МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ ИМЕНИ ГУМАРБЕКА ДАУКЕЕВА»
Институт Систем Управления и Информационных Технологий
Кафедра «Системы информационной безопасности»

«ДОПУЩЕН К ЗАЩИТЕ»
Зав.кафедрой _____
(ученая степень, звание, Ф.И.О.)
_____ «_____» _____20____г.
(подпись)

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Проектирование комплексной защиты информационной системы агенства недвижимости.

Специальность Системы Информационной Безопасности

Выполнил(а) Аманов Лачин Русланович Группа СИБ-16-2

Научный руководитель к.т.н. доцент Сатимова Елена Григорьевна

Консультанты:
по специальной части:
старший преподаватель Дмитриева Маргарита Валерьевна

_____ «_____» _____20____г.
(подпись)

по безопасности жизнедеятельности:
к.т.н. доцент Приходько Николай Георгиевич

_____ «_____» _____20____г.
(подпись)

Нормоконтролер: старший преподаватель Дмитриева Маргарита Валерьевна

_____ «_____» _____20____г.
(подпись)

Рецензент: _____
(ученая степень, звание, Ф.И.О.)
_____ «_____» _____20____г.
(подпись)

Алматы 2020

Задание на выполнение дипломного проекта

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ ИМЕНИ ГУМАРБЕКА ДАУКЕЕВА»

Институт Систем Управления и Информационных Технологий

Кафедра «Системы Информационной Безопасности»

Специальность «Системы Информационной Безопасности»

ЗАДАНИЕ
на выполнение дипломного проекта

Студенту Аманову Лачину Руслановичу

Тема проекта «Проектирование комплексной защиты информационной системы агентства недвижимости»

Утверждена приказом по университету № 147 от «11» ноября 2019 г.

Срок сдачи законченного проекта «1» июня 2020 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): Исходные материалы для выполнения задания дипломного проекта – сервер СУБД Oracle Linux, СУБД Oracle Database, средства Oracle для организации безопасности базы данных, расширение PHP – Laravel, расширение JS – Vue, расширение для CSS – SCSS, OCI-8 для взаимодействия с БД Oracle.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта: Цель работы - проектировка и организация комплексной защиты базы данных Oracle для агентства. Акцент делается на защите на уровне приложения и СУБД, средствами Laravel, Vue, Oracle и SCSS, то есть на всем том, что входит в обязанности администратора безопасности БД. Задача также – организация безопасности на и уровне сети. На уровне приложения стояла задача разработки защищенного от атак, направленных на базу данных, приложения.

Перечень графического материала (с точным указанием обязательных чертежей): исходная ER-диаграмма реляционной учебной базы данных агентства недвижимости__

Основная рекомендуемая литература: Официальная документация Laravel, веб-сайт www.laravel.com, официальная документация Vue, веб-сайт

www.vuejs.org, официальная документация Oracle Database, веб-сайт www.oracle-patches.com

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
Анализ рисков информационной безопасности	старший преподаватель Дмитриева Маргарита Валерьевна	17.02.2020 – 09.05.2020	
Безопасность жизнедеятельности	к.т.н. доцент Приходько Николай Георгиевич	17.02.2020 – 09.05.2020	

График
подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Проектирование БД агентств	17.02.2020 – 20.02.2020	
Создание пользователей. Предоставление привилегий и ролей	21.02.2020 – 28.02.2020	
Создание представлений, процедур, функций и триггеров	01.03.2020 – 08.03.2020	
Организация различных видов аудита БД	09.03.2020 - 18.03.2020	
Организация защиты сети. Распределение ролей пользователям по IP - адресам	19.03.2020 – 27.03.2020	
Написание приложения и его защита	28.03.2020 - 30.04.2020	
Анализ рисков ИБ. БЖД	01.05.2020 - 09.05.2020	

Дата выдачи задания «15» октября 2019 г.

Заведующий кафедрой _____ (Бердибаев Рат Шындалиевич)
(подпись) (ФИО)

Научный руководитель
проекта _____ (Сатимова Елена Григорьевна)
(подпись) (ФИО)

Задание принял к
исполнению студент_____ (Аманов Лачин Русланович)
(подпись) (ФИО)

Аннотация

В данной дипломной работе рассматривается проектирование комплексных мер защиты информационной системы агентства недвижимости, с использованием базы данных Oracle, фреймворков Laravel и Vue, для создания защищенного веб – приложения, исследования возможных уязвимостей и организации ее защиты.

Annotation

In this thesis, the design of integrated security measures for the information system of a real estate agency, using the Oracle database, the Laravel and Vue frameworks, to create a secure web application, study possible vulnerabilities and organize its protection is considered.

Аңдатпа

Бұл дипломдық жұмыста қорғалған веб – қосымшаны құру, ықтимал осалдықтарды зерттеу және оны қорғауды ұйымдастыру үшін Oracle деректер базасын, Laravel және Vue фреймворктерін пайдалана отырып, жылжымайтын мүлік агенттігінің ақпараттық жүйесін қорғаудың комплесдік шараларын жобалау қарастырылады.

Содержание

Введение

Сегодня ведущие агентства недвижимости демонстрируют положительные темпы роста и развития. При этом они вынуждены работать в жестких условиях рыночной экономики и нормативного регулирования деятельности.

Жизненная необходимость оставаться экономически эффективными и конкурентоспособными на фоне беспрецедентных изменений и трансформаций, постоянной реструктуризации и модернизации производства и сбыта заставляет компании заново переосмысливать стратегию и тактику бизнеса, а также разрабатывать и совершенствовать существующие корпоративные информационные системы (ИС) и системы информационной безопасности (СИБ).

Организация режима информационной безопасности (ИБ) становится критически важным стратегическим фактором развития любого агентства недвижимости.

Это, в особенности, проявляется в компаниях, использующих или работающих с сервисами онлайн-услуг,CRM-систем,1С, базы данных и веб-сайты.

Целями данной дипломной работы является созданием защищенного веб – приложения с использованием базы данных Oracle Database, изучением и дальнейшим решением проблем, уязвимостей, с которыми может столкнуться приложение, сбережением конфиденциальной информации от хищения, словом, проектированием комплексных мер защиты.

Актуальностью данного дипломного проекта заключается в том, что разработка веб – приложения является перспективным направлением информационных и телекоммуникационных систем, а агентства недвижимостей массово переходят к цифровой рабочей среде, для увеличения продуктивности рабочего процесса. Актуальностью также выражается постоянным обновлением, апгрейдом веб – фреймворков и веб – стандартов, чтобы посещение веб – ресурсов пользователями был еще более безопасным и удобным. Oracle тоже не стоит на месте и улучшает свои ресурсы, чтобы они были использованы организациями как от малого бизнеса до огромных корпораций.

1 Теоретический обзор технологий

1.1 Описание базовых технологий и их уязвимости

Перед тем как углубиться в фреймворки, которые использовались для создания приложения и формирования доступа к базе данных, были затронуты базовые концепции тех технологий, которые несут или могут нести в себе уязвимости, с целью рассмотрения опции их устранения и отладки.

HTML – Гипертекстовый язык разметки, применяется для верстки веб страниц, является главной на сегодняшний день технологией разметки веб документа. Автор использует HTML 5, являющейся последней на сегодняшний день версией данного языка разметки. В него добавлена поддержка таких семантических тегов как <section>, <header>,<footer>, <nav>, <article> и тд. Добавлена поддержка аудио и видео элементов <audio>,<video>. Сокращена форма записи DOCTYPE, а также появилось много новых атрибутов.

Веб браузер получает HTML документ в ответ от сервера или локального хранилища, а затем отображает документ в виде мультимедийной страницы.

HTML элементы - это строительные блоки для страницы. С помощью определенных тегов может быть отображены картинки, интерактивные формы, а также другие объекты.

HTML документ строится с использованием семантических тегов, таких как параграфы, заголовки, списки и ссылки. Теги HTML записываются в угловых скобках. В разметку HTML могут быть встроены программы, написанные на скриптовых языках таких как JavaScript, которые влияют на дальнейшее поведение веб страницы. Также может быть включен и CSS. В стандарте W3C указано что хранить HTML и CSS необходимо в разных файлах.

Но в сыром HTML есть много дыр связанные со скриптовыми тегами. Это связано с тем что человек не имея никаких прикладных устройств или дорогого программного обеспечения, может внедрить кусок скриптового кода, который может и не будет ничего красть, зато будет запустит цепочку событий, которые приведут к невозвратной поломке устройства пользователя. К сожалению, это можно сделать, использовав самый простой бесконечный цикл.

Например, проблема возникает в случае, когда злоумышленник вставляет какой то контент в тег script кусок кода приложений на реакте с серверным рендерингом:

```
<script>
window.__INITIAL_STATE__ = <%- JSON.stringify(initialState) %>;
</script>
```

В initialState </script> может появиться в любом месте, где данные поступают от пользователя или из других систем. JSON.stringify не будет менять такие строки при сериализации, потому что они полностью соответствуют формату JSON и Javascript, поэтому они просто попадут на страницу, позволяя злоумышленнику выполнить произвольный Javascript в браузере пользователя.

Другим примером может служить ситуация, в которой пользователь заходит на произвольный сайт, в котором отображена обычная форма отправки каких либо данных, предположительно – форма авторизации. Но, до того как была посещена страница в нее был внедрен вредоносный код:

http://xss.com/login.php?q=<script>killProcess()</script>

Этот код напоминает SQL – инъекцию, но при запросе злоумышленник отправляет тег скрипт, в которой расписан вредоносный код.

Когда человек отправит данные с формы на сервер, при авторизации, злоумышленник перехватит всю его персональную информацию. Таким образом, произвольные пользователи попадаются на один из самых распространенных видов атак XSS.

XSS – это атака, нацеленная на веб – приложения, которая подразумевает внедрение вредоносного кода на какую то ни было веб – страницу сайта и взаимодействие вредоносного кода с сервером хоста злоумышленников при открытии страницы обычным пользователем.

Основной целью межсайтового скриптинга является кражей cookies людей, пользующихся услугами веб – приложений, с помощью скрипта, инъецированного в сервер, с дальнейшей выборкой необходимых данных и использованием их для последующих атак и взломов.

Сравнив этот тип атак с SQL-инъекциями, автор делает вывод, что XSS атака безопасна для сервера, но несет угрозу для пользователей зараженного ресурса или страницы. Однако, если злоумышленник получит cookies администратора сервера, ему представится полный доступ к панели управления сайтом и его содержимому. По этим причинам, использование сырого HTML кода, не рекомендует консорциум всемирной паутины W3C.

CSS – Каскадные таблицы стилей (CSS) - это язык таблиц стилей, используемый для описания представления документа, написанного на языке разметки. Хотя чаще всего используется для установки визуального стиля веб-страниц и пользовательских интерфейсов, написанных в HTML и XHTML, язык может быть применен к любому XML-документу, включая простой XML, SVG и XUL, и применим к рендерингу в речи или к другим СМИ. Наряду с HTML и JavaScript, CSS - это краеугольная технология, используемая большинством веб-сайтов для создания визуально привлекательных веб-страниц, пользовательских интерфейсов для веб-приложений и пользовательских интерфейсов для многих мобильных приложений.

Наследование - ключевая особенность CSS. Эти таблицы влияют на отношения между предками и потомками. Наследование - это механизм, с помощью которого свойства применяются не только к определенному элементу, но и к его потомкам. Наследование основывается на дереве документов, которое является иерархией элементов XHTML на странице, основанной на вложенности. Элементы потомка могут наследовать значения свойств CSS из любого содержащего их элемента предка. В общем, элементы-потомки наследуют свойства, связанные с текстом, но свойства, связанные с коробкой, не наследуются. Свойства, которые можно унаследовать, - это цвет, шрифт, межстрочный интервал, высота строки, стиль списка, выравнивание текста, текстовый отступ, преобразование текста, видимость, пробел и интервал между словами. Свойства, которые не могут быть унаследованы, - color, font, letter-spacing, line-height, list-style, text-align, text-indent, text-transform, visibility, white-space and word-spacing и z-index. Наследование можно использовать, чтобы избежать объявления определенных свойств снова и снова в таблице стилей, что позволяет сократить CSS. Наследование в CSS - это не то же самое, что наследование в языках программирования на основе классов, в которых является возможным определить класс B «как класс A, но с изменениями». С CSS можно стилизовать элемент с «классом A, но с изменениями». Тем не менее, невозможно определить CSS-класс B, подобный этому, который затем можно было бы использовать для стилизации нескольких элементов без повторения модификаций.

