

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ»**

**(ДОННУ)**

Факультет Физико-технический

Кафедра Компьютерных технологий

Зав. кафедрой «КТ»

Т.В. Ермоленко

(подпись)

«\_\_\_» 2017 г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к выпускной квалификационной работе бакалавра на тему:

Разработка веб-приложения приема и оформления заказов типографии

Автор выпускной квалификационной работы М.Н. Миленьких

(подпись, дата)

Обозначение ВКР 09.03.01.080000.00 ВКР Группа 102321 КН-А

Направление 09.03.01 Информатика и вычислительная техника

Руководитель ВКР ст. препод. Гукай А.Е.

(подпись, дата)

Консультанты по разделам:

(наименование раздела) (подпись, дата)

(наименование раздела) (подпись, дата)

(наименование раздела) (подпись, дата)

Нормоконтроль ст. лаборант В.Г. Медведева

(подпись, дата)

Донецк

2017



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ»**

**(ДОННУ)**

Факультет Физико-технический

Кафедра Компьютерных технологий

Зав. кафедрой «КТ»

Т.В. Ермоленко

«\_\_\_» 2016 г.

**ЗАДАНИЕ**

к выпускной квалификационной работе бакалавра

Студент М.Н. Миленьких Группа 102321 КН-А

Обозначение ВКР 09.03.01.080000.00 ВКР

Тема Разработка веб-приложения приема и оформления заказов типографии

Утверждено приказом по ДонНУ № 233/08 от «15» марта 2018 г.

Срок представления ВКР к защите «15» июня 2018 г.

Исходные данные для ВКР:

1. Информация об автопарковках
2. Информация о машинах
3. Карта местности
4. Нормативно-правовая документация, касающася автомобилистов и автопарковок
5. Документация по работе с html-сервером Apache.
6. Документация по языку разработки серверных приложений РНР.
7. Документация по Фреймверку Laravel 5
8. Документация по серверу баз данных MYSQL

Содержание пояснительной записки

ВВЕДЕНИЕ:

Целью данной курсовой работы является разработка системы парковок автомобилей. Основными задачами приложения является поиск свободных парковок, бронирование места заранее и конечно же мониторинг ситуации, происходящей с автомобилем, в тот момент, когда хозяин автомобиля занимается своими делами и не может лично следить за сохранностью своего автомобиля.

Наименование и содержание разделов:

1. Анализ предметной области: состояние вопроса, функции, моделирование существующих бизнес-процессов, обоснование актуальности и цели работы.
2. Техническое задание: описание области применения и исходных данных приложения, требования к пользовательским интерфейсам, требования к аппаратным, программным и коммуникационным интерфейсам, требования к пользователям продукта, требования к адаптации на месте, функции программного продукта, ограничения.
3. Проектирование и разработка клиент-серверного взаимодействия: обзор протоколов и архитектуры веб-приложения
4. Проектирование и разработка базы данных: обоснование выбора СУБД, инфологическая, концептуальная, логическая, физическая схемы базы данных, SQL запросы
5. Разработка клиент-серверного приложения «Автостоянка»: описание процесса дизайна, верстки и разработки серверной части веб-приложения.
6. Руководство по использованию программного продукта
7. Тестирование программного продукта: описание аппаратных и программных средств создания и эксплуатации приложения, описание контрольных примеров работы с приложением.

ЗАКЛЮЧЕНИЕ:

Работа выполнена в соответствии поставленному заданию на 100%. Доработка связана с расширением функциональных возможностей.

Перечень графического материала:

1. Модель бизнес-процессов автостоянки.
2. Входные и выходные данные интернет-приложения.
3. Физическая модель базы данных.
4. Структура интернет-приложения.
5. Экранные формы администраторской части приложения.
6. Экранные формы клиентской части приложения.

Руководитель ВКР ст. препод. Гукай А.Е.

(подпись, дата)

Консультанты по разделам:

(наименование раздела) (подпись, дата)

(наименование раздела) (подпись, дата)

(наименование раздела) (подпись, дата)

Задание принял к исполнению М.Н. Миленьких

(подпись, дата)

## АННОТАЦИЯ

Отчет о курсовой работе: 93 с., 47 рис., 11 табл., 1 приложение, 11 источников.

Объектом исследования является система формирования заказа в типографии города.

Предметом исследования является веб-приложение типографии, предоставляющее функционал для оформления заказов от клиентов, подвязки дизайна изделия к заказу, передача заказа исполнителю и логистика заказа по окончанию работ.

Цель работы – разработка веб-приложения приема и оформления заказов типографии.

Метод исследования – изучение технических средств, необходимых для разработки веб-приложения, принципов работы с базой данных через СУБД MySQL и Фреймворк ORM Eloquent.

В работе были использованы технологии Laravel 5.х, AJAX, jQuery, Фреймворк Materialize, SCSS, MySQL, IDE phpStorm 2017, Photoshop, VBA.

В результате решения задачи было разработано клиент-серверное приложение для автоматизации работы типографии, база данных для приложения, веб-сайт, сопутствующий скрипт для формирования прайса типографии.

Дальнейшее развитие программы связано с расширением ее возможностей, увеличением числа опций, автоматизации процессов, происходящих внутри типографии.

WEB-ПРИЛОЖЕНИE, СУБД, MYSQL, FRAMEWORK, AJAX, HTML,CSS, JQUERY, LARAVEL, ORM ELOQUENT, JSP,PHP, MVC, VBA.

ANNOTATION

Report on the course work:.93 p 47 img, Table 11, 1 an application 1 sources 11.

The object of research is the parking system is a major metropolis and the processes taking place in the parking lot in the course of its work.

The subject of the research program is a parking management service.

The purpose of the work - a client-server application development "Parking".

Method of research - the study of the technical means necessary for the development of Web applications, principles of work with the database through MySQL database and framework ORM Eloquent.

In this work were used technology Laravel 5, AJAX, AngularJS, HTML5, CSS3, MySQL, IDE phpStorm 2016.

As a result of solving the problem of a client-server application to automate the parking lot was designed for database application site.

Further development of programs related to the expansion of its capabilities, increasing the number of options.

WEB-PRILOZHENIE, database, MYSQL, FRAMEWORK, AJAX, HTML, CSS, ANGULARJS, LARAVEL, ORM ELOQUENT, JSP, PHP, MVC.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **СОДЕРЖАНИЕ** Изм.  Лист  № докум.  Подпись  Дата  Лист  6  09.03.01.080000.000 ПЗ  Разраб.  Миленьких М.  Провер.  Гукай А.Е.    Н. Контр.  Медведева В..  Утверд.  Ермоленко Т.В.  *Разработка веб-приложения приема и оформления заказов типографии*  Лит.  Листов  93  ДОННУ  кафедра «КТ» | | | | ВВЕДЕНИЕ | | 8 | | 1 | Анализ предметной области | 10 | |  | 1.1 Состояние вопроса | 10 | |  | 1.2 Актуальность и цель работы | 11 | | 2 | Техническое задание | 15 | |  | 2.1 Область применения продукта | 15 | |  | 2.2 Требования к пользователям продукта |  | |  | 2.3 Требования к структуре приложения |  | |  | 2.4 Требования к обмену данными |  | |  | 2.5 Требования к вспомогательному программному обеспечению | 15 | | 3 | Проектирование бизнес-процессов в системе |  | | 4 | Проектирование и разработка клиент-серверного взаимодействия | 27 | | 5 | Проектирование и разработка базы данных | 30 | |  | 4.1 Обоснование выбора СУБД | 30 | |  | 4.2 Инфологическая модель базы данных | 39 | |  | 4.3 Концептуальная модель базы данных | 41 | |  | 4.4 Физическая модель базы данных | 42 | |  | 4.5 Реализация базы данных в ORM Eloquent | 43 | | 6 | Процесс разработка клиент-серверного приложения | 55 | |  | 5.1 Разработка макета веб-интерфейса | 55 | |  | 5.2 Верстка макета веб-интерфейса | 57 | |  | 5.3 Разработка серверной части веб-приложения | 58 |  |  |  |  | | --- | --- | --- | | **7** | **Руководство по использованию программного продукта**  Изм.  Лист  № докум.  Подпись  Дата  Лист  7  09.03.01.080000.000 ПЗ | **62** | |  | **6.1 Файловая структура веб-приложения** | **62** | |  | **6.2 Руководство администратора** | **64** | |  | **6.3 Руководство пользователя** | **66** | | **8** | **Оценка надежности программного продукта** | **75** | |  | **7.1 Оценка качества разработанного ПО** | **75** | |  | **7.2 Тестирование веб-интерфейса** | **77** | |  | **7.3 Тестирование серверной части и базы данных** | **79** | | **ЗАКЛЮЧЕНИЕ** | | **82** | | **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** | | **84** | | **ПРИЛОЖЕНИЕ А (обязательное) Фрагменты листинга** | | **85** | | **ВЕДОМОСТЬ РАБОТЫ** | |  | | | |
|  | |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# ВВЕДЕНИЕ

Современный мир – мир постоянного движения, мир нововведений и технологий. Всё стремительно течет и меняется, и те вещи, которые раньше требовали больших затрат человеческих усилий и времени, теперь могут быть выполнены в одно нажатие мышки на компьютере, либо вообще без каких-либо человеческих усилий, автоматически, по графику как часы. Данную возможность нам дало стремительное развитие информационных технологий в 21 веке. Рутинные задачи теперь могут быть выполнены программой, а человек в этом процессе занимает роль управленца, оператора, который только проверяет правильность процесса, и, в случае необходимости, вносит коррективы.

Работа в типографии зачастую однообразна. В общем виде её можно описать следующий образом: клиенты каждый день приходят в офис, приносят что-то готовое для печати или же просят сотрудника типографии помочь в изготовлении макета. Продукты в типографии так же зачастую однотипны: продукция различных стандартизированных форматов, которая может быть отпечатана на листах разной плотности, с использованием различных типов бумаги и краски. Чтоб облегчить этот каждодневный было решено разработать специально приспособленную под нужды конкретной типографии(с её внутренними потребностями) веб-систему, которая бы автоматизировала работу с данными по продукции, а так же упростила работу с клиентами, сведя всю работу к нескольким кликам мышки менеджера-операциониста.

Целью данной дипломной работы является разработка системы для автоматизации работы типографии, а именно автоматизация приема заказа, назначение ответственных за выполнение работ, автоматический расчет цены для клиента и прибыли для руководителя типографии. Так же в объем поставленных задач входит сбор данных о клиентах, для последующего распространения рекламно-агитационной продукции среди них, и решения логистических задач самой типографии. Наряду с основной целью есть вспомогательная цель – автоматизация подсчетов финальной цены товара в зависимости от параметров, заданных в формуле.

Объектом исследования является система формирования заказа в типографии города.

Предметом исследования является веб-приложение типографии, предоставляющее функционал для оформления заказов от клиентов, подвязки дизайна изделия к заказу, передача заказа исполнителю и логистика заказа по окончанию работ.

**1 Анализ предметной области**

* 1. Состояние вопроса

Для начала работы над приложением был поставлен вопрос «существуют ли подобные сервисы в среде интернет на текущий момент?». Для объективного анализ были сделаны всевозможные запросы со словом «типография» и «калькулятор типографии», и на основе этих запросов были найдены узкоспециализированные сервисы по распространению типографской продукции, а так же калькуляторы для расчета цен на изделия типографии. Для сравнения были выбраны три аналога: «Типография сборных типажей» (сайт: <http://tst.org.ua/>), «ВОСТОК ПРЕСС ПЛЮС» (сайт: http://xerox.dn.ua ) и Издательско-полиграфический центр «Edit». Рассмотрим детальнее каждый из сервисов.

Таблица 1 – сравнение трех случайно найденных веб-сайтов типографий

|  |  |  |  |
| --- | --- | --- | --- |
| Критерии | Типография сборных типажей | «ВОСТОК ПРЕСС ПЛЮС» | Издательско-полиграфический центр «Edit» |
| Удобство просмотра информации пользователем | 8 | 8 | 3 |
| Удобство поиска информации | 10 | 7 | 2 |
| Уровень ценовой политики | 8 | 8 | 8 |
| Удобство формирование заказа | 10 | 5 | 2 |
| Удобство логистики | 5 | 5 | 0 |
| Полнота доступных видов продукции | 10 | 10 | 3 |
| Возможность обратной связи с заказчиком(авто перезвон) | 5 | 5 | 5 |
| Информационная перегруженность | 3 | 7 | 10 |
| Внешний вид сайта | 10 | 8 | 2 |

Все критерии, которые присутствуют в таблице, характеризовались следующими цифровыми значениями от 0 до 10, где 0 – максимально неудобно и плохо, а 10 – соответственно максимально удобно, логично, понятно и хорошо сделано.

Проанализировав парковки по данным критериям было выяснено:

1. Не всегда на сайтах типографии удобно работать с выбранным продуктом. Часто итоговый продукт является суммой ряда продуктов, которые нужно как-то объединить в одно целое, прикрепить к этому свой макет и быстро, и удобно отправить на изготовление, да и еще получить обратно по указанному адресу.
2. Неудобство логистических решений во всех типографиях, участвовавших в анализе;
3. Перегруженность информацией основных страниц сайта;
4. Внешний вид сайтов иногда попросту отталкивает заказчика или же делает его работу с сайтом не комфортной;
5. Спектр услуг (продуктов) типографии ограничивается возможностью конкретно выбранной типографии и не предоставляет универсальный интерфейс для заказа продукта, в других типографиях, в случае отсутствия данной возможности в выбранной. Данная часть могла бы приносить прибыль выбранной типографии как посреднику, который нашел место, в котором будет выполнен заказ, а по завершению работ доставил бы заказчику;
6. Ценовая политика сайта не учитывает цены в этой сфере у других типографий, цена не привязана к курсу валюты, сложность в обновлении цен со стороны администратора.
   1. Актуальность и цель работы

Реализация данной работы преследует следующие цели:

1. Решение проблем существования стихийных парковок
2. Наблюдение за автопарковками и как следствие уменьшение числа преступлений, связанных с угоном автомобилей с парковок и кражей имущества из салона автомобилей
3. Учет автопарковок
4. Уменьшение пробок на улицах мегаполисов за счёт создания специализированных парковок и равномерной транспортной нагрузки на все парковки города
5. Централизация парковок в единой среде управления парковками

Можно выделить ряд задач, выполнение которых приведет к выполнению поставленных целей:

1. Разработка базы данных клиент-серверного приложения «Автопарковка»
2. Разработка Rest API для создания гибких интерфейсов отдельно взятых автопарковок
3. Разработка современного веб-интерфейса
4. Разработка системы обратной связи с автовладельцем
5. Разработка системы учета автопарковок и автомобилей на автопарковках
6. Разработка системы бронирования мест на парковке

**2 Техническое задание**

**2.1 Область применения продукта**

Данный продукт рассчитан на пользователей, которые активно пользуются собственным транспортным средством передвижения и являются владельцами устройств с выходом в интернет. Приложение позволяет оперативно просматривать состояние загруженности автопарковок выбранного региона и бронировать свободное место за некоторое время до физического прибытия автомобиля. Приложение может быть применено в мегаполисах с высокой транспортной загруженностью для разгрузки улиц города от стихийных «бросовых» парковок.

