**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Кафедра информатики и компьютерных систем**

ВОЛКОВА

Мария Николаевна

**РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ «ЛИЧНЫЙ КАБИНЕТ» ДЛЯ СИСТЕМЫ МОНИТОРИНГА ТРАНСПОРТА  
С ИСПОЛЬЗОВАНИЕМ ANGULAR**

Дипломная работа

Научный руководитель:

кандидат технических наук, доцент И. П. Стецко

Допущена к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Зав. кафедрой информатики и компьютерных систем,

доктор технических наук, профессор С. Г. Мулярчик

Минск, 2021

**ОГЛАВЛЕНИЕ**

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ 3](#_Toc73914625)

[РЕФЕРАТ 5](#_Toc73914626)

[ВВЕДЕНИЕ 8](#_Toc73914627)

[ГЛАВА 1](#_Toc73914628) [СИСТЕМА МОНИТОРИНГА ТРАНСПОРТА 10](#_Toc73914629)

[1.1 Принцип работы 10](#_Toc73914630)

[1.2 Анализ функционала приложений 11](#_Toc73914631)

[1.3 Приложение «Личный кабинет» 15](#_Toc73914632)

[ГЛАВА 2](#_Toc73914633) [АРХИТЕКТУРА WEB-ПРИЛОЖЕНИЙ 18](#_Toc73914634)

[2.1 Модель клиент-сервер 18](#_Toc73914635)

[2.2 Протоколы взаимодействия между клиентом и сервером 21](#_Toc73914636)

[2.3 Способы предоставления доступа 23](#_Toc73914637)

[2.4 Распределение функций между клиентом и сервером 27](#_Toc73914638)

[ГЛАВА 3](#_Toc73914639) [СРЕДСТВА РАЗРАБОТКИ 30](#_Toc73914640)

[3.1 Понятие фреймворка 30](#_Toc73914641)

[3.2 Сравнение фреймворков 31](#_Toc73914642)

[3.3 Общие сведения о фреймворке *Angular* 40](#_Toc73914643)

[3.4 Тестирование web-приложений 43](#_Toc73914644)

[3.5 Средства развертывания 45](#_Toc73914645)

[ГЛАВА 4](#_Toc73914646) [РАЗРАБОТКА ПРИЛОЖЕНИЯ «ЛИЧНЫЙ КАБИНЕТ» 49](#_Toc73914647)

[4.1 Инфраструктура, обеспечивающая функционирование системы 49](#_Toc73914648)

[4.2 Структура базы данных 50](#_Toc73914649)

[4.4 Авторизация пользователя в приложении «Личный кабинет» 54](#_Toc73914650)

[4.5 Функции «Личного кабинета» 56](#_Toc73914651)

[4.6 Визуализация данных 63](#_Toc73914652)

[4.7 Тестирование и сборка приложения 64](#_Toc73914653)

[ЗАКЛЮЧЕНИЕ 65](#_Toc73914654)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 66](#_Toc73914655)

# **ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

БД – база данных

ГИС – геоинформационная система

ООП – объектно-ориентированное программирование

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

*ACSII –* American standard code for information interchange(американский стандартный кодекс для информационного обмена)

*AJAX* – *Asynchronous JavaScript and XML* (асинхронный *JavaScript* и *XML*)

*API – Application Programming Interface* (программный интерфейс приложения)

*CRUD* – *Сreate, Read, Update, Delete* (cоздание, чтение, обновление, удаление)

*CSS – Cascading Style Sheets* (каскадные таблицы стилей)

*DOM – Document object model* (объектная модель документа)

*FTP –* File Transfer Protocol (протокол передачи файлов)

*GPS* – Global Positioning System (система глобального позиционирования)

*IT – Information Technology* (информационные технологии)

HTML – *Hyper Text Markup Language* (язык гипертекстовой разметки)

*HTTP – Hyper Text Transfer Protocol* (протокол передачи гипертекста)

*JSON – JavaScript Object Notation* (текстовый формат обмена данными основанный на JavaScript)

*REST – Representational State Transfer* (передача состояния представления)

*SAML* – Security assertion markup language (язык разметки декларации безопасности)

SASS – *Syntactically Awesome Stylesheets* (синтаксические таблицы стилей)

*SMS –* Short Message Service (служба коротких сообщений)

SMTP – *Simple Mail Transfer Protocol* (простой протокол передачи почты)

*SSL* – Secure Sockets Layer (уровень защищенных сокетов)

*TCP* – *Transmission Control Protocol* (протокол управления передачей)

*TLS –* Transport Layer Security (протокол защиты транспортного уровня)

*URI –* Uniform Resource Identifiers (единый идентификатор ресурсов)

*URL* – Uniform Resource Locator (система унифицированных адресов)

*XML* – *Extensible Markup Language* (расширенный язык разметки)

# **РЕФЕРАТ**

Дипломная работа: 65 страниц, 28 рисунков, 20 источников.

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ «ЛИЧНЫЙ КАБИНЕТ» ДЛЯ СИСТЕМЫ МОНИТОРИНГА ТРАНСПОРТА С ИСПОЛЬЗОВАНИЕМ ANGULAR.

*Объект исследования* – система мониторинга транспорта.

*Предмет исследования* – web-приложение «Личный кабинет» для системы мониторинга транспорта.

*Цель работы* – разработка web-приложения «Личный кабинет» для системы мониторинга транспорта.

*Практическая значимость* заключается в разработке web-приложения для системы мониторинга транспорта. Суть разрабатываемого приложения состоит в повышении эффективности рабочего процесса компаний, осуществляющих мониторинг транспорта за счёт снижения нагрузки в работе диспетчеров и передачи полномочий клиентам.

В результате выполнения работы было разработано web-приложение «Личный кабинет» для системы мониторинга транспорта. Были изучены принципы работы мониторинга транспорта, проанализирован функционал приложений компаний, предоставляющих услуги мониторинга транспорта. Были изучены и проанализированы различные архитектура для написания web-приложений, а также проведён сравнительный анализ средств разработки. Клиентская часть приложения была разработана с использованием фреймворка Angular. Для тестирование использовались модульные тесты. Разработанное приложение было собрано с использованием технологии контейнеризации docker.

**РЭФЕРАТ**

Дыпломная праца, 65 старонкi, 28 малюнкаў, 20 крыніц.

РАСПРАЦОЎКА WEB-ДАДАТКІ «АСАБІСТЫ КАБІНЕТ» ДЛЯ СІСТЭМЫ МАНІТОРЫНГУ ТРАНСПАРТУ З ВЫКАРЫСТАННЕМ ANGULAR.

*Аб'ект даследавання* - сістэма маніторынгу транспарту.

*Прадмет даследавання* - web-дадатак «Асабісты кабінет» для сістэмы маніторынгу транспарту.

*Мэта працы* - распрацоўка web-дадатку «Асабісты кабінет» для сістэмы маніторынгу транспарту.

Практычная значнасць заключаецца ў распрацоўцы web-прыкладання для сістэмы маніторынгу транспарту. Сутнасць распрацоўванага прыкладання складаецца ў павышэнні эфектыўнасці працоўнага працэсу кампаній, якія ажыццяўляюць маніторынг транспарту за кошт зніжэння нагрузкі ў працы дыспетчараў і перадачы паўнамоцтваў кліентам.

У выніку выканання работы быў распрацаван web-дадатак «Асабісты кабінет» для сістэмы маніторынгу транспарту. Былі вывучаны прынцыпы працы маніторынгу транспарту, прааналізаваны функцыянал прыкладанняў кампаній, якія прадстаўляюць паслугі маніторынгу транспарту. Былі вывучаны і прааналізаваны розныя архітэктура для напісання web-прыкладанняў, а таксама праведзены параўнальны аналіз сродкаў распрацоўкі. Кліенцкая частка прыкладання была распрацавана з выкарыстаннем фреймворка Angular. Для тэставання выкарыстоўваліся модульныя тэсты. Распрацаванае прыкладанне было сабрана з выкарыстаннем тэхналогіі контейнерызацыі docker.

**ABSTRACT**

The diploma consists of 65 pages. It contains 28 images, 20 sources.

DEVELOPMENT OF A WEB-APPLICATION «PERSONAL ACCOUNT» FOR THE TRANSPORT MONITORING SYSTEM USING ANGULAR.

*Object of research -* transport monitoring system.

*Subject of research - web-application «Personal account» for the transport monitoring system.*

*Objective – development of the web-application "Personal account" for the transport monitoring system.*

The practical significance lies in the development of a web-application for the transport monitoring system. The essence of the developed application is to increase the efficiency of the working process of the companies that monitor transport by reducing the workload of dispatchers and the transfer of authority to customers.

As a result of the work a web-application "Personal Cabinet" for the transport monitoring system was developed. The principles of transport monitoring were studied, the functionality of the applications of the companies that provide transport monitoring services was analyzed. Different architectures for writing web-applications were studied and analyzed, as well as a comparative analysis of the development tools. The client part of the application was developed using the Angular framework. Unit tests were used for testing. The developed application was built using containerization technology docker.

# **ВВЕДЕНИЕ**

Развитие современных технологий приводит к тому, что количество информации и данных, с которыми человеку приходится работать, растёт с каждым днём. В мире используются разные технологии для решения конкретных задач повседневной жизни во всех секторах, независимо от того, с какой отраслью мы работаем. Интернет стал неотъемлемой частью каждого аспекта нашей повседневной жизни. Информационные технологии привели к фундаментальным изменениям во многих сферах: бизнес, образование, правительство, здравоохранение и многое другое. Возможность подключения к Интернету через персональный компьютер, смартфон или планшет ведёт к формированию мобильной реальности, не привязанности человека к конкретному месту или устройству. В производстве стоит задача предоставить быстрый и удобный доступ к информации. Одним из решений данной задачи является разработка клиент-серверного приложения, с помощью которого пользователи могут просматривать и модифицировать информацию, производить обмен с базой данных.

Клиент-серверные приложения могут поддерживать большое количество функций, такие как рабочий процесс, хранилища данных, бизнес-аналитика, отчетность, интеграция данных. Обеспечивается гибкость системы, которая достигается использованием трех логических структур:

– представления (реализуют функцию ввода и отображения данных);

– прикладной (поддерживает прикладные функции, характерные для заданной предметной области);

– доступа к информации (реализует функцию хранения, менеджер ресурсов, управление информационно-вычислительными ресурсами).

Система способна адаптироваться к росту количества пользователей и увеличению объёма базы данных без замены программного обеспечения, а в основном за счёт увеличения аппаратных средств.

Требования современной жизни приводят к расширению возможностей Интернет-пространства. Все больше стало появляться веб-сайтов, реализующих различные функции. Не составляют исключения и личные кабинеты, предназначенные для решения различных задач. Сфера использования личных кабинетов огромна: от банков и интернет-магазинов до сайтов поиска работы. Создание удобного интерфейса, добавление возможности поиска, сортировки, доступа к скрытой от посторонних информации предоставляет пользователю возможность быстрого решения поставленных задач. Пользователь получает возможность доступа к своим личным данным вне зависимости от того, где он находится в конкретный момент времени.

**Объект исследования** – система мониторинга транспорта.

**Предмет исследования** – web-приложение «Личный кабинет» для системы мониторинга транспорта.

**Цель работы** – разработка web-приложения «Личный кабинет» для системы мониторинга транспорта.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. провести анализ систем мониторинга транспорта, основных потребностей в данной сфере;
2. практически освоить процесс создания и развёртывания web-приложений;
3. рассмотреть и выбрать средства разработки web-приложений;
4. разработать web-приложение для системы мониторинга транспорта;
5. протестировать основные функции и развернуть приложение.

**Практическая значимость** заключается в разработке web-приложения для системы мониторинга транспорта. Суть разрабатываемого приложения состоит в повышении эффективности рабочего процесса компаний, осуществляющих мониторинг транспорта за счёт снижения нагрузки в работе диспетчеров и передачи полномочий клиентам.

# **ГЛАВА 1**

# **СИСТЕМА МОНИТОРИНГА ТРАНСПОРТА**

## **1.1 Принцип работы**

Сегодня в различных отраслях экономики всё более востребованным становится мониторинг транспорта. Прежде всего это касается предприятий, имеющих парк автомобильного транспорта, где система *GPS* мониторинга транспорта может быть использована для решения широкого круга задач, в частности:

* оперативного дистанционного мониторинга местоположения и перемещения автотранспортных средств;
* оперативного дистанционного мониторинга различных технических параметров транспортных средств;
* транспортной логистики в системах управления перевозками и автоматизированных системах управления автопарками.

Использование системы мониторинга транспорта позволяет проложить оптимальный маршрут и обеспечить его безопасное прохождение, минимизировать расход топлива, повысить дисциплину в коллективе.

Система спутникового мониторинга транспорта – это программно-аппаратный комплекс, состоящий из трёх частей:

* устройств мониторинга транспортных средств;
* сервера геоинформационной системы (ГИС);
* клиентских диспетчерских мест.

Устройства мониторинга устанавливаются на транспортные средства и обеспечивают оперативное определение местоположения, перемещения на местности. Полученная телеметрическая информация проходит предварительную обработку, сохраняется и, при наличии соединения, передается на сервер. Сервер обеспечивает с помощью программного интерфейса приём и хранение навигационных и измерительных данных от мобильных объектов. В свою очередь клиент, имея доступ к серверу посредством интернет-соединения, может осуществлять контроль транспортного средства (рисунок 1.1).

Рисунок 1.1 – Система мониторинга транспорта

Принцип определение координат можно описать следующим образом:

* навигационный модуль, установленный на транспортном средстве, посредством систем глобального позиционирования определяет своё местоположение;
* с использованием сотовых каналов связи каждые 5-10 минут навигационный модуль информирует сервер о своём местоположении;
* сведения, собранные таким образом, пересылаются на сервер, где они анализируются и систематизируются при помощи программного обеспечения;
* данные, в обработанном виде поступают клиенту; используя компьютер или планшет, клиент может их просмотреть.

## **1.2 Анализ функционала приложений**

**На рынке нашей страны присутствует ряд компаний, осуществляющих мониторинг транспорта, например, Wialon, БелТрансСпутник, Ресурсконтроль, XLTracking, Navirec, АНТЕЛИС Электроникс.**

Развитие компьютерного программного обеспечения для автоматизации работы различных предприятий на протяжении многих лет привело к созданию разных категорий программного обеспечения. Система управления рабочим процессом – это совокупность программных систем, которые берут на себя все функции по управлению процессами. Взаимодействие между системами и обмен данными позволяет автоматизировать рабочий процесс, снизить затраты на техническое обслуживание в течение всего жизненного цикла. Централизованный сбор данных, всесторонний онлайн-мониторинг, визуализация рабочих характеристик составляют основу удобного детального анализа всех данных процесса для улучшения эффективности в работе предприятия.

Компании, осуществляющие мониторинг транспорта, предоставляет своим клиентам доступ к программным приложениям. Системы онлайн мониторинга позволяют отображать различные показатели работы транспортных средств такие как:

* местоположение машины;
* подтверждение пробега, прогноз прибытия;
* температурный контроль;
* наличие заправок и сливов;
* расход топлива и значение других встроенных датчиков.