Несмотря на то, что CSS не является языком программирования, и не несет в себе никаких свойств, позволяющих просматривать и модифицировать пользовательские данные, в 2009 году некто по имени Крис Эванс описал новую технику по извлечению этих самых данных путем использования обычного свойства background, просто перенаправив человека на свой сайт.

JavaScript – это высокоуровневый, интерпретируемый язык программирования. Он характеризуется как динамичный, слабо типизированный, основанный на прототипах и мультипарадигмальный. Вместе с HTML и CSS, JavaScript является одной из трех главных технологий во Всемирной Паутине Интернета. Данный скриптовый язык применяется для создания динамичных веб страниц и интерактивных онлайн программ, включая и видеоигры.

Все браузеры поддерживают JavaScript без надобности в дополнительной установке прочих плагинов. Каждый из движков JavaScript представляет своеобразную реализацию JavaScript'a, основанную на спецификации ECMAScript. Некоторые из стандартов ECMAScript все еще не поддерживаются современными браузерами.

Как мультипарадигмальный язык, JavaScript поддерживает событийное программирования (программирование на действиях таких как клик мыши) функциональное программирование, объектно-ориентированное и прототипное. Также имеет API для работы с текстовыми данными, массивами, датой, регулярными выражениями, а также манипуляции с DOM.

Изначально JavaScript являлся языком программирования на стороне клиента, но в настоящее время он используется и для написания серверных программ, а также мобильных приложений.

JavaScript язык программирования, у которого также существуют уязвимости и пробелы. Самой его главной проблемой является допущение внедрения вредоносного кода на страницу, подменяя ссылки или выведение рекламы на пораженном веб ресурсе.

PHP – Это серверный, скриптовый язык программирования, разработанный для создания веб – приложений, но также его используют как язык программирования и для других целей. PHP код может быть встроен напрямую в HTML код или может быть скомбинирован вместе с системой веб шаблонов и фреймворков. PHP код исполняется интерпретатором PHP как модуль на веб сервере или через CGI. Веб сервер комбинирует результат выполненного кода PHP, который может содержать любой тип данных, включая и изображения и сгенерированные веб страницы. PHP код также может быть запущен из командной строки и может быть использован при реализации графических приложений.

PHP является одним из лидеров в области веб разработки. Данный язык поддерживается большинством хостинг провайдеров. PHP используется такими крупными сайтами как Facebook,Wikipedia, VK. Применяется для разработки динамических веб - приложений, в которых предусмотрены запросы в базу данных и т.п. На основе PHP появилось множество фреймворков и CMS, упрощающие разработку. Особое внимание следует уделить фреймворку Laravel и CMS Magento. Которые предоставляют огромное количество инструментов для упрощения разработки и повышения качества и защищенности веб – приложений.

К сожалению, несмотря на достоинства этого языка, у него также имеется множество дыр. Примером может послужить ситуация, когда злоумышленник может получить доступ к таблицам базы данных путем использования очень популярного метода взлома – SQL-инъекций. Которую описывают, как обычную вставку SQL фрагмента, чем может быть обычный SELECT в адресной строке URL. Также случаются ситуации, когда злоумышленники могут просмотреть содержимое и название файлов, в случае сбоя опций Apache, что может привести к тому, что хакер сможет получить доступ на чтение, и запись важной информации базы данных, к примеру, учетных данных пользователей или же, непосредственно использующей эту базу данных, организации.

MySQL – Это структурированный язык запросов. SQL применяется в программировании для разработки и управления информацией в реляционной базе данных. SQL предоставляет возможность получить доступ к множеству записей в базе данных с помощью одного запроса, а также ликвидирует потребность в строгом указании, как достичь записи, с помощью индекса или без него.

MySQL содержит в себе множество типов и выражений, которые формально можно назвать подъязыками, обычно к ним причисляют: DQL, DDL, DCL, DML.

Область использования SQL включает в себя запрос информации, манипуляции информацией (добавление, удаление и изменение), создание таблиц и их модификация.

MySQL широко используется во всем мире и это все благодаря следующим преимуществам:

– позволяет пользователям получить доступ к информации в реляционной базе данных;

– позволяет пользователю описывать информацию;

– позволяет пользователям добавлять и управлять информацией;

– позволяет встраивать другие языки, используя MySQL модули, а также библиотеки;

– позволяет пользователю создавать и удалять базы данных и таблицы;

– позволяет пользователю создавать функции, процедуры и тд;

– позволяет задавать пользователю запрет на доступ другим пользователям и тд.

Но, несмотря на все эти преимущества, у MySQL есть большая уязвимость, которая позволяет запустить стороннюю библиотеку с root – правами, а для ее эксплуатации необходим доступ к СУБД с возможностью выполнять, например опции SELECT и FILE, и получать доступ к функциям работы с логами, которые должны быть доступны только пользователю admin. Через манипуляции с этими функциями записи в лог, атакующий может изменить или создать файл конфигурации my.cnf. Такая уязвимость получила название CVE-2016-6662. Этот эксплойт содержит следующий код:

```
set global general_log_file = '/etc/my.cnf';
set global general_log = on;
select '
[mysqld]
malloc_lib=/tmp/mysql_exploit_lib.so
[separator]
';
set global general_log = off;
```

1.2 Описание продвинутых технологий и их преимущества

СУБД Oracle Database 11g – база данных, спроектированная специально для работы в корпоративных сетях распределенной обработки данных GRID, которая предназначена для производительного развертывания на базе различных типов техники, от малых серверов до мощных симметричных мультипроцессорных серверных структур.

Oracle Database 11g характеризуется высокой производительностью для веб – приложений, мощностью, и безопасного хранения данных. Эта версия дает возможность использования криптографических средств на уровне

столбцов, что предполагает применение шифрования табличного пространства Tablespace, что соответственным образом применяются для кодирования всех таблиц с данными, индексов и других объектов базы, хранимых в этом табличном пространстве. Шифрование обеспечивается и для хранящихся в базе данных объектов LOB, которая является типом данных, предназначенная для работы с огромными текстовыми или графическими данными.

К тому же Oracle 11g предоставляет различные инструменты для разработчиков, а также несложный процесс создания веб – приложений, максимально производительно реализующий ключевые функциональные особенности СУБД, среди которых можно отметить – Client Side Caching (кэширование на стороне клиента), Binary XML для оптимизации работы приложений, которые обрабатывают, хранят и выполняют XML файлы. Кроме всего прочего Oracle имеет встроенную интеграцию с такими, на данный момент незаменимыми для разработки приложений, как Visual Studio, поддерживает экспорт данных в Microsoft Access.

Oracle Database содержит в себе словарь данных, являющимся его ядром, набор внутренних таблиц, которые содержат всю информацию, необходимую серверу базы данных для ее управления. К счастью владельцем этих таблиц является пользователь SYS и могут быть изменены только им. Набор представлений только для чтения предоставляются для осмысленного отображения содержимого внутренних таблиц, а также они позволяют пользователям Oracle запрашивать данные словаря без необходимости прямого доступа к нему.

СУБД Oracle представляет ряд механизмов для поддержки управления безопасностью базы данных. Он содержит более 80 различных системных привилегий. Каждая системная привилегия позволяет пользователю выполнять определенную операцию с базой. Если у пользователя нет привилегий, он не может выполнять какие-либо операции, в том числе подключение к базе данных.

Пользователи с высокими привилегиями базы данных получают возможность выполнять административные функции предоставляя определенные привилегии системы. Другим пользователям предоставляется только минимальный набор привилегий, позволяющий им подключаться к базе данных и получать доступ к необходимые данные. Управление безопасностью Oracle Database 11g может быть делегировано любому количеству пользователей. Специфичные для сайта роли могут быть определены для делегирования административных обязанностей на основе организационных структур. Когда пользователь подключен как SYSOPER или SYSDBA, пользователь имеет право на выполнение специальных операций с базой данных. Авторизация для подключения как SYSDBA или SYSOPER создается с помощью механизмов ОС (т. е. членств в определенной группе ОС и требует, чтобы пользователь проходил аутентификацию в ОС). Пользователь, подключенный как SYSOPER, имеет право выполнять запуск,

завершение работы базы данных, создать файл параметров сервера и операции резервного копирования. Пользователь, подключенный через SYSDBA, имеет те же полномочия, что и SYSOPER, с дополнительными возможностями для создания базы данных и выполнения операций, разрешенные всеми системными привилегиями опции Админа. Пользователи, которые подключаются через AS SYSDBA, имеют доступ ко всем данным словарных таблицы и могут предоставлять и / или отзывать объектные привилегии для объектов других пользователей.

Безопасность данных, управляемых сервером данных Oracle Database 11g, зависит не только в безопасном администрировании Oracle Database 11g, но и в правильном администрирование базовой платформы ОС и любых других узлов, подключенных к распределенной среде.

Основные распределенные функции, включенные в сервер Oracle Database 11g, используют ссылки к базе данных для определения пути подключения к удаленной базе данных Oracle. Когда подключение к удаленной базе данных, информация в базе данных ссылка определение используется для предоставления идентификационной и аутентификационной информации удаленного сервера. Удаленный сервер создает сеанс базы данных для пользователя со ссылкой на базу данных (если пользователь авторизован для доступа к удаленной базе), а затем принимает свои решения по управлению доступом на основе этого пользователя и его привилегий в базе.

Используя ссылки на базу данных для определения имен объектов схемы, пользователь в локальной базе данных может:

– выбирать, объединять данные из таблиц в любом количестве удаленных баз данных Oracle;

– использовать операторы DML для обновления таблиц в удаленных база данных (Oracle Database 11g автоматически реализует протокол двухфазной фиксации);

– выполнять программные модули.

Laravel является бесплатным PHP веб фреймворком с открытым исходным кодом. Он предназначен для разработки веб - приложений и построен на архитектурном шаблоне model-view-controller(MVC).

Особенности Laravel:

– модульная система упаковки;

– специализированный менеджер зависимостей;

– своеобразные методы доступа к базе данных;

– утилиты, которые помогают в размещении на хостинге, поддержке проекта и ориентация на упрощение взаимодействия;

– безопасная работа с данными.

Laravel имеет множество встроенных функций, плюсом является встроенная система авторизации, аутентификации и регистрации пользователя.

Eloquent ORM- это продвинутая реализация шаблона Active Record, который позволяет обращаться к базе данных через методы. Active Record представляет каждую таблицу в виде класса, а каждый отдельный объект класса как строку в этой таблице.

Querybuilder предоставляет прямой доступ к базе данных и является альтернативой Eloquent ORM. Вместо того чтобы писать запросы в базу данных напрямую, Laravel Querybuilder имеет множество классов и методов для доступа к базе данных.

Restful controllers представляет разделение логики HTTP, GET и POST запросов на (GET, POST, PUT, DELETE). GET запрос используется для того чтобы запросить определенную информацию от сервера, POST запрос используется для внесения новой информации, PUT используется для обновление информации, а DELETE для ее удаления.

Class autoloading предоставляет автозагрузку PHP классов без ручного ввода пути к классу. Также загружаются только классы, которые используются, загрузка ненужных классов не производится.

Blade templating engine – связывает один или более шаблонов с определенной информацией для вывода на дисплей с помощью транспонирования шаблоны в кэшированный PHP код для повышения производительности. Blade предоставляет множество встроенных механизмов таких как операторы ветвления и циклы, которые совпадают с их аналогами PHP.

Database seeding – предоставляет способ для наполнения таблиц базы данных с выбранным типом данных, технология очень полезная для тестирования.

Unit testing – предоставлена в виде интегрированной части Laravel. Используется для тестирования веб - приложения и запускается с помощью командной строки Laravel Artisan CLI.

Automatic pagination упрощает задачу реализации разделения информации на страницы.

Homestead – это виртуальная машина Vagrant, которая предоставляет разработчикам Laravel все необходимые инструменты встроенные в сборку, включая Ubuntu и Gulp и другими полезными инструментами для разработки.