**2.2 Требования к пользователям продукта**

Пользователи системы:

1) заказчик (по умолчанию)

2) менеджер активных продаж

3) менеджер-операционист

4) закупщик\приемщик заказов

5) бухгалтер

6) логист

7) дизайнер

8) директор

9) Администратор системы

Таблица 1 – Функции пользователей в зависимости от их роли

|  |  |  |
| --- | --- | --- |
| № п\п | Роль | Функция |
| 1 | Заказчик | В случае реализации внешнего магазина для клиентов будет иметь права просмотра каталога товаров, добавления их в корзину и оформления заказа |
| 2 | менеджер активных продаж | делает просчёты, продаёт создание дизайна и полиграфию |
| 3 | менеджер-операционист | Работает с входящими запросами, делает просчеты |
| 4 | закупщик\приемщик заказов | В просчете заполняет информацию по возможности выполнения заказа, работает с менеджерами по запросам, передает заказ на печать в соответствующую типографию |
| 5 | Бухгалтер | Видит все колонки, связанные с финансовыми операциями |
| 6 | Логист | Вносит логистическую информацию: дата доставки, итоговая сумма продажи со скидкой или наценкой клиенту, выбор службы доставки, описание доставки, стоимость доставки, трек-номер, имя и номер телефона курьера, информацию по дополнительной службе доставки (цена и телефон). Может просматривать рекомендации по |
| 7 | Дизайнер | Просмотр заказа и файлов, связанных с заказом, редактирование цены выполнения дизайна |
| 8 | Директор | Суперадмин, имеет возможность добавления, изменения, удаления всех полей во всех таблицах. |
| 9 | Администратор системы | Добавление, редактирование, удаление информации из таблиц |

Таблица 2 – Доступный функционал в зависимости от роли пользователя

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Роли  Функции | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Кнопка и форма «СДЕЛАТЬ ПРОСЧЁТ» | - | + | + | + | + | - | - | + | - |
| Кнопка и форма «СДЕЛАТЬ ДИЗАЙН» | - | + | + | + | + | - | + | + | - |
| Кнопка и форма «ЗАКАЗ НА ПЕЧАТЬ» | - | + | + | + | - | - | + | + | - |
| Кнопка и форма «ИНФОРМАЦИЯ ПО ВСЕМ ЗАКАЗАМ» | - | + | + | + | + | + | + | + | - |
| Форма «таблица с ценами» | - | + | + | + | + | - | - | + | - |
| Форма «калькулятор расчета стоимости» | - | + | + | + | + | - | - | + | - |
| Создание индивидуального просчета | - | - | + | + | - | - | - | + | - |
| Добавление информации о клиенте | - | + | + | + | - | - | - | + | - |
| Редактирование индивидуального просчета | - | - | + | + | - | - | - | + | - |
| Указание финальной цены в просчете | - | - | - | + | - | - | - | + | - |
| Формирование заказа | - | - | + | - | - | - | - | + | - |
| Добавление дизайна к просчету | - | - | + | - | - | - | - | + | - |
| Просмотр и редактирование плановой цены дизайна | - | - | - | - | + | - | - | + | - |
| Просмотр и редактирование «стоимость дизайна» | - | - | - | - | + | - | + | + | - |
| Работа с файлами внутри файловой системы | - | - | + | + | - | - | + | + | + |
| Добавление и просмотр логистической информации | - | - | - | + | + | + | - | + | - |
| Просмотр рекомендаций для логиста | - | - | - | - | - | + | - | + | - |
| Просмотр рекомендаций для менеджера | - | + | + | - | - | - | - | + | - |
| Просмотр рекомендаций для закупщика | - | - | - | - | + | - | - | + | - |
| Просмотр неоплаченных заказов | - | - | + | - | + | + | - | + | - |
| Статус оформления доставки | - | - | - | - | - | + | - | + | - |
| Добавление ролей сотрудникам | - | - | - | - | - | - | - | + | + |
| Добавление новых сотрудников | - | - | - | - | - | - | - | + | + |
| Управление таблицами системы | - | - | - | - | - | - | - | + | + |

**2.3 Требования к структуре приложения**

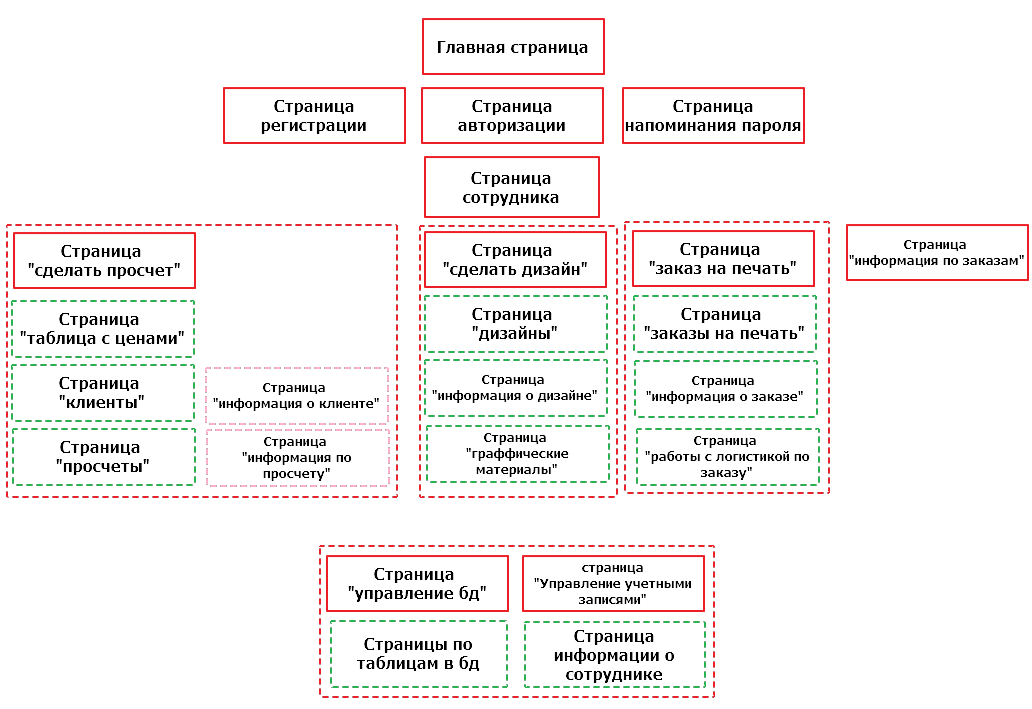


Рисунок 1 – Карта сайта. Общая структура.

Карта маршрутов в проекте (CRUD-система)

CRUD (сокр. от англ. create, read, update, delete — «создать, прочесть, обновить, удалить») — акроним, обозначающий четыре базовые функции, используемые при работе с персистентными хранилищами данных: создание, чтение, редактирование, удаление.

Таблица 3 – основные маршруты для реализации системы.

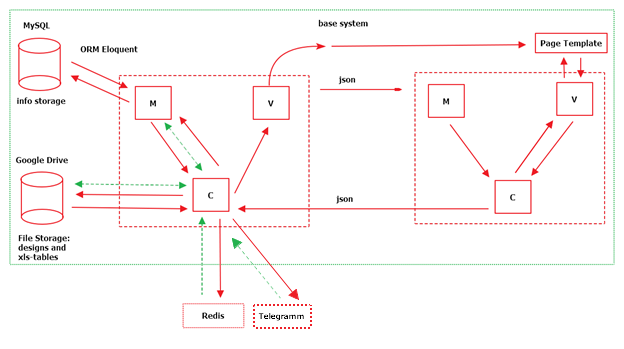
|  |  |  |
| --- | --- | --- |
| Method | Путь | Комментарий |
| 1 | 2 | 3 |
| get | / | Главная страница |
| get | /login | Страница входа |
| get | /registration | Страница регистрации |
| get | /remind | Страница напоминания пароля |
| get | /logout | Путь для выхода из системы |
| get | /price | Открывает окно с таблицей цен на изделия |
| get | /calc | Запускает калькулятор расчетов |
| post | /telegram/getupdates | Получаем информацию о сообщении в телеграме |
| get | /admin/make | Выводит оcновной cлужебный функционал для админиcтратора базы данных. |
| get | /admin/make/backup | Сохраняет данные бд на гугл диcк. |
| post | /admin/make/restore | Восстанавливает данные из выбранного бэкапа |
| post | /admin/make/reload | Загружает обновления таблиц из гугл диcка |
| post | /admin/make/calc/pdf | Сгенерировать печатную версию расчёта из калькулятора |
| get | /admin | Вход на основную рабочую страницу персонала |
| get | /admin/calculation | Страница «сделать просчет» |
| get | /admin/calculation/list | Страница со списком всех просчетов в системе. Доступ к функционалу просчетов для сотрудников. |
| post | /admin/calculation/list/{page?} | JSON-данные о просчетах по страницам. |
| get | /admin/calculation/get/{id} | Страница выбранного просчета. |
| post | /admin/calculation/get/{id} | JSON-данные по выбранному просчету. |
| get | /admin/calculation/del/{id} | Удаление выбранного просчета. |
| get | /admin/calculation/add | Страница добавления просчета |
| post | /admin/calculation/add | Метод добавления просчета в базу данных |
| get | /admin/calculation/update/{id} | Страница обновления данных в просчете |
| post | /admin/calculation/update/{id} | Метод обновления данных в таблице просчетов |
| post | /admin/calculation/filter | Фильтрация данных\поиск по нужному критерию |
| get | /admin/clients | Страница работы с клиентами компании |
| get | /admin/clients/list | Страница списка клиентов компании |
| post | /admin/clients/list/{page} | JSON-данные о клиентах по страницам. |
| get | /admin/clients/get/{id} | Страница выбранного клиента. |
| post | /admin/clients/get/{id} | JSON-данные по выбранному клиенту. |
| get | /admin/clients/del/{id} | Удаление выбранного клиента. |
| get | /admin/clients/add | Страница добавления нового клиента |
| post | /admin/clients/add | Метод добавления клиента в базу данных |
| get | /admin/clients/update/{id} | Страница обновления данных клиента |
| post | /admin/clients/update/{id} | Метод обновления данных в таблице клиентов |
| post | /admin/clients/filter | Фильтрация данных\поиск по нужному критерию |
| get | /admin/design | Страница работы с дизайнами клиентов |
| get | /admin/design/list | Страница списка дизайнов |
| post | /admin/design/list/{page?} | JSON-данные о дизайнах по страницам. |
| get | /admin/design/get/{id} | Страница выбранного дизайна. |
| post | /admin/design/get/{id} | JSON-данные по выбранному дизайну. |
| get | /admin/design/del/{id} | Удаление выбранного дизайна. |
| get | /admin/design/add | Страница добавления нового дизайна |
| post | /admin/design/add | Метод добавления дизайна в базу данных |
| get | /admin/design/update/{id} | Страница обновления данных выбранного дизайна |
| post | /admin/design/update/{id} | Метод обновления данных в таблице дизайна |
| post | /admin/design/filter | Фильтрация данных\поиск по нужному критерию |
| get | /admin/files | Страница работы с файлами |
| get | /admin/files/list | Страница списка файлов |
| post | /admin/files/list/{page?} | JSON-данные о файлах по страницам. |
| get | /admin/files/get/{id} | Страница выбранного файла. |
| post | /admin/files/get/{id} | JSON-данные по выбранному файлу. |
| get | /admin/files/del/{id} | Удаление выбранного файла. |
| get | /admin/files/add | Страница добавления нового файла |
| post | /admin/files/add | Метод добавления файла в базу данных |
| post | /admin/files/filter | Фильтрация данных\поиск по нужному критерию |
| get | /admin/orders | Страница работы с заказами компании |
| get | /admin/orders/list | Страница списка заказов компании |
| post | /admin/orders/list/{page?} | JSON-данные о заказах по страницам. |
| get | /admin/orders/get/{id} | Страница выбранного заказа. |
| post | /admin/orders/get/{id} | JSON-данные по выбранному заказу. |
| get | /admin/orders/del/{id} | Удаление выбранного заказа. |
| get | /admin/orders/add | Страница добавления нового заказа |
| post | /admin/orders/add | Метод добавления заказа в базу данных |
| get | /admin/orders/update/{id} | Страница обновления данных заказа |
| post | /admin/orders/update/{id} | Метод обновления данных в таблице заказов |
| post | /admin/orders/filter | Фильтрация данных\поиск по нужному критерию |

Работа с таблицами будет осуществляться аналогично работы с основными страницами.

tableName – название таблицы, для которой будет реализованы маршруты

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| get | /admin/tableName | Страница работы с данными в таблице |
| get | /admin/tableName/list | Страница списка данных из таблицы |
| post | /admin/tableName/list/{page?} | JSON-данные по страницам |
| get | /admin/tableName/get/{id} | Страница выбранного поля из базы данных |
| post | /admin/tableName/get/{id} | JSON-данные по выбранному полю из базы данных. |
| get | /admin/tableName/del/{id} | Удаление выбранного поля. |
| get | /admin/tableName/add | Страница добавления нового элемента в конкретную таблицу. |
| post | /admin/tableName/add | Метод добавления данных в базу данных |
| get | /admin/tableName/update/{id} | Страница обновления данных |
| post | /admin/tableName/update/{id} | Метод обновления данных в таблице |
| post | /admin/tableName/filter | Фильтрация данных\поиск по нужному критерию |

**2.4 Требования к обмену данными**



Клиент

Сервер

Рисунок –схема обмена данными между клиентом и сервером

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

По данной схеме и сервер, и клиент работаю с использованием паттерна MVC. Данные для клиента отдаются в виде JSON-массивов. Однако, в вместе с этим, сервер отдает еще страницы, содержащие внешний вид, в который будут подставлены данные, содержащие JSON. Данные, которые вносятся сотрудниками на гугл диск автоматически переносятся в базу данных проекта при помощи внешнего сервиса очередей (отложенного выполнения задачи).

Для общения между менеджером и закупщиком можно использовать телеграмм, чтоб не реализовывать собственные функции чата.

**2.5 Требование к входящим и исходящим данным**

Для корректного ввода данных в поля форм сотрудниками компании применим возможности html5 – паттерны, принимающие на вход регулярные выражения в виде шаблона, который ограничивает возможность ввода сторонних данных в систему.

Таблица - Паттерны для данных, принимаемых от сотрудников

|  |  |  |
| --- | --- | --- |
| № | Паттерн | Использование |
| 1 | ^[a-zA-Zа-яА-Я][a-zA-Zа-яА-Я0-9-\_\.]{1,255}$ | Наименование макета |
| 2 | ^[a-zA-Zа-яА-Я]{1,255}$ | Ф.И.О. клиента |
| 3 | [\+]\d{2}[\(]\d{3}[\)]\d{3}[\-]\d{2}[\-]\d{2} | Телефон клиента\получателя |
| 4 | ^[a-zA-Zа-яА-Я][a-zA-Zа-яА-Я0-9-\_,\.]{1,255}$ | Адресс доставки |
| 5 | (0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])[- /.](19|20)\d\d | Дата доставки |
| 6 | (0[0-9]|1[0-9]|2[0-3])(:[0-5][0-9]){2} | Время доставки |
| 7 | ^[a-zA-Zа-яА-Я]{1,255}$ | Получатель |
| 8 | ^[0-9]{1,255}[,.][0-9]{1,2}$ | Оплата\стоимость |
| 9 | ^[a-zA-Zа-яА-Я][a-zA-Zа-яА-Я0-9-\_\.]{1,255}$ | Бумага |
| 10 | ^[a-zA-Zа-яА-Я][a-zA-Zа-яА-Я0-9-\_\.]{1,255}$ | Ламинация |
| 11 | ^[a-zA-Zа-яА-Я][a-zA-Zа-яА-Я0-9-\_\.]{1,255}$ | Формат |
| 12 | ^[0-9]{1,255}$ | Тираж |
| 13 | ^[a-zA-Zа-яА-Я][a-zA-Zа-яА-Я0-9-\_\.]{1,255}$ | цветность |
| 14 | ^[a-zA-Zа-яА-Я][a-zA-Zа-яА-Я0-9-\_\.]{1,255}$ | После печатная обработка |
| 15 | https?:\/\/.+ | Ссылка на сайта |
| 16 | ^(?:0|\(?\+33\)?\s?|0033\s?)[1-79](?:[\.\-\s]?\d\d){4}$ | Вайбер\watsup |
| 17 | ^[a-zA-Z0-9\.,@\_-]{1,255}$ | Skype |
| 18 | [a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,3}$ | e-mail |
| 19 | ^[a-zA-Z0-9\.,@\_-]{1,255}$ | Telegram |
| 20 | (?=^.{8,}$)((?=.\*\d)|(?=.\*\W+))(?![.\n])(?=.\*[A-Z])(?=.\*[a-z]).\*$ | Пароль |
| 21 | ^[a-zA-Z][a-zA-Z0-9-\_\.]{1,20}$ | Логин |

**2.6 Требования к вспомогательному программному обеспечению**

Для разработки и технического обслуживания данного веб-приложения необходимо следующее программное обеспечение:

Таблица 2– требование к программным средствам обслуживающего персонала

|  |  |
| --- | --- |
| Название программного средства | Функции |
| Веб-браузер | 1. Просмотр содержимого веб-страницы; 2. Отладка веб-интерфейса; 3. Добавление, удаление, редактирование данных в базе данных; |
| База данных | Хранение и выдача информации для веб-приложение |
| Система контроля версией | 1. Отслеживание последних изменений в коде приложения; 2. Своевременное обновление функционала веб-приложения; 3. Загрузка исправлений ошибок веб-приложения; |
| Система загрузки зависимостей Composer | Обновление библиотек веб-приложения |
| Файловый менеджер FileZilla | Обновление данных веб-приложения на удаленном хостинге |
| Локальный сервер | Отладка и тестирование версий веб-приложения |
| Операционная система | Windows 7 и выше, либо UNIX |
| Почтовый клиент | Оперативный обмен информацией с клиентами |
| Среда разработки | Обновление функционала веб-приложения, исправление возникших в процессе работы веб-приложения ошибок |

Для использования веб-приложения достаточно наличие веб-браузера и соединения интернет.

**2.7 Требования к программным интерфейсам**

При разработке интерфейса сайта будут учитываться существующие стандарты дизайна, а также потребности будущих пользователей. Каждая страница должна иметь четкую, ясную и предельно простую организацию, которая не вызовет вопросов у пользователя.

**2.8 Функции продукта**

1. Регистрация нового пользователя в системе
2. Авторизация зарегистрированных пользователей
3. Предоставление полной информации о студентах
4. Возможность печати заявок, квитанций
5. Оплата проживания в общежитии через выбранную платежную систему
6. Доступ к информации на трех уровнях: студента, модератора, администратора
7. Просмотр и изменение личной пользовательской информации
8. Организация системы оповещения студентов
9. Обеспечение обратной связи между студентами и персоналом студенческого городка
10. Возможность электронной заявки на вызов технического персонала

**2.6 Требования к базам данных**

База данных для приложения должна содержать данные о месте расположения парковок, текущую наполняемость парковок, информацию об автомобилях, расположенных на проковках.

Также потребуется обеспечить связи между таблицами. База данных должна хранить большие объемы данных и обеспечить удовлетворительную скорость работы.

Для выбора подходящей базы данных необходимо произвести сравнительных анализ популярных баз данных по желаемым критериям и выбрать необходимую.

## **3 Моделирование бизнес-процессов**

Моделирование бизнес-процессов является крайне важным этапом разработки, так как позволяет оценить работу предприятия и в дальнейшем оптимизировать его работу.