**Система *GPS-*мониторинга транспорта Wialon предоставляет своим пользователям около 20 различных приложений, которые позволяют осуществлять те или иные функции, в зависимости от поставленных задач. Приложение «Диспозиция» позволяет оперативно получать информацию по всему автопарку. При загрузке приложения автоматически формируется единая таблица-сводка, содержащая всю необходимую информацию. Возможность детального просмотра перемещения объекта, с указанием его скорости, координат, а также основных параметров датчиков. Отображение информации по всему автопарку в одном окне. Возможность осуществление быстрого поиска.**

**«Личный кабинет» от компании Wialon предоставляет пользователям возможность контролировать весь документооборот через приложение. В приложении пользователю доступны 5 активных вкладок: состояние счёта, документация, заявка на услуги, объекты, переход в системы мониторинга. Через электронный кабинет пользователь может управлять объектами, что позволяет существенно сократить телефонные звонки и работу с бумажной документацией. Приложение предоставляет пользователям возможность самостоятельно добавлять новые объекты или изменять уже имеющиеся. Минус приложения: доступ предоставляется только клиентам компании Wialon.**

**БелТрансСпутник – *IT* компания с собственными разработками в области управления транспортом, аналитики и логистики. Система управления включает инновационные технологии на базе искусственного интеллекта и анализа данных. Компания имеет программное обеспечение, которое предоставляет следующие функции: онлайн мониторинг транспорта, построение маршрутов, отображение показателей датчиков, контроль топлива, учёт сотрудников и постановка им задач.**

**Компания Ресурсконтроль предоставляет клиентам эффективный инструмент для контроля транспорта и учёта рабочего времени сотрудников. Система позволяет оптимизировать время поездок и сократить маршруты. Программное обеспечение компании позволяет получать отчёты, осуществлять мониторинг транспорта онлайн, контролировать расход топлива.**

**Программный комплекс компании XLTracking – эффективный инструмент спутникового мониторинга транспорта, снижающий расходы, увеличивающий продуктивность работы персонала. Сервис позволяет провести анализ автоперевозок, сократить топливные расходы путём предоставления детальных отчётов о перемещении транспортных средств. Приложение имеет возможности добавлять несколько компаний в одном аккаунте, добавлять сотрудников и распределять их по уровням доступа, осуществлять управление автопарком, контролировать прохождение технического обслуживания, вести учёт водителей, возможность группировать, документировать данные, полученные с датчиков и контролировать их показателей.**

**Компания** **Navirec предоставляет клиентам приложения, которые позволяют осуществлять следующие функции: определять местоположение транспорта в реальном времени, время в пути и стоянок, возможность создавать собственные отчёты и группировать данные, управлять техническим обслуживанием автопарка, осуществлять контроль и бронирование транспортного средства с указанием водителя и типа поездки, возможность управления транспортом из любой точки.**

**Компания Антелис Электроникс предлагает комплексное решение проблем дистанционного мониторинга подвижных объектов. Компания предоставляет программное обеспечение, которое позволяет осуществлять следующие функции: мониторинг транспорта в режиме реального времени, расход топлива, формирование отчётов, группировка транспортных средств, показания датчиков за разные периоды времени.**

**Таким образом, проанализировав программные средства, предоставляемые разными компаниями, осуществляющими мониторинг транспорта, можно сформировать сравнительную таблицу 1.1.**

**Таблица 1.1 – Сравнение функций приложений разных компаний**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Wialon** | **БелТрансСпутник** | **Ресурсконтроль** | **XLTracking** | **Navirec** | **Антелис Электроникс** |
| **онлайн мониторинг** | **+** | **+** | **+** | **+** | **+** | **+** |
| **формирование отчётов** | **+** | **+** | **+** | **+** | **+** | **+** |
| **контроль расхода топлива** | **+** | **+** | **+** | **+** | **+** | **+** |
| **возможность добавлять подкомпании** | **–** | **–** | **–** | **+** | **–** | **–** |
| **управление автопарком** | **+** | **–** | **–** | **+** | **–** | **–** |
| **учёт документов** | **+** | **–** | **–** | **+** | **–** | **–** |
| **контроль технического обслуживания** | **–** | **–** | **–** | **+** | **–** | **–** |
| **самостоятельное добавление датчиков в систему** | **–** | **–** | **–** | **–** | **–** | **–** |

**Проанализировав уже имеющиеся приложения, разработанные компаниями, осуществляющими услуги мониторинга транспорта, можно сделать следующий вывод: большинство приложений предоставляют пользователю оперативно получать текущую информацию по всему автопарку, контролировать документооборот, формировать отчёты и контролировать расход топлива. Однако большинство приложений не покрывают весь рабочий процесс: нет возможности управлять пользователями, добавлять и сортировать транспортные средства, добавлять навигационные модули и самостоятельно изменять связи (пользователь – транспортное средство), (транспортное средство – навигационный модуль). К большинству приложений имеют доступ только клиенты компании.**

Учёт различных рабочих процессов на предприятии, внесение в базу данных информации об аренде датчиков и навигационных модулей, закрепление датчиков за машинами, управление машинами предприятий, добавление новых предприятий, как правило, осуществляется вручную диспетчерами компании реализующих средства для мониторинга транспорта.

Для упрощения работы и автоматизации процесса было решено разработать специальное приложение – «Личный кабинет». Приложение предоставит клиентом компании возможность с легкостью управлять услугами мониторинга.

## **1.3 Приложение «Личный кабинет»**

«Личный кабинет» – персональная страница пользователя сайта, где он может выполнять различные действия и отслеживать связанную с ним и скрытую от посторонних информацию.

«Личный кабинет» для компаний, пользующихся услугами мониторинга транспорта – это браузерное интернет-решение, использующиеся для управления автопарком в одном месте, контроля закрепления разнообразных датчиков. Пользователь может зайти в приложение через веб-браузер в любое время из любого места, с разных устройств (настольный ПК, ноутбук, смартфон) и, таким образом, получить доступ к необходимой информации, оперативно осуществлять всевозможные операции. Пользователям могут быть назначены разные права доступа, как это необходимо для конкретной области ответственности пользователя.

Преимущества использования «Личного кабинета»:

* снижение нагрузки в работе диспетчеров – компании не требуется нанимать и обучать новых диспетчеров, тратить деньги на содержание их рабочих мест, так как большинство операций имеется в приложении, и специалисты клиента смогут осуществлять их сами;
* автоматизация процесса заполнения базы данных и установки связей между различными таблицами в базе данных (БД);
* уменьшение количества ошибок – все данные, которые вносит пользователь, перед записью в БД проверяются на сервере;
* информация по всему автопарку расположена в одном окне;
* удобный интерфейс приложения для осуществления сложных операций в несколько шагов;
* быстрый поиск необходимой информации, всех машин, датчиков и их привязок.

В приложении каждому пользователю в процессе регистрации назначается роль: администратор или пользователь. Роль администратора определяется для операторов компании, предоставляющей услуги мониторинга транспорта. В приложении для администратора должны быть доступны следующие функции:

1. просмотр списка всех пользователей:
2. добавление нового пользователя;
3. просмотр личные данные любого пользователя;
4. изменение роли;
5. изменение статуса пользователей (блокировка / разблокировка);
6. просмотр списка всех машин:
7. добавление новых машин любому пользователю;
8. редактировка информации о машине;
9. разделение списка машин на группы;
10. архивирование машины.
11. просмотр списка всех навигационных модулей:
12. добавление нового навигационного модуля;
13. редактирование информации о навигационном модуле;
14. изменение привязки навигационный модуль – пользователь и навигационный модуль – машина;
15. списание старых блоков.

Роль пользователя должна определятся для компаний, которым предоставляются услуги мониторинга транспорта. Приложение «Личный кабинет» должно позволять пользователям осуществлять следующие функции:

1. редактирование личных данных;
2. просмотр списка всех машин:
3. сортировка машин по группам;
4. добавление новой машины;
5. редактирование информации о машине.
6. просмотр списка всех навигационных модулей (датчиков):
7. добавление нового навигационного модуля;
8. редактирование информации о навигационном модуле;
9. изменение привязки навигационный модуль – пользователь и навигационный модуль – машина;
10. списание старых блоков.

Вход в «Личный кабинет» осуществляется через Интернет. В отсутствии связи это может оказаться недостатком, так как пользователь не будет иметь возможность самостоятельно осуществить необходимые функции.

# **ГЛАВА 2**

# **АРХИТЕКТУРА WEB-ПРИЛОЖЕНИЙ**

## **2.1 Модель клиент-сервер**

Модель клиент-сервер является наиболее распространенной формой сетевой архитектуры, используемой для передачи данных [1]. В современном Интернет-пространстве модель стала популярной, благодаря использованию её во многих приложениях. Концепция архитектуры клиент-сервер разделяет выполнение функциональных единиц работы между действиями, которые инициируются непосредственно пользователями (клиент), и ответами на запросы со стороны сервера. Клиент – инициатор запроса, сервер – поставщик услуг (рисунок 2.1).

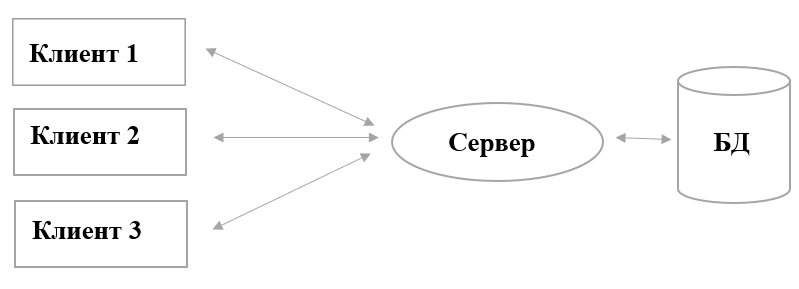


Рисунок 2.1 – Модель клиент-сервер

В модели клиент-сервер клиентом является персональный компьютер пользователя или рабочая станция. В большинстве приложений, построенных на основе клиент-серверной архитектуры, обработка данных осуществляется сервером, а результаты возвращается клиенту, что позволяет увеличить производительность всего приложения [2].

Клиент-сервер – это приложение совместной обработки, в котором взаимодействие пользователя с вычислительной средой (сервером) осуществляется через программируемую рабочую станцию (клиента), которая выполняет некоторую часть приложения [1].

Клиент-серверная архитектура состоит из сервера приложения, сервера базы данных и ПК. Основные типы архитектуры: двухуровневая и трёхуровневая архитектуры.

Двухуровневая архитектура модели клиент-сервер – это архитектура, которая включает сервер базы данных и клиентский ПК (рисунок 2.2). В двухуровневой архитектуре пользователь запускает приложение на своём ПК, который подключается через сеть к серверу. Клиент имеет доступ к базе данных напрямую без участия каких-либо посредников. Клиентское приложение осуществляет кодирование и бизнес-логику, а затем отображает все полученные данные пользователю.

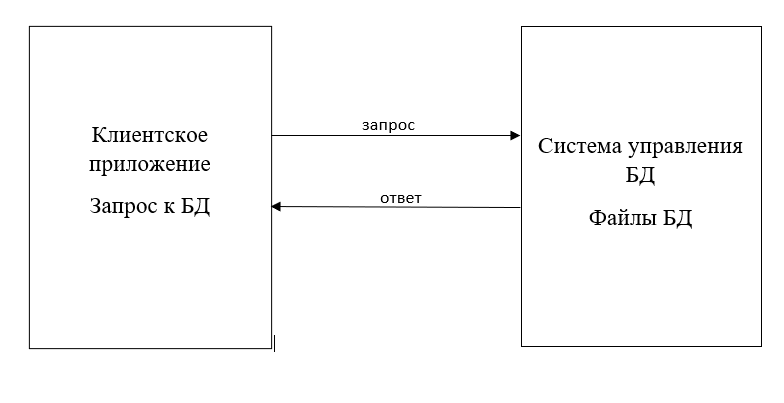


Рисунок 2.2 – Двухуровневая клиент-серверная архитектура

Трехуровневая архитектура модели клиент-сервер включает клиентский ПК, сервер приложения, сервер базы данных (рисунок 2.3). Трехуровневая архитектура может быть расширены до N-уровней за счет использования нескольких серверов приложения. В этой архитектуре клиент поддерживает только логику представления, поэтому на стороне клиента требуется меньше ресурсов и кода. Основная логика приложения осуществляется на сервере приложения. Приложение содержит один сервер, который способен поддерживать множество клиентов и предоставляет множество ресурсов на сервере. Таким образом, приложение включает посредника (север приложений) также известного как промежуточное ПО.

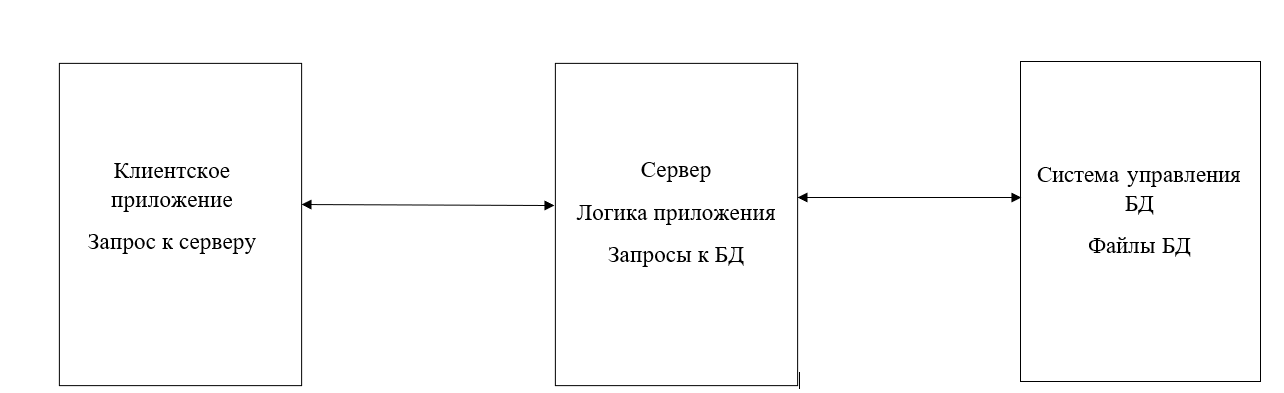


Рисунок 2.3 – Трехуровневая архитектура сервер-клиент

Промежуточное программное обеспечение – это отдельное программное обеспечение, работающие на отдельной машине и выполняющие логику приложения. Промежуточным программным обеспечением является сервер приложения, который выступает в качестве посредника между клиентским ПК и базой данных.

Преимущества использования модели клиент-сервер:

* архитектура разделяет обработку приложения на несколько машин;
* логика приложения выполняется на сервере, что позволяет упростить совместное использование ресурсов от клиента к серверу;
* использование данной архитектуры уменьшает репликацию данных за счёт хранения данных на сервере, а не на каждом клиенте;
* данная архитектура позволяет использовать одни и те же данные для разных представлений.