Artisan CLI- интерфейс командной строки Laravel, который был добавлен в Laravel 2.

В данный момент с помощью Artisan выполняется: генерация кода PHP, а именно возможность генерировать контроллеры, модели, а также создавать Factoryдля наполнения базы данных данными или созданию Migration. Также имеет множество команд для управления системой роутинга веб – приложения.

Validation – система валидации Laravel. Laravel предоставляет несколько методов для валидации данных поступающих от клиента. По умолчанию Контроллеры Laravel используют трейт Validate Request который

предоставляет удобные и мощные правила для валидации информации поступающей с помощью HTTP запроса.

Mail – Laravel имеет удобный и простой API для работы с библиотекой SwiftMailer с использованием SMTP драйвера, позволяет быстро отправлять email с помощью локального или облачного сервиса в зависимости от вашего выбора.

CSRFprotection – Laravel также имеет механизм для защиты от кросс сайтовой подделки запроса. Laravel генерирует специальный токен для каждой активной сессии пользователя. Этот токен нужен для того, чтобы определить действительно ли авторизированный пользователь выполняет данный запрос. Рекомендуется токен вставлять к каждой форме на странице.

Migrations (миграции) являются контролем версии для вашей базы данных, позволяют вашей команде разработчиков легко изменять и делиться схемой базы данных вашего приложения. Решает многие проблемы как которые могут возникнуть, как и у одного программиста, так и команды. Строитель схемы базы данных предоставляет поддержку создания, управления и удаления таблиц базы данных. Данная логика также содержится в классах Laravel. Миграции генерируются с помощью командной строки Artisan CLI.

Для критически важных приложений существует два уровня безопасности: защита приложений и защита серверов. Laravel - это фреймворк для разработки, поэтому он делает более безопасным и сервер и приложение. Функции Laravel позволяют использовать все безопасно. Все данные очищаются там, где это необходимо.

Laravel защищает от SQL инъекций, поскольку в нем всегда работает библиотека Fluent Query Builder или Eloquent. Laravel осуществляет это, создав подготовленные операторы, которые избегают любого пользовательского ввода, который может прийти через ваши формы. Если хакеры добавляют новый ввод в форму, они могут попытаться вставить цитату, а затем запустить собственный запрос SQL, чтобы повредить или прочитать базу данных приложения. Однако это не сработает, так как используется Eloquent. Eloquent избежит этой команды SQL, и неверный запрос будет просто сохранен в виде текста в базе данных.

Laravel также защищает куки пользователей. Для этого фреймворк генерирует ключ приложения. Ключ приложения или ключ шифрования использует классы шифрования и cookie для создания безопасных зашифрованных строк и хэшей. Чрезвычайно важно, чтобы этот ключ оставался секретным и никому не передавался. Кроме того, он создается из 32 символов случайной генерации символов, чтобы никто не мог угадать его, поскольку Laravel использует этот ключ для проверки куки. Класс cookie использует ключ приложения для создания безопасных зашифрованных строк и хэшей. Laravel защитит куки, используя хэш и будет следить за тем, чтобы никто не вмешивался в них.

Чтобы защитить приложение от CSRF-атак , Laravel использует метод токенов - Form Classes, который создает уникальный токен в форме. Токен гарантирует, что запрос исходит от нашего приложения, а не откуда-то еще. С этим токеном можно убедиться, что он проверяет поддельный запрос. В Laravel по умолчанию включена CSRF-защита.

Vue.js является реактивной библиотекой, которая дает возможность внедрять интерактивное поведение в любой контекст, в котором выполняются функции по обеспечению безопасного пользования приложением, путем защиты куков и локальных данных пользователя таких как Local Storage Application, что является достаточно уязвимой частью всех веб сайтов. Vue состоит из экземпляров, компонентов, что делает весь ее код нечитаемым и неизменяемым при выполнении каких либо действий злоумышленников на клиентской стороне.

Vue можно использовать как на отдельных страницах, решая простые задачи, так и в качестве фундамента для полноценных промышленных приложений с безопасной авторизацией, использующей токены, регистрацией, и обработкой данных.

Сама по себе реактивность программирования не нова. Однако приложения ее начали использовать относительно недавно, в основном благодаря стараниям фреймворка Vue. Чтобы веб – приложение можно было назвать реактивным, оно должно выполнять следующие функции:

– отслеживать изменения в своем состоянии;

– распространять уведомления об изменении своих компонентов;

– автоматически рендерить представления в ответ на изменения состояния;

– своевременно реагировать на действия пользователя.

Существует огромное количество примеров по безопасности Vue.

При использовании шаблонов или render-функций содержимое экранируется автоматически. Это значит, что для шаблона:

```
<h1>{{ userProvidedString }}</h1>
```

Если userProvidedString содержит:

```
‘<script>alert(“hi”)</script>’
```

то он будет экранирован в следующий HTML:

```
&lt;script&gt;alert(&quot;)&lt;/script&gt;
```

Таким образом предотвращая внедрение вредоносного скрипта. Экранирование осуществляется с помощью нативного API браузера, такого как textContent, поэтому уязвимость возможна только в случае, если сам браузер уязвим.

Аналогичным образом, динамические привязки к атрибутам также автоматически экранируются. Это значит, что для шаблона:

```
<h1 v-bind:title="userProvidedString">hello</h1>
```

Если userProvidedString содержит:

```
onclick=”alert(\’hi\’)'
```

то он будет экранирован следующим образом:

```
&quot; onclick=&quot;alert('hi')
```

тем самым предотвращая преждевременное закрытие атрибута title для добавления нового, произвольного. Экранирование выполняется с помощью нативного API браузера, такого как setAttribute.

Также автор обращает внимание на этот пример:

```
<a v-bind:href="sanitizedUrl" v-bind:style="userProvidedStyles">Ссылка</a>
```

Он предполагает, что sanitizedUrl был санитизирован и это действительно настоящий URL, а не JavaScript. Но используя userProvidedStyles, злоумышленники всё еще могут предоставить CSS для так называемой атаки click jack, то есть стилизовать ссылку в виде прозрачного блока поверх кнопки входа. В таком случае, если https://example.com куда ведёт ссылка, создан таким, чтобы визуально повторять на страницу авторизации вашего приложения, появляется возможность перехвата например логинов и паролей пользователей.

В случае, если пользователям позволят определять содержимое элемента <style>, это создаст ещё большую уязвимость, предоставив полный контроль над стилями страницы. Поэтому Vue не стоит отрисовывать теги стилей внутри шаблонов, например так:

```
<style>{{ userProvidedStyles }}</style>
```

Чтобы полностью обезопасить юзеров от атак click jacking, Консорциум Всемирной Паутины W3C и Vuejs.org рекомендуют дать доступ полного контроля над CSS только внутри изолированного iframe.

2 Разработка защищенного веб – приложения

2.1 Создание базы данных Oracle

Разработку веб - приложения открывает создание базы данных Oracle [1] с дальнейшей её конфигурацией, рисунок 2.1.

```
set sqlblanklines on
create tablespace estate_agency datafile '/u01/app/estate_agency.dat' size 200M reuse autoextend on next 10M maxsize 200M;
CREATE USER estate_agency
IDENTIFIED BY qwerty2905 DEFAULT TABLESPACE estate_agency;
GRANT DBA TO estate_agency;
ALTER USER estate_agency QUOTA 1000M ON estate_agency;

conn estate_agency/qwerty2905@orcl;
```

Рисунок 2.1 – Создание базы данных Oracle

Создается табличное пространство estate_agency, которое является выделением пространства в базе данных, которое в свою очередь может содержать объекты схемы. С созданием табличного пространства указывается файл данных, который будет использоваться для хранения данных, включая пользовательские данные размером 200 Мб и возможностью расширения до 300М. Далее создается пользователь estate_agency с привилегиями администратора, созданием пароля и привязкой к ранее зарегистрированному табличному пространству. Также пользователю присваивается квота на это табличное пространство, ограничивающее используемое пользователем место в нем, рисунок 2.2.

```
CREATE ROLE agency_admin
IDENTIFIED USING agency_admin_role_check;
GRANT SELECT, INSERT, UPDATE, DELETE ON cities TO agency_admin;

CREATE OR REPLACE PROCEDURE agency_admin_role_check
 AUTHID CURRENT_USER
AS
BEGIN
    IF (SYS_CONTEXT ('USERENV','IP_ADDRESS')
        BETWEEN '192.168.10.0' and '192.168.10.15')
    THEN
    EXECUTE IMMEDIATE 'SET ROLE agency_admin';
END
IF;
END;
/
CREATE USER estate_agent
IDENTIFIED BY qwerty2905;
GRANT EXECUTE ON agency_admin_role_check TO estate_agent;
GRANT CREATE SESSION TO estate_agent;
COMMIT;
```

Рисунок 2.2 – Создание роли и пользователя Oracle

Выше происходит создание роли agency_admin для контроля доступа к приложению, которой предоставляются привилегии на чтение, запись, изменение и удаление данных таблицы CITIES в созданном ранее табличном пространстве. Назначается процедуру, которая устанавливает свойство

AUTHID пользователю, чтобы использовать права роли. Процедура проверяет пользователя с помощью SYS_CONTEXT, функцию SQL, которая используется для получения информации о текущем сеансе пользователей. Это условие строится на простом тесте, который предполагает что пользователи, работают с базой в пределах ip адресов, указанных в условии. В данном случае это между 192.168.10.0 и 192.168.10.15. В случае если пользователь прошел тест, ему предоставляется роль.

Эта роль, которая может быть включена только авторизированным пользователям пакетом PL/SQL или, как в моем случае, процедурой agency_admin_role_check. Этот метод создания роли ограничивает включение роли этого типа вызывающему приложению. Например, приложение может выполнять аутентификацию и настраиваемую авторизацию, например, проверять, подключился ли пользователь через прокси-сервер. Этот тип роли повышает безопасность, поскольку пароли не внедряются в исходный код приложения и не хранятся в таблице, рисунок 2.3.

```
C:\Users\user>sqlplus estate_agency/qwerty2905@orcl

SQL*Plus: Release 11.2.0.3.0 Production on Sat May 2 22:40:14 2020

Copyright (c) 1982, 2011, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> @estate_agency.sql

Role created.


Grant succeeded.


Procedure created.


User created.


Grant succeeded.


Grant succeeded.


Commit complete.

SQL>
```

Рисунок 2.3 – Создание роли, процедуры, пользователя и сессии

Этот пример показывает, что пользователь, который выполнил процедуру, имеет IP адрес, который подходит условию процедуры. Что позволяет ему выбрать, изменять или удалять данные с этой сущности. Также обращается внимание на то, что ранее созданной роли были даны права к

одной таблице, но не к другим, что демонстрируется на рисунке 2.4, путем обычной выборки данных из таблицы STREETS, вернувшую ошибку об отсутствии этой таблицы.

```
SQL> conn estate_agent@orcl
Enter password:
Connected.
SQL> execute estate_agency.agency_admin_role_check;

PL/SQL procedure successfully completed.

SQL> select name from estate_agency.cities;

NAME
--------------------------------------------------------------------------------
Almaty
Astana
Atyrau
Aktau

SQL> select * from estate_agency.streets;
select * from estate_agency.streets
                            *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> SELECT SYS_CONTEXT('USERENV','IP_ADDRESS') FROM dual;

SYS_CONTEXT('USERENV','IP_ADDRESS')
--------------------------------------------------------------------------------
192.168.10.11

SQL>
```

Рисунок 2.4 – Выполнение процедуры

Дальше настраивается аудит пользователей базы данных. Чтобы это работало при подключении и отключении пользователя используются триггеры, которые срабатывали автоматически при авторизации и выходе, рисунки 2.5 – 2.8.

```
create table audit_users
(
    username varchar2(80),
    timestmp timestamp
);
create table audit_users_logoff
(
    username varchar2(80),
    timestmp timestamp
);

conn sys/Qwerty2905@orcl as sysdba;

create or replace TRIGGER AUDIT_LOGOFF_t
BEFORE LOGOFF ON database
BEGIN
    IF USER IN ('ESTATE_AGENCY') THEN
    INSERT INTO estate_agency.audit_users_logoff
    VALUES
        (USER, SYSDATE);
END
IF;
END AUDIT_LOGOFF_t;
/
create or replace TRIGGER AUDIT_LOGON_t
AFTER LOGON ON database
BEGIN
    IF USER IN ('ESTATE_AGENCY') THEN
    INSERT INTO estate_agency.audit_users
    VALUES
        (USER, SYSDATE);
END
IF;
END AUDIT_LOGON_t;
/
commit;
```

