды.

Важный момент, о котором никогда нельзя забывать – закон. Центробанк довольно внимательно контролирует сферу кредитования юридических лиц и для этого, ими были разработаны различные нормативы. Один из таких нормативов – Н25.

Этот норматив максимального размера риска на связанное с банком лицо (группу связанных с банком лиц) регулирует (ограничивает) кредитный риск банка в отношении связанного с ним лица (группы связанных с ним лиц) и определяет максимальное отношение совокупной суммы обязательств лица (лиц, входящих в группу лиц) перед банком и обязательств перед третьими

лицами, вследствие которых у банка возникают требования в отношении указанного лица (лиц, входящих в группу лиц), к собственным средствам (капиталу) банка.

Говоря простыми словами, как бы группа связанных заемщиков не была финансово стабильна и надежна, она не может получить в банке кредитов на сумму, более чем 20% от их собственного капитала. В связи с этим, при оценке вероятности одобрения важно учитывать, что некоторые суммы кредита не будут одобрены по закону, а не из-за финансовых показателей.

С учетом всего вышеизложенного, был разработан алгоритм расчета вероятности одобрения кредита. За вероятность одобрения кредита в 100% принимается индекс Альтмана выше, чем 3.5, а за 0% – 1. И, естественно, при превышении суммы, разрешенной к получению по закону, вероятность получения кредита считается равной 0%.

При запросе оценки пользователь получает страницу, содержащую вероятность одобрения кредита, заполненного ранее. Форма представлена на рисунке 4.18.

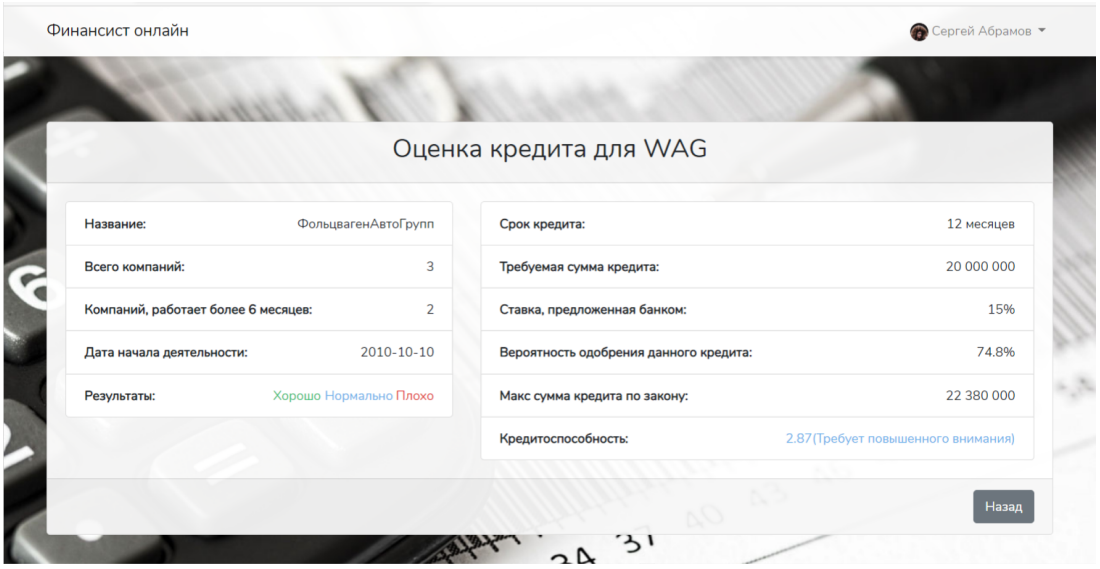


Рисунок Рефакторинг и доработка финансового веб-сервиса.32 – Вероятность одобрения кредита

На этом проект полностью готов к использованию и можно приступить к его развертыванию на VPS.

4.5 Развертывание проекта на VPS

Для размещения веб-приложения ранее было принято решение использовать виртуальный приватный сервер – VPS. Выбор хостера для аренды VPS довольно индивидуален, у кого-то дешевле, у кого-то выше надежность и более оперативная поддержка. Нами был выбран хостер – AlienVDS. Это достаточно молодой хостер, но при этом на 2020 год он обладает одним из самых больших процентов uptime, т.е. времени работы за год без сбоев, приближаясь практически к 100%. При этом стоимость услуг гораздо ниже конкурентов. На 1 апреля 2020 года стоимость самой дешевой VDS составляет 1 доллар в месяц, при этом предоставляя характеристики не хуже конкурентов, а канал в 100 Мегабит за данные деньги найти у конкурентов вообще не предоставляется возможным. К тому же все виртуальные сервера работают на твердотельных накопителях – SSD, что значительно повысит производительность веб-приложения.