Недостатки модели клиент-сервер:

* серверы довольно дорогие: серверы, разрабатываемые в соответствие с высокими стандартами, надежны и имеют лучшую производительность;
* количество серверов: в большинстве приложениях с моделью клиент-сервер задействовано несколько серверов, такое приложение довольно сложно настроить;
* требуется квалифицированный опыт: плохо построенная модель клиент-сервер плохо управляется; настройка модели клиент-сервер настолько сложна, что для ее обслуживания требуются квалифицированные специалисты и инженеры по обслуживанию;
* безопасность: клиентская операционная система легко доступна для серверов, и это подвергает клиентскую систему ряду проблем; сообщение, которым обмениваются клиент и сервер создают множество проблем с безопасностью; безопасность включает в себя множество аспектов, некоторые из которых включают: физический ущерб, угрозы и вирусные атаки; для того, чтобы иметь надежные сети, необходимо соблюдать некоторые основные правила.

Современным способом снижения расходов является использование «облачных серверов», которые позволяют использовать виртуальные машины, установленные в облаке провайдера. Использование физических серверов имеет ряд недостатков, описанных выше. Облачные сервера позволяют сократить затраты на обслуживания, обеспечить стабильность работы и безопасность.

## **2.2 Протоколы взаимодействия между клиентом и сервером**

Обмен данными между клиентом и сервером осуществляется посредством стандартизованных протоколов, таких как:

– протокол передачи файлов (*FTP*);

– простой протокол передачи почты (*SMTP*);

– протокол передачи гипертекста (*HTTP*).

Протокол передачи файлов (*FTP*) – используется для передачи файлов между клиентом и сервером. Использует *TCP* в качестве транспортного протокола, два порта: порт данных и порт команд. По умолчанию порт команд – 21, порт данных – 20. *FTP* может работать в двух режимах: пассивном и активном. При передаче данных протокол *FTP* не шифрует их, поэтому все имена и пароли пользователей передаются в открытом виде, что является небезопасным. Использование данного протокола позволяет хранить файлы на сервере. Могут быть сохранены любые файлы, такие как изображения, фильмы, музыка. Для обеспечения безопасности используется протокол передачи файлов поверх безопасного соединения *SSH* [3].

Протокол передачи почты (S*MTP*) – это протокол, который по умолчанию используется для отправки и получение почты. Команды, исходящие от клиента и ответы от сервера происходят в *SMTP* сеансе. *SMTP* – это текстовый протокол, ориентированный на соединение в котором передача данных осуществляется по безопасному каналу. Протокол использует *TCP*-соединение и 25 порт по умолчанию. Почта отправляется серией сообщений запроса и ответа между клиентом и сервером. Посылаемое сообщение состоит из заголовка и тела, которое представляет собой последовательность символов *ASCII* [3].

*HTTP* – протокол обмена веб-сообщениями. Один из широко используемых протоколов приложений в Интернете, использующихся для передачи текста, видео и других информационных ресурсов при работе веб-приложений. *HTTP* – текстовый протокол, не определяет размера заголовка и полезной нагрузки сообщения. Использует *TCP* в качестве транспортного протокола по умолчанию и *TLS* или *SSL* для безопасности. *HTTP* использует универсальный идентификатор ресурса (*URI*) [4].

Для упрощения организации взаимодействия между клиентом и сервером используется архитектурный стиль *REST*, использующий в основе *HTTP*-протокол*.*

*REST* – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. *REST* – согласованный набор ограничений, учитываемых при проектировании распределённых систем, что приводит к повышению производительности и упрощению архитектуры [5].

*REST* предлагает разработчикам ряд архитектурных принципов для проектирования *web*-сервисов, включая обработку и передачу состояния по *HTTP* между клиентской и серверной частями приложения, написанных на различных языках. *REST* архитектура использует *HTTP*-методы в соответствии с определением протокола, устанавливая однозначное соответствие между операциями *create*, *read*, *update*, *delete*. Согласно этому соответствию:

* *POST*-запрос используется для создания ресурса на сервере;
* *GET*-запрос используется для извлечения ресурса;
* *UPDATE*-запрос используется для изменения и обновления состояния ресурса;
* *DELETE*-запрос используется для удаления ресурса.

При проектировании многих *WEB API* методы *HTTP* используются не по назначению. Зачастую в некоторых *WEB API* *GET*-запрос используется некорректно, выполняя разного рода транзакции на сервере, например, для добавления записей в базу данных. В подобных случаях запросы не используются вне соответствуют REST архитектуре.

Для реализации приложений, использующих *REST* архитектуру необходимо соблюдать некоторые ограничения:

* использование модели клиент-сервер;
* отсутствие состояния – информация между запросами не хранится на сервере, сервер получает необходимую информацию от клиента; когда клиент готов перейти в новое состояние, он инициирует отправку запросов;
* кэширование – ответы от сервера, содержащие явное обозначение, могут быть кэшированы на стороне клиента;
* использование унифицированного интерфейса позволяет создавать несколько серверов и устанавливать между ними взаимосвязи;
* возможность использования промежуточных серверов для повышения масштабируемости и разделения нагрузки.

Использование REST архитектуры упрощает организацию взаимодействия между различными частями приложения и имеет следующие преимущества:

* масштабируемость;
* производительность;
* прозрачность системы взаимодействия;
* простота интерфейсов;
* лёгкость внесения изменений;
* способность эволюционировать, приспособление к новым требованиям [5].

## **2.3 Способы предоставления доступа**

Для входа пользователя в приложение и получения доступа к выполнению определённых действий, ему необходимо выполнить ряд взаимосвязанных процедур: идентификацию, аутентификацию и авторизацию.

**Идентификация** – процедура распознавания субъекта. При входе пользователя на первую страницу приложение – страницу регистрации, пользователь предъявляет системе свой идентификатор (логин, имя, почта). Система проверяет его по своей базе данных. Если такой субъект существует, то для системы он считается легальным, все остальные нелегальными.

**Аутентификация** – это процедура проверки подлинности субъекта, которая позволяет достоверно убедиться в том, что субъект, предъявивший свой идентификатор, на самом деле является именно тем субъектом, идентификатор которого он использует. Для этого пользователю необходимо предоставить система информацию, которая может быть известно только ему одному (пароль, секретный вопрос, отпечаток пальцев и т. д.).

**Авторизация** – процедура предоставления субъекту определённых прав доступа к ресурсам системы после прохождения их процедуры аутентификации. Для каждого пользователя определяется ряд прав, которые он может осуществлять при обращении к ресурсам системы [6].

Существуют разные типы аутентификации пользователя:

1. по сертификатам;
2. по паролю;
3. по одноразовым паролям;
4. по ключам доступа;
5. по токенам.

Аутентификация по сертификатам осуществляет посредством набора свойств и подписей, которые выступают в роли посредника между клиентом и сервером, позволяют идентифицировать владельца сертификата и проверить его подлинность. Сертификат имеет криптографический ключ, который на стороне клиента может хранится на отдельном физическом устройстве, в файле, в браузере, в операционной системе компьютера. Сервер проверяет сертификат на подлинность, и он должен обладать тремя основными свойствами:

1. иметь действительную дату;
2. быть подписанным *certification authority*;
3. сертификат не должен быть в списке исключений.

Использование сертификатов для аутентификации пользователя в системе является более надежным способом, чем аутентификация по паролю и зачастую используется в приложениях, в которых требуется обеспечить безопасность данных (интернет-банкинг). Однако трудности с распространением закрытых ключей и поддержка сертификатов делает их малодоступными для широкой аудитории.

Аутентификация по паролю осуществляется с помощью набора символов, которые пользователь определяет в системе при регистрации. Для аутентификации по паролю существует два основных протокола, которые традиционно используются в *web*-приложениях:

1. *HTTP authentication*;
2. *Forms authentication*.

*HTTP* является стандартным протоколом, который активно применяется в работе web-приложений. При первом обращении неавторизированного пользователя к системе, сервер возвращает браузеру статус «*401 Unauthorized*». В ответ на это сообщение браузер отображает пользователю страницу для ввода логина/пароля. После успешной авторизации пользователя в системе и при следующих обращениях клиента к серверу браузер добавляет заголовки «Authorization», в которых передаются данный пользователя для аутентификации.

Существуют несколько схем аутентификации по *HTTP* протоколу, которые отличаются по степени безопасности: *Basic*, *Digest*, *NTLM, Negotiate*.

*Forms authentication* – протокол аутентификации, не имеющей определённого стандарта. Схемы реализации такого протокола являются специфичными для конкретного приложения. Работа осуществляется следующим образом: веб-приложение при входе пользователя в систему отображает форму для ввода логина и пароля. После этого данные отправляются на сервер и при успешной авторизации на сервере для пользователя создается *session token*. *Token* обратно возвращается пользователю и при каждом запросе передаётся на сервер для проверки подлинности пользователя [7].

Описанные выше протоколы являются стандартным способом аутентификации пользователя, однако при разработке нередко применяются нестандартные протоколы и данные в запросе передаются в других местах.

Аутентификация не является безопасным способом, так как недостаточно сложный пароль может быть легко подобран. При непродуманной разработке, приложение может иметь множество уязвимых мест, упрощающих несанкционированный доступ к личным данным пользователя.

Ряд уязвимостей, которые могут привести к взлому приложения:

1. при хранении паролей в базе данных приложение не использует хэш-функции и пароли хранятся в открытом виде;
2. между клиентом и сервером пароли пересылаются в незашифрованном виде по незащищённому соединению;
3. при регистрации пользователя в системе приложение самостоятельно генерирует пароль и отправляет пользователю на почту, а при первом входе пользователя в систему приложение не требует от пользователя сменить пароль;
4. приложение не проверяет создаваемые пользователем пароли на сложность, в результате чего они могут быть подобранны злоумышленниками;
5. при совершении важных операций приложение не требует от пользователя пройти повторную аутентификацию;
6. Web-приложение создаёт долгосрочные токены (день, неделю) и не уничтожает их при долгой неактивности пользователя;
7. приложение создаёт токены, которые легко могут быть воспроизведены.

Таким образом, аутентификация с использованием паролей при правильной организации является относительно безопасным способом для тех приложений, где не требуется обеспечить повышенную безопасность в хранениях данных пользователя.

Аутентификация по одноразовым паролям является частью аутентификации по паролю и используется для организации двухфакторной аутентификации. Пользователю для входа в систему необходимо предоставить два фактора: пароль и ключ, который будет генерировать одноразовой пароль.

Способы генерирования одноразовых паролей:

1. программный или аппаратный – приложение *Google Authenticator* является мобильным приложением для осуществления двухфакторной аутентификации; приложение генерирует пароль, который пользователь должен ввести для входа в приложение; примером аппаратной генерации токена является *RSA SecurID*.
2. случайные коды, которые передаются пользователю на электронную почту или через *SMS*;
3. список заранее известных кодов, которые выдаются пользователю при регистрации – при совершении пользователям действий, которые требуют повышенной безопасности система требует от пользователя ввести пароль под определёнными номером.

Аутентификации по одноразовым паролям имеет уровень безопасности выше, чем аутентификация по паролям.

Аутентификация по ключам доступа в качестве ключа применяет длинные строки, которые являются уникальными и по своей сути заменяют логин и пароль. Ключи генерируются сервером по запросу и хранятся в приложениях. Использование ключей вместо паролей повышает уровень безопасности приложения, так как ключ значительно сложнее подобрать, чем пароль. Если ключ используется для аутентификации по незащищенному соединению, то в приложениях может использоваться две части ключа: публичная и секретная. Секретная часть используется для генерирование секретной подписи, а публичная позволяет аутентифицировать пользователя в системе.

Аутентификация по токенам осуществляется следующим образом: при входе в приложение пользователь вводит один из возможных способов защиты (пароль, ключ, сертификат) и клиенту выдаётся токен, используя который, он получает доступ к приложению. При аутентификации таким способом должны быть выполнены следующие проверки:

* дата и срок действия токена действителен;
* токен достоверен и предназначен конкретному приложению;
* токен не был изменен.

Токены могут иметь разные форматы:

1. *Simple Web Token* – простой токен, использующий пару ключ-значение, зашифрованных в *HTML*-форме; существует несколько ключей, которые являются обязательными для работы токена: *Issuer, Audience, ExpiresOn*; для подписи токена используется симметричный ключ;
2. *JSON Web Token* – состоит из трёх элементов: заголовок, поля, подписи; заголовок и поля передаются в *json*-формате; подписи при генерации используют различные алгоритмы: симметричные и асимметричные;
3. *Security Assertion Markup Language* – включает эмитенты, субъекты, подписи и передаются в *XML*-формате; подписи таких токенов генерируются при помощи ассиметричной криптографии [7].

## **2.4 Распределение функций между клиентом и сервером**

Сервер представляет собой приложение, выполняющее расчётные операции и различные процессы в ответ на запросы клиента. Сервер отвечает за логику работы приложения, обрабатывает запросы, получаемые от клиента, выполняет необходимые операции и отсылает ответ клиенту. Для хранения информации используется база даных. Сервер формирует запросы к базе данных, позволяя клиенту добавлять новую информацию, редактировать и получать доступ к существующей информации.

Клиентская часть веб-приложения представляет пользовательский интерфейс приложения и может рассматриваться как набор поведений, выполняемых пользователем. Для описания структуры и графического интерфейса веб-страницы используется «язык гипертекстовой разметки» – *HTML*. Для художественного оформления страницы веб-приложения используется таблица стилей *CSS* или метаязык на основе *CSS* – *SCSS*. Реализация основных функций пользовательского интерфейса с доступом к объектам приложения, управления встроенными страницами осуществляется с помощью языка *JavaScript.* Язык *JavaScript* предоставляет средства для динамического взаимодействия с пользователями. При использовании данного языка могут возникнуть некоторые проблемы, обусловленные различиями в способах реализации языка, выбранные разработчиками браузеров [8].

Веб-страницы могут быть разделены на статические и динамические. В зависимости от предлагаемого контента и сферы использования в разрабатываемых приложениях применяются те или иные страницы. Первое поколение веб-приложений состояло из набора статических веб-страниц, содержащих информацию о компании и продукте, рекламные материалы. Такие веб-страницы со временем стали заменяться динамическими с функциями, предназначенными для доступа к данным интерактивно, выполняя транзакции в режиме онлайн и обмениваясь информацией. Преимущества использования динамических веб-приложений заключается в том, что данные и презентацию можно разделить. Это оказывает положительный эффект на производительность и качество всего приложения [9].

Разрабатываемое приложение «Личный кабинет» является динамическим веб-приложением. Страницы в приложении будут создаваться динамически, используя фиксированный шаблон для разных типов данных, что позволит производить автоматическое обновления данных не перерисовывая страницу полностью.

В web-приложениях пользователь работает с так называемым «*фронтендом*» (от *Front End*), который реагирует на действия. После этого данные отправляются в так называемый «*бэкенд*» (от *Back End*), где они непосредственно обрабатываются. В контексте модели клиент-сервер «*фронтенд*» – это клиент, «*бэкенд*» – сервер.

Написать клиентскую часть приложения можно с использованием языка программирования *JavaScript* и языка разметки *HTML*, но с повышением сложности приложений, повышается трудоемкость разработки. Проектировать, разрабатывать и развёртывать приложения становится сложнее с использованием только языка программирования.