Рисунок 2.5 – Создание таблиц и триггеров на аудит пользователя

```
SQL> @estate_agency.sql

Table created.


Table created.

Connected.

Trigger created.


Trigger created.


Commit complete.

SQL>
```

Рисунок 2.6 – Выполнение транзакций

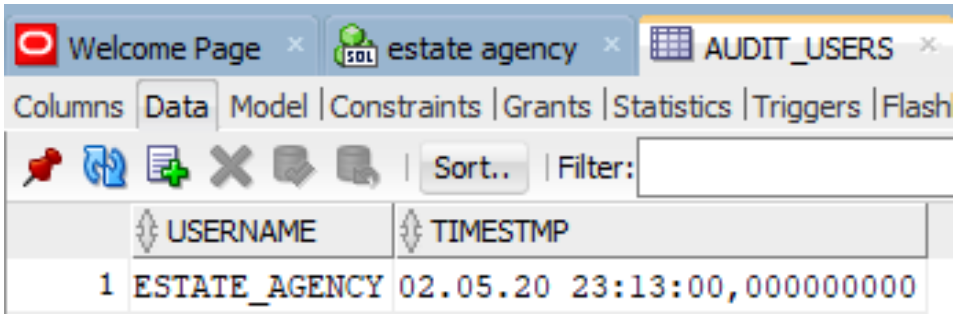

Рисунок 2.7 – Успешный аудит на авторизацию

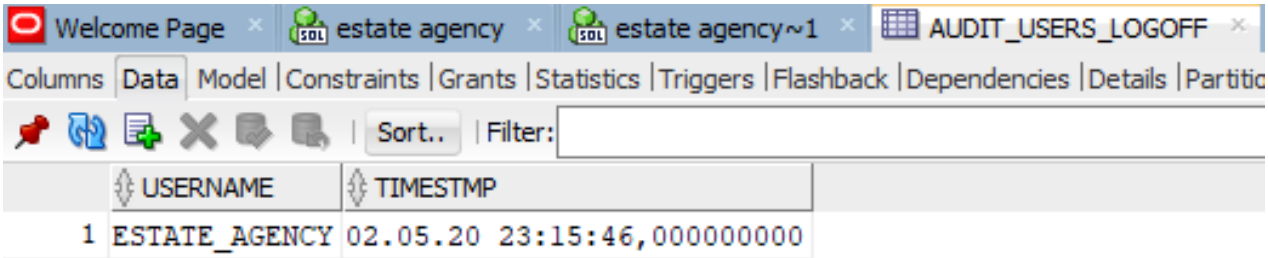

Рисунок 2.8 – Успешный аудит на деавторизация

2.2 Настройка и создание серверной части, Laravel

Проект плавно подходит к созданию веб – приложения, в котором будут использованы самые передовые фреймворки – Laravel, Voyager, SCSS, Vue.

Первым делом установим Laravel [2] и создадим проект, используя консоль GitBash и следующие команды:

- composer global require laravel/installer;
- laravel new.

Спустя некоторое время в папке появится директория с фреймворком Laravel. Установив фреймворк необходимо настроить доступ к базе данных. Доступ к настройке базы данных находится в корневой папке Laravel, в файле env. На рисунке 2.9 отображены эти настройки.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:nLnxwGOVNwePLVQkh7WtWOGk8c2nob0VURQG9Se/5wM=
APP_DEBUG=true
APP_URL=http://127.0.0.1:8000

LOG_CHANNEL=stack

DB_CONNECTION=oracle
DB_HOST=192.168.10.111
DB_PORT=1521
DB_DATABASE=orcl
DB_USERNAME=estate_agency
DB_PASSWORD=qwerty2905
```

Рисунок 2.9 – Настройки файла env для подключения к базе

Далее происходит реализация создания административной панели, которая предназначена, для администрирования базы данных проекта. Но перед этим, рассматриваются несколько миграций к базе, которые создают её сущности. Пример небольшого отрывка кода одной из таких миграций, рисунок 2.10.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateStreetsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('streets', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->unsignedBigInteger('district_id');
            $table->foreign('district_id')->references('id')->on('districts');
            $table->string('name', 100);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('streets');
    }
}
```

Рисунок 2.10 – Создание миграции Street

Эта миграция, создает таблицу STREETS в базе с такими полями, как id, district_id, name, здесь же district_id является вторичным ключом к таблице districts, которая также создается путем мигрирования из Laravel. Миграции расположены в корне приложения, в папке database/migrations, рисунок 2.11. Выполняются миграции командой php artisan migrate. Результат миграций и ER-диаграмма показаны на рисунках 2.12 – 2.15.