Для разработанного веб-приложения была выбрана VDS на KVM-архитектуре с следующими характеристиками:

– 1024 Мб оперативной памяти;

– одноядерный процессор с частотой 2600 MHz;

– SSD диск ёмкостью 15 GB;

– канал подключения 200 мб/с;

– один белый IPv4.

При этом стоимость данной VDS составит всего 2.5$ в месяц или 28$ в год (с учетом скидки за срок аренды), что вполне удовлетворяет финансовым требованиям проекта. Характеристик, предоставляемых на данном тарифе, хватит для использования веб-приложения не только на начальном этапе, но и с большим запасом производительности на будущее.

Также для доступа к веб-приложению по короткому адресу принято решение использовать домен finansist.ga, арендованный ранее для проекта. А для доступа к консоли phpMyAdmin – субдомен pma.finansist.ga

1. Алгоритм развертывания веб-приложения на новом VPS очень простой:

2. Подключаемся к серверу по SSH.

3. Устанавливаем Docker и Docker Compose на VDS.

4. Клонируем актуальную версию проекта на сервер удобным способом, в нашем случае через GitHub. При необходимости переносим директорию с базой данных.

5. Указываем все необходимые environment-переменные, т.е. логины и пароли базы данных, домены, запросы к которым необходимо обработать, настройки Laravel, указывающие, что веб-приложение запущено в production.

6. В корне проекта выполняется команда docker-compose up.

И после автоматической сборки всех контейнеров – проект запущен в работу. Убедиться в этом можно по адресу – finansist.ga (рисунок 4.19).

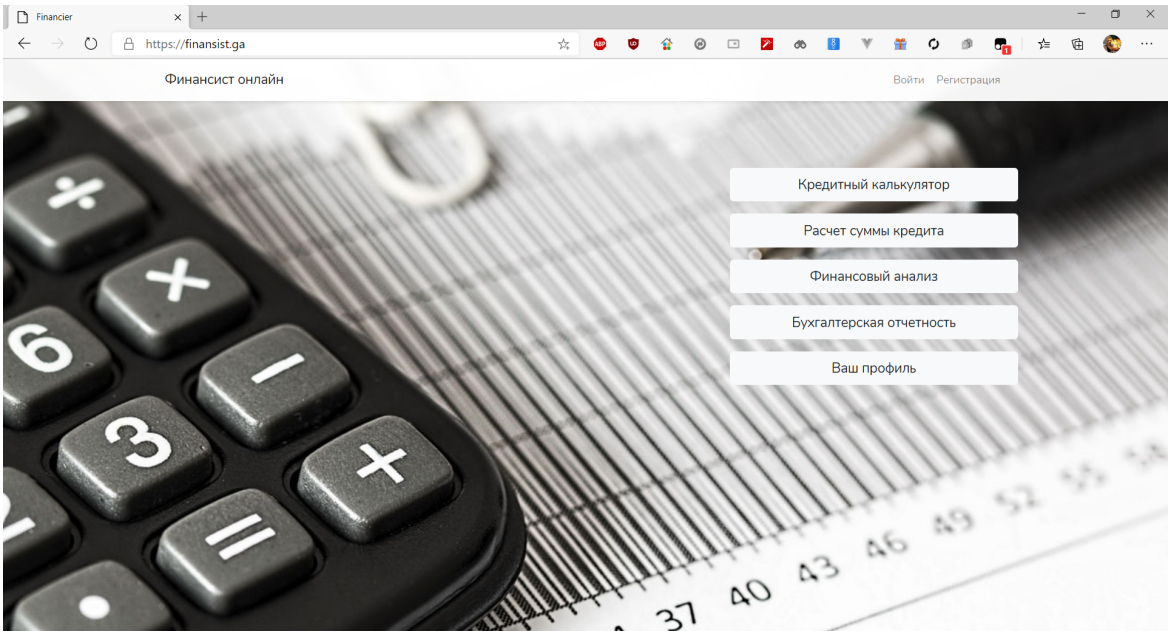


Рисунок Рефакторинг и доработка финансового веб-сервиса.33 – Запущенное в работу веб-приложение

Как можно заметить, развертывание приложения требует лишь настройки переменных окружения и выполнения всего лишь одной команды. При этом приложение работает в среде, полностью идентичной тем, что использовались на компьютерах разработчиков.

Веб-приложение «Финансист Онлайн» полностью запущено в работу.