Для разработки в таких случаях предпочтительнее использовать так называемый «*фреймворк*» (от *Framework*), который предоставляет готовые компоненты, модели и шаблоны, позволяющие исключить дублирование кода и ускорить процесс. Исходя из этого, для разработки клиентской части приложения «Личный кабинет» использовался фреймворк.

# **ГЛАВА 3**

# **СРЕДСТВА РАЗРАБОТКИ**

## **3.1 Понятие фреймворка**

Как было сказано выше, клиентскую часть приложения «Личный кабинет» целесообразно разрабатывать с использованием фреймворка.

Фреймворк представляет собой специальную программную оболочку или совокупность компонентов и моделей, предоставляющих разработчику возможность использования шаблонов, характерных для данного языка, что позволяет упростить и ускорить процесс разработки. Фреймворк включает шаблоны для представления информации приложения, язык программирования для создания интерактивного доступа к данным и интерфейс для доступа к ресурсам приложения. Для разработчика использование фреймворка – это хорошая практика при реализации программы на выбранном языке программирования и использование уже существующих решений [10].

Практически для каждого языка программирования существуют фреймворки со своими плюсами и минусами. Фреймворки разрабатываются и тестируются целыми веб-сообществами, имеют хорошую документацию и техническую поддержку со стороны сообществ. Сообщества пользователей и разработчиков, возникающих вокруг определенного фреймворка, обмениваются знаниями и идеями, что позволяет упростить процесс разработки. Фреймворк избавляет разработчика от необходимости писать много повторяющегося кода, который будет использоваться в различных местах. Это позволяет повысить эффективность разработки. Большинство библиотек, которые используются фреймворками, являются бесплатными и имеют техническую поддержку со стороны сообществ, это позволяет разработчику быстрее кодировать и уменьшить расходы для конечного пользователя [10].

Каждый фреймворк имеет определённую структуру, и при решении конкретной задачи будет использоваться один из них. Сравнение возможностей каждого из фреймворка, выявление преимуществ и недостатков позволяет разработчику принять грамотное решение при выборе. Для этого необходимо сравнить фреймворки по 3 категориям: популярность, обучаемость и производительность.

## **3.2 Сравнение фреймворков по популярности**

Фреймворки клиентской части используются для создания одностраничных приложений и отвечают за пользовательский интерфейс. Наиболее популярные и удобные для использования и создания крупных приложений из них следующие: Angular, React, Vue.js.

Фреймворк *Angular* – открытая платформа, разработанная компании Google в 2016 году. В качестве основного языка разработки фреймворк использует *TypeScript*. *Angular* используется для создания динамических одностраничных веб-приложений. Фреймворк предоставляет большие возможности для создания как уникальных, так и универсальных приложений, которые работают в Интернете, на мобильных устройствах или на компьютере. Это многоцелевой фреймворк, но особенно ярко его возможности проявляются при реализации приложений типа *CRUD*[11].

*React* – библиотека *JavaScript* для создания пользовательских интерфейсов. В настоящие время React разрабатывается *Facebook* и сообществом отдельных пользователей. Библиотека предназначена для улучшения разработки пользовательского интерфейса, упрощая обновления данных путем разделения приложения на более мелкие компоненты, которые объединяются для создания сложных пользовательских интерфейсов [12].

Фреймворк *Vue.js* – *JavaScript* фреймворк, разработанный в 2014 году Эваном Ю. Используется для создания одностраничных приложений. Фреймворк можно использовать как для создания отдельных страниц, так и в качестве основы для разработки полноценного приложения. Основная идея фреймворка – привязка данных и компонентов. *Vue.js* уделяет внимание пользовательскому опыту, что упрощает разработку [13].

Популярность каждого из фреймворка можно оценить путём сбора и анализа данных из основных платформ, связанных с разработкой программного обеспечения, таких как *GitHub, NPM Trends, Stack Overflow*.

*GitHub* – платформа для разработки программного обеспечения, которую используют более 65 миллионов разработчиков для размещения бизнес проектов и проектов с открытым исходным кода. Размещение кода, написанного разными разработчиками для множества целей, делает *GitHub* уникальным и надежным источником программного обеспечения. В таблице 3.1 представлена информация о репозиториях *Angular. React, Vue* на *GitHub*.

Таблица 3.1 – Популярность фреймворков по данным *GitHub*

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Angular* | *React* | *Vue* |
| *GitHub Stars* | 73300 | 169000 | 184000 |
| *GitHub Fork* | 19200 | 29100 | 34000 |
| *Repositories* | 1630131 | 5294660 | 1828481 |
| *GitHub Issue* | 2300 | 532 | 332 |

*NPM* – крупнейший в мире диспетчер пакетов программного обеспечения, который позволяет разработчикам делится или заимствовать пакеты с открытым исходным кодом, написанным на языке *JavaScript*. *NPM* *Trends* – веб-сайт, предоставляющий информацию о количестве пакетов, загруженных через *NPM* за всё время.

*StackOverflow* – это онлайн платформа, где разработчики могут задавать вопросы о программировании. На платформе разработчики и технические энтузиасты обсуждают различные языки программирования, что делает платформу идеальным местом для понимания того, что использует глобальное сообщество разработчиков.

На рисунках 3.1 и 3.2 показаны графики популярности Angular, React и Vue.js по данным NPM Trends и StackOverflows соответственно.

Рисунок 3.1 – Популярность фреймворков по данным *NPM Trends*

Рисунок 3.2 – Популярность фреймворков по данным *StackOverflow*

Таким образом, анализируя выше сказанное, можно сделать вывод, что популярность *React* выше, чем у *Angular и Vue*. Однако сделать выбор в пользу того или иного фреймворка только по одному этому параметру является неправильным.

## **3.3 Сравнение фреймворков по обучаемости**

Для исследования параметра обучаемости бралась информация из документации фреймворков. Так как это *JavaScript* фреймворки, и они довольно схожи по структуре и рабочим процессам, то информацию по каждому из них можно оценивать по 5 пунктам: синтаксис, архитектура, данные управления, жизненный цикл и сторонние библиотеки.

*Angular* имеет обширную документацию и учебные пособия. Для разработки фреймворк использует *TypeScript* и *HTML*. *TypeScript* – язык программирования, разработанный и поддерживаемый *Microsoft*, который в отличии от простого *JavaScript* добавляет возможности статической типизации.

Приложение, написанное на *Angular*, состоит из модулей, которые являются контейнером для кода. Модули могут содержать компоненты, сервисы, провайдеры. Модули могут быть экспортированы или импортированы из одной части кода в другую. Каждое приложение имеет корневой модуль, с которого начинается загрузка приложения. *Angular* частично использует концепцию *Model View Controller*, имея компоненты в качестве контроллера и шаблоны в качестве представления [11].

Фреймворк поддерживает двухстороннюю привязку, что позволяет динамически изменять данные в шаблоне при их изменении в компоненте.

Каждый компонент имеет жизненный цикл. *Angular* создаёт, отображает и обновляет компоненты при изменении данных. Фреймворк предоставляет возможность пользователям взаимодействовать с кодом на различных этапах жизненного цикла.

Сторонние библиотеки расширяют возможности разработчика, добавляя готовый стили и функции [11].

*React* имеет официальную документацию с несколькими пошаговыми руководствами на странице фреймворка. В разработке использует синтаксис *JSX* – расширение *JavaScript*, представляющее смесь *JavaScript* и *HTML* и напоминает язык шаблонов. *JSX* используется для создания элементов, составляющих компоненты. При изменении элементов реагирует виртуальный *DOM*, который заботиться об обновление реального *DOM*. Виртуальный *DOM* – это концепция, в которой «виртуальные» представления пользовательского интерфейса хранятся в памяти, а «настоящий» *DOM* синхронизируется через библиотеку *React DOM*, что позволяет производить обновления динамически.

Приложение *React* состоит из компонентов. Компоненты могут быть использованы повторно. Каждый компонент может иметь состояние. Состояние является локальными инкапсулированным. Состояния могут изменяться, при этом компоненты будут обрисовываться заново.

*React* имеет встроенные функции жизненного цикла, обеспечивающие контроль над событиями такими как создание, обновление элементов. Объектная модель виртуального документа хранится в кэш. Изменения в данных передаются сверху вниз, затем *React* сравнивает виртуальный *DOM* и *React DOM* и, при необходимости, вносит изменения. В официальной документации нет полного объяснения жизненного цикла.

Сторонние библиотеки и пакеты могут быть установлены через *NPM*. Наиболее часто используемая сторонняя библиотека для *React* – *Redux*. Библиотека представляет собой контейнер состояния для приложений JavaScript. В *Redux* всё состояние хранится в дереве объектов [12].

*Vue.js* имеет подробную документацию и примеры на совей официальной странице. Основные концепции подробно объяснены и проиллюстрированы примерами.

*Vue.js* использует синтаксис шаблона на основе *HTML*, который привязывается к базовому экземпляру *JavaScript*. Приложение *Vue* состоит из повторно используемых компонентов, называемых экземплярами. Приложение представляет дерево вложенных компонентов.

Данные приложения хранятся в функции данных компонентов, которые аналогичны состояниям компонентов. Данные могут передаваться из одного компонента в другой.

Подобно *React* *Vue* использует виртуальный *DOM* для обновления страницы. Каждый из компонентов проходит несколько шагов инициализации, где каждый шаг соответствует своему этапу в жизненном цикле, что позволяет пользователям работать с кодом на определённых этапах жизненного цикла.

Сторонние компоненты могут быть установлены через *NPM* или импортированы как зависимости. Одна из наиболее широко используемых сторонних библиотек *VueX* – библиотека управления состоянием [13].

## **3.4 Сравнение фреймворков по производительности**

Для оценки производительности каждого из фреймворков были написаны небольшие тестовые приложения на *Angular, React и Vue.js*, имеющие одинаковую логику и структуру. Приложения имеет меню, таблицу с данными (рисунок 3.3) и осуществлять следующие функции: создание 1000 строк, добавление 1000 строк, создание 10000 строк, добавление 10000 строк и удаление всего.

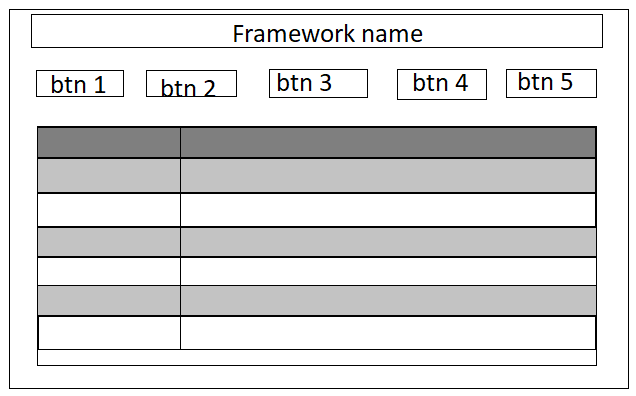


Рисунок 3.3 – Интерфейс тестового приложения для оценки производительности

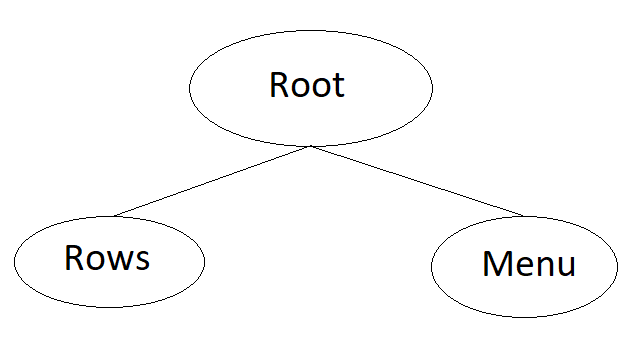
**Приложение разделено на три отдельных компонента (рисунок 3.4).

Рисунок 3.4 – Взаимосвязь компонентов в тестовом приложении

Скорость работы приложения была протестирована и измерена в браузере *Google Chrome*, работающем локально в операционной системе Windows, установленной на ноутбуке *HP* 15-da0236ur 4PS99EA (процессор 8-го поколения *Intel Core* 1,8 ГГц, оперативная память 8 ГБ, HDD 1000 Гб).

Было проведено 3 теста, по результатам которых были рассчитаны средние значения. Результаты эксперимента представлены в таблицах 3.2 – 3.4.

Таблица 3.2 – Производительность *Angular*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Angular* | t1, мс | t2, мс | t3, мс | <t>,мс |
| загрузка страницы | 779 | 1100 | 1110 | 996 |
| создание 1000 строк | 261,8 | 253,4 | 239,6 | 252 |
| добавление 1000 строк | 280,3 | 327,8 | 346,3 | 318 |
| создание 10000 строк | 1300,6 | 1587,3 | 1139,6 | 1342 |
| добавление 10000 строк | 1478,3 | 1474,2 | 1381,8 | 1445 |
| удаление | 2221,8 | 1830,2 | 1495,9 | 1849 |

Таблица 3.3 – Производительно *React*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *React* | t1, мс | t2, мс | t3, мс | <t>, мс |
| загрузка страницы | 554 | 529 | 531 | 538 |
| создание 1000 строк | 346,2 | 332,9 | 320,0 | 333 |
| добавление 1000 строк | 414,6 | 356,5 | 347,3 | 373 |
| создание 10000 строк | 1909,3 | 1847,9 | 2015,1 | 1924 |
| добавление 10000 строк | 1478,3 | 1474,2 | 1381,8 | 1444 |
| удаление | 512,6 | 570,9 | 437,9 | 507 |

Таблица 3.4 – Производительно *Vue.js*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Vue.js* | t1, мс | t2, мс | t3, мс | <t>, мс |
| загрузка страницы | 657 | 570 | 581 | 603 |
| создание 1000 строк | 387.3 | 359,7 | 361,4 | 370 |
| добавление 1000 строк | 480,2 | 468,7 | 414,5 | 455 |
| создание 10000 строк | 1365,2 | 1070 | 1285,1 | 1240 |
| добавление 10000 строк | 1478,3 | 1474,2 | 1381,8 | 1445 |
| удаление | 2215,8 | 2137,4 | 2231,4 | 2197 |

## **3.5 Выбор фреймворка для реализации проекта**

Следует отметить, что сравнение программных средств и выбор из них лучших затруднено многими фактами. Так как каждое программное средство имеет широкий спектр использования, то одно из них лучше в одной сфере, другое – в другой.

Для сравнения популярности были проанализированы данные с ведущих платформ, которые активно используются разработчиками. Согласно этим результатам, самым популярным фреймворком является *React*.

Для анализа сложности обучения фреймворки были сравнены по 5 пунктам: синтаксис, архитектура, управления данными, жизненный цикл и использование сторонних библиотек.

С точки зрения синтаксиса, *Angular* является самым сложным для обучения, так как он использует не просто *JavaScript*, а типизированный язык *TypeScript*. *React* и *Vue.js* по синтаксису очень схожи.

Три фреймворка имеют схожую архитектуру. Каждое приложение может быть представлено в виде дерева компонентов. В *Angular* приложениях компоненты разделены на модули, которые содержат вспомогательные файлы кода.

Данные в сравниваемых фреймворках передаются от компонента к компоненту. *Angular* отличается от *React* и *Vue.js* наличием двусторонней привязки.