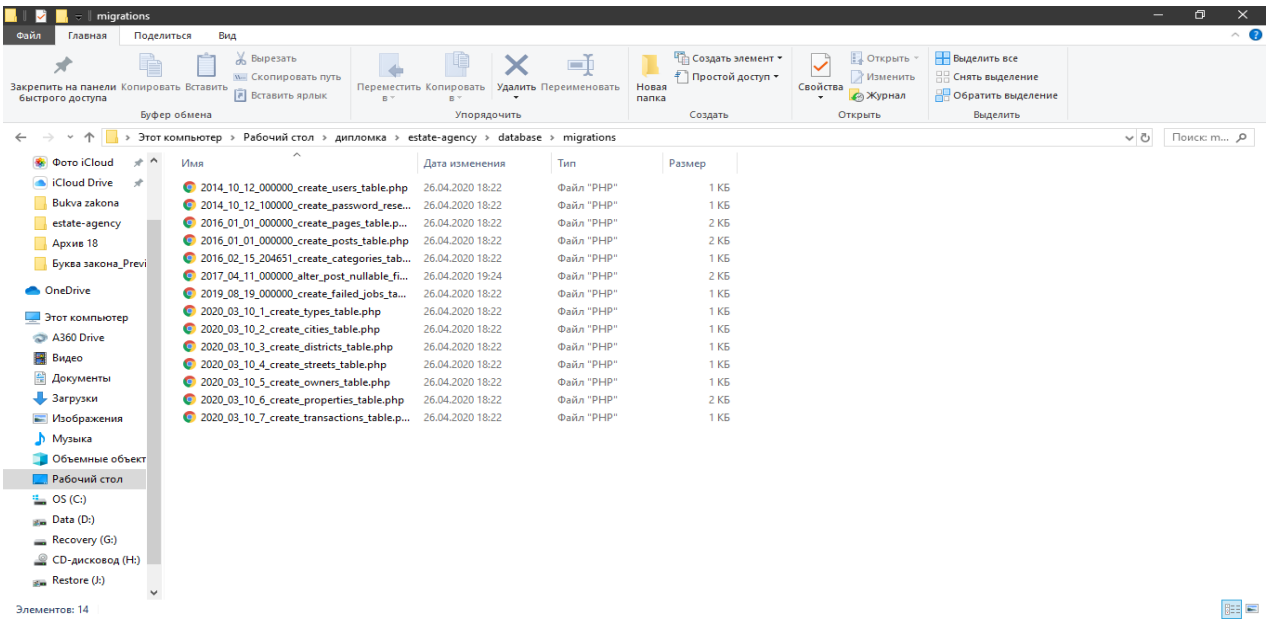

Рисунок 2.11 – Расположение миграций базы данных

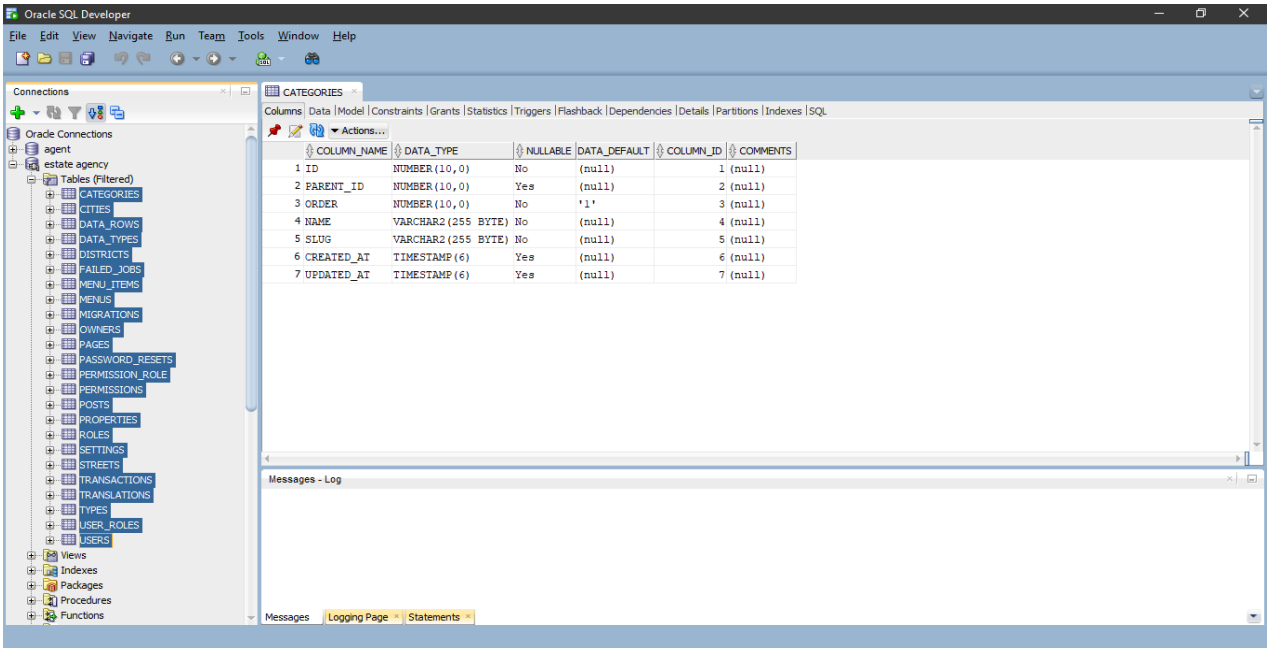

Рисунок 2.12 – Результат миграций

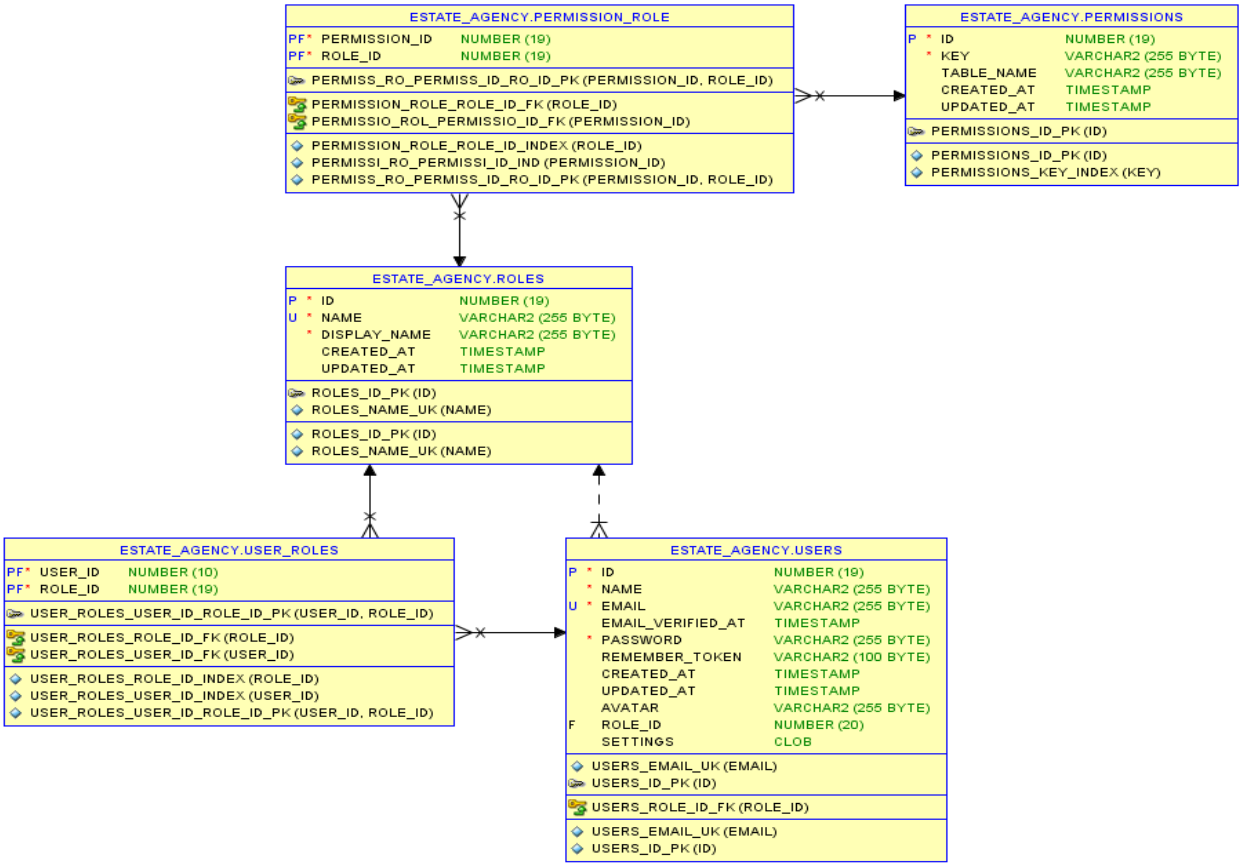

Рисунок 2.13 – ER-диаграмма, часть 1

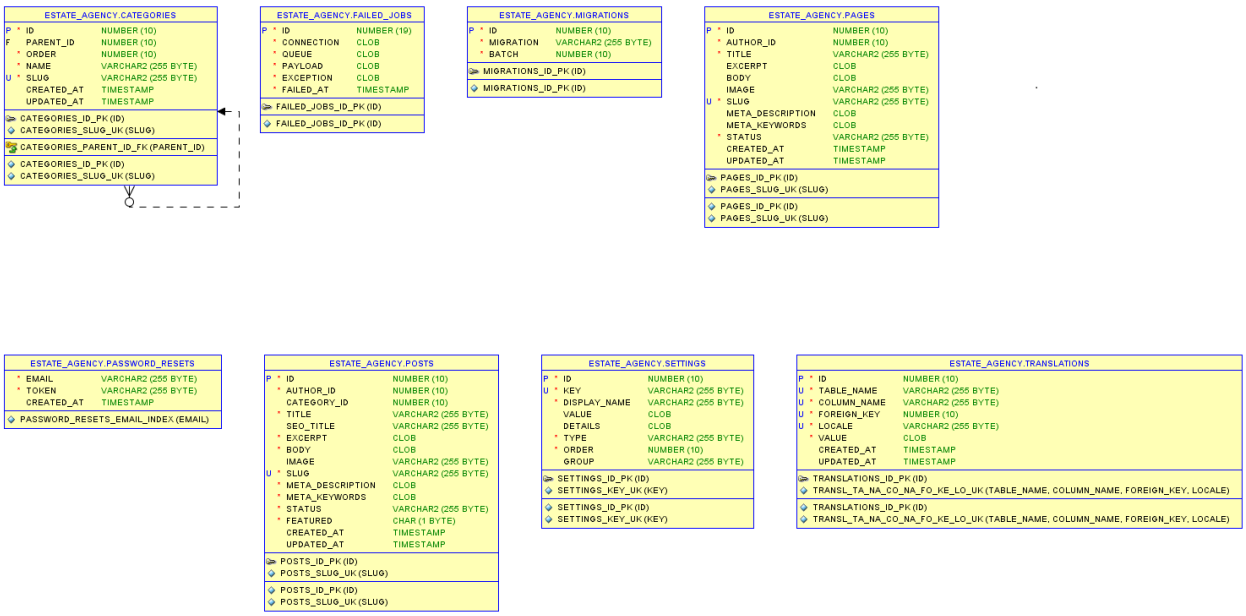

Рисунок 2.14 – ER-диаграмма, часть 2

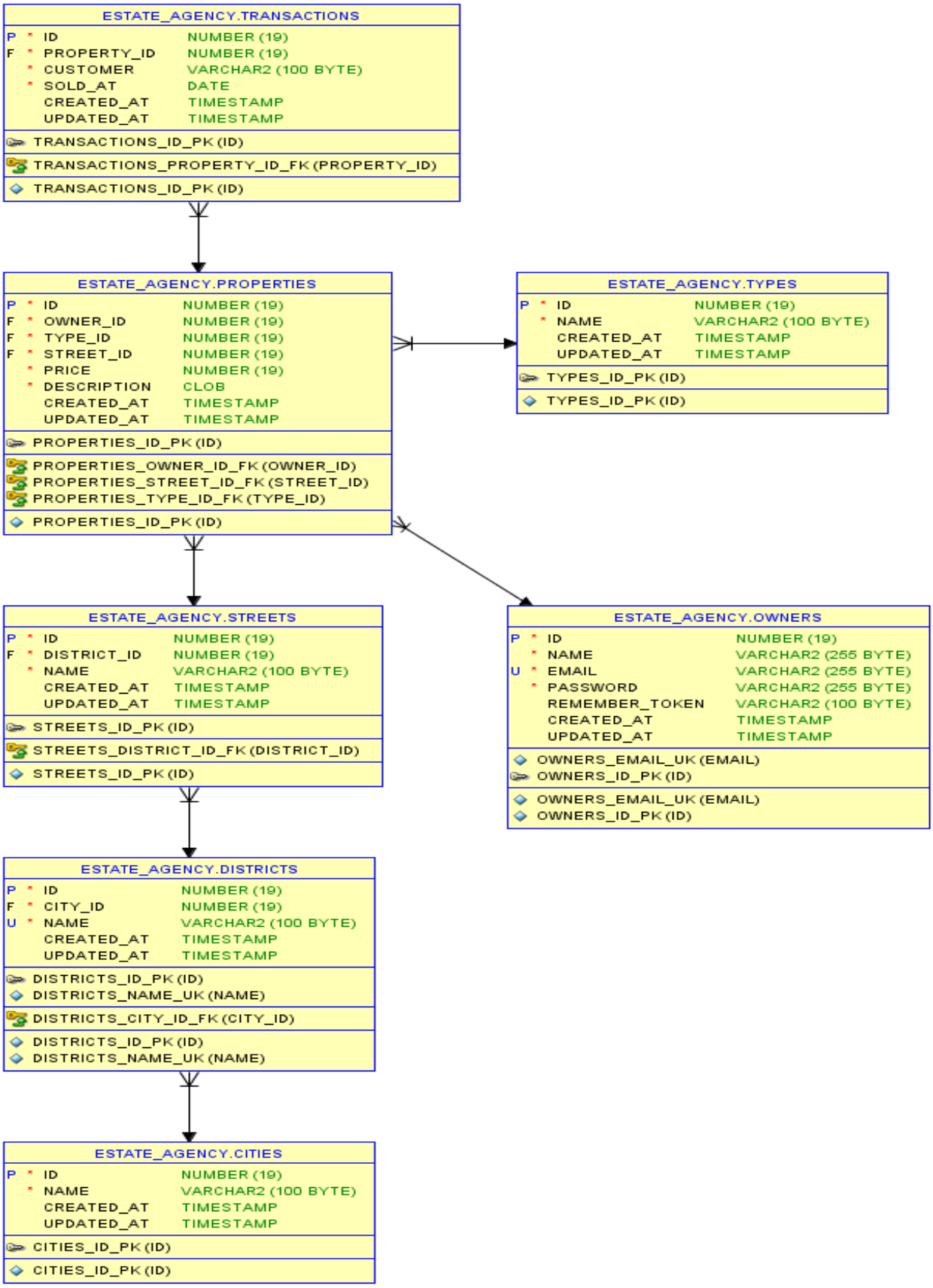

Рисунок 2.15 – ER-диаграмма, часть 3

После того как закончен весь процесс миграций и таблицы были созданы и смонтированы, наступает момент создания административной панели – Voyager [3]. Для создания административной панели используют команду php artisan voyager:install. После установки и небольшой настройки администратору предоставляется возможность зайти в панель Voyager через адрес локального хоста, указанный в env-файле – http://127.0.0.1/admin. Страница авторизации отображена на рисунке 2.16.

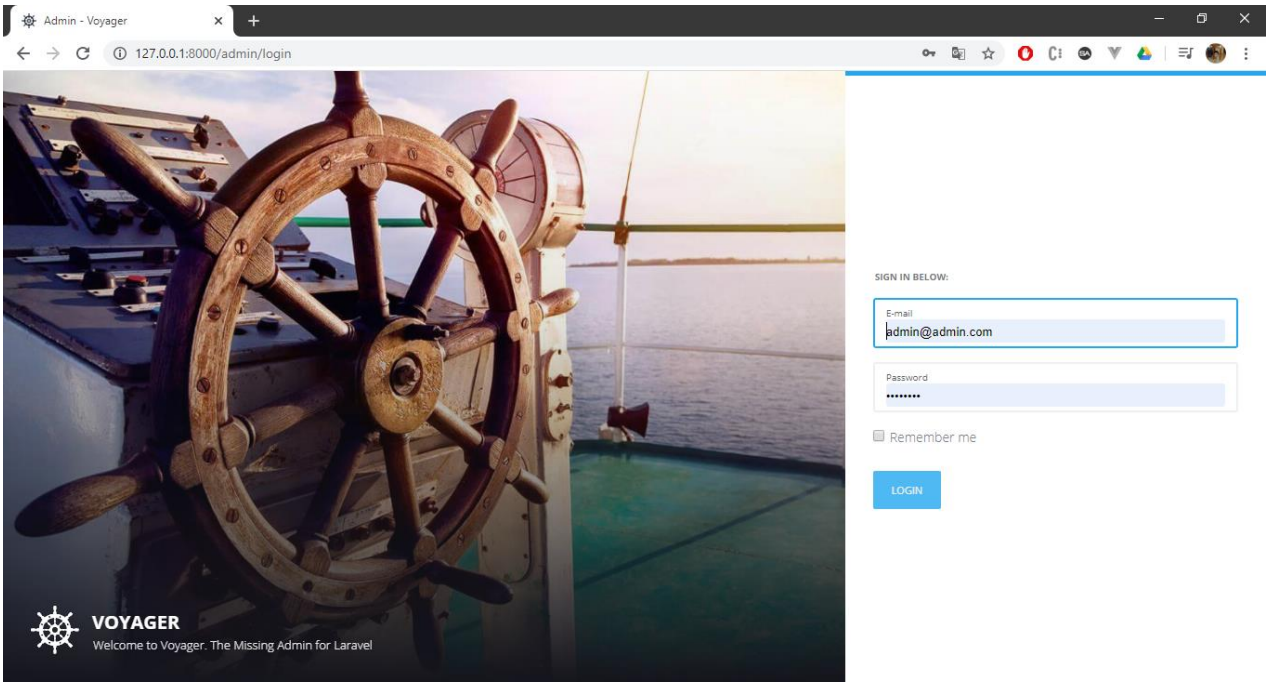

Рисунок 2.16 – Административная панель, страница авторизации

После авторизации администратору предоставляется множество вариантов по управлению базой данных, рис. 2.17 – 2.22.

Административная панель состоит из вкладок:

а) вкладка “Панель управления”;

б) вкладка “Роли”, в которой хранятся роли, которые создает Voyager, для пользователей, по умолчанию создаются роли администратор и пользователь. У администратора есть права на чтение, редактирование, запись и удаление любых данных с таблиц базы. У пользователя напротив ограничены права к существующим сущностям;

в) вкладка “Пользователи”, в которой расположены существующие пользователи администраторской системы;

г) вкладка инструменты, которая в свою очередь из таких вкладок, как:

1) “База данных”, в которой отображена вся структура базы, и в зависимости от типа авторизованного пользователя, вкладка может предоставить доступ, к чтению, изменению и созданию таблиц.