Для описания бизнес-процессов для данного курсового проекта была выбрана нотация IDEF0, которая позволяет описать процессы в их функциональном аспекте. Главными компонентами моделей являются блоки и дуги. В свою очередь дуги и их вхождение обуславливают тип интерфейса:

1. Если дуга входит в блок сверху – это управляющая информация
2. Если дуга входит в блок слева – это входная информация
3. Если дуга выходит из блока справа – это результаты
4. Если дуга выходит из блока снизу – это механизм

В основе выбранной нотации лежит принцип о том, что каждый компонент модели может быть разложен на мелкие детали (декомпозирован) с целью детализации процесса, наглядного представления.

На рисунке 1 отображена декомпозиция контекстной диаграммы. Ее задача – отображение в целом процессов, которые описывают структуру и функции проекта. На данной диаграмме реализованы следующие интерфейсы:

1. Управляющая информация – перечень услуг, нормативная документация
2. Входная информация – информация от клиента, информация об автомобиле
3. Результаты – постановка автомобиля под охрану, выдача автомобиля в том же состоянии, в котором он был изначально заявлен владельцем.
4. Механизм – оплата услуг, персонал, обслуживающий автопарковку

Общий блок разбивается на следующие дочерние: парковочная площадка, парковщик, кассовый аппарат, охрана автомобиля. Схема работы такова: клиент приезжает на парковочную площадку и передает автомобиль парковщику, который руководствуясь перечнем услуг и нормативно-правовой документаций, размещает автомобиль на парковке под охраной, при этом выдает счет-фактуру согласно перечню услуг на оплату предоставляемой услуги. После оплаты счета автомобиль полноценно помещается под охрану. Клиент может забрать автомобиль в любое удобное ему время. Если автомобиль размещался дольше оплаченного срока, то клиент получает повторно счет-фактуру, для оплаты времени размещения автомобиля и доп. услуг помимо уже оплаченного.

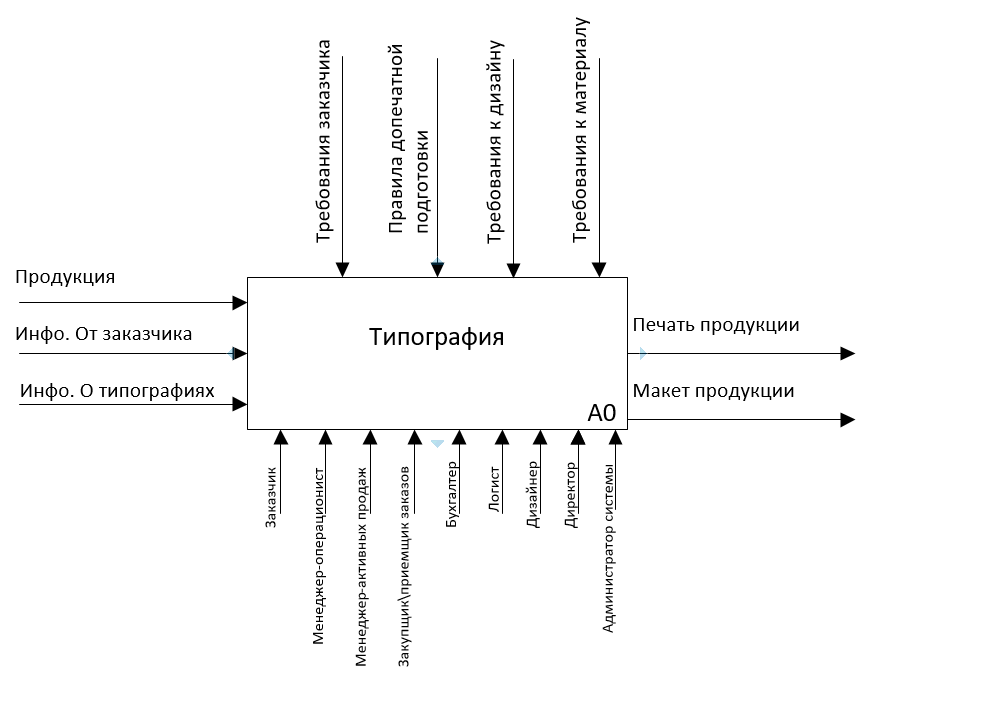


Рисунок 1 – Описание функциональной модели верхнего уровня

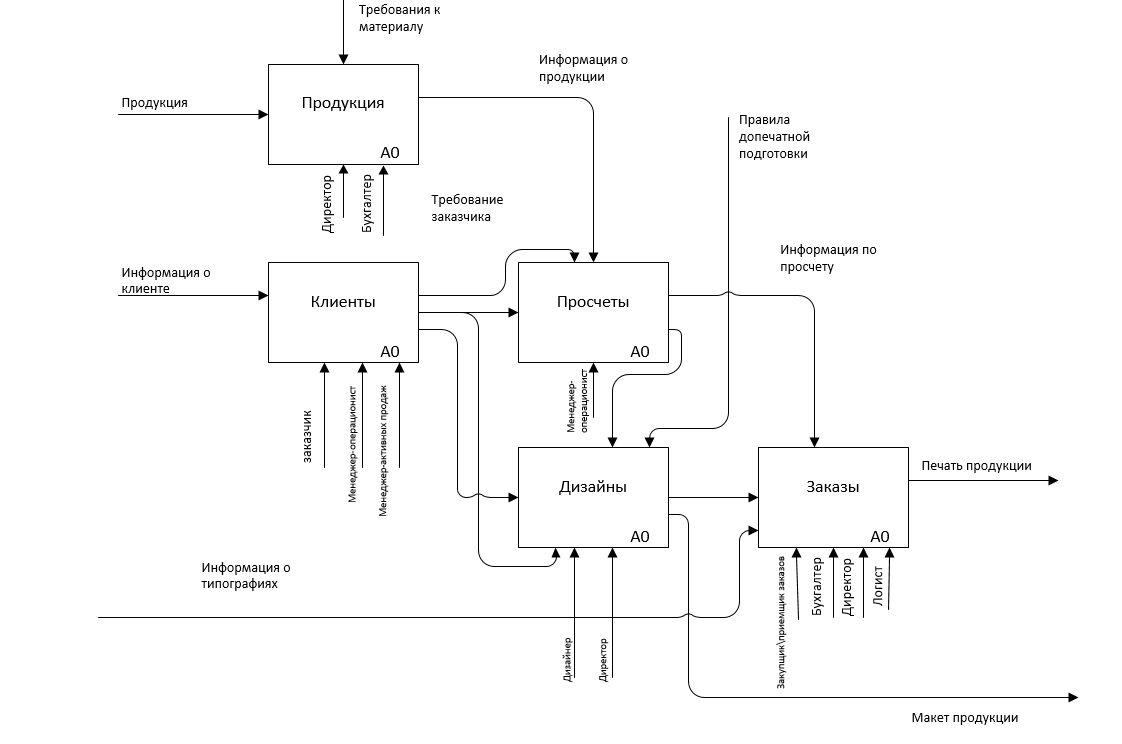


Рисунок 2 – Описание функциональной модели бизнес процесса второго уровня

На схеме наглядно видно, на каком этапе какие управляющие элементы и какие механизмы задействованы.

Так клиенты, которые приходят в типографию, оставляют информацию о себе, которая в дальнейшем будет использована для в различных отделах типографии.

Сперва просчёт заказа клиенту, затем изготовление дизайна, затем отправка готового дизайна в печать. Каждый просчет в специальной таблице связан с заданиями на дизайн, и с заказами на печать. Вся информация по просчётам, дизайнам, заказам на печать заносится в сводную таблицу. У каждого типа сотрудника есть права на просмотр лишь некоторых колонок. Просчёт может быть выполнен двумя способами: просто заглянув в таблицу с ценами или сделав просчёт в специальном калькуляторе.

Менеджер активных продаж смотрит в таблицу и говорит клиенту цену за изделие. А менеджер-операционист, каждый просчёт фиксирует в таблице просчетов.

**3 Проектирование и разработка клиент-серверного взаимодействия**

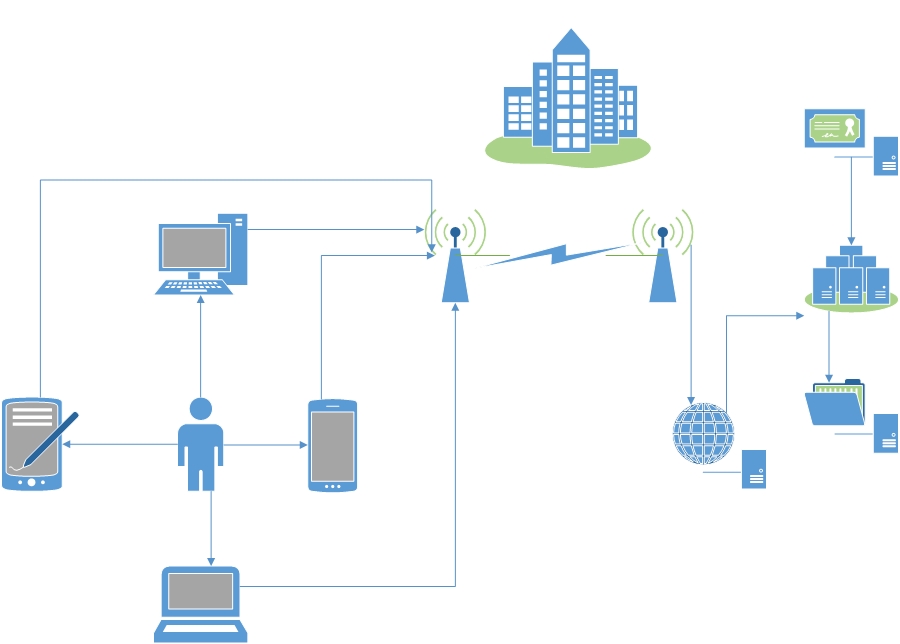


Рисунок 12 – схема сетевого взаимодействия

На рисунке 12 иллюстрирован процесс сетевого взаимодействия в приложении.

Сначала пользователь подключается к сети Интернет через одно из своих устройств. Далее происходит обмен данными с сервером по протоколу TCP/IP.

Пользователи различных групп могут добавлять/изменять/удалять определенные данные с учетом уровня доступа и сохранять свои правки. Например, потенциальный клиент сервиса Атопарковки пожелал забронировать себе заранее место под свою машину, тогда он ищет подходящую для него автопарковку и оставляет заявку в случае если есть свободные места, или ищет парковку со свободными местами.

Алгоритм процесса сетевого взаимодействия:

1. Пользователь (потенциальный клиент сервиса) подключается к интернету через одно из своих устройств.

2. По протоколу TCP\IP происходит обмен данными с сервером.

3. Просмотр автопарковок выбранной местности

4. Заполнение информации о себе и об автомобиле

5. Размещение заявки на парковочное место (бронирование)

6. Обратная связь ответственного за выбранную автопарковку с клиентом по средствам почты, sms-сообщения, звонка

Протокол обмена текстовыми сообщениями SMTP

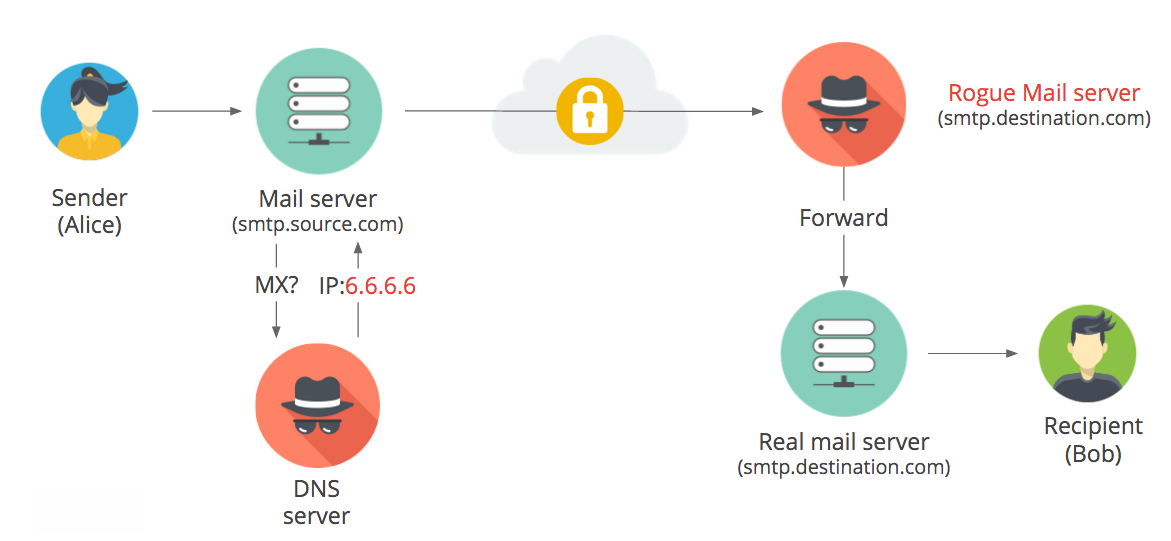


Рисунок 13 – схема обмена сообщениями по протоколу SMTP

Laravel предоставляет простой интерфейс к популярной библиотеке [SwiftMailer](http://swiftmailer.org/). Главный файл настроек - app/config/mail.php - содержит всевозможные параметры, позволяющие менять SMTP-сервер, порт, логин, пароль, а также устанавливать глобальный адрес from для исходящих сообщений. Можно использовать любой SMTP-сервер (на рисунке 13 представлена схема обмена данными по данному протоколу), либо стандартную функцию PHP mail - для этого устанавливаем параметр driver в значение mail. Кроме того, доступен драйвер sendmail.

Если в файле app/config/mail.php установить опцию driver в 'log', то все отправляемые письма будут записываться в лог-файл фреймворка и не будут рассылаться. Этот вариант используется для отладки.

Метод Mail::send используется для отправки сообщения:

Mail::send('emails.welcome', array('key' => 'value'), function($message)

{

$message->to('foo@example.com', 'Джон Смит')->subject('Привет!');

});

Первый параметр - имя шаблона, который должен использоваться для текста сообщения. Второй - ассоциативный массив переменных, передаваемых в шаблон. Третий - функция-замыкание, позволяющая внести дополнительные настройки в сообщение.

При разработке приложения иногда необходимо отключить доставку отправляемых сообщений. Для этого можно либо вызывать метод Mail::pretend, либо установить параметр pretend в значение true в файле настроек config/mail.php. Когда это сделано, сообщения будут записываться в файл журнала приложения, вместо того, чтобы быть отправленными получателю. Если необходимо отладить вид отсылаемых писем, то можно воспользоваться сервисом MailTrap.

**4 Проектирование и разработка базы данных**

4.1 Выбор и обоснование СУБД

Базы данных - это универсальное специально разработанное приложение, которое позволяет хранить данные различных типов. В каждой БД есть определенная модель (реляционная, документно-ориентированная), которая обеспечивает простой и удобный доступ к данным. Системы управления базами данных (СУБД) - специальные приложения (или библиотеки) управляющее базами данных различной сложности и позволяющее удобно выполнять действия над данными в БД.

Реляционная система управления базами данных (РСУБД)

СУБД, которая будет использована в проекте, должна обеспечивать реляционную модель работы с данными. Модель подразумевает определенный тип связи между сущностями из различных таблиц. Чтобы работать с данными, такой тип СУБД должен содержать определенную структуру (таблицы). Таблицы зачастую могут содержать столбцы с разными типами данными. Каждая строка состоит из множества элементов (столбцов) и имеет уникальный ключ, который лежит в той же таблице - все эти данные связаны между собой, согласно описанию реляционной модели.

Типы данных и отношений между ними

Отношения в базах данных можно рассматривать как математическое множество, содержащее в себе число атрибутов, которые суммарно представляют собой базу данных и информацию, хранящуюся в ней (фраза для тех, кто понимает, что такое математическое множество).

При разработке структуры таблицы каждое поле записи должно иметь свой собственный тип (например, набор символов, вещественное значение и т.д.). Все СУБД содержат различные типы данных в своём наборе для использования, которые могут быть не всегда взаимозаменяемы. При работе с СУБД всегда приходится сталкиваться с подобными ограничениями.

Рассмотрим три популярные, часто используемые и бесплатные СУБД:

SQLite – мощная встраиваемая система управления

MySQL - популярная и распространённая СУБД, используемая для веб

PostgreSQL – СУБД с большим числом возможностей и хорошей технической поддержкой

СУБД SQLite

Легковесная и удобно встраиваемая база данных. Эта система строится на файлах. Она предоставляет широкий набор инструментов для работы с ней, по сравнению с другими СУБД. При работе с этой СУБД обращения происходят напрямую к файлам (в этих файлах хранятся данные), вместо портов и сокетов в СУБД. Именно поэтому SQLite очень быстрая, а также мощная благодаря технологиям обслуживающих библиотек.

Таблица 3 – описание типов данных СУБД SQLite

|  |  |
| --- | --- |
| Тип данных | Описание |
| NULL | Данные типа NULL |
| INTEGER | 1. знаковое целочисленное значение, использует 1, 2, 3, 4, 6, или 8 байт в зависимости от порядка числа |

Окончание таблицы 3

|  |  |
| --- | --- |
| REAL | 1. число с плавающей точкой, занимает 8 байт для хранения числа в формате IEEE |
| REAL - TEXT | 1. текстовая строка, при хранении используются кодировки UTF-8, UTF-16BE или UTF-16LE |
| BLOB | тип данных BLOB, массив двоичных данных (предназначенный, в первую очередь, для хранения изображений, аудио и видео) |

Преимущества SQLite

1. Файловая структура – база данных, состоящая из одного файла, что облегчает её перенос на другие машины
2. Используемые стандарты – не смотря на то что данная СУБД является файловой она всё равно поддерживает стандарты SQL. Некоторые функции опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), но основные все-таки поддерживаются
3. Отличная при разработке и тестировании - в процессе разработки приложений часто появляется необходимость расширения возможностей базы данных. SQLite обеспечивает легкую масштабируемость за счёт своего малой файловой структуры.