ЗАКЛЮЧЕНИЕ

Все больше и больше различных предприятий сейчас признают, что веб-приложения – это не просто побочный продукт деятельности, а полноценный канал для ведения бизнеса. В связи с этим, требования к веб-приложениям со стороны бизнеса и пользователей продолжают расти, веб-сайты становятся полноценными программными продуктами, которые по функционалу и сложности не уступают классическим десктопным приложениям. И для того, чтобы справиться с растущей сложностью веб-приложений, в практически любом серьезном проекте приходится переходить к командной разработке.

На стадии начальной разработки нового проекта все довольно просто: кода мало, понятно назначение каждого модуля и их взаимосвязи между собой. Затем проект развивается, какие-то функции добавляются, какие-то изменяются, и в какой-то момент разработчики сами начинают путаться в собственным кодом. Если же к проекту подключится новый программист, то разобраться в структуре и логике незнакомого приложения ему будет очень сложно. Поэтому для эффективной командной разработки крайне важен правильный выбор архитектура проекты. Приложение должно строиться на основе стандартных общепризнанных подходов, с которыми знакомы большинство профессионалов-программистов.

Однако при командной разработке появляются и другие трудности, не связанные с архитектурой и структурой проекта. Например, проблемы возникает из-за неконсистентности окружений у разных разработчиков. Всего лишь одна ошибка из-за разных версий используемых инструментов может остановить разработку проекта на недели.

Поэтому важно так организовать окружение, чтобы оно было полностью идентично у всех разработчиков и при этом разворачивалось и настраивалось на их компьютерах за минимальное время.

Целями магистерской диссертации являлись разработка окружения для командной работы над веб-приложением, рефакторинг существующего проекта

финансового веб-сервиса для повышения внутреннего качества программного обеспечения и расширение функционала данного сервиса.

В ходе работы были изучены современные способы организации окружения для веб-приложений, методы размещения веб-сервисов на боевых серверах, а также методики построения проектов, позволяющие повысить эффективность командной разработки.

Было создано окружение, позволяющее запускать проект на компьютерах всех разработчиков, а также непосредственно на сервере, в полностью идентичной среде. При этом в окружение уже включены все необходимые инструменты для веб-разработки и поддержки приложения на сервере. Это позволяет развернуть окружение на чистой операционной системе и максимально быстро и просто получить среду, полностью готовую к разработке.

Был произведен полный рефакторинг веб-сервиса, разработанного в рамках бакалаврской выпускной квалификационной работы, архитектура проекта была построена на базе фреймворка Laravel. Были реализованы дополнительные алгоритмы проведения финансового анализа деятельности компаний, а также разработан механизм для оценки вероятности одобрения кредита.

Созданное окружение веб-приложения может послужить основой для других PHP-приложений, как написанных на фреймворках, так и на чистом языке PHP. При этом гарантируется полная идентичность окружения на всех компьютерах, что полностью исключит ошибки несоответствия окружения. Перевод проекта на фреймворк Laravel обеспечил хорошую основу для дальнейшего перехода проекта в командную разработку. Реализованный веб-сервис для расчета финансовых показателей может быть внедрен в организации с самыми разнообразными отраслями деятельности. С помощью финансового анализа, предоставляемого данным сервисом, компании могут повысить эффективность управления, а также значительно снизить риск ухудшения

кредитной истории за счет проведения оценки вероятности одобрения кредита еще до посещения финансовой организации.

В ходе разработки мною был изучен большой объем документации как клиентских, так и серверных технологий. Получены навыки в построении окружений для веб-приложений, в разработке клиент-серверных приложений на современных архитектурах и организации проектов для повышения эффективности командной разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Анежа Д. Container Technologies Overview / DZone [Электронный ресурс] / Д. Анежа // DZone: dzone.com. – [Б. м. : б. и.], 2019. – Режим доступа: https://dzone.com/articles/container-technologies-overview – Загл. с экрана.

2 Анкудинов С. Laravel 7 / Хабр [Электронный ресурс] / С. Анкудинов // Habrahabr: Habrahabr.ru. – [Б. м. : б. и.], 2020. – Режим доступа: https://habr.com/ru/post/502058/ – Загл. с экрана.

3 Бернстайн Л. Анализ финансовой отчетности/ Л. Бернстайн. – 1-е изд. – М. : Финансы и статистика, 2003. – 624с.

4 Ганичев А. Docker. Начало / Хабр [Электронный ресурс] / А. Ганичев // Habrahabr: Habrahabr.ru. – [Б. м. : б. и.], 2018. – Режим доступа: https://habr.com/ru/post/353238/ – Загл. с экрана.