2) “Документация”, являющаяся справочником админки. Содержит команды, ресурсы и логии.

3) “Bread”, позволяет администратору создавать так называемые хлебные крошки к админке, с помощью которых можно, опять же в зависимости от привилегий, читать, создавать, редактировать и удалять данные с таблиц.

д) вкладка “Настройки”, имеет множество настроек к административной панели.

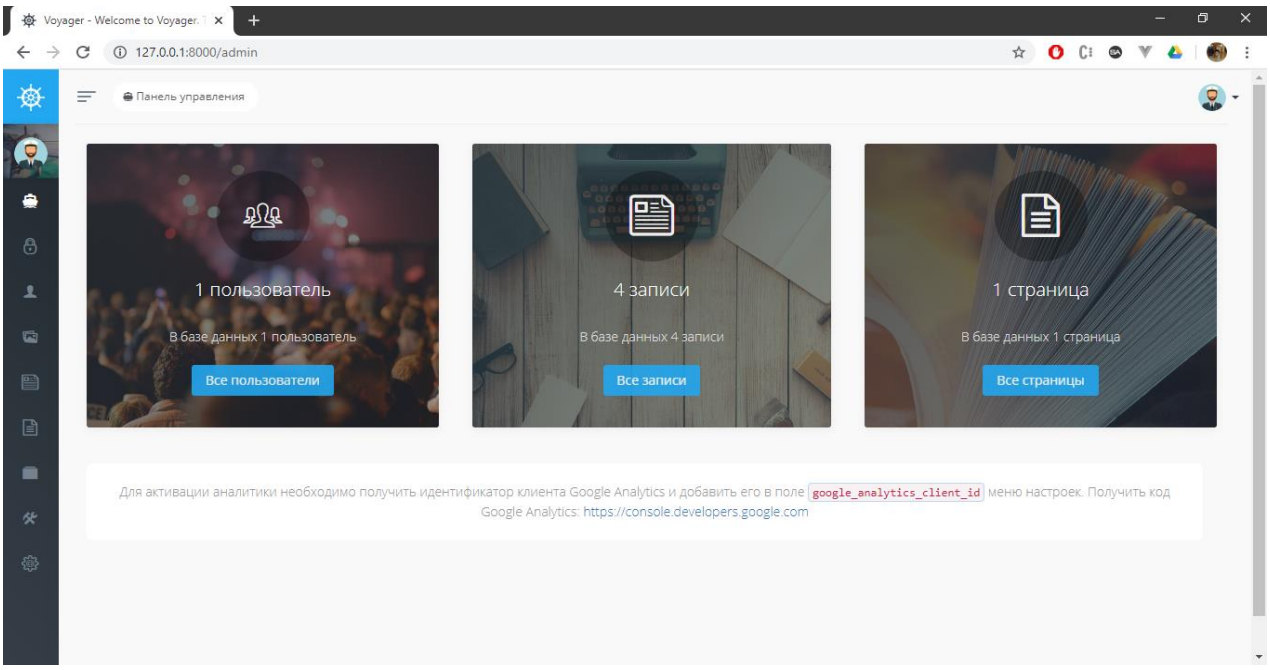

Рисунок 2.17 – Административная панель, Панель управления

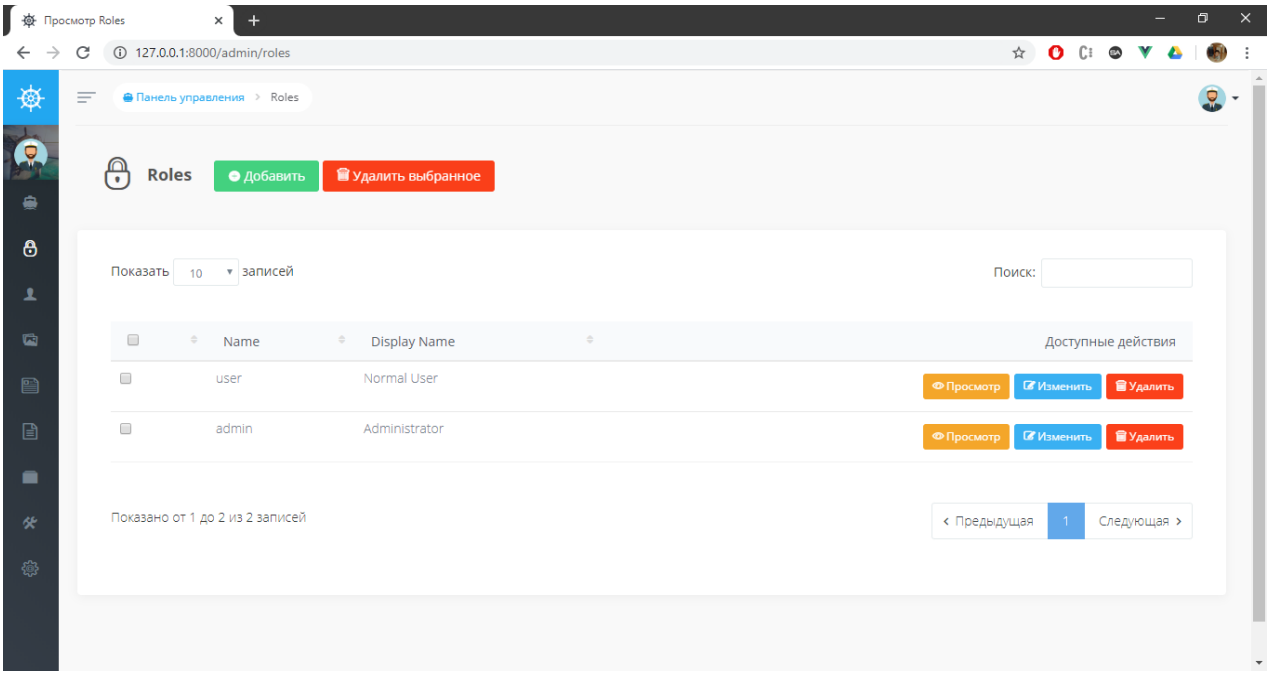

Рисунок 2.18 – Административная панель Роли

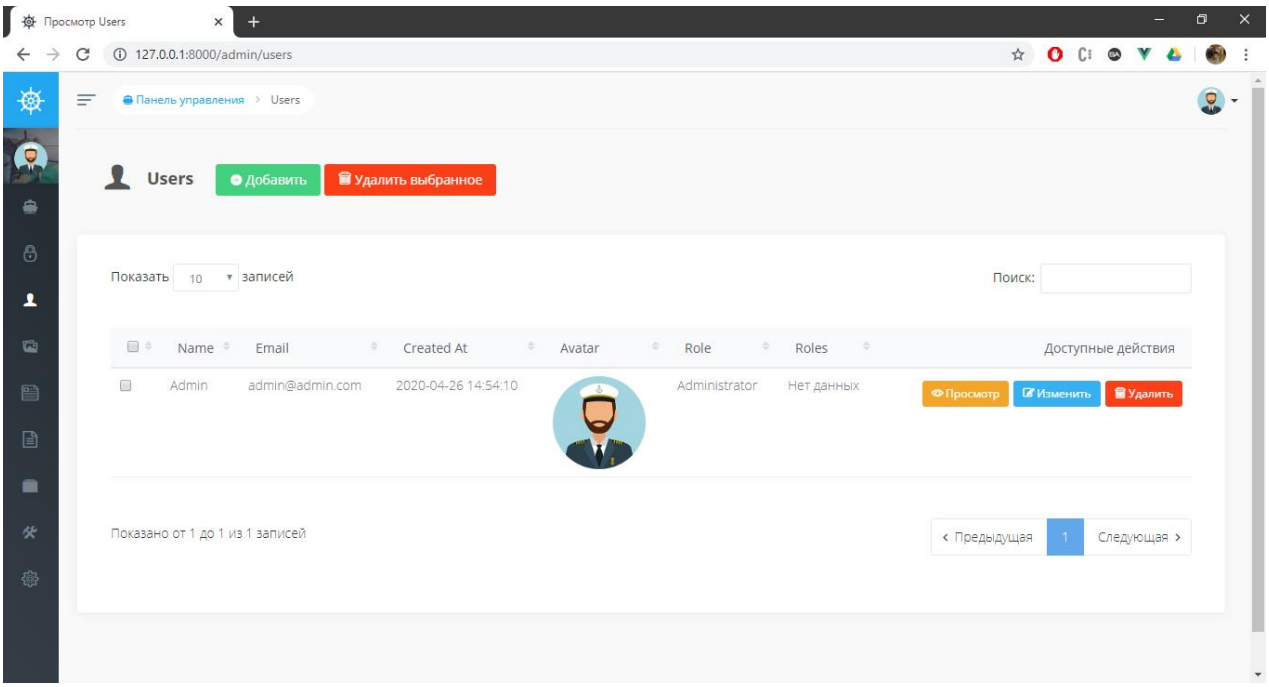

Рисунок 2.19 – Административная панель Пользователи

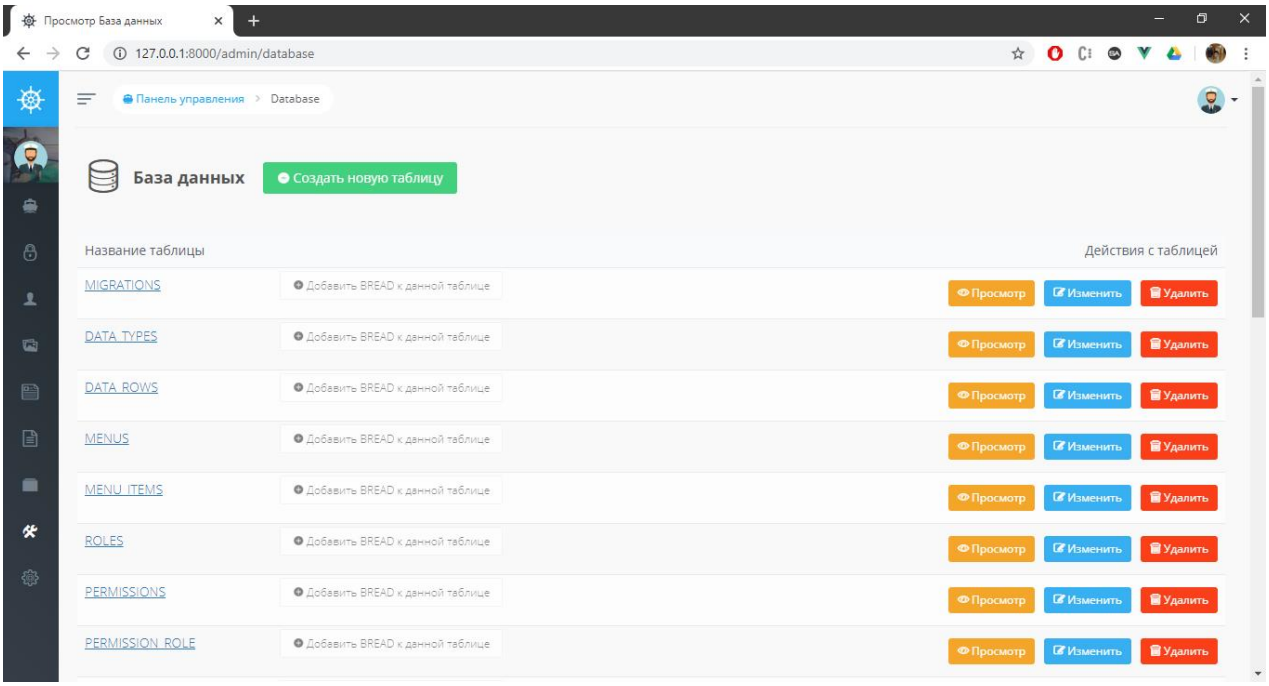

Рисунок 2.20 – Административная панель База данных

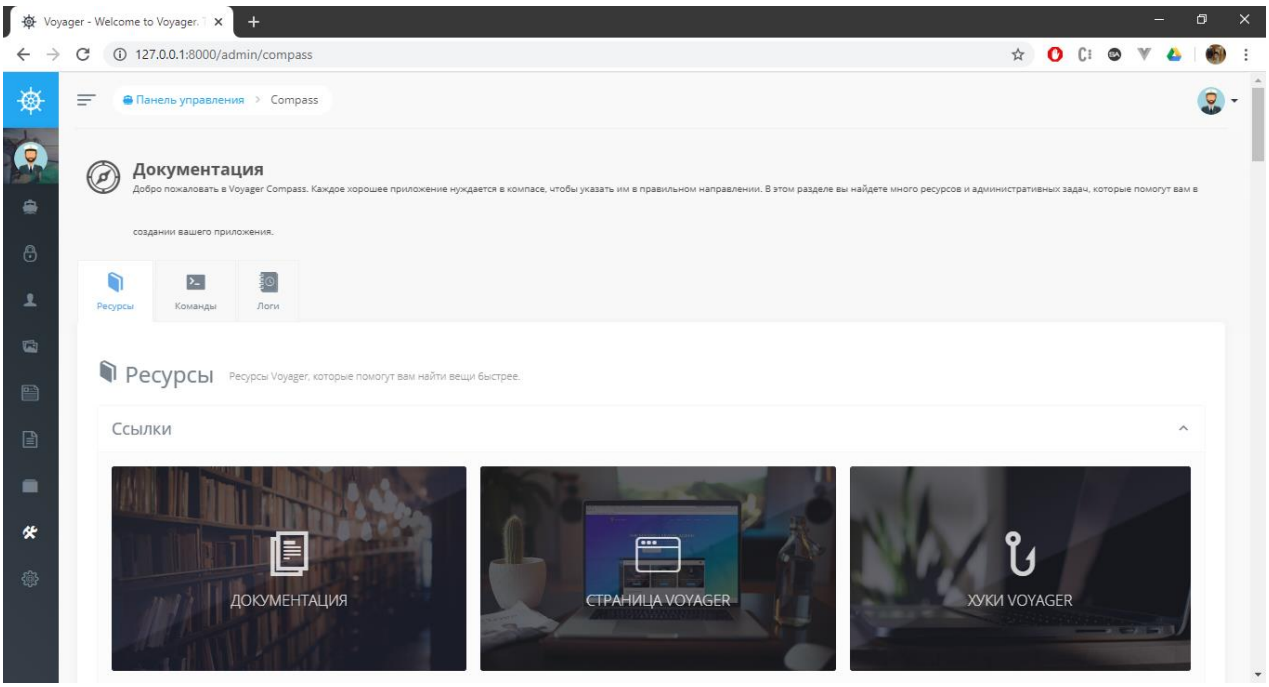

Рисунок 2.21 – Административная панель Документация

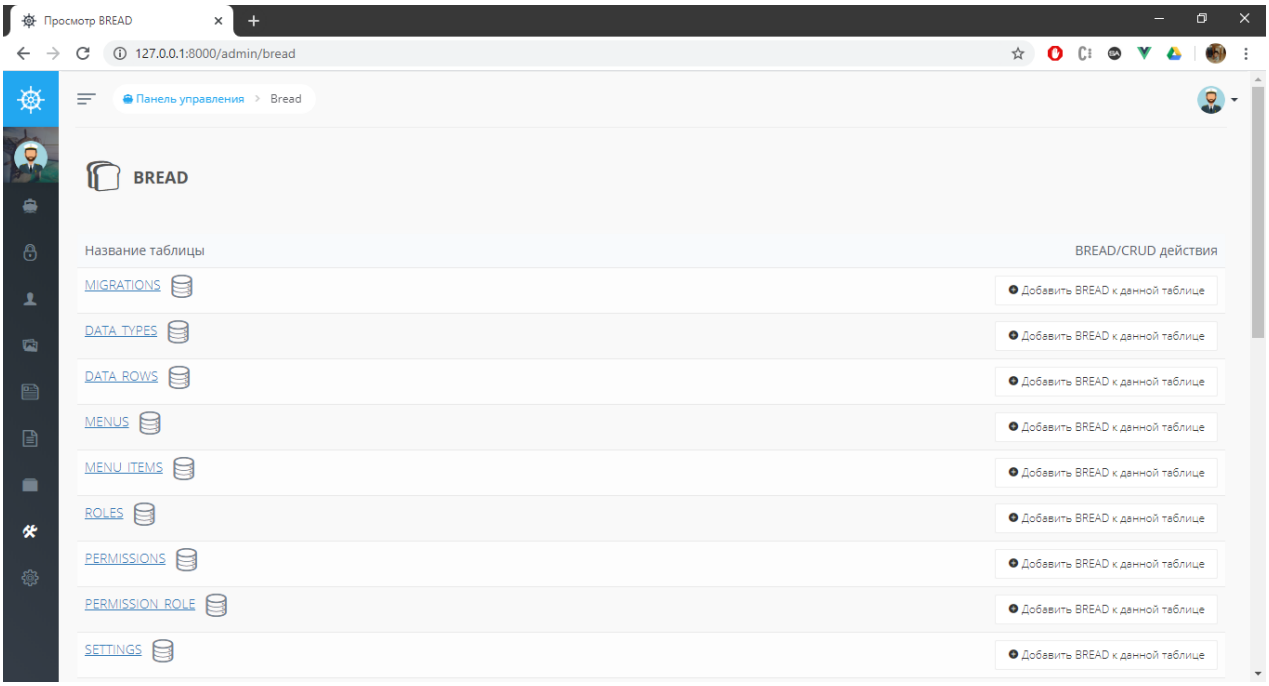

Рисунок 2.22 – Административная панель BREAD

Административная панель успешно создана и смонтирована к базе данных. Следующим шагом является создание веб – интерфейса к приложению. Перед этим отмечается, что должно быть в приложении агентства недвижимости.

Техническое задание:

– приложение должно позволять пользователям регистрироваться и авторизовываться;

– исходя из авторизации должен быть личный кабинет, с функцией добавления недвижимости в объявление;

– просмотр объявлений, несмотря на авторизацию;

– просмотр определенного объявления, также несмотря на авторизацию.

Но, прежде чем начнется разработка клиентской части приложения, обращается внимание на серверную обработку информации.

Для регистрации и авторизации нужна модель для создания пользователей в базе и модель на проверку входных данных при авторизации.

Таким образом для начала создается контроллер с названием OwnerController для пользователей, и расписывается функция создания пользователя, рисунок 2.23.

```
public function create(Request $request)
{
    $owner = new Owner;
    $owner->name = $request->name;
    $owner->email = $request->email;
    $owner->password = Hash::make($request->password);
    $owner->remember_token = Str::random(60);
    $owner->save();

    return response()->json([
        'name' => $owner->name,
        'email' => $owner->email,
        'token' => $owner->remember_token,
        'id' => $owner->id,
        'message' => 'Вы успешно зарегестрировались'
    ]);
}
```

Рисунок 2.23 – Серверная часть регистрации

Здесь идея состоит в том, что при вводе нужных полей на странице регистрации, пользователь отдает эти данные в контроллер в котором выполняется эта функция, которая создает новую учетную запись в таблице Owners и, при успешной проверке данных возвращает сообщение клиентской части.

Функция обрабатывает такие поля, как:

– имя;

– e-mail;

– пароль;

– csrf-токен.

Обращается внимание на две вещи, на то как шифруется пароль и как создается токен.

Пароль шифруется с помощью функции Hash, которая обеспечивает безопасное хэширование Bcrypt. Bcrypt, разработанная Нильсом Провосом и

Дэвидом Мациросэм, впервые была представлена в 1999 году, является самым надежным выбором хэширования для хранения паролей на данный момент, так как сложность его вычисления настраивается, это значит, что можно увеличить время генерации хэша при увеличении мощности железа.

CSRF-токен генерирует строковое значение, необходимое для авторизации и защиты данных пользователя и предотвращения CSRF-атак, формируется путем создания строки длиной в 60 различных символов. Как это будет использоваться в процессе можно объяснить тремя шагами:

– пользователь отправляет свои данные на странице авторизации;

– сервер сверяет полученный токен с токеном который есть в таблице пользователей;

– после проверки сервер отправляет клиентской части ответ.

Начинается процесс разработки авторизации пользователя, рисунок 2.24.

```
public function show(Request $request)
{
    $owner = Owner::where('email', '=', $request->email)->first();
    if ($owner) {
        if (Hash::check($request->password,  $owner->password)) {
            return response()->json([
                'name' => $owner->name,
                'email' => $owner->email,
                'token' => $owner->remember_token,
                'id' => $owner->id,
            ]);
        } else {
            return response()->json(['error' => 'Пароль введен неправильно']);