Фреймворки имеют одинаковый жизненный цикл. Самым простым для изучения жизненного цикла является *Vue.js*, так как техническая документация представляет полное описание каждого из циклов. Вторым самым простым в освоении является *Angular*. Самым трудным для изучения жизненного цикла является *React*.

Для загрузки сторонних библиотек используется *NPM*. Установка библиотек позволяет добавлять некоторые жизненно важные функции для *React* и *Vue.js*, которые по умолчанию существуют в *Angular*.

Изучение нового фреймворка является субъективным для каждого человека и не может быть выражено количественно. Vue имеет самый простой синтаксис. С точки зрения архитектуры *React* и *Vue.js* проще чем *Angular*. Самым большим и трудным для изучения из трёх фреймворков является *Angular.*

Для сравнения производительности на фреймворках было написано простое приложение. Затем приложение было протестировано на скорость загрузки результат представлен в таблице 3.5.

Таблица 3.5 – Сравнение производительности фреймворков

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Angular* | *React* | *Vue.js* |
| загрузка страницы | 996 | 538 | 603 |
| создание 1000 строк | 252 | 333 | 370 |
| добавление 1000 строк | 318 | 373 | 450 |
| создание 10000 строк | 1342 | 1924 | 1240 |
| добавление 10000 строк | 1445 | 1444 | 1445 |
| удаление | 1849 | 507 | 2197 |

По скорости загрузки страницы *React* был самым быстродействующим. По скорости выполнения функции и отображения результата самым быстрым был *Angular*. Стоит отметить, что производительность фреймворков по скорости может изменяться в зависимости от размера и сложности приложения.

Таким образом, *React* является самым популярным и простым для изучения фреймворком. *Angular* – самый сложный в освоении и медленный по загрузки фреймворк, но недостатки компенсируются удобством использования и преимуществами, которые предоставляет *Angular* разработчикам в создании крупномасштабных приложений. Исходя из этого для создания клиентской части приложения «Личный кабинет» было принять решение использовать фреймворк *Angular*.

## **3.6 Общие сведения о фреймворке *Angular***

*Angular* нацелен на создание одностраничных приложений (*single page application).* Преимущество использования одностраничных приложений состоит в том, что для обеспечения работы весь необходимый код (*HTML, CSS, JavaScript*) загружается вместе с загрузкой самой страницы, либо нужные ресурсы подключаются динамически и добавляются к странице по необходимости, обычно в ответ на действия пользователя. Страница не перезагружается во время работы [14].

Angular, являясь полноценной интерфейсной средой, имеет собственный набор библиотек для построения сложных форм. Последняя версия Angular предоставляет две стратегии построения форм:

* управляемые шаблоном формы;
* модельные или реактивные формы.

Обе технологии принадлежат библиотеке @*angular/forms* и основаны на одних и тех же классах управления формой. Тем не менее, они заметно отличаются по своей философии, стилю программирования и технике.

Реактивные формы предполагают создание модели формы в классе компоненты, связывание этой модели с *DOM* элементами в шаблоне компоненты, используя ряд доступных директив. Вся логика работы с формой содержится в компоненте, а не в шаблоне. Обновление данных происходит синхронно и контролируется программистом. Главная идея состоит в том, что элементы формы и сами формы обеспечивают *observable-based API*. Реактивные формы и их отдельные элементы могут рассматриваться как постоянный источник данных на изменение, на который можно подписываться и обрабатывать, используя определённый набор методов [11].

При разработке больших и сложных приложений возникает необходимость в хранении состояния и реагирование на его изменение. Для приложений, разработанных с использованием *Angular*-фреймворка, существует контейнер состояний на основе *RxJS*, использующий *Redux*, находящейся в библиотеке *@ngrx/store*.

*Redux* – контейнер состояний (*state*) для *JavaScript* приложений. Выполнение последовательность действий для модификации состояний является хорошим подходом при проектировании сложных веб-приложений.

Основные постулаты *Redux*:

1. одно хранилище для всего состояния приложения;
2. состояние доступно только для чтения;
3. изменения делаются «чистыми» функциями, к которым предъявляются следующие требования:
4. функции не должны делать внешних вызовов по сети или базе данных;
5. функции не должны менять аргументы;
6. вызов чистой функции с теми же аргументами всегда возвращает одинаковый результат.

Основная идея заключается в том, чтобы хранить каждый фрагмент данных в централизованном контейнере и изменять его только с помощью набора определённых действий. *Redux* – система передачи сообщений, которая встречается в объектно-ориентированном программировании, но она не встроена непосредственно в язык, а реализована с помощью библиотеки. Подобно ООП, *Redux* переводит контроль от вызывающего объекта к получателю – интерфейс не управляет состоянием напрямую, а передаёт ему сообщение для обработки [15].

Преимущества такого подхода:

* благодаря инверсии контроля, исчезает необходимость обновлять пользовательский интерфейс, как только меняется состояние; добавлять функции отмены или возвращение назад становится проще;
* в *Redux* есть дерево состояний, независимое от пользовательского интерфейса; при обработке состояний вне интерфейса легче поддерживать постоянство, ведь сериализация в *localStorage* или *URL* проводится в единственном месте;
* *Reducers* дают огромную свободу в работе с действиями, которые можно сочетать, отправлять, обрабатывать одновременно.

Любое приложение *Angular* должно иметь возможность обмениваться данными с внешним миром, например, вносить изменения не сервер, обычно включая асинхронную работу, такую как выполнение запросов *AJAX*. Для реализации этих целей используются «побочные эффекты» (рисунок 3.5).

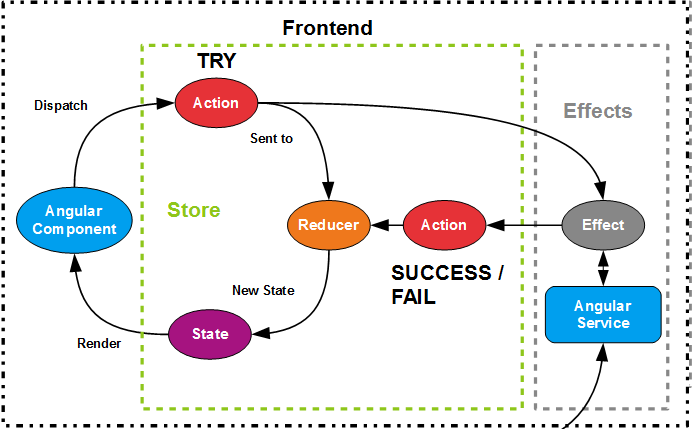


Рисунок 3.5 – *Redux* в действии

Промежуточное *ПО Redux* позволяет перехватывать отправленные действия и добавлять вокруг них дополнительное сложное поведения. Когда пользователь совершает действие, эффект перехватывает это действие и отправляет запрос на сервер. После ответа с сервера совершается новое действие, которое сигнализирует приложение об успешном или неуспешном ответе. После этого изменяется состояние приложения [15].

## **3.7 Тестирование web-приложений**

Тестирование программы – процесс анализы программного средства с целью выявления дефектов и повышения качества продуктов [16].

Функционирование программы в соответствии с поставленными задачами определяет надёжность продукта. На начальных этапах разработки программного обеспечения задаются условия работы и задачи, которые должны выполняться в процессе работы программы. Для тестирования проекта первоначально необходимо определиться с требованиями, которым должен соответствовать проект. Условно, требования можно подразделить на следующие требования: пользовательские, функциональные, нефункциональные, бизнес-логики и качества. Например, задачи, которые пользователь может выполнять с использованием программы, относятся к пользовательским требованиям. Определяя пользовательский сценарий работы, можно построить тесты соответствующим образом. Поведение системы описываются функциональными требованиями. При проектировании программы требования влияют на дизайн системы [16].

В зависимости от структуры и сложности проекта, чем позже будет обнаружена проблема, тем сложнее и дороже может оказаться её решение. Качества и стоимость программного обеспечения зависит от проверочных испытаний. Тестирование программы осуществляется на разных этапах разработки. Небольшие «куски кода» тестируются непосредственно программистом, часть тестирования проводится специалистами–тестировщиками. Готовые программы отдаются клиентам, которые, в свою очередь, в ходе работы могут обнаружить ошибки, которые не были исправлены на первых двух этапах.

Виды тестирования:

1. модульное тестирование – осуществляется разработчиками, реализующими модули; перед интегрированием модулей в систему каждый из них должен быть протестирован; примером тестирования модулей является проверка корректности работы функций;
2. интеграционные тесты – проверяют корректность совместной работы между модулями;
3. системные тесты – проверяют работу системы как единого целого; при таком типе тестирования рассматриваются части системы, непосредственно доступные пользователю без углубления в код; тестирование может осуществлять как клиент, так и разработчик;
4. регрессионное тестирование – используется после внесений изменений в систему для любых типов программного обеспечения; цель тестирования – убедится в том, что внесённые изменения не порождают ошибок системы; первоначально запускают уже существующие тесты для того, чтобы убедится в том, что функционал системы сохраняется; затем тестируют измененные модули и функциональность системы в целом;
5. нагрузочные тесты – используют для проверки соответствия системы технически требованиям, после чего проверяют производительность; цель таких тестов – найти места, в которых может произойти перезагрузка системы и оптимизировать их [17].

Тестирование в Angular приложениях включает два типа тестов: модульные тесты (*unit tests*) и системные тесты (*e2e tests*).

Модульные тесты используются для проверки одного блока исходного кода: функции, методы, объекты, типы, значения и др.

Системные тесты *e2e* используются для проверки функциональности приложения, имитируя поведение конечного пользователя. Данный вид тестов позволяет проверить правильно ли отображает страница определённые элементы при загрузке, например, кнопки или текст.

Каждый из типов тестов имеет свои преимущества и недостатки.

* скорость – модульные тесты работают с небольшими фрагментами кода, они могут выполнятся быстро; тесты *e2e* основаны на тестировании через браузер, поэтому они медленнее;
* надёжность – тесты *e2e* включают больше зависимостей и сложных взаимодействий, могут быть нестабильными, что приводит к ложным срабатываниям; запуск модульных тестов редко приводит к ложным срабатываниям; если модульный тест не пройден – проблема в коде;
* помощь в обеспечении качества кода – одним из основных преимуществ написания тестов является то, что они помогают обеспечить качества кода; написание модульных тестов может помочь выявить сложный код, который трудно проверить; *E2E* тесты не улучшают качества кода, так как они тестируют код не напрямую, а со стороны браузера;
* экономическая эффективность – написание *e2e* тестов занимает больше времени, так как они основаны на сложных взаимодействиях, следовательно затраты на разработку выше;
* имитация взаимодействия с пользователем – модульные тесты могут моделировать взаимодействие с пользователем, но написание *E2E* тестов в этом случае будет проще, потому что для этой цели они и созданы.

Оба типа тестов важны для проверки функциональности приложения. Большая часть кода может быть проверена модульными тестами, но ключевые функциональные возможности должны быть протестированы с помощью *e2e* тестов.

В настоящие время *Jasmine* и *Karma* – это основные инструменты, используемые для тестирования базового кода *Angular*. Фреймворк *Jasmine* имеет множество функциональных возможностей для написания различных видов тестов.

*Karma* – инструмент, позволяющий создавать браузеры и запускать внутри них тесты *Jasmine* из командной строки. Результаты тестов также отображаются в командной строке. Karma следит за изменениями в файлах разработки и автоматически перезапускает тесты. Тесты *Jasmine* запускаются как часть цепочки инструментов разработки, которая требует запуска тестов и проверки через командную строку.

## **3.8 Средства развертывания**

После разработки, web-приложение необходимо развернуть и запустить. Запуск программы происходит на физическом сервере и для этого необходимо выполнить следующую последовательность шагов:

* собрать программы из исходников;
* установить все необходимые зависимости;
* скомпилировать программу;
* запустить программу.

Одним из способов упрощения этого процесса является развертывание в контейнерах, которые содержат исполняемый код, среды выполнения и зависимости от операционной системы, необходимые для запуска приложения. Одним из способов контейнеризации является использование набора инструментов Docker.

*Docker* – технология контейнеризации, разработанная для упрощения создания, развертывания и запуска приложения. *Docker* создаёт автономные пакеты, называемые контейнерами. Контейнеры позволяют разработчику упаковывать приложение со всеми необходимыми ему частями, например, библиотеки и другие зависимости, и отправлять всё это как один пакет. Использование контейнеров гарантирует работу кода на любом компьютере вне зависимости от пользовательских настроек. *Docker* позволяет использовать разработанную программу, как часть другой программы, обеспечивает гибкость приложения и потенциально уменьшает количество систем, позволяя разработчику создавать, развёртывать, копировать и перемещать контейнеры из среду в среду, что помогает оптимизировать приложение для облака [18].

*Docker* схож с виртуальной машиной. Отличие состоит в разделении или выделении ресурсов. *Docker* использует многоуровневую файловую систему, которая позволяет контейнерам совместно использовать общие части. Вместо создания виртуальной операционной системы, приложения используют то же ядро *Linux*, что и у системы, на которой они запущены, и требует запуска только того, что не запущено на главном компьютере. Это значительно повышает производительно и уменьшает размер приложения.

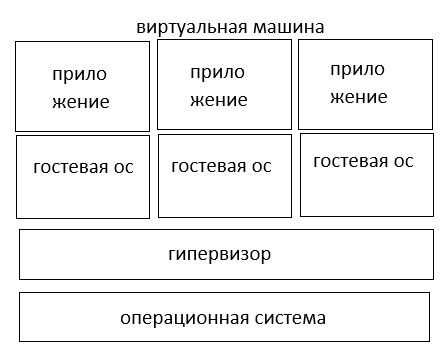
Виртуальные машины используют гипервизор для эмуляции аппаратного обеспечения, которое позволяет нескольким операционным системам работать бок о бок (рисунок 3.6).

Рисунок 3.6 – Виртуальная машина

Это не так просто, как при использовании контейнеров. Когда ресурсы ограничены, то приложения должны быть лёгкими. Контейнеры изначально работают в операционной системе, разделяя ее между всеми контейнерами, что позволяет приложениям и серверам оставаться легковесными и быстро работать параллельно.

Контейнер – набор одного или несколько процессов, изолированных от остальной части системы. Контейнер, в котором находится приложение, содержит необходимые библиотеки, зависимости и файлы (рисунок 3.7).

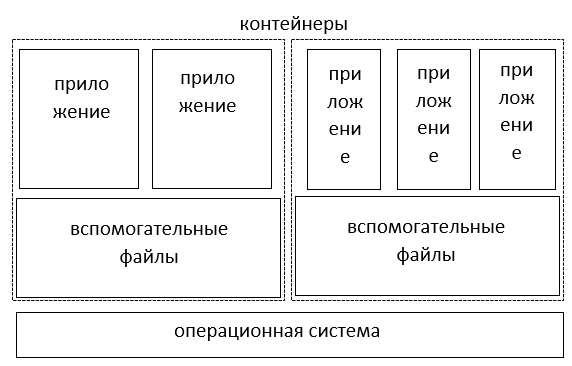


Рисунок 3.7 – Контейнеры *Docker*

Таким образом, все файлы, необходимые для запуска, предоставляются из отдельного образа, что означает, что контейнер является переносимым и согласованным при переходе от разработки к тестированию и, наконец, к производству. При создании приложения разработчик имеет среду разработки с определёнными конфигурациями. Разрабатываемое приложение опирается на эту конфигурацию и зависит от конкретных библиотек, зависимостей и файлов. Перед разработчиком стоит задач максимально эмулировать эти среды локально, но без дополнительных затрат на воссоздание серверных сред. Решением данной задачи является использование контейнеров [19].