5 Губернаторов А. Теория, методология и практика оценки кредитоспособности предприятия-заемщика / А. Губернаторов. – 1-е изд. – М. : КНОРУС, 2018. – 157с.

6 Донцова Л. Анализ финансовой отчетности/ Л. Донцова. – 1-е изд. – М. : Дело и сервис, 2008. –368с.

7 Дронов В. Laravel. Быстрая разработка современных динамических Web-сайтов на PHP, MySQL, HTML и CSS / В. Дронов. – 1-е изд. – М. : Дело и сервис, 2008. –368с.

8 Жданов В. Финансовый анализ предприятия с помощью коэффициентов и моделей. Учебное пособие / В. Жданов. – 1-е изд. – М. : Проспект, 2020. – 176с.

9 Зандстра М. PHP: объекты, шаблоны и методики программирования / М. Зандстра. – 4-е изд. – М. : И.Д. Вильямс, 2015. – 576с.

10 Золотарев О. PHP в 2019: лучше, чем вы о нём думаете / Хабр [Электронный ресурс] / О. Золотарев // Habrahabr: Habrahabr.ru. – [Б. м. : б. и.], 2019. – Режим доступа: https://habr.com/ru/company/funcorp/blog/456024/. – Загл. с экрана.

11 Камысовская С. Бухгалтерская финансовая отчетность. Формирование и анализ показателей. Учебное пособие / С. Камысовская. – 1-е изд. – М. : Инфра-М, 2014. – 432с.

12 Карпович В. 20 Eloquent ORM трюков / Хабр [Электронный ресурс] / В. Карпович // Habrahabr: Habrahabr.ru. – [Б. м. : б. и.], 2018. – Режим доступа: https://habr.com/ru/post/354036/ – Загл. с экрана.

13 Краткий обзор Symfony. Актуальность. Стоит ли попробовать? [Электронный ресурс]. – [Б. м. : б. и.], 2017. – Режим доступа: https://habr.com/ru/company/otus/blog/344726/ – Загл. с экрана.

14 Литова Е. ЦБ усложнит банкам кредитование связанных заемщиков [Электронный ресурс] / Е. Литова // РБК: rbc.ru. – [Б. м. : б. и.], 2018. – Режим доступа: rbc.ru/finances/25/06/2018/5b2cf8bd9a794706e5cfd15c – Загл. с экрана.

15 Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. / Р. Мартин – 1-е изд. – СПб. : Питер, 2018. – 352с.

16 Мерай К. The Smart Way To Handle Request Validation In Laravel [Электронный ресурс] / К. Мерай// Medium: medium.com. – [Б. м. : б. и.], 2018. – Режим доступа: https://medium.com/@kamerk22/the-smart-way-to-handle-request-validation-in-laravel-5e8886279271. – Загл. с экрана.

17 Моуэт Э. Использование Docker / Э. Моуэт – 1-е изд. – М. : ДМК Пресс, 2017. –354 с.

18 Мэнн С. What's in a Production Web Application?/ Stephen Mann [Электронный ресурс] / С. Мэнн // Stephen Mann: stephenmann.io. – [Б. м. : б. и.], 2019. – Режим доступа: https://stephenmann.io/post/whats-in-a-production-web-application/– Загл. с экрана.

19 Мэтт С. Laravel. Полное руководство/ С. Мэтт. – 2-е изд. – СПб. : Питер, 2020. – 512с.

20 Ньяупане С. PHP Software Architecture Part 1: MVC [Электронный ресурс] / С. Ньяупане// Medium: medium.com. – [Б. м. : б. и.], 2017. – Режим доступа: https://medium.com/@sameernyaupane/php-software-architecture-part-1-mvc-1c7bf042a695. – Загл. с экрана.

21 Рог В. Что такое Yii и с чего начать [Электронный ресурс] / В. Рог // WebforMyself: WebforMyself.com. – [Б. м. : б. и.], 2017. – Режим доступа: https://webformyself.com/chto-takoe-yii-i-s-chego-nachat/. – Загл. с экрана.

22 Сухова Л. Практикум по анализу финансового состояния и оценке кредитоспособности банка-заемщика / Л. Сухова. – 1-е изд. – М. : Финансы и статистика, 2003. – 152с.

23 Фаулер М. Рефакторинг кода на JavaScript: улучшение проекта существующего кода / М. Фаулер. – 2-е изд. – СПб. : Диалектика, 2019. – 464с.

24 Физерс М. Эффективная работа с унаследованным кодом. / М. Физерс. – 1-е изд. – М. : И.Д. Вильямс, 2009. – 400с.

25 Чернышов В. Docker: как развернуть фуллстек-приложение и не поседеть / В. Чернышов. – 2-е изд. – М. : Инфра-М, 2008. – 176с.