        }
    } else {
        return response()->json(['error' => 'E-mail введен неправильно']);
    }
    if ($owner === null) {
        // Owner doesn't exist
    }
}
```

Рисунок 2.24 – Серверная часть авторизации

На странице авторизации предполагаются такие поля как ввод почты и пароля, исходя от этого был дописан контроллер к таблице пользователей, на авторизацию. Функция проверяет входные данные как уже было сказано ранее, на совпадение полей e-mail и хэшами паролей, а также токенов, в случае если все данные введены корректно, то пользователь авторизовывается и отправляется в личный кабинет.

Но что же администратору делать с данными авторизованного пользователя или, если их нет, как добавлять данные? Для этого подготавливается API и настраиваются ресурсы и контроллеры для таких моделей как STREETS, DISTRICTS, CITIES, PROPERTIES, TYPES (Улицы,

Районы, Города, Недвижимости, Типы), рисунки 2.25 – 2.29. API является интерфейсом прикладного программирования для веб – сервера и веб – браузера. Позволяет клиентской и серверной части взаимодействовать друг с другом без перезагрузки страницы.

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Resources\Json\JsonResource;
use App\Http\Resources\DistrictResource;
use App\District;

class StreetResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    public function toArray($request)
    {
        return [
            'id' => (string) $this->id,
            'name' => $this->name,
            'district' => new DistrictResource(District::where('id', '=', $this->district_id)->first()),
        ];
    }
}
```

Рисунок 2.25 – Ресурс модели Streets

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Resources\Json\JsonResource;
use App\Http\Resources\CityResource;
use App\City;

class DistrictResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    public function toArray($request)
    {
        return [
            'id' => (string) $this->id,
            'name' => $this->name,
            'city' => new CityResource(City::where('id', '=', $this->city_id)->first()),
        ];
    }
}
```

Рисунок 2.26 – Ресурс модели Districts

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Resources\Json\JsonResource;

class CityResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    public function toArray($request)
    {
        return [
            'id' => (string) $this->id,
            'name' => $this->name,
        ];
    }
}
```

Рисунок 2.27 – Ресурс модели Cities

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Resources\Json\JsonResource;
use App\Http\Resources\StreetResource;
use App\Street;
use App\Type;

class PropertyResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    public function toArray($request)
    {
        return [
            'id' => (string) $this->id,
            'street' => new StreetResource(Street::where('id', '=', $this->street_id)->first()),
            'attributes' => [
                'type' => new TypeResource(Type::where('id', '=', $this->type_id)->first()),
                'owner_id' => $this->owner_id,
                'price' => $this->price,
                'description' => $this->description,
                'created_at' => $this->created_at,
            ],
        ];
    }
}
```

Рисунок 2.28 – Ресурс модели Properties

```php
<?php

use Illuminate\Http\Request;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::resource('properties', 'PropertyController');

Route::resource('streets', 'StreetController');

Route::resource('districts', 'DistrictController');

Route::resource('cities', 'CityController');

Route::resource('types', 'TypeController');

Route::resource('profile', 'PropertyController')->only([
    'store'
]);

Route::middleware('auth:api')->get('/user', function (Request $request) {
    return $request->user();
});
```

Рисунок 2.29 – Подготовка API

При формировании API, в контроллер приходят данные с ресурсов, которые были продемонстрированы выше. API будет выводить данные на определенные запросы. Адрес на который будет отправляться запрос всегда будет содержать в URL /api, а далее название модели и ресурса, к примеру, http://localhost:8000/api/streets.

С помощью программного обеспечения Postman производится запрос на проверку данных, возвращаемых сервером, рисунки 2.30, 2.31.