Образы контейнеров обеспечивают портативность и контроль версий, помогая гарантировать, что то, что работает на компьютере разработчика, также работает у клиента (например, в производственном процессе).

По сравнению с виртуальными машинами, работающий контейнер менее ресурсоемок, имеет стандартный интерфейс (запуск, остановка, переменные среды и т. д.), сохраняет изоляцию приложения и более прост в управлении как часть более крупного приложения (несколько контейнеров).

Преимущество контейнеров *Docker*:

* модульность – подход *Docker* к контейнеризации ориентирован на способности удалять часть приложения, обновлять или восстанавливать его без ненужного удаления всего приложения;
* слои и контроль версий изображений – каждый файл изображения *Docker* состоит из серии слоёв, эти слои объединены в одно изображение, слой создаётся при изменении изображения; каждый раз, когда указывается команда, такая как запуск или копирование, создаётся новы слой; *Docker* повторно использует эти слои для новых сборок контейнеров, что значительно ускоряет процесс сборки; промежуточные изменения распределяются между изображениями, дополнительно улучшая скорость, размер и эффективность; каждый раз, когда появляются новые изменения, имеется встроенный журнал изменений – полный контроль над изображениями контейнера;
* отмена – это способность откатываться назад; каждое изображение имеет слои; имеется возможность откатиться на предыдущую, если текущая не устраивает; это поддерживает гибкий подход к разработке и помогает сделать непрерывную интеграцию и развертывание;
* быстрое развертывание – подготовка нового оборудования, его запуск и доступность занимают несколько дней; контейнеры на основе Docker помогают сократить время развертывания до секунд; создав контейнер для каждого процесса, можно быстро поделиться схожими процессами с новыми приложениями; поскольку ОС не требует загрузки для добавления и перемещения контейнера, время развертывания существенно сокращается.

Таким образом, технология контейнеризации *Docker* – это более детальный, управляемый, основанный на микросервисах подход, который существенно ускоряет процесс сборки и развёртывания приложения [20].

# **ГЛАВА 4**

# **РАЗРАБОТКА ПРИЛОЖЕНИЯ «ЛИЧНЫЙ КАБИНЕТ»**

В рамках настоящей работы, с использованием фреймворка Angular было разработано приложение «Личный кабинет», которое предоставляет клиентам компании, реализующей услуги мониторинга транспорта,возможность с легкостью управлять услугами.

**4.1 Инфраструктура, обеспечивающая функционирование системы**

Для разработки приложение «Личный кабинет» использовалась трехуровневая архитектура клиент-сервер, которая включает следующие части (рисунок 4.1):



Рисунок 4.1 – Архитектура разработанного приложения «Личный кабинет»

* сервер базы данных – на сервере размещаются все данные о пользователях системы и данные, приходящие от мобильных навигационных устройств; для управления данными используется реляционная система *MySQL*;
* сервер приложения – позволяет соединить клиентскую часть и базу данных; на сервере реализуется логика работы и осуществляются функции для управления данными; сервер приложения был разработан с использованием фреймворка *ASP.NET CORE*;
* сервер представления – пользовательский веб-интерфейс в браузере, реализованный с использованием фреймворка *Angular*; сервер представления взаимодействуют с сервером приложения через *HTTP*-запросы.

## **4.2 Структура базы данных**

Разработанное приложение «Личный кабинет» включает список пользователей. За каждым пользователем могут быть закреплены транспортные средства. На транспортном средстве могут быть установлены различные устройства, например, датчики, системы мониторинга. При добавлении транспортного средства в систему пользователь указывает дату. Приложение имеет функцию разбиения транспортных средств на группы и архивации тех, которые сейчас не используются. При закреплении системы мониторинга за определённым пользователем и транспортным средством, необходимо указать дату установки и снятия. Каждый пользователь может добавлять подразделения.

Для реализации вышеприведенной структуры в базе данных создаются таблицы (рисунок 4.2), которые условно можно разделить на таблицы, содержащие данные, и таблицы, содержащие связи между данными.

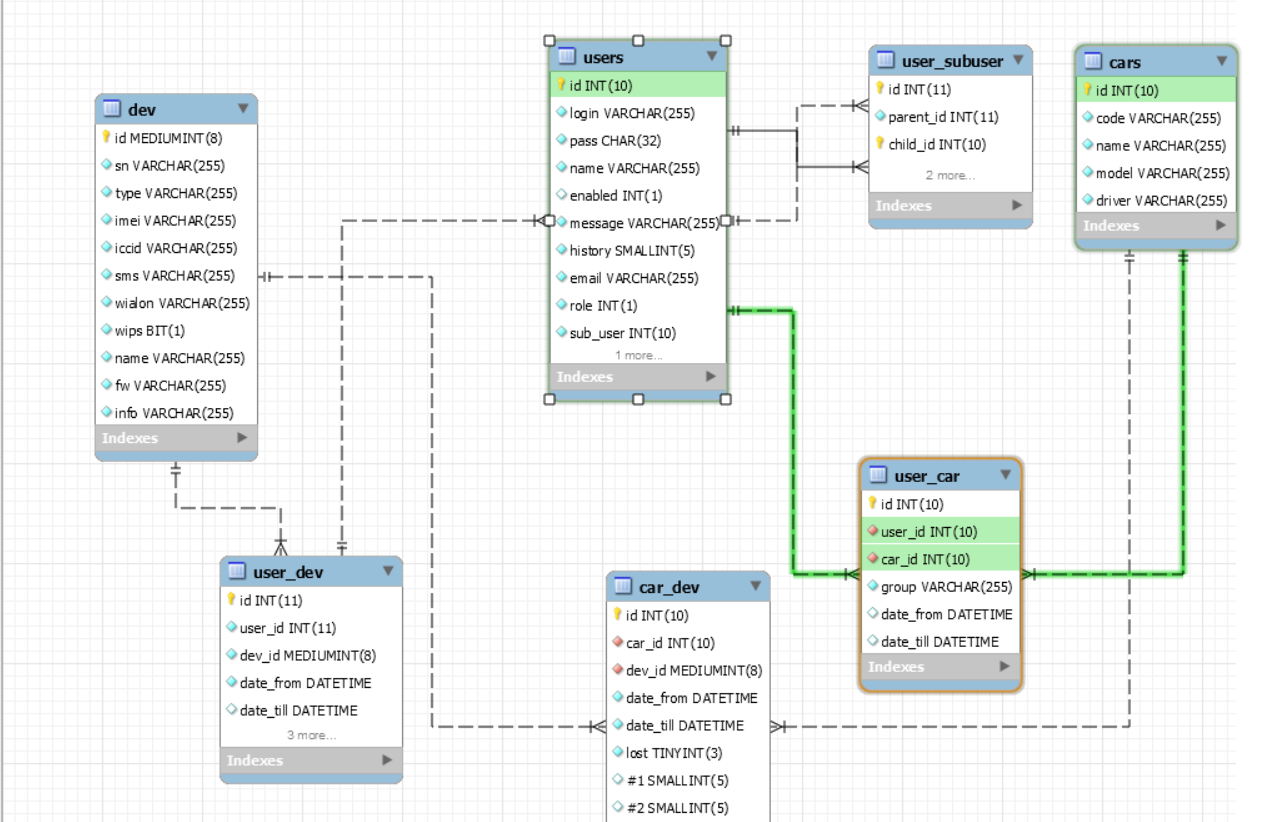


Рисунок 4.2 – Структура базы данных

Данные:

* таблица пользователей (организаций) – *users*;
* справочник машин организаций – *cars*;
* справочник навигационных модулей – *devs*;

Связи:

* Пользователь – машина – *user\_car*;
* Пользователь – навигационный модуль – user\_dev;
* Машина – навигационный модуль – *car\_dev*;
* Пользователь – подпользователь – user\_subuser;

**4.3 Структура приложения**

Приложения, написанные с использованием фреймворка *Angular*, имеют модульную структуру. Модуль – самостоятельная и независимая единица. В разработанном приложении «Личный кабинет» условно можно выделить следующие модули:

* + - 1. *login*;
      2. *common*;
      3. *objects*;
      4. *tasks*.

Модуль *login* отвечает за авторизацию пользователя. Содержит сервис, в котором реализованы функции отправки данных пользователя на сервер и получение токена, чтобы пользователю не пришлось авторизоваться в системе каждый раз при выполнении операций.

*Common* модуль отвечает за компоненты, которые могут повторяться в различных частях приложения, к ним относятся меню, таблицы, фильтры, визуализация истории, функции, которые могут использоваться во всех частях приложения.

*Objects* модуль содержит компоненты, отображающие информациюо пользователях, транспортных средствах, навигационных модулях.

Модуль *tasks* содержит компоненты, позволяющие осуществлять различные операции такие как: добавление в систему новых объектов, создание различных привязок и т.д.

Основной сущностью *Angular*-приложения является компонент. «Личный кабинет» содержит множество компонентов, предоставляющих пользователю удобный интерфейс для осуществления необходимых функций.

В приложении был реализован сервис, выполняющий функции маршрутизатора. Сервис содержит массив, сопоставляющий *URL*-адреса и компоненты. Например, *URL*-адрес – ‘*/login’*, соответствует *login.component.* Неавторизированный пользователь может зайти только на *URL*-адрес ‘/*login*’, а соответственно он имеет доступ к *login*.*component*. В случае, когда неавторизированный пользователь пытается перейти на другие *URL*-адреса, он будет перенаправлен на страничку входа в личный кабинет.

Для обработки запросов на серверной части приложения используются контроллеры – это классы, написанные на языке *C#.* В приложении «Личный кабинет» были реализованы следующие контроллеры:

* *AuthController* – обрабатывает запросы неавторизированных пользователей;
* *AccountController* – обрабатывает запросы неавторизированных пользователей;
* *UsersController* – обрабатывает все запросы, касающиеся пользователя, редактирования данных, добавление новых и т. д.;
* *DeviceController –* обрабатывает запросы, касающиеся навигационных модулей;
* *CarsController –* используется для обработки запросов транспортных средств.

В контроллерах происходит вызов функций классов и соответствующих сервисов, где формируются запросы к БД.

Рисунок 4.3 – Структура приложения

привязка модуля к пользователю

set-user

меню пользователя

menu

component

удаление привязку

unset-dev

component

добавление в архив

archive-car

component

страница пользователя

user-page

component

редактор

пользователя

edit-user

component

cтраница пользователя

user-page

component

список тран.

ср-в

cars-list component

страница входа

login

component

привязка

модуля к транспорту

set-dev

pедактор

транспорта

edit-car

component

меню

menu

component

список пользователей

users-list

component

список модулей

devs-list

component

добавление

транспорта

add-car

component

cтраница

тран-ого ср-ва

car-page

component

добавление пользователя

add-user

component

страница

модуля

dev-page

component

добавление

модуля

add-dev

component

редактор

модуля

edit-dev

component

привязка

модуля к транспорту

set-dev

списание

блока

unset-user

component

## **4.4 Авторизация пользователя в приложении «Личный кабинет»**

Для работы в разработанном приложении, пользователю необходимо авторизироваться. После перехода в приложение по ссылке, для пользователя открывается страница с формой входа, содержащая поля для логина и пароля (рисунок 4.4).

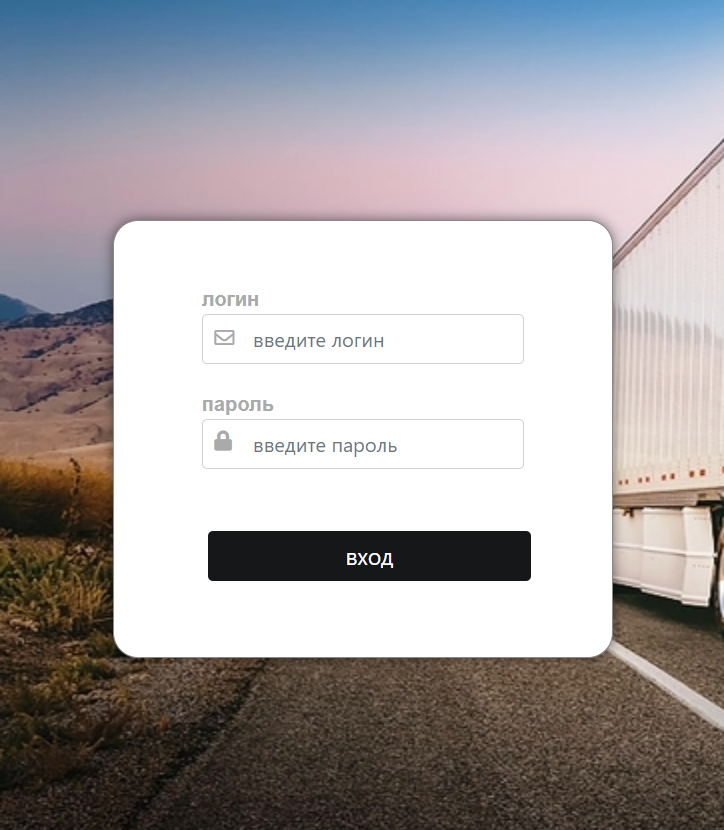


Рисунок 4.4 – Форма для авторизации пользователя

После заполнения формы и нажатия на кнопку «вход», пароль шифруется и вместе с логином отправляются на сервер. На сервере происходит проверка корректности введённых данных, и если они верны, то происходит вход в приложение. При правильности введённых данных на сервере для каждого пользователя формируется «session token», который содержит некоторую информацию о пользователи, например, идентификатор и роль пользователя.

При следующих запросах на сервер токен отправляется в заголовке запроса, что позволяет определить системе можно ли выполнять запрос или отослать ошибку пользователю. Каждый токен имеет определённое время существования. Если время токена истекло, то он автоматически обнуляется и пользователю необходимо заново авторизироваться. Токен доступа аннулируется и автоматически удаляется на «*фронтенде*», если пользователь добровольно выходит из системы, нажимая на кнопку выхода на странице (рисунок 4.5).

**

Рисунок 4.5 – Кнопка выхода из приложения

На рисунке 4.6 приведена блок-схема логики авторизации пользователя.