Недостатки SQLite

1. Нет возможностей для работы с пользователями - более массивные СУБД поддерживают функции управления правами доступа пользователей. Для данной СУБД это не так критично, так как она, зачастую, применяется в приложениях небольшого размера.
2. Нет возможности увеличить скорость работы СУБД - опять, исходя из проектирования, довольно сложно выжать что-то более производительное из этой СУБД.

СУБД MySQL

MySQL - это самая популярная полнофункциональная серверная СУБД. MySQL содержит огромный набор реализованных функций. Она свободно распространяемая и успешно работает с различными сайтами и веб приложениями. Данная СУБД очень проста для обучения, огромное количество методических обучений по MySQL можно отыскать на просторах сети интернет (справочники, методические пособия, книги, видео уроки и т.д.).

Несмотря на то, что в ней не реализован весь SQL функционал, MySQL предлагает довольно много инструментов для разработки приложений. Так как это серверная СУБД, приложения для доступа к данным, в отличии от SQLite работают со службами MySQL.

Таблица 4 – описание типов данных СУБД MySQL

|  |  |
| --- | --- |
| Тип данных | Описание |
| TINYINT | 0 до 255 (1 байт) |
| SMALLINT | -2^15 (-32,768) до 2^15-1 (32,767) (2а байта) |
| MEDIUMINT | -8388608 до 8388607 (3 байта) |
| INT или INTEGER | -2147483648 до 2147483647 (4 байта) |
| BIGINT | -9223372036854775808 до 9223372036854775807 |
| FLOAT | Небольшие значения с плавающей точкой |
| DOUBLE, BOUBLE PRECISION, REAL | * Значения с плавающей точкой двойной точности |
| DECIMAL, NUMERIC | * распакованное значение с плавающей точкой, всегда знаковое. |

Окончание таблицы 4

|  |  |
| --- | --- |
| DATE | дата |
| DATETIME | дата и время в одном значении |
| TIMESTAMP | временная отметка timestamp |
| TIME | время |
| YEAR | год, 2 или 4 числа (4 – по умолчанию) |
| CHAR | строковое значение фиксированной длины, справа всегда добавляются пробелы до указанной длины при сортировке |
| VARCHAR | строковое значение переменной длины |
| TINYBLOB, TINYTEXT | значение типа BLOB или TEXT, 255 (2^8 - 1) символов - максимальная длина |
| BLOB, TEXT | * значение типа BLOB или TEXT, 65535 (2^16 - 1) символов - максимальная длина |
| MEDIUMBLOB, MEDIUMTEXT | * значение типа BLOB или TEXT, 16777215 (2^24 - 1) символов - максимальная длина |
| LONGBLOB, LONGTEXT | * значение типа BLOB или TEXT, 4294967296 (2^32 - 1) символов - максимальная длина |
| ENUM | Перечисление |
| SET | множество |

Преимущества СУБД MySQL

1. Удобство и простота в работе – установка MySQL занимает мало времени. Вспомогательные приложения, например, веб-GUI, позволяет облегчить работу с базой данных
2. Обширный функционал - MySQL поддерживает большинство функционала SQL.
3. Безопасность - большое количество функций обеспечивающих безопасность, которые поддерживается по умолчанию
4. Масштабируемость - MySQL легко работает с большими объемами данных и легко масштабируется
5. Скорость - упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

Недостатки MySQL

1. Известные ограничения - изначально в MySQL заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях.
2. Проблемы с надежностью - из-за некоторых способов обработки данных MySQL (связи, транзакции, аудиты) иногда уступает другим СУБД по надежности.
3. Медленная разработка - Хотя MySQL технически открытое ПО, существуют жалобы на процесс разработки.

СУБД PostgreSQL

PostgreSQL является мощной СУБД с открытым исходным кодом. Это объектно-реляционная система управления базами данных. Она имеет более чем 15 лет активного развития и проверенной архитектуры, которая заслужила ему высокую репутацию надежности, целостности данных и корректности работы. Она работает на всех основных операционных систем, включая Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) и Windows. Она полностью соответствует ACID, имеет полную поддержку внешних ключей, объединений, представлений, триггеров и хранимых процедур (на нескольких языках). Она включает в себя большинство SQL: 2008 типы данных, в том числе INTEGER - числовой, BOOLEAN, CHAR, VARCHAR, DATA, INTERVAL, и TIMESTAMP. Она также поддерживает хранение больших двоичных объектов, включая рисунки, звуки или видео. Она имеет собственные интерфейсы программирования для C / C ++, Java, .NET, Perl, Python, Ruby, Tcl, ODBC, среди прочего, и исключительной документации.

База данных корпоративного класса, PostgreSQL имеет сложные функции, такие как Multi-Version управления параллелизмом (MVCC), момент времени восстановления, табличными, асинхронная репликация, вложенные транзакции (точки сохранения), Интернет / горячее резервное копирование, сложный планировщик / оптимизатор запросов, и обеспечение отказоустойчивости. Он поддерживает международные наборы символов, многобайтовые кодировки символов, Unicode, и др, а также поддержка индивидуальной чувствительности и форматирования. Она отличается высокой масштабируемостью как в чистом количестве данных, которыми она может управлять, так и количества одновременно работающих пользователей, которые она может вместить. Есть активные системы PostgreSQL в производственной среде, которые управляют более 4 терабайт данных. Некоторые общие ограничения PostgreSQL включены в приведенной ниже таблице.

Таблица 5 – описание типов данных СУБД PostgreSQL

|  |  |
| --- | --- |
| Тип данных | Описание |
| bigint - | знаковое 8-ми битное целочисленное значение |
| bigserial - | автоматически инкрементируемое 8-ми битное целочисленное значение |
| bit[(n)] | строка постоянной длины |
| bit varying [(n)] | строка переменной длины |
| boolean | булево значение (true/false) |

Продолжение таблицы 5

|  |  |
| --- | --- |
| box | прямоугольник на плоскости |
| bytea | бинарные данные (массив байтов) |
| character varying [(n)] | строковое значение переменной длины |
| character [(n)] | строковое значение постоянной длины |
| cidr | IPv4/IPv6 сетевой адрес |
| circle | круг на плоскости |
| date | календарная дата (год, месяц, день) |
| double precision | число с плавающей точкой двойной точности (8 байт) |
| inet | IPv4/IPv6 адрес хоста |
| integer | знаковое 4-ех байтовое целочисленное значение |
| interval [fields][(p)] | отрезок времени |
| line | бесконечная прямая на плоскости |
| lseg | отрезок на плоскости |
| macaddr | MAC адрес |
| mony | валютное значение |
| numeric [(p, s)] | точное численное значение с выбранной точностью |
| path | геометрическая кривая на плоскости |
| point | геометрическая точка на плоскости |
| polygony | многоугольник на плоскости |
| real | число с плавающей точкой одинарной точности (4 байта) |
| smallint | знаковое целочисленное значение (4 байта) |
| serial | автоматические инкрементируемое целочисленное значение (4 байта) |
| text | строковое значение переменной длины |

Окончание таблицы 5

|  |  |
| --- | --- |
| time [(p)] [without time zone] | время суток (без часового пояса) |
| time [(p)] with time zone | время суток (включая часовой пояс) |
| timestamp [(p)] [without time zone] | дата и время (без часового пояса) |
| timestamp [(p)] with time zone | дата и время (с часовым поясом) |
| tsquery | текстовый поисковый запрос |
| tsvector | документ текстового поиска |
| txid\_snapshot | пользовательский снимок транзакции с ID |
| uuid | универсальный уникальный идентификатор |
| xml | XML данные |

Достоинства PostgreSQL

1. Открытое ПО соответствующее стандарту SQL - PostgreSQL - бесплатное ПО с открытым исходным кодом. Эта СУБД является очень мощной системой.
2. Большое сообщество - существует довольно большое сообщество, в котором вы запросто найдёте ответы на свои вопросы
3. Большое количество дополнений - несмотря на огромное количество встроенных функций, существует очень много дополнений, позволяющих разрабатывать данные для этой СУБД и управлять ими.
4. Расширения - существует возможность расширения функционала за счет сохранения своих процедур.
5. Объектность - PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого

Недостатки PostgreSQL

1. Производительность - при простых операциях чтения PostgreSQL может значительно замедлить сервер и быть медленнее своих конкурентов, таких как MySQL
2. Популярность - по своей природе, популярностью эта СУБД похвастаться не может, хотя и присутствует довольно большое сообщество.
3. Хостинг - в силу выше перечисленных факторов иногда довольно сложно найти хостинг с поддержкой этой СУБД.

Большинство сайтов и онлайн приложений прекрасно работают с MySQL несмотря на некоторые ограничения этой СУБД. Будучи легкой в настройке и масштабируемой системой - MySQL проверена временем. Эта СУБД сочетает в себе продвинутый функционал и свободный доступ к исходному коду. Система безопасности MySQL включает в себе простые и в то же время достойные способы защиты доступа к данным.

Исходя из результатов сравнения было решено выбрать в качестве основной базы данных для разработки проекта СУБД MySQL.

4.2 Инфологическая модель базы данных

Инфологическая модель базы данных – наиболее естественные способы сбора и представления той информации, которую предполагается хранить в создаваемой базе данных.

Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

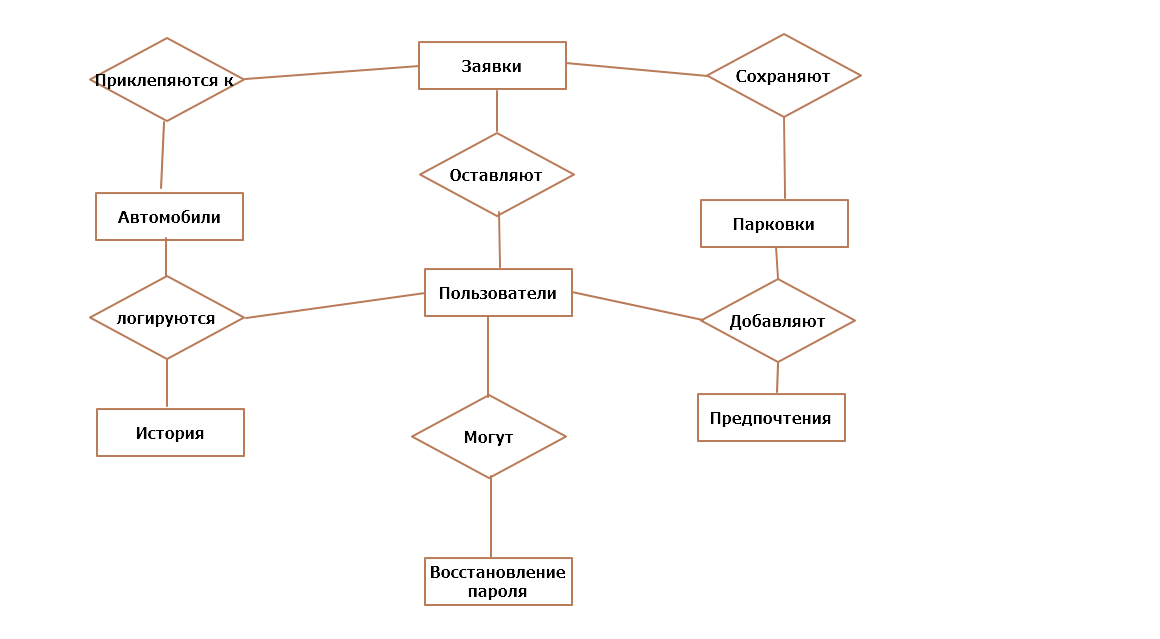


Рисунок 15 – инфологическая модель базы данных

На схеме выделено 7 сущностей: Пользователь – основная сущность, заявки, парковки, предпочтения, автомобили, история и восстановление пароля. Пять основных сущностей и две вспомогательные сущности. Пользовали могут взаимодействовать со всеми сущностями либо непосредственно, либо через другие сущности.

4.3 Концептуальная модель базы данных

Концептуальная модель - это отражение предметной области, для которой разрабатывается база данных.

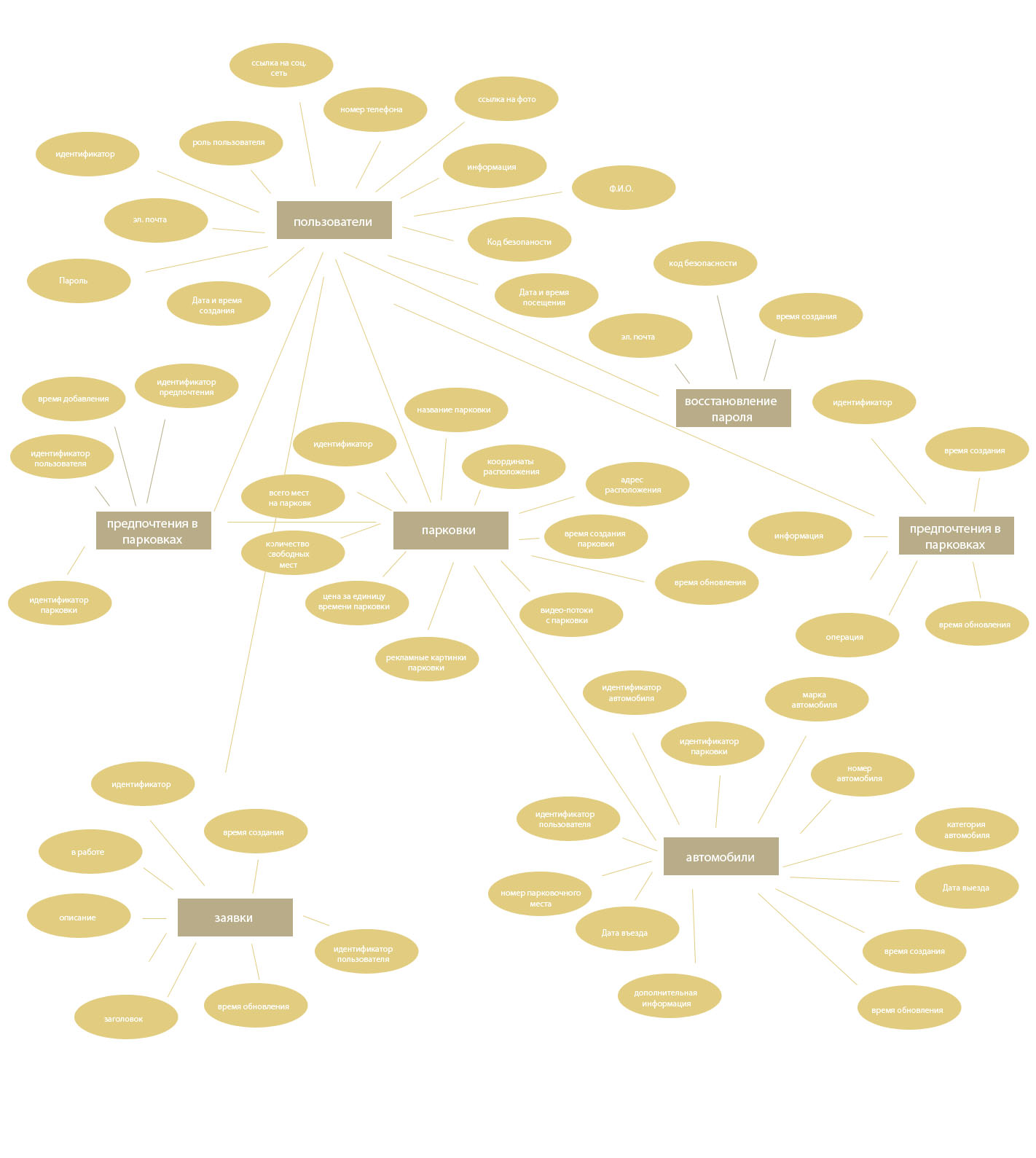


Рисунок 16 – концептуальная схема базы данных

На рисунке N представлено 7 сущностей и атрибуты этих сущностей, которые отражают будущий вид базы данных.

4.4 Физическая модель базы данных

Физическая модель данных описывает реализацию объектов логической модели на уровне объектов конкретной базы данных.