26 Шеремет А. Методика финансового анализа деятельности коммерческих организаций / А. Шеремет. – 2-е изд. – М. : Инфра-М, 2008. – 176с.

27 Эмельянов С. Laravel Mix – что это и как им пользоваться? / Эмельянов [Электронный ресурс] / С. Эмельянов // Sergeyem: sergeyem.ru. – [Б. м. : б. и.], 2019. – Режим доступа: https://sergeyem.ru/blog/35. – Загл. с экрана.

28 Get Started, Part 1: Orientation and setup [Электронный ресурс] : documentation for Docker. – [Б. м. : б. и.], 2020. – Режим доступа: https://docs.docker.com/get-started/. – Загл. с экрана.

29 Installation – Laravel – The PHP Framework For Web Artisans [Электронный ресурс] : documentation for Laravel. – [Б. м. : б. и.], 2020. – Режим доступа: https://laravel.com/docs/7.0. – Загл. с экрана.

30 PHP: Руководство по PHP [Электронный ресурс] : руководство по PHP. – [Б. м. : б. и.], 2020. – Режим доступа: http://www.php.net/manual/ru/. – Загл. с экрана.

31 VPS: что это такое? Просто о сложном [Электронный ресурс]. – [Б. м. : б. и.], 2020. – Режим доступа: https://spbit.ru/news/n180329/ – Загл. с экрана.

ПРИЛОЖЕНИЕ А

(обязательное)

Конфигурация окружения

Файл docker-compose.yml, содержащий конфигурацию Docker-окружения:

```
version: '3.7'
services:
  web:
    image: nginx:1.17.9-alpine
    container_name: financier_web
    volumes:
      - ${APP_PATH_HOST}:${APP_PATH_CONTAINER}
      - ${PHPMYADMIN_PATH}:${PHPMYADMIN_PATH_CONTAINER}
      - ${NGINX_CONFIG}:${NGINX_CONFIG_CONTAINER}
      - ${CERTBOT_CERTS_PATH}:${NGINX_SSL_CERTS_PATH}:ro
      - ${CERTBOT_WEBROOT_PATH}:${NGINX_SSL_WEBROOT_PATH}:ro
    ports:
      - "80:80"
      - "443:443"
    links:
      - app
      - phpmyadmin
    working_dir: ${APP_PATH_CONTAINER}
    restart: unless-stopped
    depends_on:
      - certbot
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!};
nginx -s reload; done & nginx -g \"daemon off;\"'"
  app:
    build: ./app
    container_name: financier_app
    volumes:
      - ${APP_PATH_HOST}:${APP_PATH_CONTAINER}
    working_dir: ${APP_PATH_CONTAINER}
    restart: unless-stopped
    user: www-data
    links:
      - database
  database:
    image: mysql:8.0.19
    container_name: financier_db
    restart: unless-stopped
    env_file:
        - .env_mysql
    volumes:
```

```
      - ${DB_PATH_HOST}:/var/lib/mysql
      - ${DB_CONFIG}:/etc/mysql/conf.d
```

Продолжение ПРИЛОЖЕНИЯ А

```
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:fpm-alpine
    container_name: financier_pma
    restart: unless-stopped
    volumes:
      - ${PHPMYADMIN_PATH}:${PHPMYADMIN_PATH_CONTAINER}
    working_dir: ${PHPMYADMIN_PATH_CONTAINER}
    links:
      - database:db
  composer:
    image: composer:latest
    volumes:
      - ${APP_PATH_HOST}:${APP_PATH_CONTAINER}
    working_dir: ${APP_PATH_CONTAINER}
    command: composer install --ignore-platform-reqs
  certbot:
    build: ./certbot
    restart: unless-stopped
    environment:
      - DOMAINS=${CERTBOT_DOMAIN1} ${CERTBOT_DOMAIN2}
      - EMAIL=${CERTBOT_EMAIL}
    volumes:
      -
${CERTBOT_CERTS_PATH}:${CERTBOT_CERTS_PATH_CONTAINER}:rw
      -
${CERTBOT_WEBROOT_PATH}:${CERTBOT_WEBROOT_PATH_CONTAINER}:rw
```