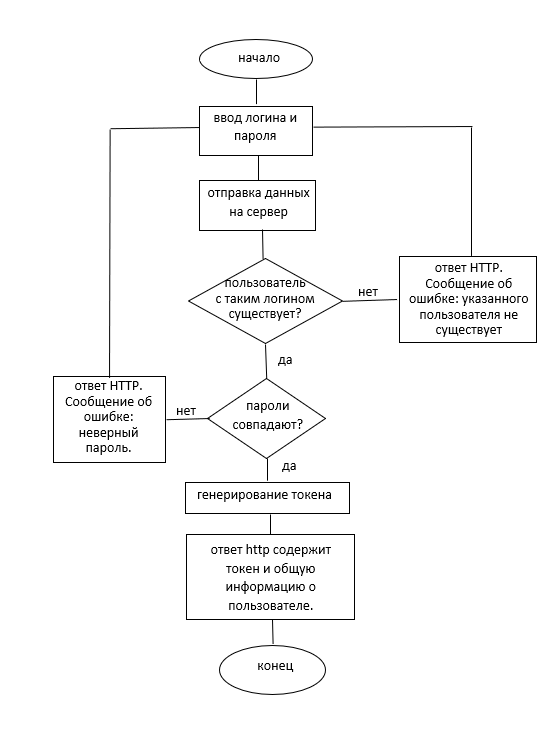


Рисунок 4.6 – Блок-схема авторизации пользователя

## **4.5 Функции «Личного кабинета»**

Все пользователи приложения, в зависимости от роли, которая устанавливается для каждого в процессе регистрации, могут быть разделены на три группы:

* администратор;
* пользователь;
* подразделение.

Роль администратора определяется для операторов компании, предоставляющей услуги мониторинга транспорта. В приложении для администратора доступны следующие функции:

* просматривать список всех пользователей, добавлять нового пользователя, просматривать личные данные любого пользователя, изменять роль, изменять статус пользователей (блокировать / разблокировать);
* просматривать список всех машин, добавлять новую машину любому пользователю, редактировать информацию о машине, разделять машины на группы, архивировать машины;
* просматривать список всех навигационных модулей, добавлять новый, редактировать информацию о навигационном модуле, изменять привязку навигационный модуль – пользователь и навигационный модуль – машина, списывать старые блоки.

Роль пользователя определяется для компаний, которым предоставляются услуги мониторинга транспорта. Каждый пользователь может создавать в своём аккаунте подразделения – это могут быть отделения компании, над которыми в приложении пользователь может осуществлять управление. В приложении для пользователя доступны следующие функции:

* редактировать личные данные;
* создавать подразделения, осуществлять управление подразделениями: редактировать, добавлять им машины и навигационные модули, создавать различные типы связей;
* просматривать список машин, принадлежащих конкретному пользователю, редактировать информацию о машине, добавлять новую машину;
* просматривать список навигационных модулей, принадлежащих данному пользователю, просматривать информацию о навигационном модуле, добавлять себе новый навигационный модуль;
* изменять привязки машина – навигационный модуль и навигационный модуль – машина.

Информация о роли содержится в токене, который генерируется на сервере после авторизации и передаётся в клиентскую часть приложения. Если пользователь имеет роль администратора, то после входа в систему он может просматривать свои личные данные, список остальных пользователей, переходить на страницы других пользователей и редактировать их данные (рисунок 4.7).

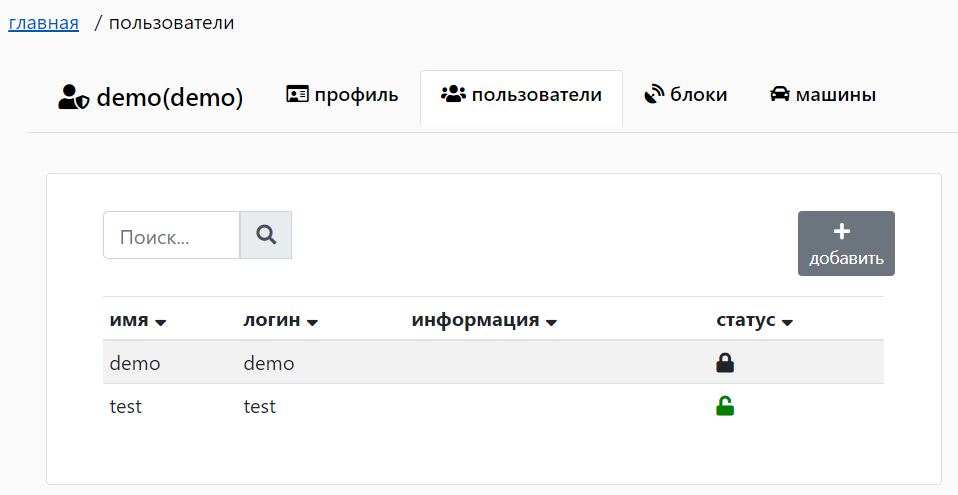
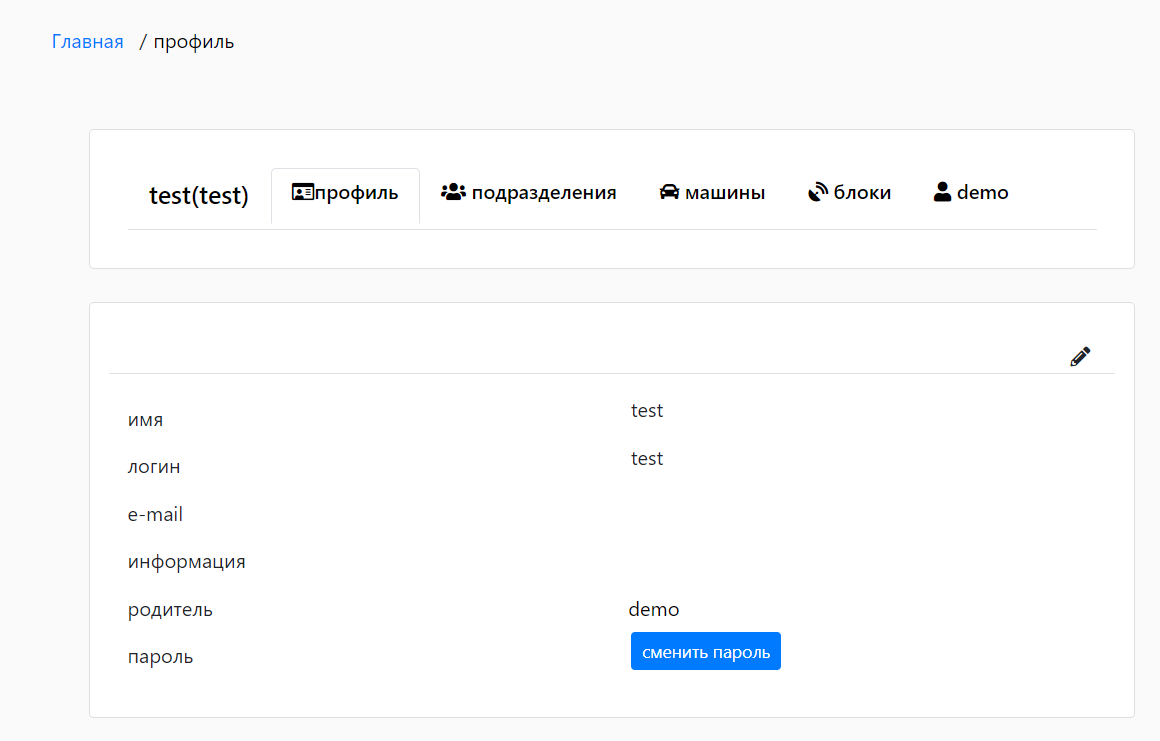


Рисунок 4.7 – Список всех пользователей на странице администратора

Если пользователь не имеет роль администратора, то после входа в разработанное приложение он сможет просматривать свои личные данные, список своих машин и навигационных модулей, добавлять подразделения (рисунок 4.8).

Рисунок 4.8 – Интерфейс приложения для обычного пользователя

Разработанное приложение «Личный кабинет» предоставляет пользователю различные функции для эффективного выполнения услуг мониторинга. При работе с приложением, для пользователя доступны следующие функции:

* редактор данных;
* добавление пользователей, транспортных средств, навигационных модулей;
* группировка транспортных средств;
* передача навигационного модуля от одного пользователя к другому;
* открытие и закрытие привязок машина – навигационный модуль или навигационный модуль – машина;
* списание старых навигационных модулей.

Для осуществления вышеперечисленных функций пользователю необходимо выполнить ряд шагов. Рассмотрим выполнение функции передачи навигационного модуля от одного пользователя к другому. Пользователь заходит на страницу выбранного навигационного модуля и выбирает необходимую функцию (рисунок 4.9).

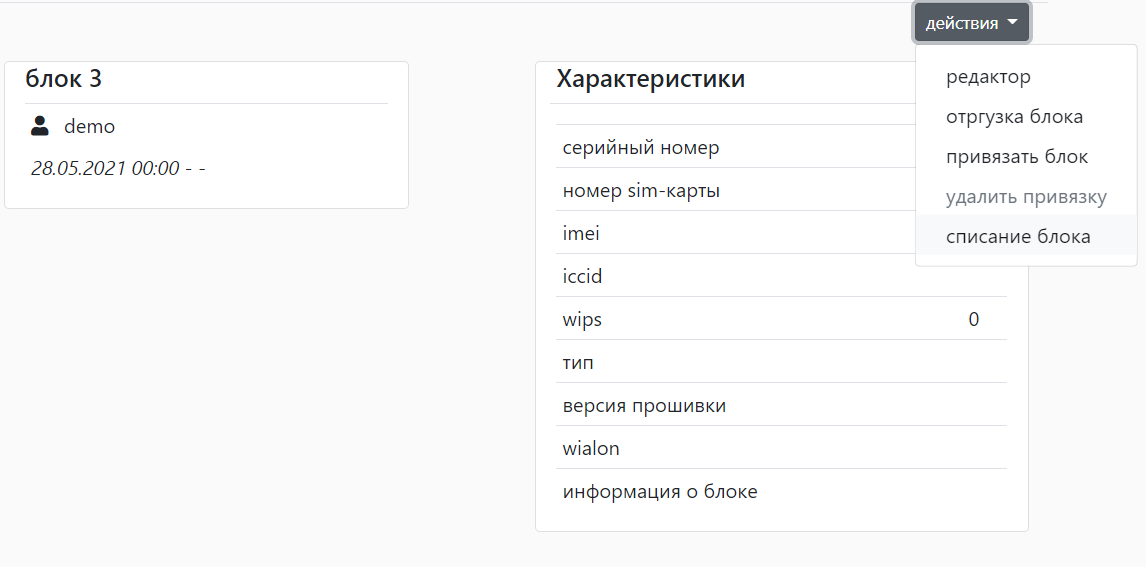


Рисунок 4.9 – Выбор функции на страницы навигационного модуля

Пользователь переходит на страницу выбранной функции и осуществляет ряд последовательных шагов. Данные отправляются на сервер. Сервер проверяет данные, которые ввёл пользователь. Если в данных содержится ошибка, например, пользователь установил дату передачи, которая не удовлетворяет определённые критерии, то операция не будет завершена, а пользователь получит сообщение об ошибке (рисунок 4.10).

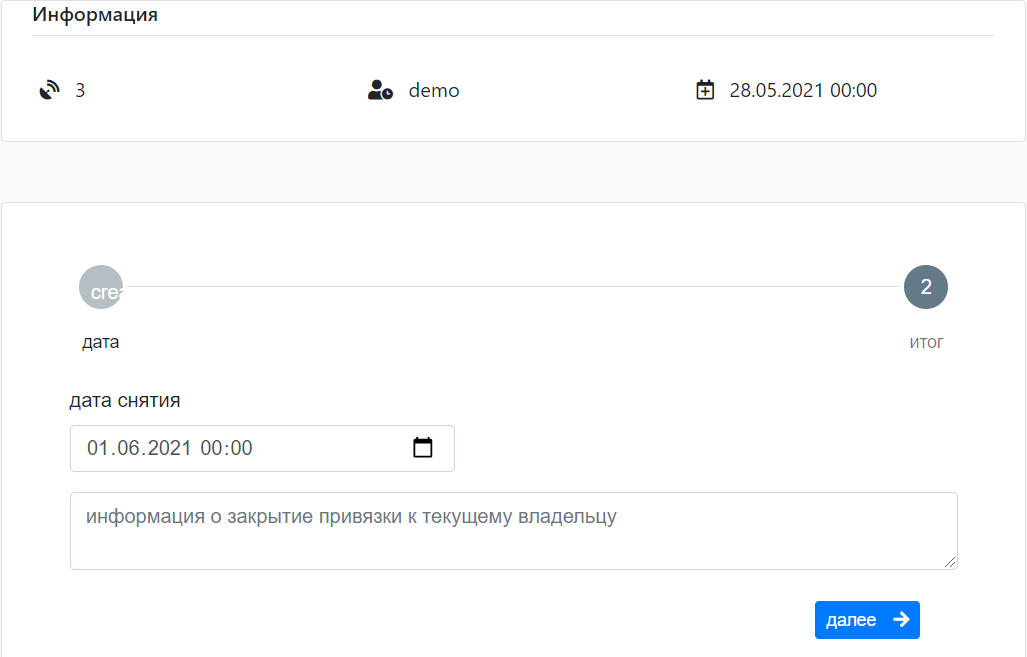


Рисунок 4.10. – Операция списания навигационного модуля

Если навигационный модуль установлен на транспортное средство, то перед осуществлением операции пользователю необходимо будет закрыть текущую привязку навигационный модуль – транспортное средство, а затем уже осуществлять процесс передачи. На рисунке 4.11 показана логика процесса.

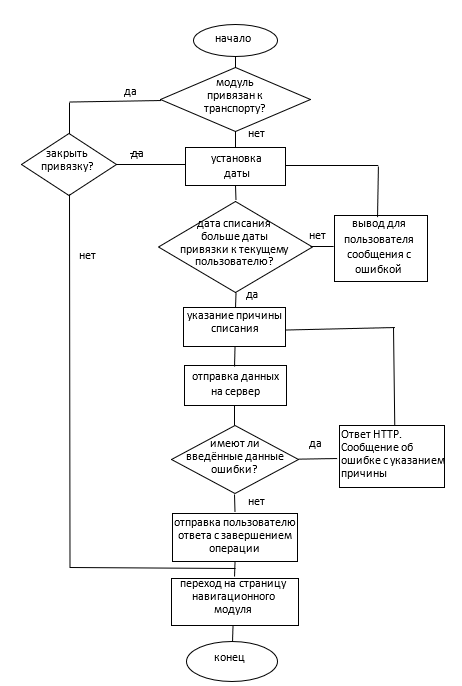


Рисунок 4.11 – Блок-схема операции «списания навигационного модуля»

Для создания связи транспортное средство – навигационный модуль необходимо выполнить несколько шагов.

1) Зайти на информационную страницу транспортного средства или навигационного модуля (рисунок 4.12), в зависимости от удобства пользователя.