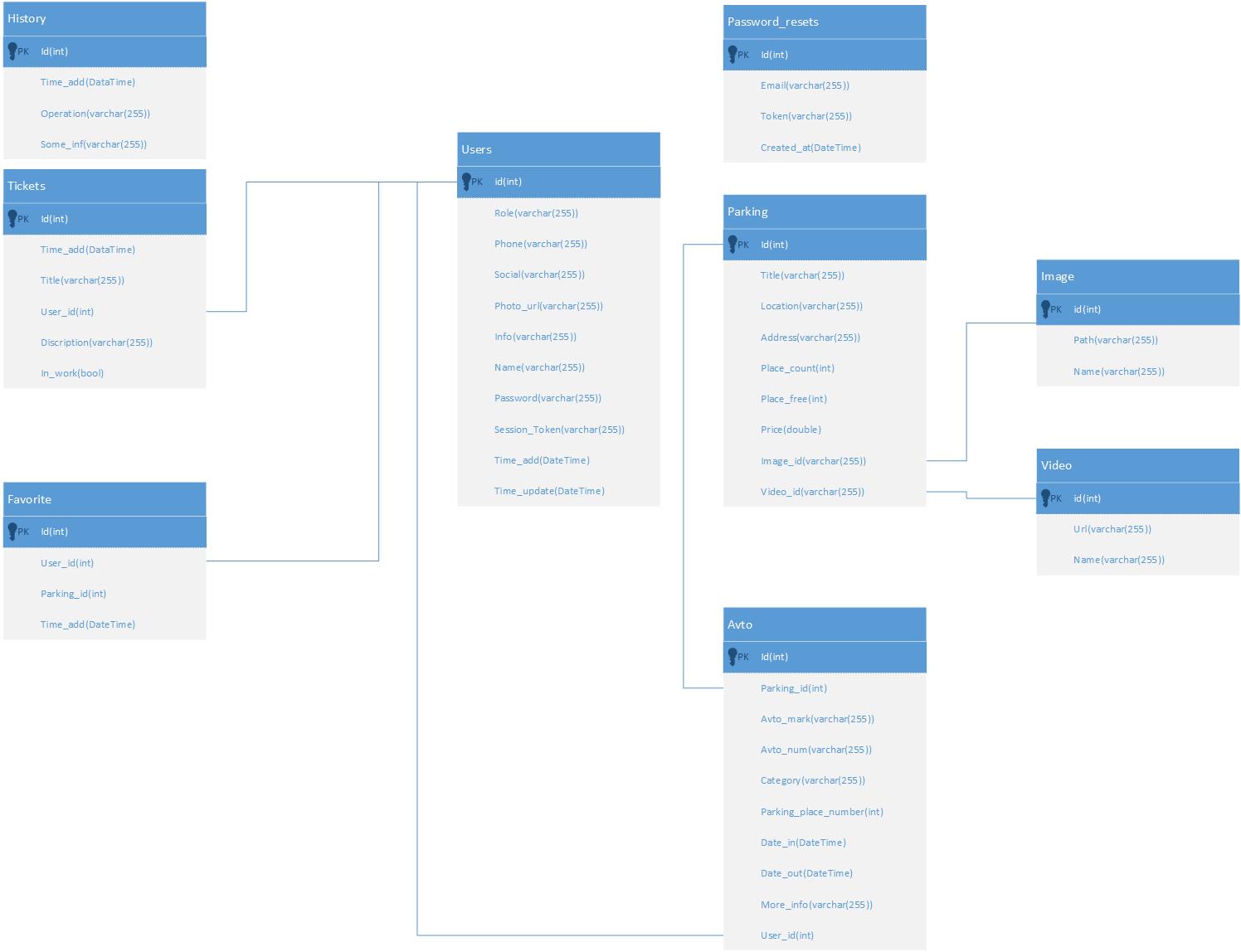


Рисунок 17 – физическая модель базы данных

В процессе разработки было добавлено две вспомогательные таблицы: image и video – которые хранят набор ссылок фотогалереи и видео галереи для каждой парковки. При этом число фото и видео материала для одной парковки не ограничено.

4.5 Реализация базы данных в ORM Eloquent

ORM Eloquent поставляется в комплекте с Laravel и обеспечивает красивую и простую реализацию ActiveRecord для работы с базой данных. Каждая таблица базы данных имеет соответствующую "Модель", которая используется для взаимодействия с этой таблицей. Модели позволяют запрашивать данные в таблицах, а также вставлять новые записи в таблицу.

Для начала работы с ORM Eloquent необходимо указать верные настройки в файле .env или в файле database.php в директории config проекта Laravel. Содержимое файла database приведено ниже на рисунке 18.

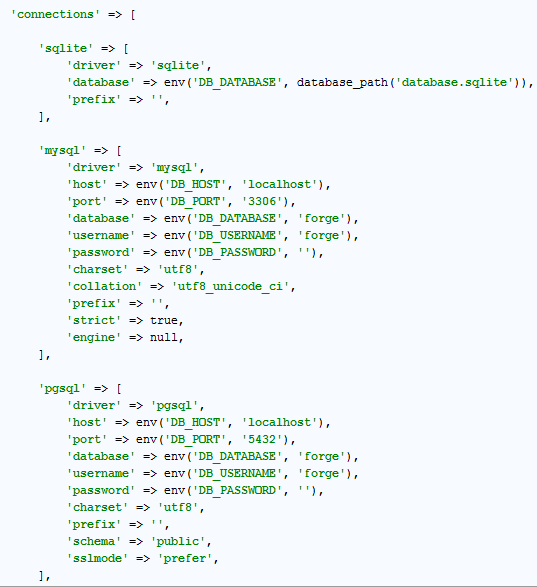


Рисунок 18 – содержимое файла настроек базы данных

Как видим ORM Eloquent поддерживает работу с различными базами данных. Разработчики изначально позаботились о совместимости с наиболее популярными базами данных. На рисунке 19 показана структура файла .env. В зависимости от типа соединения с базой данных, данные из этого файла автоматически в процессе работы подставляются в соответствующие поля в файле database.php, описанном ранее.

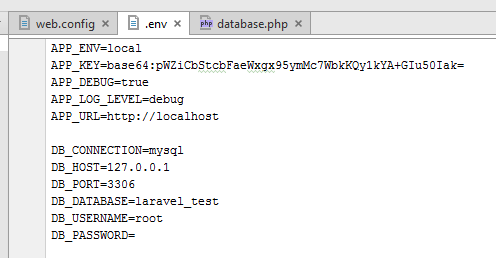


Рисунок 19 – структура файла .env

Создание моделей в Laravel

Сопоставление SQL и команд ORM Eloquent

В таблице N приведено сопоставление команд, используемых в ORM и их аналогов, написанных на языке SQL. Все, команды, которые приведены в таблице были использованы в разработке проекта.

Таблица 6 - Сопоставление SQL и команд ORM Eloquent

|  |  |  |
| --- | --- | --- |
| Описание | SQL | ORM Eloquent |
| Выбор всех значений из таблицы | Select \* from users | DB::table('users')->get(); |

Продолжение таблицы 6

|  |  |  |
| --- | --- | --- |
| Выборка по 1му и по двум столбцам из таблицы | Select name from users,  Select name, age from users | $query = DB::table('users')->select('name');  $users = $query->addSelect('age')->get(); |
| Получает список данных, значение которых в колонке votes лежит между 1 и 100 | Select \* from users where votes between 1 and 100 | DB::table('users') ->whereBetween('votes', [1, 100])->get() |
| Выбирает данные от 1 до 3 включительно | Select \* from users where id=1 or id=2 or id=3 | DB::table('users') ->whereIn('id', [1, 2, 3])->get(); |
| Выбирает все данные, кроме, тех, которые содержат указанный id | Select \* from users where id<>1 and id<>2 and id<>3 | DB::table('users') ->whereNotIn('id', [1, 2, 3])->get(); |
| Выбирает только те данные, где поле updated\_at пустое | Select \* from users where updated\_at=null | DB::table('users') ->whereNull('updated\_at')->get(); |
| Выбирает данные одновременно по нескольким указанным данным | Select \* from users where id=2 and email='john@doe.com' | DB::table('users') ->whereIdAndEmail(2, 'john@doe.com') ->first(); |

Окончание таблицы 6

|  |  |  |
| --- | --- | --- |
| Выдает список, отсортированный по имени, где параметр count > 100, сгрупированный по count | Select \* from users order by name desc where count > 100 group by count | DB::table('users') ->orderBy('name', 'desc') ->groupBy('count') ->having('count', '>', 100) ->get(); |
| Выборка данных в определенном количестве с выбранным шагом |  | DB::table('users')->skip(10)-> take(5)->get(); |
| Выборка данных из нескольких таблиц |  | DB::table('users') -> join('contacts', 'users.id', '=', 'contacts.user\_id')-> join('orders', 'users.id', '=', 'orders.user\_id')-> select('users.id', 'contacts.phone', 'orders.price')->get(); |
| Выборка данных из нескольких таблиц |  | DB::table('users')-> leftJoin('posts', 'users.id', '=', 'posts.user\_id')->get(); |
| Выборка данных из нескольких таблиц | select \* from users where name = 'John' or (votes > 100 and title <> 'Admin') | DB::table('users')-> where('name', '=', 'John') -orWhere(function($query) {$query->where('votes', '>', 100)->where('title', '<>', 'Admin'); })->get() |

Миграции

Миграции – это набор средств для быстрого создания и обновления структуры базы данных. Фасад Laravel Schema обеспечивает поддержку создания и изменения таблиц в независимости от используемой БД. Он предоставляет единый, выразительный и гибкий API для всех поддерживаемых в Laravel СУБД. Для создания новой миграции используется Artisan-команда make:migration. Миграция будет помещена в папку database/migrations и будет содержать метку времени, которая позволяет фреймворку определять порядок применения миграций. Для нашего проекта нам необходимо создать семь таблиц-миграций. Ниже приведен код, который выполняет данное действие.

1. php artisan make:migration create\_users\_table – таблица пользователей нашей системы
2. php artisan make:migration create\_password\_resets\_table – вспомогательная таблица для восстановления пароля
3. php artisan make:migration create\_favorite\_table – таблица предпочтений пользователя (список понравившихся парковок)
4. php artisan make:migration create\_parking\_table – таблица парковок
5. php artisan make:migration create\_avto\_table – таблица автомобилей на парковках
6. php artisan make:migration create\_history\_table – таблица записи въездов и выездов на парковку
7. php artisan make:migration create\_tickets\_table – таблица пользовательских заявок

Для запуска всех необходимых миграций приложения используем Artisan-команду migrate.

Таблица 7 – основные Artisan-команды для работы с миграциями

|  |  |
| --- | --- |
| Команда | Описание |
| migrate | Выполнение всех миграций проекта |
| migrate:rollback | Отмена изменений, сделанных последней миграцией |
| migrate:reset | Отмена всех изменений миграций |
| migrate:refresh | Откат всех изменений и повторное применение |
| php artisan migrate:refresh --seed | Откат всех изменений и повторное применение, и заполнение тестовыми данными |

Структура миграций

Класс миграций содержит два метода: up() и down(). Метод up() используется для добавления новых таблиц, столбцов или индексов в БД, а метод down() просто отменяет операции, выполненные методом up().

Для создания новой таблицы БД используем метод create() фасада Schema. Метод create() принимает два аргумента. Первый — имя таблицы, второй — замыкание, которое получает объект Blueprint, используемый для определения новой таблицы. При создании таблицы мы можем использовать любые методы для работы со столбцами построителя структур.

Таблица 8 – доступные типы столбцов при создании миграции

|  |  |
| --- | --- |
| Команда | Описание |
| $table->bigIncrements('id'); | Инкрементный ID (первичный ключ), использующий эквивалент "UNSIGNED BIG INTEGER" |

Продолжение таблицы 8

|  |  |
| --- | --- |
| $table->bigInteger('votes'); | Эквивалент BIGINT для базы данных |
| $table->binary('data'); | Эквивалент BLOB для базы данных |
| $table->boolean('confirmed'); | Эквивалент BOOLEAN для базы данных |
| $table->char('name', 4); | Эквивалент CHAR для базы данных |
| $table->date('created\_at'); | Эквивалент DATE для базы данных |
| $table->dateTime('created\_at'); | Эквивалент DATETIME для базы данных |
| $table->decimal('amount', 5, 2); | Эквивалент DECIMAL с точностью и масштабом |
| $table->double('column', 15, 8); | Эквивалент DOUBLE с точностью, всего 15 цифр, после запятой 8 цифр |
| $table->enum('choices', ['foo', 'bar']); | Эквивалент ENUM для базы данных |
| $table->float('amount'); | Эквивалент FLOAT для базы данных |
| $table->increments('id'); | Инкрементный ID (первичный ключ), использующий эквивалент "UNSIGNED INTEGER" |
| $table->integer('votes'); | Эквивалент INTEGER для базы данных |
| $table->json('options'); | Эквивалент JSON для базы данных |
| $table->jsonb('options'); | Эквивалент JSONB для базы данных |
| $table->longText('description'); | Эквивалент LONGTEXT для базы данных |
| $table->mediumInteger('numbers'); | Эквивалент MEDIUMINT для базы данных |

Окончание таблицы 8

|  |  |
| --- | --- |
| $table->mediumText('description'); | Эквивалент MEDIUMTEXT для базы данных |
| $table->morphs('taggable'); | Добавление столбца taggable\_id INTEGER и taggable\_type STRING |
| $table->nullableTimestamps(); | Аналогично timestamps(), но разрешено значение NULL |
| $table->rememberToken(); | Добавление столбца remember\_tokenVARCHAR(100) NULL |
| $table->smallInteger('votes'); | Эквивалент SMALLINT для базы данных |
| $table->softDeletes(); | Добавление столбца deleted\_at для мягкого удаления |
| $table->string('email'); | Эквивалент VARCHAR |
| $table->string('name', 100); | Эквивалент VARCHAR с длинной |
| $table->text('description'); | Эквивалент TEXT для базы данных |
| $table->time('sunrise'); | Эквивалент TIME для базы данных |
| $table->tinyInteger('numbers'); | Эквивалент TINYINT для базы данных |
| $table->timestamp('added\_on'); | Эквивалент TIMESTAMP для базы данных |
| $table->timestamps(); | Добавление столбцов created\_at и updated\_at |
| $table->uuid('id'); | Эквивалент UUID для базы данных |

Помимо перечисленных типов столбцов существует несколько других «модификаторов» столбцов, которые можно использовать при добавлении столбцов.

Таблица 9 – типы модификаторов столбцов

|  |  |
| --- | --- |
| Модификатор | Описание |
| ->first() | Помещает столбец "первым" в таблице (только MySQL) |
| ->after('column') | Помещает столбец "после" указанного столбца (только MySQL) |
| ->nullable() | Разрешает вставлять значения NULL в столбец |
| ->default($value) | Указывает значение "по умолчанию" для столбца |
| ->unsigned() | Делает столбцы integer беззнаковыми UNSIGNED |

Таблица 9 - Доступные типы индексов

|  |  |
| --- | --- |
| Команда | Описание |
| $table->primary('id'); | Добавление первичного ключа |
| $table->primary(['first', 'last']); | Добавление составных ключей |
| $table->unique('email'); | Добавление уникального индекса |
| $table->index('state'); | Добавление базового индекса |

Laravel также поддерживает создание ограничений для внешнего ключа, которые используются для обеспечения ссылочной целостности на уровне базы данных. Также можно указать требуемое действие для свойств "on delete" и "on update" ограничений. Для удаления внешнего ключа используется метод dropForeign(). Ограничения внешнего ключа используют те же принципы именования, что и индексы.

Разработанные миграции:

1. Миграция таблицы users

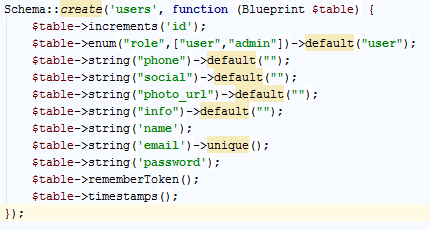


Рисунок 20 – миграция таблицы users

1. Миграция таблицы password\_resets

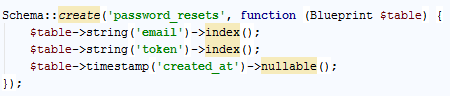


Рисунок 21 – миграция таблицы password\_resets

1. Миграция таблицы parking

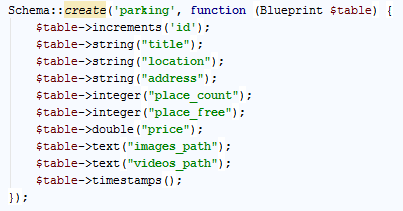


Рисунок 22 - миграция таблицы parking

1. Миграция таблицы avto

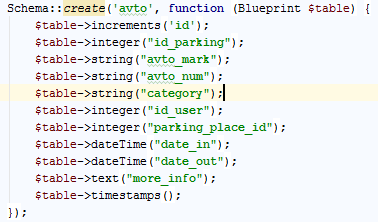


Рисунок 23 - Миграция таблицы avto

1. Миграция таблицы history

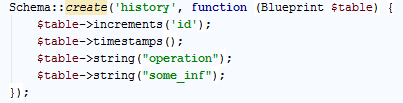


Рисунок 24 - Миграция таблицы history

1. Миграция таблицы tickets

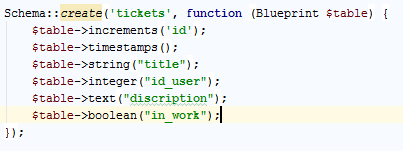


Рисунок 25 - миграция таблицы tickets

1. Миграция таблицы favorite

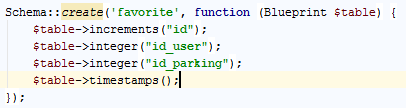


Рисунок 26 – миграция таблицы favorite

**5 Разработка клиент-серверного приложения «Автостоянка»**

* 1. Разработка макета веб-интерфейса

Определение целей проекта - это фундаментальная часть процесса создания дизайна. Как и в любой поездке, мы должны знать пункт назначения. С другой стороны, мы могли бы ездить в разных направлениях, но так мы могли бы легко потеряться. Устанавливая понятные цели для проекта, мы знаем, куда двигаться и когда проект будет готов. Гораздо проще будет оценить работу и понять сделали ли мы работу хорошо, задав один простой вопрос: «Выполнили мы все поставленные цели?»