Файл .env, содержащий environment-переменные окружения:

```
#app
APP_PATH_CONTAINER=/var/www/html/site
APP_PATH_HOST=./src
#web
NGINX_CONFIG=./web/vhost.conf
NGINX_CONFIG_CONTAINER=/etc/nginx/conf.d/default.conf
NGINX_SSL_CERTS_PATH=/etc/letsencrypt
NGINX_SSL_WEBROOT_PATH=/usr/share/nginx/html
#phpmyadmin
PHPMYADMIN_PATH_CONTAINER=/var/www/html/phpmyadmin
PHPMYADMIN_PATH=./phpmyadmin
#database
DB_CONFIG=./db
DB_PATH_HOST=./database
#certbot
CERTBOT_DOMAIN1=FINANSIST.GA
CERTBOT_DOMAIN2=PMA.FINANSIST.GA
CERTBOT_EMAIL=abramizsaransk@gmail.com
CERTBOT_CERTS_PATH_CONTAINER=/etc/letsencrypt
CERTBOT_CERTS_PATH=./certbot/certbot_certs
```

```
CERTBOT_WEBROOT_PATH_CONTAINER=/webroot
CERTBOT_WEBROOT_PATH=./certbot/certbot_webroot
```

Продолжение ПРИЛОЖЕНИЯ А

Файл vhost.conf, содержащий конфигурацию сервера Nginx:

```
server {
  listen      80;
  server_name .finansist.ga;
  location ^~ /.well-known/acme-challenge/ {
  auth_basic off;
  allow all;
  root /usr/share/nginx/html;
  try_files $uri =404;
  break;
  }
  # Redirect to HTTPS
  location / { return 301 https://finansist.ga$request_uri; }
}

server {
  listen      80;
  server_name .pma.finansist.ga;
  location ^~ /.well-known/acme-challenge/ {
  auth_basic off;
  allow all;
  root /usr/share/nginx/html;
  try_files $uri =404;
  break;
  }
  # Redirect to HTTPS
  location / { return 301
https://pma.finansist.ga$request_uri; }
}
server {
  listen      443 ssl http2;
  server_name www.finansist.ga;

  # SSL
  ssl_certificate
/etc/letsencrypt/live/finansist.ga/fullchain.pem;
  ssl_certificate_key
/etc/letsencrypt/live/finansist.ga/privkey.pem;
  # Redirect to HTTPS
  location / { return 301 https://finansist.ga$request_uri; }
}
server {
  listen      443 ssl http2;
  server_name www.pma.finansist.ga;
  ssl_certificate
/etc/letsencrypt/live/pma.finansist.ga/fullchain.pem;
```

```
       ssl_certificate_key
/etc/letsencrypt/live/pma.finansist.ga/privkey.pem;
       # Redirect to HTTPS
  location / { return 301 https://pma.finansist.ga$request_uri; }
```

Окончание ПРИЛОЖЕНИЯ А

```
    }
    server {
        listen      443 ssl http2;
        server_name finansist.ga;
        index index.php index.html index.htm;
        root /var/www/html/site/public;
        # SSL
       ssl_certificate
/etc/letsencrypt/live/finansist.ga/fullchain.pem;
       ssl_certificate_key
/etc/letsencrypt/live/finansist.ga/privkey.pem;
        location / {
            try_files $uri /index.php?$args;
        }
        location ~ \.php$ {
            try_files $uri $uri/ /index.php last;
            fastcgi_split_path_info ^(.+\.php)(/.+)$;
            fastcgi_pass app:9000;
            fastcgi_index index.php;
            include fastcgi_params;
            fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
            fastcgi_param PATH_INFO $fastcgi_path_info;
        }
        location ~* .php/ { rewrite  (.*.php)/ $1 last; }
    }
    server {
        listen      443 ssl http2;
        server_name pma.finansist.ga;
        root /var/www/html/phpmyadmin;
        index index.php index.html index.htm;
        # SSL
       ssl_certificate
/etc/letsencrypt/live/pma.finansist.ga/fullchain.pem;
       ssl_certificate_key
/etc/letsencrypt/live/pma.finansist.ga/privkey.pem;

        location / {
            try_files $uri /index.php?$args;
        }
        location ~ \.php$ {
            fastcgi_split_path_info ^(.+\.php)(/.+)$;
            fastcgi_pass phpmyadmin:9000;
            fastcgi_index index.php;
            include fastcgi_params;
```

```
            fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
            fastcgi_param PATH_INFO $fastcgi_path_info;
        }
        location ~* .php/ { rewrite  (.*.php)/ $1 last; }
    }
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Функции для финансового анализа и оценки вероятности одобрения кредита

Функции для расчета коэффициентов финансового анализа:

```
//КОЭФФИЦИЕНТЫ
public function k1($credit = false)
{
 $balance_dates = $this->actual_balance_dates();
 $ds_id = $this->findArticleIdByCode('1250');
 $vcb_id = $this->findArticleIdByCode('1240');
 $k1 = $balance_dates->avg(function ($balance_date) use
($ds_id, $vcb_id, $credit) {
  $ds = $this-
>findResultValueByDateAndArticleId($balance_date, $ds_id);
  $vcb = $this-
>findResultValueByDateAndArticleId($balance_date, $vcb_id);
  return ($ds + $vcb) / ($this->ko($credit, $balance_date));
 });
 return round($k1, 2, PHP_ROUND_HALF_UP);
}

public function k2($credit = false)
{
 $balance_dates = $this->actual_balance_dates();
 $ds_id = $this->findArticleIdByCode('1250');
 $vcb_id = $this->findArticleIdByCode('1240');
 $po_id = $this->findArticleIdByCode('1260');
 $k1 = $balance_dates->avg(function ($balance_date) use
($ds_id, $vcb_id, $po_id, $credit) {
  $ds = $this-
>findResultValueByDateAndArticleId($balance_date, $ds_id);
  $vcb = $this-
>findResultValueByDateAndArticleId($balance_date, $vcb_id);
  $po = $this-
>findResultValueByDateAndArticleId($balance_date, $po_id);
  return ($ds + $vcb + $po) / ($this->ko($credit,
$balance_date));
 });
 return round($k1, 2, PHP_ROUND_HALF_UP);}

public function k3($credit = false)
{
 $balance_dates = $this->actual_balance_dates();
 $razdel_id = $this->findArticleIdByCode('12');
 $k1 = $balance_dates->avg(function ($balance_date) use
($razdel_id, $credit) {
```

```
        $razdel = BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($razdel_id)->first();
        $sum1200 = $balance_date->get_Razdel_Sum($razdel)->value;
        return ($sum1200) / ($this->ko($credit, $balance_date));
       });
       return round($k1, 2, PHP_ROUND_HALF_UP);
      }

      public function k4($credit = false)
      {
       $balance_dates = $this->actual_balance_dates();
       $razdel3_id = $this->findArticleIdByCode('13');
       $razdel4_id = $this->findArticleIdByCode('14');
       $razdel5_id = $this->findArticleIdByCode('15');
       $k1 = $balance_dates->avg(function ($balance_date) use
($razdel3_id, $razdel4_id, $razdel5_id, $credit) {
        $razdel3 =
BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($razdel3_id)->first();
        $razdel4 =
BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($razdel4_id)->first();
        $razdel5 =
BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($razdel5_id)->first();
        $v1300 = $balance_date->get_Razdel_Sum($razdel3)->value;
        $v1400 = $balance_date->get_Razdel_Sum($razdel4)->value;
        $v1500 = $balance_date->get_Razdel_Sum($razdel5)->value;
        if ($credit) {
         $gsz = $this->gsz;
         $v1300 = $v1300 + $gsz->credit_info->sum / count($gsz-
>company_work6Month());
        }
        if ($v1300 == 0) $v1300++;
        return ($v1400 + $v1500) / $v1300;
       });
       return round($k1, 2, PHP_ROUND_HALF_UP);}

      public function k5($credit = false)
      {
       $balance_dates = $this->actual_balance_dates();
       $pp_id =
FinanceReportArticle::wherecode('2200')->first()->id;
       $vp_id =
FinanceReportArticle::wherecode('2110')->first()->id;
       $k5 = $balance_dates->avg(function ($balance_date) use
($pp_id, $vp_id) {
        $pp =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($pp_id)->first()->value;
```

```
        $vp =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($vp_id)->first()->value;

        if ($vp == 0) $vp++;
        return $pp / $vp;
       });
       return round($k5, 2, PHP_ROUND_HALF_UP);
      }

      public function k5_status()
      {
       $balance_dates = $this->actual_balance_dates();
       $pp_id =
FinanceReportArticle::wherecode('2200')->first()->id;
       $vp_id =
FinanceReportArticle::wherecode('2110')->first()->id;
       $k1 = $balance_dates->sortBy('date')->reduce(function
($carry, $balance_date) use ($pp_id, $vp_id) {
        $pp =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($pp_id)->first()->value;
        $vp =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($vp_id)->first()->value;

        if ($vp == 0) $vp++;
        $k5 = $pp / $vp;
        if ($k5 > $carry['k']) $carry['status'] = true;
        else $carry['status'] = false;
        $carry['k'] = $k5;
        return $carry;
       }, [
        'status' => false,
        'k' => -999999999,
       ]);
       return $k1['status'];
      }


      public function ko($credit, $balance_date)
      {
       $ko1_id = $this->findArticleIdByCode('1510');
       $ko2_id = $this->findArticleIdByCode('1520');
       $ko5_id = $this->findArticleIdByCode('1550');
       $ko1 = $this-
>findResultValueByDateAndArticleId($balance_date, $ko1_id);
       $ko2 = $this-
>findResultValueByDateAndArticleId($balance_date, $ko2_id);
```