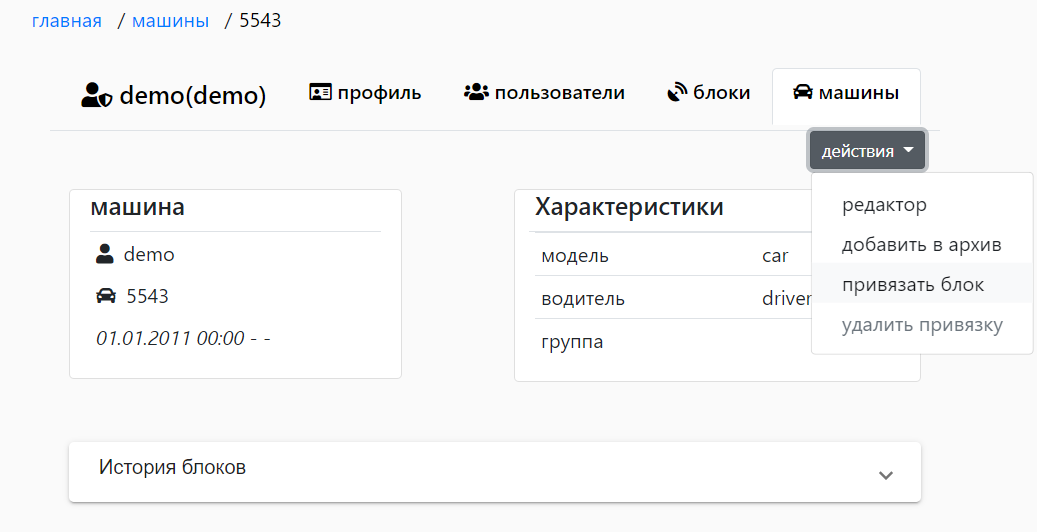


Рисунок 4.12 – Информационная страница транспортного средства

2) Выбрать необходимую операцию (рисунок 4.13).

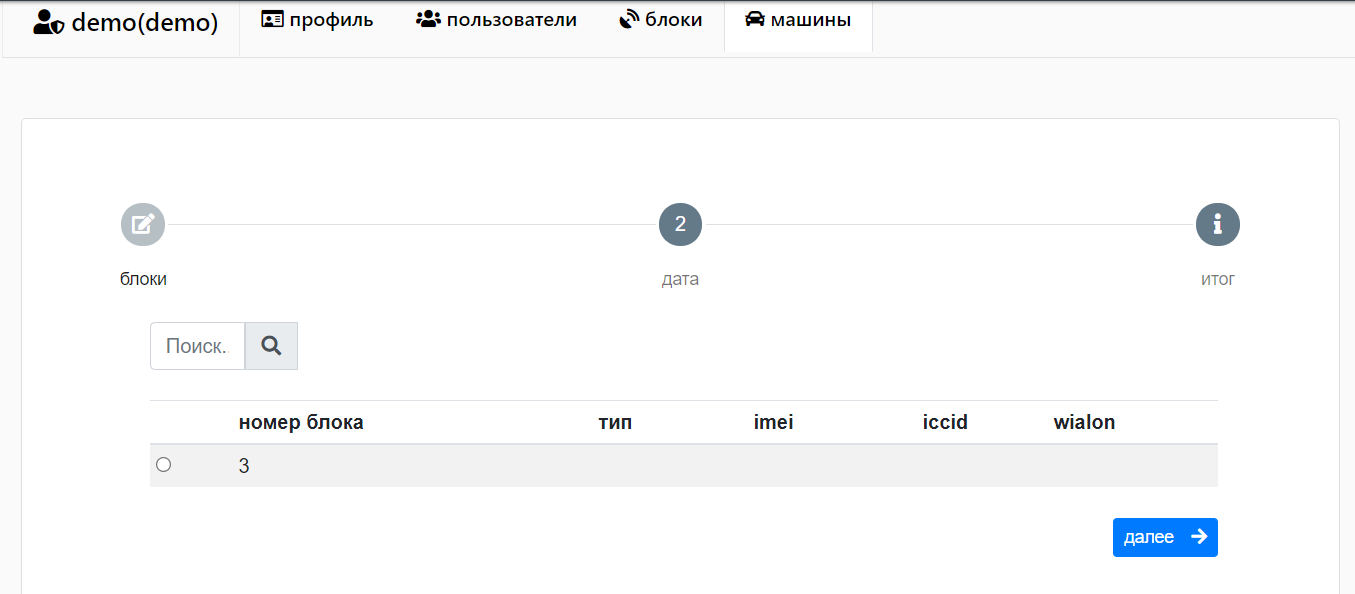


Рисунок 4.13 – Операция «привязки навигационного модуля»

3) Ввести данные и отправить их не сервер. На сервере происходит проверка данных, которые ввёл пользователь, и если всё правильно, то пользователь получает сообщением об успешном завершении операции. Логика операции представлена на рисунке 4.14.

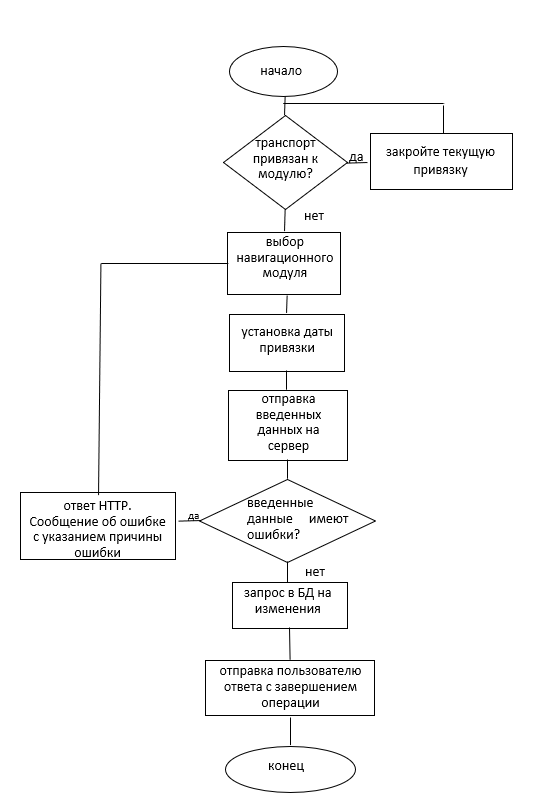


Рисунок 4.14 – Блок схема операции создания привязки транспорт- навигационный модуль

## **4.6 Визуализация данных**

Разработанное приложение предоставляет пользователю удобные средства для просмотра данных. Был реализован компонент timeline (рисунок 4.15) для просмотра истории привязок. В компоненте пользователь может самостоятельно устанавливать дату просмотра истории, а в строке поиска – выбирать машину, историю которой он хочет посмотреть.

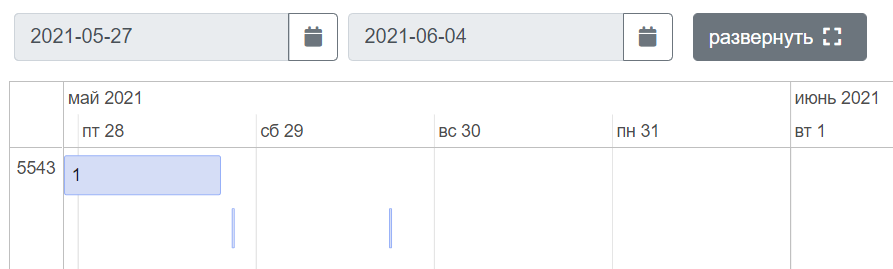


Рисунок 4.15 – Компонент timeline

Для пользователей, которые имеют большой список машин или навигационных модулей, приложение предоставляет удобные средства для фильтра данных по каким-либо критериям (рисунок 4.16).

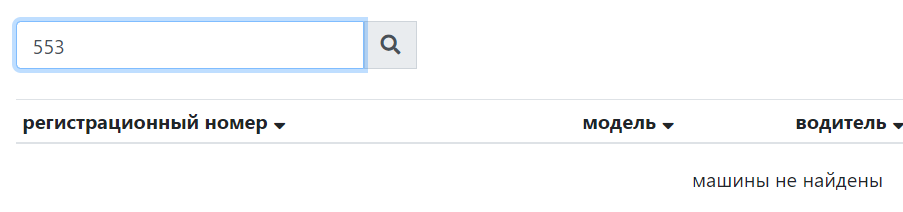


Рисунок 4.16 – Поиск данных в приложении

## **4.7 Тестирование и сборка приложения**

Проверка основных функций в приложении осуществляется с использованием unit-тестов, написанных при помощи фреймворков *Jasmine* и *Karma*. Были проверены основные функции клиентской части приложения в отдельности.

В Angular есть встроенный инструмент, который позволяет разработчику проверить, насколько покрыты тестами различные части кода (рисунок 4.17).

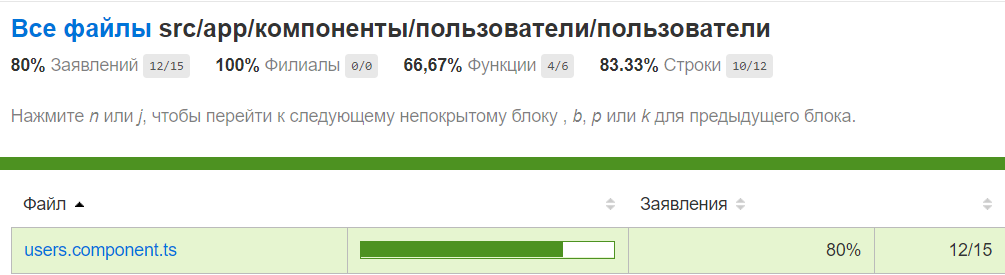


Рисунок 4.17 – Степень покрытия тестами

Для сборки приложения была использована технология контейнеризации docker. Для этого в приложении были созданы три каталога: backend, соответствующий серверной части приложения, frontend – клиентская часть приложения и database – база данных приложения.

В каждой папке содержится *dockerfile*, в котором прописаны необходимые директивы для запуска конкретной части приложения. *Dockerfile* – это контейнер. В этом файле определяются среды различных частей приложения, для запуска из в разных мест.

Инструкции для запуска и развертывания приложения были определены в файле *docker-compose.yml*. В этом файле были прописаны папки, названия контейнеров и номера портов, что позволяет запускать всё приложение вместе вне зависимости от среды.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения дипломной работы было разработано web-приложение «Личный кабинет» для системы мониторинга транспорта и решены все поставленные задачи.

* + - 1. Была изучена система мониторинга транспорта и проанализированы основные потребности данной сферы. На основе проделанной работы были определены основные функции разработанного web-приложения.
      2. Был изучен процесс создания и развёртывания web-приложений. В результате была выбрана архитектура будущего приложения и средства разработки.
      3. Было разработано web-приложение для системы мониторинга транспорта, в котором были реализованы следующие функции:
* редактирование личных данных пользователя, просмотр списка всех пользователей, создание нового пользователя;
* просмотр списка машин, принадлежащих конкретному пользователю, распределение машины по соответствующим группам, создание и добавление машины;
* просмотр списка навигационных модулей, редактирование информации о навигационном модуле, привязка или открепление блока от(к) машине или пользователю;
* просмотр и редактирование текущих привязок навигационного модуля к машине, навигационного модуля к пользователю;
* просмотр историй навигационных модулей, машин.

1. С использование фреймворка в разработанном приложении «Личный кабинет» были протестированы основные функции. Приложение «Личный кабинет» было собрано с использованием *(чего?)* docker.

*(возможно, переделать или добавить с учётом пожеланий на предзащите)*

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. M David Hanson The Client/ server Architecture // Server Management by editor Gilbert Held – march, 2000 – section 1, p. 3.
2. Kambalyal C. 3 Tier Architecture [Electronic resource] // USA: Sushil Consultants Inc. – An Information Technology & Management Services company - Mode of access: <https://channukambalyal.tripod.com/NTierArchitecture.pdf> – Date of access: 19.04.2021.
3. Harshwardhan. P.Shitole Secure Email Software using e-SMTP/ Harshwardhan .P.Shitole, Prof.S.Y.Divekar // International Research Journal of Engineering and Technology – 2019 – Vol. 06 №3 – pp.3967-3971.
4. Grigorik I. Making the web faster with HTTP 2.0/ I. Grigorik // Communications of the ACM – 2013 – Vol. 56 no. 12 pp. 42-49.
5. REST // Национальная библиотека им. Н. Э. Баумана [Электронный ресурс]– 2017 – Режим доступа: <https://ru.bmstu.wiki/REST#cite_note-1> – Дата доступа: 27.0502021.
6. Аутентификация. Теория и практика обеспечения безопасного доступа к информационным ресурсам. Учебное пособие для вузов / А.А.Афанасьев, Л. Т. Веденьев, А.А. Воронцов и др., под ред. А.А. Шелупанов, С.Л. Груздева, Ю.С. Нахаев. – 2-ое изд., стереотип. – М.: Горячая линия – Телеком, 2012. – 550 с.
7. Выростков Д. Обзор способов и протоколов аутентификации в веб-приложениях // Технологический консалтинг и разработка ПО [Электронный ресурс] – 2015 – Режим доступа: https://www.dataart.com.ua/news/obzor-sposobov-i-protokolov-autentifikatsii-v-veb-prilozheniyah/ - Дата доступа – 20.04.2021.
8. Никсон Р. Создаём динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS / Р. Никсон – 2-е изд. – СПб.: Питер, 2013 – 560 с.
9. Filippo Ricca Using Clustering to Support the Migration from Static to Dynamic Web Pages / Filippo Ricca, Paolo Tonella // International Conference on Program Comprehension – 2003 – Vol. 1 pp. 207.
10. Schwarz M. Design and analysis of web application frameworks [Electronic resource] // Aarhus University – 2013 – Mode of access: <https://pure.au.dk/portal/files/54599671/PH.D_dissertation_Mathias_Schwarz.pdf> – Date of access: 18.05.2021.
11. Официальный документация Angular [Электронный ресурс] / Angular – Режим доступа: https://angular.io/ - Дата доступа:04.05.2020.
12. Официальная документация React [Электронный ресурс] / React – Режим доступа: https://ru.reactjs.org/ – Дата доступа: 04.05.2020.
13. Официальная документация Vue.js [Электронный ресурс] / Vue.js – Режим доступа: https://vuejs.org/ – Дата доступа: 04.05.2020.
14. Фримен А. Angular для профессионалов. /А. Фримен – 2-е изд. – СПБ.:Питер, 2017 – 800с. -ил. – (Серия для профессионалов).
15. What’s So Great About Redux? [Electronic resource] /J. Falcone // freeCodeCamp – 2017- Mode of access: https://medium.freecodecamp.org/whats-so-great-about-redux-ac16f1cc0f8b. – Date of access:20.05.2020.
16. Куликов С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов – Минск: Четыре четверти, 2017 – 312 с.
17. Тестирование и внедрение системы // European Certificate of Informatics Professionals [Электронный ресурс] – 2011 – Режим доступа: <https://eopearhiiv.edu.ee/e-kursused/eucip/arendus_vk/14____.html> – Дата доступа: 10.05.2021.
18. Learn Docker – from the beginning, part I images and containers. [Electronic resource] / C. Noring // Microsoft Azure – 2019 - Mode of access: https://dev.to/azure/docker-from-the-beginning-part-i-28c6 – Date of access: 20.05.2020.
19. What is Docker [Electronic resource] // RedHat, Inc – Mode of access: https://www.redhat.com/en/topics/containers/what-is-docker – Date of access: 20.05.2020.
20. What are Linux containers? [Electronic resource] // RedHat, Inc – Mode of access:https://opensource.com/resources/what-are-linux-containers?intcmp=7016000000127cYAAQ – Date of access: 20.05.2020.