Рисунок 27 – палитра цветов будущего сайта

На рисунке 27 находятся цвета, которые будут хорошо сочетаться друг с другом в нашем проекте. Все цвета взяты из окружающей природы и органично будут смотреться на сайте, не вызывая раздражения у пользователей, а наоборот притягивая внимание потенциального пользователя к определенным частям сайта.

Цвета: бледно-желтый, бордовый, коричневый, красный, оранжевый, оттенки желтого, оттенки осени, палитра для осени, подбор цвета, подбор цвета для осени, шафрановый, яркий красный.

Выбранные цвета были использованы для разработки макета сайта в программе Photoshop (представлено на рисунке 28). Будущий макет сайта разрабатывается на основе набросков, сделанных в техническом задании проекта.

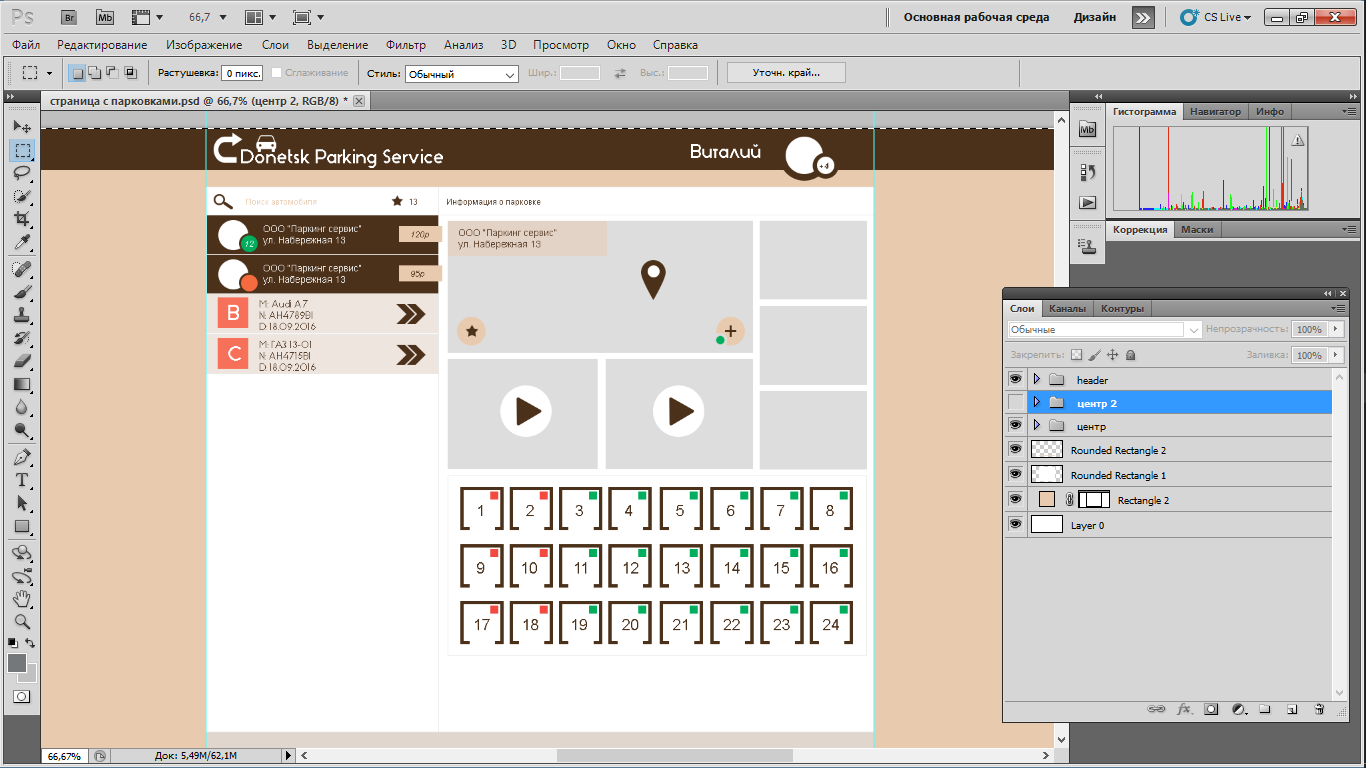


Рисунок 28 – разработанный макет сайта

На основе разработанного макета можно приступить к верстке сайта и привязке функционала к элементам веб-интерфейса.

* 1. Верстка макета веб-интерфейса

Для верстки сайта создадим будущую файловую структуру сайта. Каталоги: img – каталог для изображений, fonts – каталог для шрифтов, css – каталог для стилей проекта, js – каталог для скриптов.

В корневом каталоге создадим файл index.html – файл, в котором будет описан каркас сайта. Для верстки сайта используем блочную верстку с html5 тегами. Подобный поход позволяет сделать точный дизайн по макету, и, в дальнейшем, адаптировать сайт под различные типы устройств (изначально сайт разрабатывается не адаптивным). Для облегчения процесса стилизации сайта стиль будет написан на языке scss – данный подход позволяет уменьшить число строк кода, сделать более точные связи вложенных элементов. Структура scss файла повторяет иерархию html каркаса, что упрощает процесс верстки.

Верстка административной части будет осуществлена при помощи Фреймворка Bootstrap 3 – административная часть не требует привлекательного оригинального дизайна, в связи с этим верстка будет осуществляться стандартными элементами этого Фреймворка.

Для реализации скриптовой части веб-интерфейса подключаем к проекту AngularJS – благодаря этому Фреймворку мы сможем реализовать все CRUD функции в проекте. Так же его использование позволит реализовать асинхронный обмен данными между клиентом и сервером.

В дальнейшем html файлы с готовым веб-интерфейсом будут подключены к серверной части и размещены в файлах \*.blade.php – файлы шаблона веб-страницы, используемы в веб-приложениях на основе Фреймворка Laravel.

* 1. Разработка серверной части веб-приложения

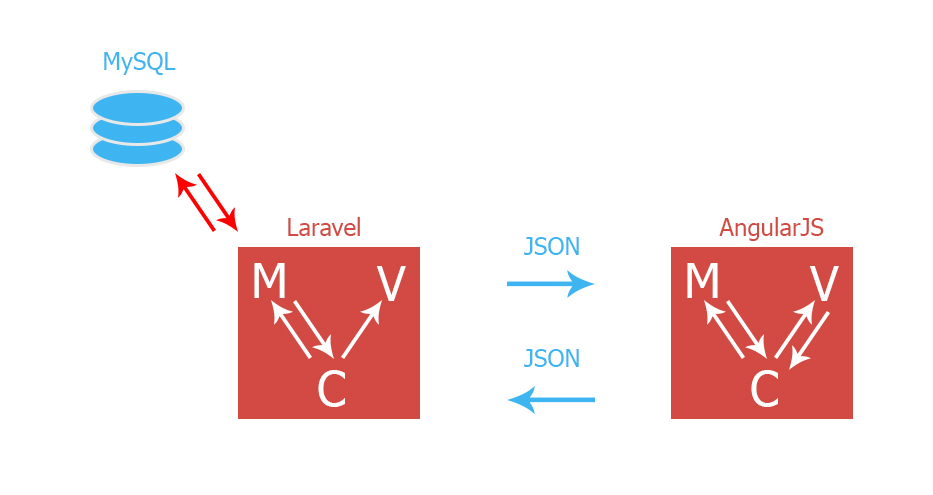


Рисунок 29 – процесс работы клиент-серверного приложения «Автостоянка»

Основными компонентами системы являются база данных, сервер и клиентская часть. Клиентская и серверная часть берут за основу шаблон MVC, реализованный при помощи Фреймверков. Для клиентской части – AngularJS, для серверной – Laravel5.

Шаблон MVC описывает простой способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса. В результате, приложение легче масштабируется, тестируется, сопровождается и конечно же реализуется.

Рассмотрим концептуальную схему шаблона MVC на рисунке30:



Рисунок 30 – концептуальная схема MVC

В архитектуре MVC модель предоставляет данные и правила бизнес-логики, представление отвечает за пользовательский интерфейс, а контроллер обеспечивает взаимодействие между моделью и представлением.

Общая последовательность работы MVC-приложения может быть описана следующим образом:

* 1. При заходе пользователя на веб-ресурс, скрипт инициализации создает экземпляр приложения и запускает его на выполнение.
  2. При этом отображается вид, скажем главной страницы сайта.
  3. Приложение получает запрос от пользователя и определяет запрошенные контроллер и действие. В случае главной страницы, выполняется действие по умолчанию (index).
  4. Приложение создает экземпляр контроллера и запускает метод действия,
  5. в котором, к примеру, содержаться вызовы модели, считывающие информацию из базы данных.
  6. После этого, действие формирует представление с данными, полученными из модели и выводит результат пользователю.

Модель — содержит бизнес-логику приложения и включает методы выборки (это могут быть методы ORM), обработки (например, правила валидации) и предоставления конкретных данных, что зачастую делает ее очень толстой, что вполне нормально.

Модель не должна напрямую взаимодействовать с пользователем. Все переменные, относящиеся к запросу пользователя должны обрабатываться в контроллере.

Модель не должна генерировать HTML или другой код отображения, который может изменяться в зависимости от нужд пользователя. Такой код должен обрабатываться в видах.

Одна и та же модель, например: модель аутентификации пользователей может использоваться как в пользовательской, так и в административной части приложения. В таком случае можно вынести общий код в отдельный класс и наследоваться от него, определяя в наследниках специфичные для под приложений методы.

Вид — используется для задания внешнего отображения данных, полученных из контроллера и модели.

Виды содержат HTML-разметку и небольшие вставки PHP-кода для обхода, форматирования и отображения данных.

Не должны напрямую обращаться к базе данных. Этим должны заниматься модели.

Не должны работать с данными, полученными из запроса пользователя. Эту задачу должен выполнять контроллер.

Может напрямую обращаться к свойствам и методам контроллера или моделей, для получения готовых к выводу данных.

Виды обычно разделяют на общий шаблон, содержащий разметку, общую для всех страниц (например, шапку и подвал) и части шаблона, которые используют для отображения данных выводимых из модели или отображения форм ввода данных.

Контроллер — связующее звено, соединяющее модели, виды и другие компоненты в рабочее приложение. Контроллер отвечает за обработку запросов пользователя. Контроллер не должен содержать SQL-запросов. Их лучше держать в моделях. Контроллер не должен содержать HTML и другой разметки. Её стоит выносить в виды.

В хорошо спроектированном MVC-приложении контроллеры обычно очень тонкие и содержат только несколько десятков строк кода. Логика контроллера довольно типична и большая ее часть выносится в базовые классы.

Модели, наоборот, очень толстые и содержат большую часть кода, связанную с обработкой данных, т.к. структура данных и бизнес-логика, содержащаяся в них, обычно довольно специфична для конкретного приложения.

Схема работы приложения «Автопарковка»:

* 1. Все данные хранятся в модели
  2. По запросу URL к контролеру данные из модели поступают в вид виде JSON массива.
  3. JSON массив передается на сторону клиента, где становится моделью для Фреймворка AngularJS
  4. Контроллер Фреймворка AngularJS выставляет данные в специально отведенные под них места в html коде.
  5. Данные отображаются пользователю.

Схема передачи данных от клиента к серверу диаметрально противоположна.

**6 Руководство по использованию программного продукта**

6.1 Файловая структура веб-приложения

Для разработки веб приложения используется Laravel 5, построенный по архитектуре MVC.

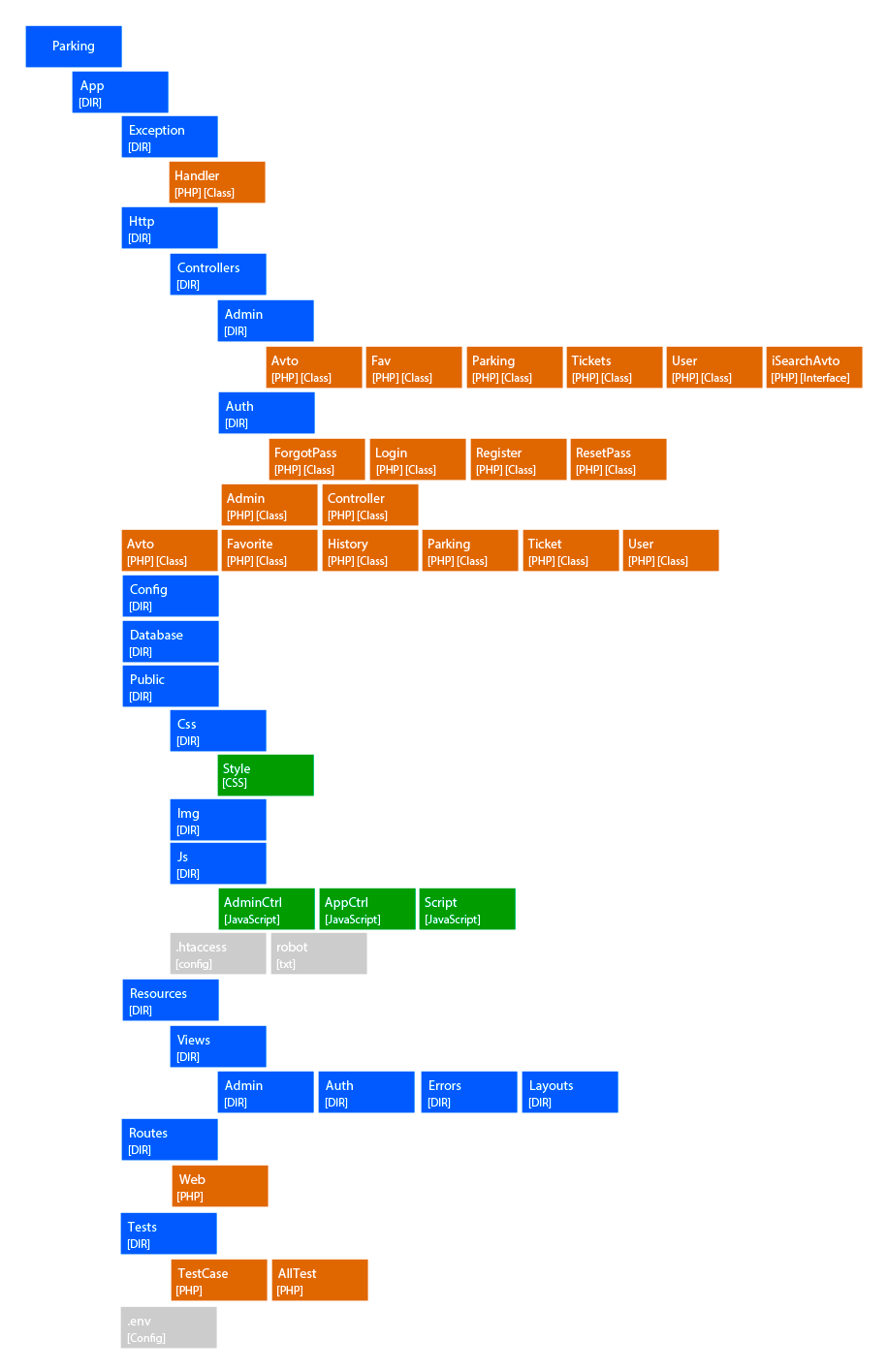


Рисунок 31 – файловая структура проекта

Так как framework несет в себе ряд служебных директорий и файлов, помимо пользовательских, то на схему вынесены только те директории и файлы, которые непосредственно были задействованы в процессе разработки приложения. Паттерн MVC – модель-представление-контроллер – подразумевает наличие трёх основных папок: папка controllers, в которой располагаются основные контроллеры (обработчики запросов) системы, папка database (и папка App с моделями), в которой расположены миграции, сиды и фабрики для создания и заполнения базы данных, а так же папочка views, содержащая шаблоны php страниц, которые в дальнейшем будут использованы для отображения информации пользователю системы. Вся информация, касающаяся оформления системы, изображения, файлы скриптов и шрифтов хранятся в папке public. Именно эта директория доступна пользователям из веб. Ограничить доступ к определенным файлам и папкам можно через файл .htaccess.

В качестве моделей в системе выступают файлы Avto, Favorite, Parking, Tickets, History, Users – данные файлы используются для обращения к базе данных не через SQL запрос, а непосредственно через методы моделей.

Для отображения информации в проекте используются файлы-шаблоны с расширением \*.blade.php. Шаблоны расположены в папке resources – views. Для удобства обращения и логического структурирования проекта внутри папки view есть еще деление на логические разделы, такие как отображения для администратора, отображения для главной страницы, отображения для ошибок, отображения для авторизации и регистрации пользователей, а так же общий универсальный шаблон для страниц пользователя и для страниц администратора. Все файлы ресурсов для этих страниц хранятся в папке public.

Для взаимосвязи отображения и модели используется контроллер, а для связи контроллера и «внешнего мира» используются маршруты. На рисунке 32 приведена структура файла маршрутов проекта автопарковки.

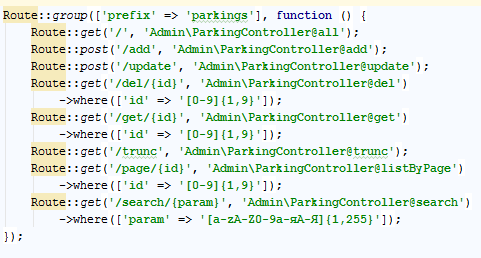


Рисунок 32 – структура файла маршрутов. Маршруты парковки.