```
      $ko5 = $this-
>findResultValueByDateAndArticleId($balance_date, $ko5_id);
```

Продолжение ПРИЛОЖЕНИЯ Б

```
      $ko = $ko1 + $ko2 + $ko5;
      if ($credit) {
       $gsz = $this->gsz;
       $ko = $ko + ($gsz->credit_info->with_stavka()) / count($gsz-
>company_work6Month());
      }
      if ($ko == 0) $ko++;
      return $ko;
     }

     public function findArticleIdByCode($code){
      return BalanceArticle::wherecode($code)->first()->id;
     }

     public function
findResultValueByDateAndArticleId($balance_date,
$balance_article_id)
     {
      return BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($balance_article_id)->first()->val-
ue;
     }
```

Функция для вычисления индекса Альтмана:

```
     public function class_company($credit = false)
     {
      $balance_dates = $this->actual_balance_dates();
      $s1200_id = $this->findArticleIdByCode('12');
      $s1300_id = $this->findArticleIdByCode('13');
      $s1400_id = $this->findArticleIdByCode('14');
      $s1500_id = $this->findArticleIdByCode('15');
      $ss2110_id =
FinanceReportArticle::wherecode('2110')->first()->id;
      $ss2300_id =
FinanceReportArticle::wherecode('2300')->first()->id;
      $ss2400_id =
FinanceReportArticle::wherecode('2400')->first()->id;
      $x1 = $balance_dates->avg(function ($balance_date) use
($s1200_id,$s1500_id, $credit) {
       $v1200 = BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($s1200_id)->first();
       $v1500 = BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($s1500_id)->first();
       $v1200 = $balance_date->get_Razdel_Sum($v1200)->value;
       $v1500 = $balance_date->get_Razdel_Sum($v1500)->value;
       $v1600 = $balance_date->get_Balance_Active_Sum()->value;
       if ($v1600==0) $v1600++;
```

```
        if ($credit) {
         $gsz = $this->gsz;
```

Окончание ПРИЛОЖЕНИЯ Б

```
         $v1500 = $v1500 + ($gsz->credit_info->with_stavka()) /
count($gsz->company_work6Month());  }
        return ($v1200 - $v1500) / ($v1600);
       });
       $x2 = $balance_dates->avg(function ($balance_date) use
($ss2400_id, $credit) {
        $ss2400 =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($ss2400_id)->first()->value;
        $v1600 = $balance_date->get_Balance_Active_Sum()->value;
        if ($v1600==0) $v1600++;
        return $ss2400 / $v1600;
       });
       $x3 = $balance_dates->avg(function ($balance_date) use
($ss2300_id, $credit) {
        $ss2300 =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($ss2300_id)->first()->value;
        $v1600 = $balance_date->get_Balance_Active_Sum()->value;
        if ($v1600==0) $v1600++;
        return $ss2300 / $v1600;
       });
       $x4 = $balance_dates->avg(function ($balance_date) use
($s1300_id,$s1400_id,$s1500_id, $credit) {
        $v1300 = BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($s1300_id)->first();
        $v1400 = BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($s1400_id)->first();
        $v1500 = BalanceResult::wherebalance_date_id($balance_date-
>id)->wherebalance_article_id($s1500_id)->first();
        $v1300 = $balance_date->get_Razdel_Sum($v1300)->value;
        $v1400 = $balance_date->get_Razdel_Sum($v1400)->value;
        $v1500 = $balance_date->get_Razdel_Sum($v1500)->value;
        if ($credit) {
         $gsz = $this->gsz;
         $v1500 = $v1500 +  ($gsz->credit_info->with_stavka()) /
count($gsz->company_work6Month());  }
        if ($v1400+$v1500==0) $v1500++;
        return ($v1300) / ($v1400+$v1500); });
       $x5 = $balance_dates->avg(function ($balance_date) use
($ss2110_id, $credit) {
        $ss2110 =
FinanceReportResult::wherebalance_date_id($balance_date-
>id)->wherefinance_report_article_id($ss2110_id)->first()->value;
        $v1600 = $balance_date->get_Balance_Active_Sum()->value;
        if ($v1600==0) $v1600++;
        return $ss2110 / $v1600;
       });
```

```
 $class = $x1*0.717+$x2*0.847+$x3*3.107+$x4*0.420+$x5*0.998;
 return round($class, 2, PHP_ROUND_HALF_UP);
}
```