6.2 Руководство администратора

Для того чтоб начать работать с системой как администратор необходимо чтоб ваша учетная запись имела роль администратора. По умолчанию в системе есть учетная запись с логином admin и паролем admin.

После того как вы вошли в учетную запись администратора вас перенаправит на адрес /admin. На странице администратора находится компонент «вкладки», название которых соответствует таблицам в базе данных. Функции администратора, которые можно выполнить в данном разделе: просмотр всех данных в каждой таблице, причем изначально загружается только определенная порция данных и уже при прокрутке администратор может подгрузить последующие порции данных. Так же доступен полнотекстный поиск по базе данных по всем полям в таблицах. Администратору доступны функции добавления, редактирования и удаления данных во всех таблицах.

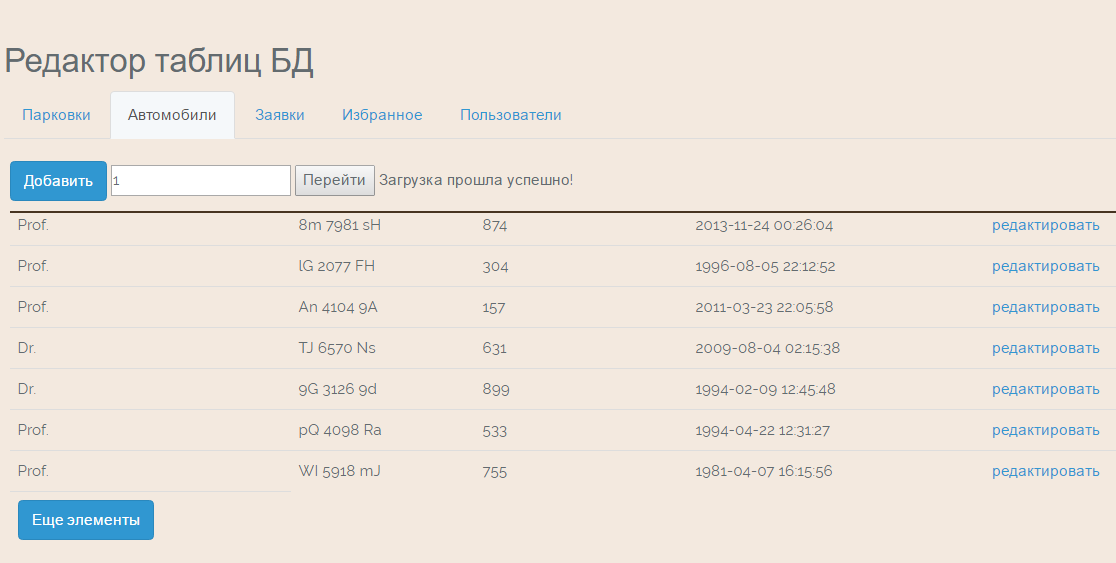


Рисунок 33 – общий вид административной панели приложения

Для редактирования или добавления элемента необходимо нажать на кнопку «редактировать» или «добавить», после чего откроется модальное окно, содержащее пустые поля при добавлении, либо заполненные поля, содержащие выбранный для редактирования элемент, которые можно в последствие модернизировать и записать в базу данных. Все функции административной панели реализованы через технологию AJAX, т.е. после выполнения операции перегружать окно администратора для просмотра изменений не требуется.

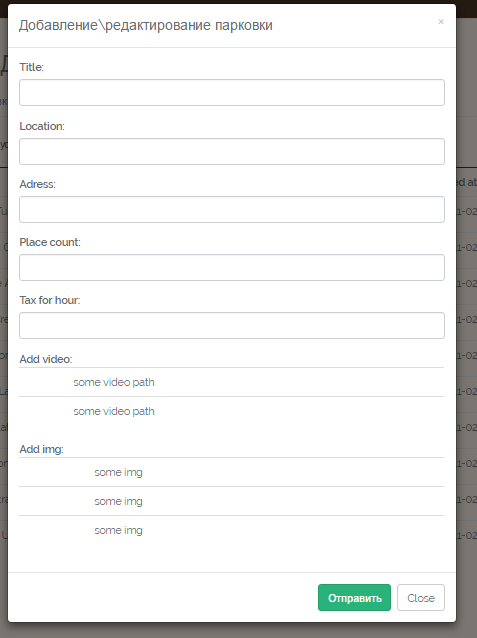


Рисунок 34 – окно добавления\редактирования парковки

6.3 Руководство пользователя

Если вы хотите стать пользователем данного клиент-серверного приложения, то необходимо чтоб у вас на устройстве, с которого будет осуществляться доступ к странице приложения был установлен браузер, поддерживающий JavaScript, html5 и css3. Зачастую к таким браузерам относятся все современные браузеры. Все окна приложения являются адаптивными и подстраиваются под размер экрана любого устройства. Карта сайта представлена на рисунке 35.

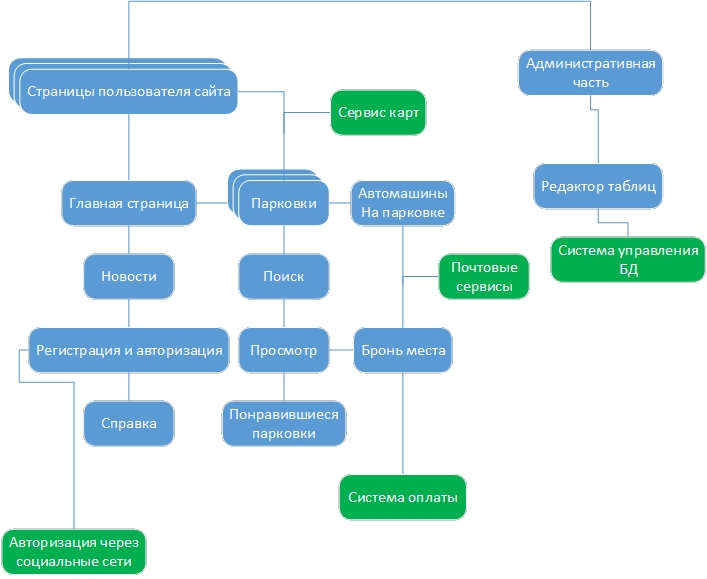


Рисунок 36 – Карта сайта приложения «Автопарковка»

Для начала работы с приложением необходимо перейти на главную страницу приложения. Данной приложение разработана на технологии AJAX, так что вся информация открывается в рамках одной страницы. Можно выделить основные части данного приложения:

1. Первый вход на страницу, когда пользователь еще не авторизовался и не просматривал никаких парковок

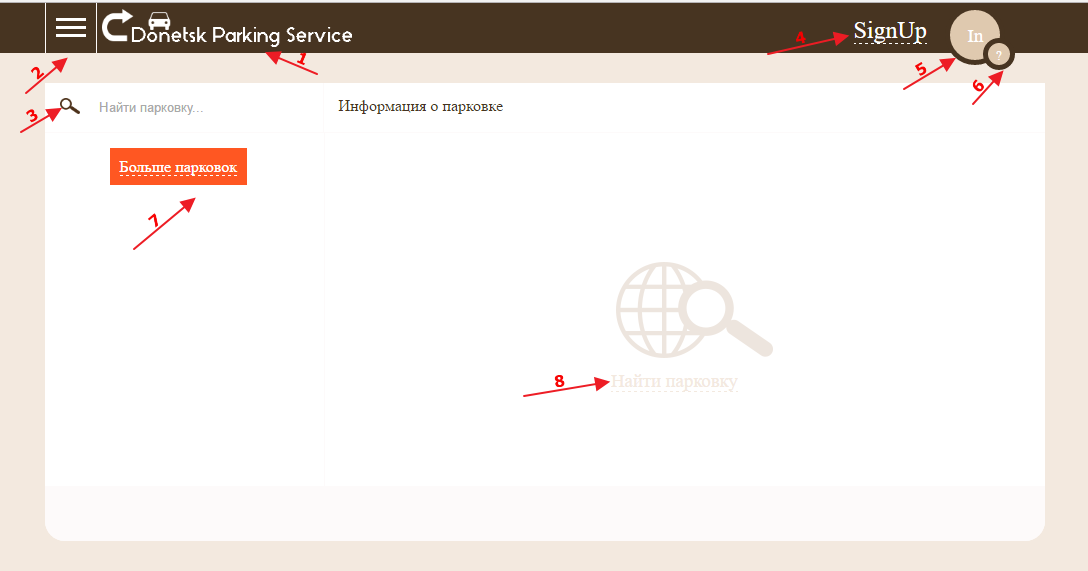


Рисунок 37 – Внешний вид главного рабочего окна пользователя

Описание функционала страницы:

1. Место под логотип веб-страницы
2. «Гамбургер меню» или главное меню приложения
3. Поиск парковок
4. Кнопка регистрации
5. Кнопка авторизации
6. Кнопка справки и технической поддержки
7. Кнопка отображения парковок в регионе
8. Кнопка перехода к поиску парковок

Использование данного программного продукта допустимо без регистрации и авторизации. Однако при авторизации пользователь получает ряд возможностей, которые облегчают использование данного приложения, такие как «понравившиеся парковки». Авторизированный пользователь может формировать понравившиеся ему места автопарковок в группу и затем быстро находить интересующую его автопарковку.

1. Просмотр списка автопарковок

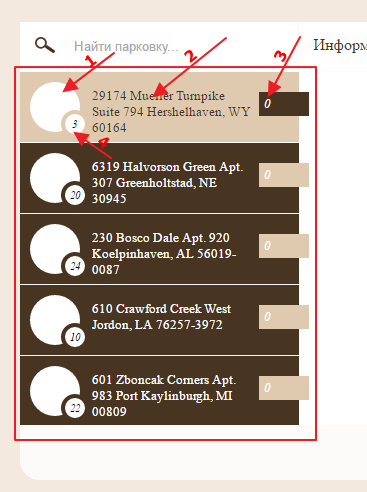


Рисунок 38 – Просмотр списка автопарковок

1. Иконка выбранной автопарковки
2. Физический адрес автопарковки
3. Начальная цена за проковочное место в единицу времени
4. Количество свободных мест на автопарковке

Данный список парковок можно при помощи колёсика мышки прокручивать вниз. Если вы достигли конца списка можно подгрузить еще список парковок, который дополнит уже ранее открытый список.

Если двойным кликом нажать на одну из парковок, то пользователю откроется список автомобилей в данный момент на этой парковке (Рисунок 39), свободные парковочные места, а также информация о парковке.

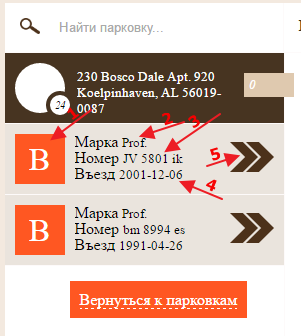


Рисунок 39 – Список автомобилей на выбранной парковке

1. Просмотр информации об автопарковке

Данный раздел позволяет просмотреть полную детальную информацию о выбранной автопарковке. Сверху страницы располагается карта с меткой, указывающей на физическое расположение автопарковки и текстом написан точный адрес. Помимо этого, с левой стороны блока с адресом есть цветная полоска. В случае если парковка «свободна» цвет выставляется в зеленый, если парковка загружена больше половины, то цвет оранжевый, если парковка заполнена полностью, то цвет красный.

Отдельно на карте присутствует кнопка добавления в избранное. Если пользователь считает что данная парковка может понадобится в дальнейшем, то можно добавить её в избранное и открывать сразу из списка избранных парковок.

Под картой располагаются два блока с видео-потоками. Если парковка имеет в наличии видео камеры, то администратор подключает видео-поток и пользователю доступно круглосуточно обзор всей парковки и своего автомобиля в частности. С правой стороны сайта расположены три рекламные фотографии парковки. Данная информация размещается администратором на своё усмотрение. В случае отсутствия видео- или фотоматериала система подставит в эти места шаблонные изображения.

Чуть ниже информационной части автопарковки идут ряды с местами для автомобилей. Пользователь может нажать мышкой на любое парковочное место и просмотреть информацию о нём если оно занят (кем занят и приблизительное время освобождения – пользователь может не забрать в срок свой автомобиль и до момента физического выезда автомобиля с места парковки таймер времени парковки будет прибавляться на один час, и соответственно оплата за парковочное место так же будет прибавляться в размере, указанном как минимальный размер оплаты в информации о парковке), либо зарезервировать парковочное место для себя. Парковочное место можно резервировать бесплатно за один час до физического въезда на парковку, либо платно, если въезд планируется, например, на следующий день.

Завершается страница парковки блоком с комментариями пользователей. Любой зарегистрированный пользователь может оставить свой отзыв.

1. Регистрация в приложении

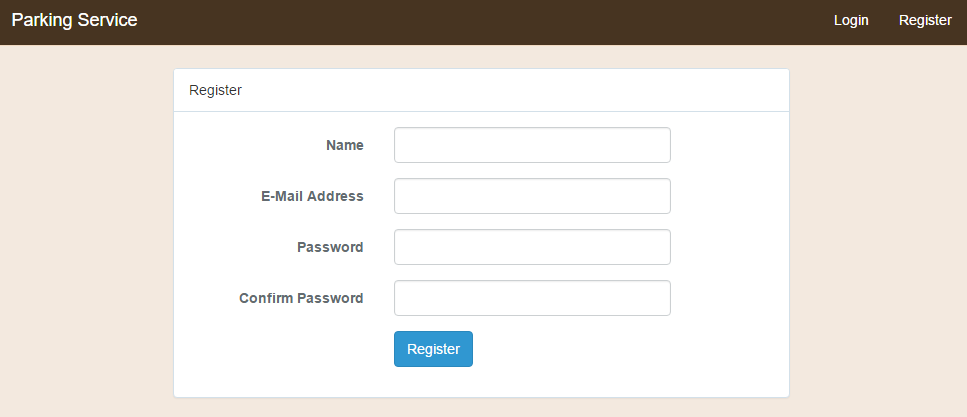


Рисунок 40 – окно регистрации в приложении

1. Вход в приложение осуществляется через отдельную страницу. После входа в приложение пользователя перекидывает на главную страницу. При этом функционал главное страницы расширяется.

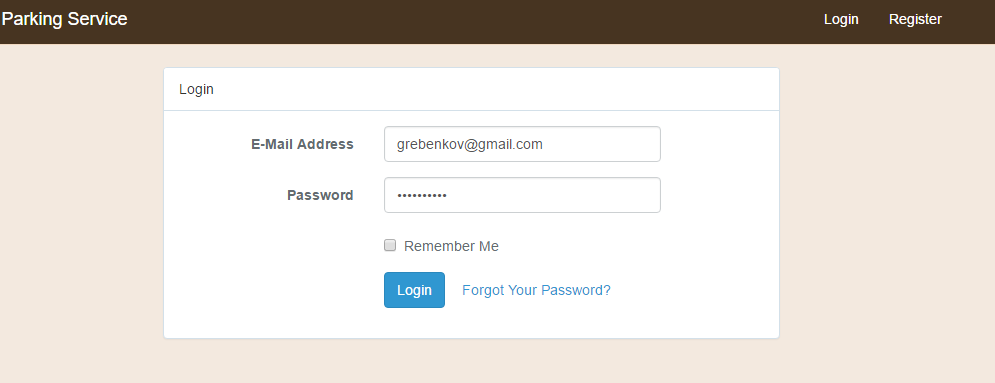


Рисунок 41 – окно входа в приложение

Помимо классического сценария входа есть возможность оставаться в системе после закрытия вкладки, а также есть возможность напоминания пароля в случае его утери.

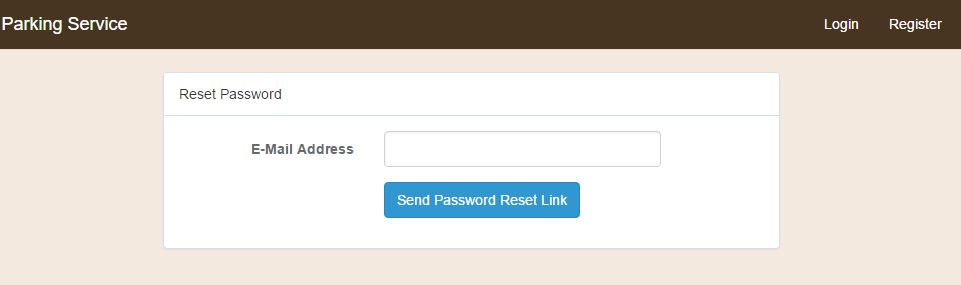


Рисунок 42 – окно восстановления пароля

1. Окно справки и технической поддержки

Данное окно содержит текущее руководство пользователя в полном объеме, а также форму для отправки вопросов администратору сайта. Ответ от администратора приходит на электронный адрес пользователя в течении одной рабочей недели.

Общий концепт работы приложения

Пользователь хочет найти парковку на которой есть свободные места. Он обращается к веб-сайту сервиса автопарковки через ноутбук или мобильный телефон и вводит искомые данные. Выбрав подходящую парковку, которая удовлетворяет искомому запросу, пользователь бронирует место и\или физически едет на выбранную парковку, и оставляет свой автомобиль.

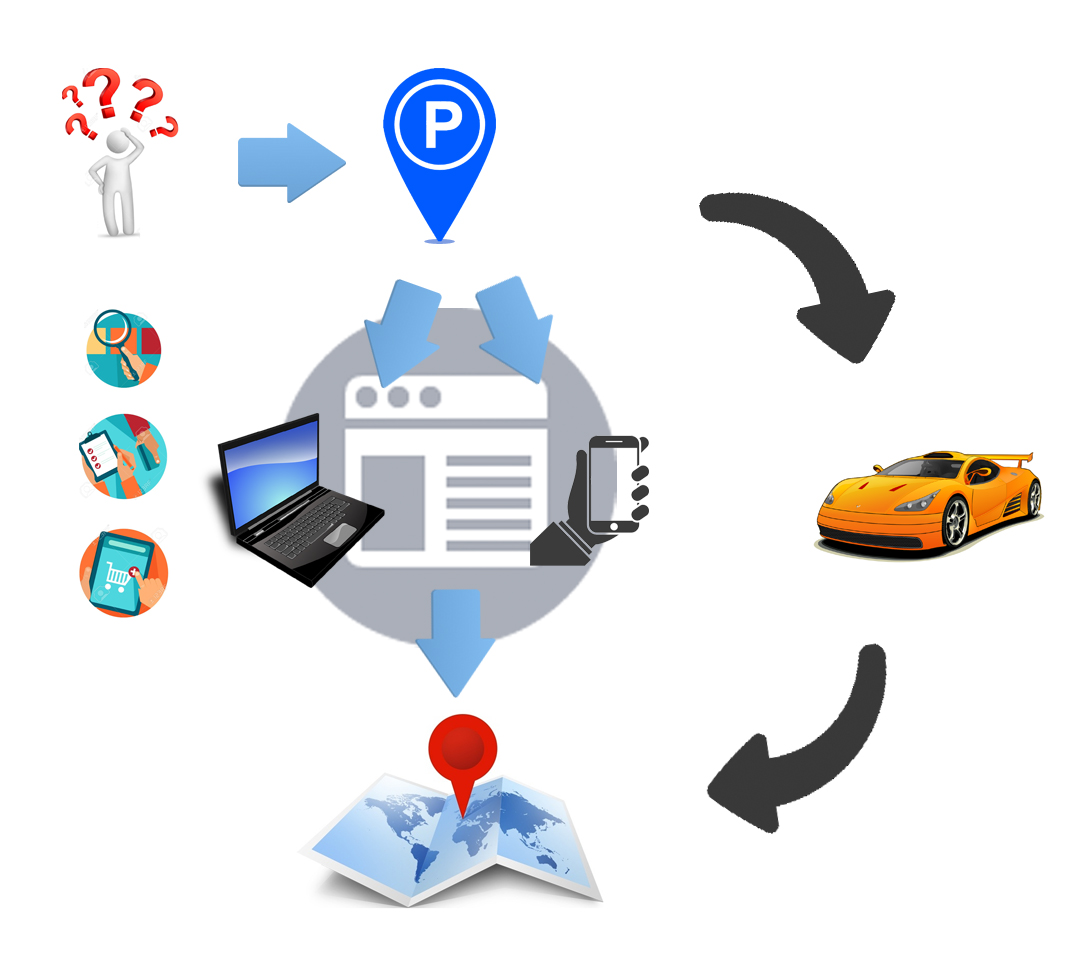


Рисунок 43 – концептуальная схема работы приложения

## **7 Тестирование и внедрение**

## 7.1 Анализ надежности программного продукта

Для проверки надежности и отказоустойчивости продукта будем вводить различные данные в открытые поля ввода. Данные будут вводится в течении определенного периода времени и если в течении этого времени система не выдаст непредвиденную ошибку, то результат тестирования можно считать успешным. Если же в процессе ввода данных появилась ошибка, которая не была нами отловлена, то результат теста будет считаться провальным (отказ). Данные для тестирования, время тестирования и результаты выполнения теста отображены в таблице 10.

Таблица 10 – Анализ надежности программного продукта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № Теста | Данные для теста  (вводимая информация) | Время функциони-рования программы (мин.) | Результат (успешно\отказ) | Результат при успешном выполнении |
| 1 | Ввод корректного электронного адреса для восстановления  пароля | 5 мин | успешно | Получение данных о восстановлении пароля |
| 2 | Ввод некорректного электронного адреса | 2 мин | успешно | Вывод сообщения об ошибке о недопустимом типе введенного значения |

Продолжение таблицы 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | Ввод корректного и существующего в базе адреса, но не существующего на SMTP-сервере | 10 мин | успешно | На электронный адрес администратора будет отправлено сообщение об ошибке |
| 4 | Ввод корректного проверочного слова при регистрации | 0,5 мин | успешно | Добавление пользователя в базу |
| 5 | Ввод некорректного проверочного слова при регистрации | 0,5 мин | успешно | Вывод сообщения об ошибке, повторный запрос ввода проверочного  слова |
| 6 | Не все обязательные поля при регистрации заполнены | 2 мин | успешно | Вывод подсказки о незаполненном обязательном поле под конкретным полем |
| 7 | Поля фамилия, имя, отчество введены не кириллицей | 2 мин | успешно | Вывод подсказки о некорректном заполнении |
| 8 | Введенные значения в текстовые поля превышают максимальную длину | 2 мин | успешно | Вывод подсказки о некорректной длине поля |
| 9 | Ввод длины пароля меньше минимального значения | 0,5 мин | успешно | Вывод подсказки о некорректной длине пароля (должен быть не менее 6 символов) |

Окончание таблицы 10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | Поле логина содержит некорректные символы |  | 0,5 мин | успешно | Вывод ошибки о некорректном введении логина |
| 11 | Ввод букв в поле суммы пополнения баланса |  | 0,5 мин | успешно | Вывод подсказки о некорректном заполнении |
| 12 | Выполнение операции оплаты общежития с недостаточным балансом |  | 5 мин | успешно | Вывод ошибки о недостатке средств на балансе |

## 7.2 Тестирование скорости работы сайта

Используя средства для разработчика Google было проведено нагрузочное тестирование веб-приложения, которое определило неоптимизированные аспекты в приложении.

Таблица 11 – Тестирование загрузки страницы при разных скоростях интернета

|  |  |
| --- | --- |
| Тип интернета | Скорость загрузки страницы, мс |
| Offline | 600 (отправка запроса) |
| GPRS (50 KB/s 500ms RTT) | 150000 |
| Regular 2G (250 KB/s 300ms RTT) | 35000 |
| Good 2G (450 KB/s 150ms RTT) | 19000 |
| Regular 3G (750 KB/s 100ms RTT) | 12000 |

Продолжение таблицы 11

|  |  |
| --- | --- |
| Good 3G (1 MB/s 40ms RTT) | 7000 |
| Regular 4g (4 MB/s 20ms RTT) | 3000 |
| DSL (2 MB/s 5ms RTT) | 5250 |
| WiFi (30 MB/s 2ms RTT) | 3100 |

Для проверки компонентов сайта на скорость их загрузки при разных типах интернета были использованы средства браузера Google Chrome, которые позволили отследить временные интервалы, в пределах которых загружаются компоненты сайта. На рисунке 44 представлена скорость загрузки страницы администратора при максимальной скорости интернета.

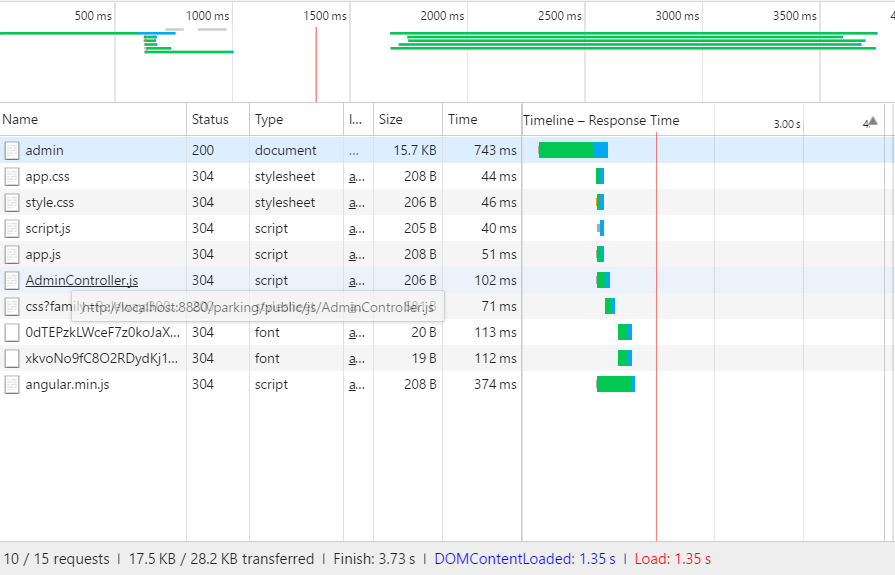


Рисунок 44 – Скорость загрузки страницы администратора при максимальной скорости интернета

Как видно из результатов на рисунке 44 общая скорость загрузки страницы администратора при максимально хорошем интернете составляет 1.5 секунды.

В ходе тестирования методом черного ящика приложение показало работоспособность и высокую степень надежности. При тестировании загрузки сайта с различными типами интернета были выявлены проблемы с не оптимальностью размера ресурсов, что повлияло на скорость загрузки сайта.

7.3 Тестирование серверной части и базы данных

PHPUnit является одним из старейших и наиболее известных пакетов тестирования модулей для PHP. Он в первую очередь предназначен для модульного тестирования, позволяет проводить тестирование кода мельчайших компонентов. PHPUnit включает в себя множество простых и гибких утверждений, которые позволяют легко проверить код. Чтобы сделать процесс тестирования для разработчиков в рамках Laravel PHP включает в себя набор тестовых приложений-хелперов , которые позволяют писать очень простые тесты PHPUnit для тестирования сложных частей приложения.

Создание нового теста

Первый шаг при использовании PHPUnit это создание нового класса теста. Основным нюансом тестовых классов является, то что они хранятся в каталоге приложения ./tests/. Внутри этой папки, каждый тестовый класс назван <name>Test.php. Этот формат позволяет РНРunit найти каждый тестовый класс - он будет игнорировать все, что не заканчивается Test.php.

В новом приложении Laravel, размещено по умолчанию два файла в каталоге /tests/: ExampleTest.php и TestCase.php. TestCase.php файл представляет собой загрузочный файл для настройки среды Laravel в тестах. Это позволяет использовать Laravel фасады при тестирование, и обеспечивает основу для тестирования хелперов.

Чтобы создать новый тестовый класс, мы можем создать новый файл вручную - или запустить Artisan make:test команду, представленную Laravel.

Для тестирования нашего приложения мы создадим класс BasicTest, в котором проведем тестирование основных страниц нашего веб-приложения. Для проверки работы с базой данных необходимо убедится что данные присутствуют в запросе.

<?php

**use** App\Parking;

**class** **BasicTest** **extends** TestCase

{

**public** **function** testDataInJSON()

{

$this->visit('/admin/users')->see('id');

$this->visit('/admin/avto')->see('id');

$this->visit('/admin/parking')->see('id');

$this->visit('/admin/fav')->see('id');

$this->visit('/admin/history')->see('id');

$this->visit('/admin/ticket')->see('id');

}

}

Рисунок 45 – пример кода для тестирования работы базы данных по средствам проверки наличия данных в JSON запросе

public function testApplication()

{

$response = $this->call('GET', '/');

$this->assertEquals(200, $response->status());

}

Рисунок 46 - пример кода для тестирования корректной загрузки главной страницы приложения

public function testDatabase()

{

$this->seeInDatabase('users', ['email' => 'grebenkov.p@example.com']);

}

Рисунок 47 - пример кода для тестирования базы данных на наличие соответствующих данных в столбце email таблицы users

**ЗАКЛЮЧЕНИЕ**

Для реализации курсового проекта были изучены процессы происходящие на автопарковках города, а также проведен сравнительный анализ функционала аналогичных систем и сделаны выводы о том, что необходимо людям, пользующихся автомобилем, на автопарковках города. На основе полученных данных было спроектировано приложение и в последствии реализовано в виде веб-системы, обслуживающей потребности автопарковок города и региона.

В процессе разработки веб-приложения были изучены технологии клиент-серверной разработки, такие как: html5&css3 – для верстки макета сайта, JavaScript & AngularJS – для разработки прослойки между клиентом и сервером (а также технология асинхронного обмена данными AJAX) и технология – Laravel5 Framework – для разработки серверного приложения, - и изучен соответствующий книжный материал.

В ходе реализации курсового проекта были реализованы следующие задачи:

1. Была разработана база данных клиент-серверного приложения «Автопарковка», состоящая из шести таблиц: таблица пользователей, таблица восстановления пароля учетной записи пользователя, таблица автомобилей, размещенных на автопарковке, таблица автопарковок, таблица понравившихся автопарковок, таблица истории действий парковок, которая необходима для ведения статистик по всем автопарковкам, таблица заявок на парковку\техническое обслуживание.
2. Разработано Rest API для создания гибких интерфейсов отдельно взятых автопарковок. В качестве основного формата передачи данных был взят JSON.
3. Разработан современный веб-интерфейс системы с использованием технологии SCSS для верстки пользовательской части веб-приложения и Bootstrap3 Framework – для верстки административной части веб-приложения.
4. Разработана система обратной связи с автовладельцем по средствам обмена сообщения на почту автовладельца через протокол SMTP.
5. Разработана система учета автопарковок и автомобилей на автопарковках.
6. Разработана системы бронирования мест на парковке.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Официальный сайт автопарковки «Борисполь» / система автоматизации автопарковок в аэропорту Борисполь в городе Киев

URL: http://bac.in.ua/ (дата обращения: 30.10.2016)

2. Официальный сайт AngularJS/ [ANGULARJS]

URL: https://angularjs.org/ (дата обращения: 30.10.2016)  
 3. Рафал Т. Основы Web-Дизайна. Руководство / Т. Рафал. – RafalTomal.com, 2015. – 211 с.

4. Хоган Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения / Б. Хоган. – Изд. 2-е – Спб.: Питер, 2014. – 320 c.  
 5. Официальная документация Laravel 5 / Web MVC framework  
URL: https://laravel.com/docs/5.3/installation (дата обращения: 12.10.2016)  
 6. Образовательный проект Laravel по-русски // [Laravel 5] / Web MVC Framework– основные понятия, архитектура, разбор примеров написания кода URL: https://laravel.ru/docs/v5 (дата обращения: 13.10.2015)  
 7. Образовательный проект w3school // [PHP] – изучение синтаксиса языка PHP URL: http://www.w3schools.com/php/default.asp (дата обращения: 13.10.2015)  
 9. Официальное сообщество набора разработчика XAMPP / Начало работы с PHP URL: https://www.apachefriends.org/ru/index.html  
(дата обращения: 15.10.2016)  
 10. Официальный сайт Bootstrap 3/ [HTML5&CSS3]  
URL: http://getbootstrap.com/ (дата обращения: 17.10.2016)  
 11. Sébastien Eustace Eloquent Documentation Release 0.4, 2015. – 53 c.

## **ПРИЛОЖЕНИЕ А**

## (обязательное)

**Фрагменты листинга**

Листинг А.1 – Файл Routes -> web.php

# 

|  |
| --- |
| Кол.листов |

|  |
| --- |
| № Строки |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *1* |  |  |  |  |  |  |
| *2* |  |  | *Документация общая* |  |  |  |
| *3* |  |  |  |  |  |  |
| *4* |  |  | *Вновь разработанная* |  |  |  |
| *5* |  |  |  |  |  |  |
| *6* | *А4* | *09.03.01.080000.000 ПЗ* | *Пояснительная записка* | *93* |  |  |
| *7* |  |  |  |  |  |  |
| *8* |  |  |  |  |  |  |
| *9* |  |  |  |  |  |  |
| *10* |  |  |  |  |  |  |
| *11* |  |  |  |  |  |  |
| *12* |  |  |  |  |  |  |
| *13* |  |  |  |  |  |  |
| *14* |  |  | *Применённая* |  |  |  |
| *15* |  |  |  |  |  |  |
| *16* |  | *Пр. № 233/08 от15.03.2016* | *Задание на ВКР* | *1* |  |  |
| *17* |  |  |  |  |  |  |
| *18* |  |  |  |  |  |  |
| *19* |  |  |  |  |  |  |
| *20* |  |  | *Прочая документация* |  |  |  |
| *21* |  |  |  |  |  |  |
| *22* | *А4* | *09.03.01.080000.000 Д* | *Презентация работы* | *7* |  |  |
| *23* |  |  |  |  |  |  |
| *24* |  |  |  |  |  |  |
| *25* |  |  |  |  |  |  |
| *26* |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

№ экз.

Обозначение

|  |
| --- |
| Формат |

Наименование

Приме-

чание

1

Листов

Лит.

Утверд.

Толстых В.К.

Н. Контр.

Медведева В.Г

Провер.

Петров П.П.

1

Лист

Дата

Подпись

№ докум.

Лист

Изм.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
| Инв. № подл. |

|  |
| --- |
| Подпись и дата |

|  |
| --- |
| Взам. инв. № |

|  |
| --- |
| Инв. № дубл. |

|  |
| --- |
| Подпись и дата |

|  |
| --- |
| Справ. № |

|  |
| --- |
| Перв. примен. |

Разраб.

Иванов И. И.

09.03.01.080000.000 ВР

«*Разработка*

*интернет-приложения «Название интернет-приложения»*»

(Ведомость работы)

*ДОННУ*

Кафедра «КТ»