Welcome Page | estate agency | STREETS

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	ID	DISTRICT_ID	NAME	CREATED_AT	UPDATED_AT
1	1	1	улица Коперника	01.05.20 16:08:32,000000000	01.05.20 16:08:32,000000000
2	2	1	улица Тянь-Шанская	01.05.20 16:08:45,000000000	01.05.20 16:08:45,000000000
3	3	1	улица Фурманова	01.05.20 16:09:49,000000000	01.05.20 16:09:49,000000000
4	4	1	улица Гагарина	01.05.20 16:10:02,000000000	01.05.20 16:10:02,000000000
5	5	1	улица Жандосова	01.05.20 16:10:14,000000000	01.05.20 16:10:14,000000000

Рисунок 2.30 – Данные с таблицы Streets

GET http://localhost:8000/api/streets Send Save

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": "1",
        "name": "улица Коперника",
        "district": {
            "id": "1",
            "name": "Медеуский район",
            "city": {
                "id": "1",
                "name": "Алматы"
            }
        }
    },
    {
        "id": "2",
        "name": "улица Тянь-Шанская",
        "district": {
            "id": "1",
            "name": "Медеуский район",
            "city": {
                "id": "1",
                "name": "Алматы"
            }
        }
    },
```


Рисунок 2.31 – Ответ сервера на запрос

Как только подготавливаются все API для будущих запросов в базу, начинается создание навигации по сайту. Это осуществляется в файле web.php путем выстраивания роутов ко всем отображающим клиентскую часть файлам.

```
Route::get('/properties', function () {
    return view('properties.index');
});

Route::post('/properties', 'PropertyController@create');

Route::get('/properties/{id}', function ($id) {
    $property = DB::table('properties')->select('id')->where('id', '=', $id)->first();
    return view('properties.show', compact('property'));
});

Route::get('/create', function () {
    return view('properties.create');
});

Route::get('/profile', function () {
    return view('profile.index');
});

Route::get('/register', function () {
    return view('profile.register');
});

Route::get('/login', function () {
    return view('profile.login');
});

Route::post('/owners', 'OwnerController@create');

Route::post('/owner', 'OwnerController@show');

Route::get('/', function () {
    return view('welcome');
});

Route::get('/event-card', function () {
    return view('event.index');
});
```

Рисунок 2.32 – Настройка роутов

На рисунке 2.32 указаны настройки навигации по всем страницам веб - приложения. Роут берет значение url в виде ‘/properties’ и ищет файл который будет показываться по данному адресу, в данном случае properties.index, где properties – это папка, index – это файл, который на самом деле называется index.blade.php. Но тут автора больше интересует функция, url которого в будущем будет представлять ‘/properties/{id}’, где {id} – является входящим параметром этой функции, id недвижимости. Этот роут достает данные с таблицы Properties, путем задания условия, которая ждет пока база не вернет все, что связано с входящим параметром. К примеру, при посещении страницы http://127.0.0.1/properties/1, база вернет данные, связанные с этой id, которая будет равна 1. А клиентская часть будет эти данные обрабатывать и демонстрировать пользователю.

2.3 Настройка и создание клиентской части, Vue

И вот наконец, плавно подходит разработка клиентской части. Всё, что связано с клиентской частью, т. е. компоненты Vue [4], Scss [5] будут находиться в папке public в корне проекта.

Перед началом создания компонентов на Vue, нужно создать сервисный файл js, который будет отправлять все запросы и импортироваться в те компоненты которым нужно один или несколько из этих запросов отправить, назвав их сервисом нашей клиентской части. Но тут возникает вопрос, а как вообще отправить запрос на сервер? И будет ли это безопасно? На оба этих вопроса отвечает Axios [6], JavaScript библиотека, которая имеет встроенную защиту от мошенничества на разных сайтах (XSRF) и устанавливает ограничения на количество запросов, что предотвращает, например, DoS – атаки, которые как раз таки нацелены на то чтобы отправлять кучу запросов. Но по счастью эта функция также присутствует и в Laravel, и они абсолютно друг другу не мешают, а напротив только дополняют друг друга.

XSRF расшифровывается как подделка сайтов. Это обычный сценарий, когда, пользователь входит в свою банковскую учетную запись, на которую получает токен сеанса, хранящаяся в локальном хранилище. Через какое-то время пользователь получает электронное письмо от злоумышленника. В письме содержится ссылка, по которой, ничего не подозревающий пользователь, нажимает, и тем самым, запускает некоторый вредоносный код, который будет вести себя так как запрограммировал его хакер. Но вот в нашем случае Axios полностью защитит приложение и пользователей от таких вот писем, потому что каждый запрос фильтруется и проходит проверку прежде чем отправит ajax – запрос, рисунки 2.33, 2.34.

```
import axios from "axios";
import NProgress from "nprogress";

const apiClient = axios.create({
    baseURL: "http://127.0.0.1:8000",
    withCredentials: false,
    headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
    }
});

apiClient.interceptors.request.use(config => {
    NProgress.start();
    return config;
});

apiClient.interceptors.response.use(response => {
    NProgress.done();
    return response;
});
```

Рисунок 2.33 – Подключение библиотеки axios

```
export default {
    getProperties(page) {
        return apiClient.get("/api/properties?page=" + page);
    },
    getProperty(id) {
        return apiClient.get(`/api/properties/${id}`);
    },
    getStreets() {
        return apiClient.get("/api/streets");
    },
    getDistricts() {
        return apiClient.get("/api/districts");
    },
    getCities() {
        return apiClient.get("/api/cities");
    },
    getTypes() {
        return apiClient.get("/api/types");
    },
    postProperty(property) {
        return apiClient.post("/properties", property);
    },
    postUser(credentials) {
        return apiClient.post("/owners", credentials);
    },
    checkUser(credentials) {
        return apiClient.post("/owner", credentials);
    },
    getOwnerProperty(id) {
        return apiClient.post("/api/profile", id);
    }
};
```

Рисунок 2.34 – Создание запросов

После созданных сервисов, создается хранилище Vue, называемое Vuex [7]. Vuex является местом, в котором хранится состояние нашего приложения. В этом контейнере как уже стало понятно, есть состояние, хранящее данные, мутаций которые при необходимости будут задавать значения для данных состояния, действий которые могут вызвать мутации и другие функции, и геттеров, с помощью которых данные могут быть модифицированы, вычислены на основе состояния, рисунки 2.35 – 2.37.

```
import Vue from "vue";
import Vuex from "vuex";
import PropertyService from "../services/PropertyService.js";
const Swal = require("sweetalert2");
Vue.use(Vuex);
```

Рисунок 2.35 – Подключение Vuex

```
export default new Vuex.Store({
    state: {
        properties: [],
        streets: [],
        districts: [],
        cities: [],
        types: [],
        propertiesTotal: null,
        property: {},
        ownerProperties: []
    },
    mutations: {
        ADD_PROPERTY(state, property) {
            state.properties.push(property);
        },
        SET_PROPERTIES(state, properties) {
            state.properties = properties;
        },
        SET_TOTAL(state, total) {
            state.propertiesTotal = total;
        },
        SET_PROPERTY(state, property) {
            state.property = property;
        },
        SET_USER_DATA(state, data) {
            localStorage.setItem("user", JSON.stringify(data));
        },
        CLEAR_USER_DATA(state) {
            localStorage.removeItem("user");
            location.reload();
        },
        SET_OWNERS_PROPERTIES(state, properties) {
            state.ownerProperties = properties;
        }
    },
```

Рисунок 2.36 – Состояние и мутации

```
    actions: {
        logout({ commit }) {···
        },
        login({ commit }, credentials) {···
        },
        register({ commit }, credentials) {···
        },
        fetchOwnerProperties({ commit }, id) {···
        },
        fetchProperties({ commit }, page) {···
        },
        fetchProperty({ commit }, id) {···
        },
        createProperty({ commit }, property) {···
        }
    },
    getters: {
        getTypes: state => {···
        },
        getStreets: state => {···
        },
        getDistricts: state => {···
        },
        getCities: state => {···
        },
        getPropertyById: state => id => {···
        },
        loggedIn() {···
        }
    }
});
```

Рисунок 2.37 – Действия и геттеры

Начинается этап создания компонентов на Vue. Первым делом создается страница с недвижимостью – каталог, которая будет показывать пользователям карточки с недвижимостью и пагинацию. Этот компонент будет содержать в себе другой компонент – карточку недвижимости. Суть всех компонентов в том что, хоть они и собираются по кусочкам, они обрабатывают одни и те же данные, заданные в хранилище, которое напоминает панцирь черепахи, так как метод хранения данных в нем не позволяет злоумышленникам читать или модифицировать информацию с помощью каких бы то ни было типов атак.

Структура компонента на рисунке 2.38.

```
<template>
  <div class="pt-5">
    <h1>Недвижимости</h1>
    <item-component v-for="property in properties" :key="property.id" :property="property"></item-component>
    <ul class="pagination">
      <template v-if="page != 1">
        <li>
          <a :href="`/properties?page=${page - 1}`" rel="prev">Предыдущая страница</a>
        </li>
      </template>
      <template v-if="total > this.page * 5">
        <li>
          <a :href="`/properties?page=${page + 1}`" rel="next">Следующая страница</a>
        </li>
      </template>
    </ul>
  </div>
</template>

<script>
import ItemComponent from "./ItemComponent.vue";
import { mapState } from "vuex";
const urlParams = new URLSearchParams(window.location.search);
export default {
  components: {
    ItemComponent
  },
  data() {
    return {};
  },
  computed: {
    page() {
      return parseInt(urlParams.get("page")) || 1;
    },
    ...mapState({
      properties: "properties",
      total: "propertiesTotal"
    })
  },
  methods: {},
  created() {
    this.$store.dispatch("fetchProperties", this.page);
  }
};
</script>
```

Рисунок 2.38 – Структура компонента Каталога

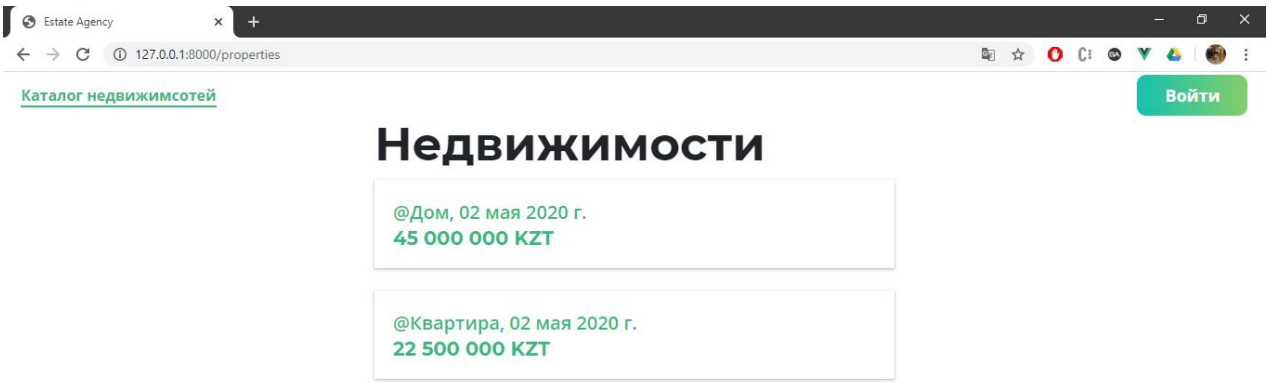

Рисунок 2.39 – Страница каталога

Структура, показанная в template не особо тривиальная, но процесс получения данных, самих недвижимостей из базы данных может заинтересовать, потому что именно тут отправляется первый запрос веб – приложения, следующие строки кода:

```
created() {
  this.$store.dispatch("fetchProperties", this.page);
}
```

Рисунок 2.40 – Страница каталога

При создании компонента, вызывается метод fetchProperties, который отправляет запрос к API и записывает данные с помощью мутации SET_PROPERTIES и SET_TOTAL в объект состояния во Vuex, хранилище, результат этих мутаций показан на рисунках 2.41 – 2.44.

```
fetchProperties({ commit }, page) {
    PropertyService.getProperties(page)
        .then(response => {
            commit("SET_TOTAL", response.data.meta.total);
            commit("SET_PROPERTIES", response.data.data);
        })
        .catch(error => {
            console.log(error);
        });
},
```

Рисунок 2.41 – Действие мутации

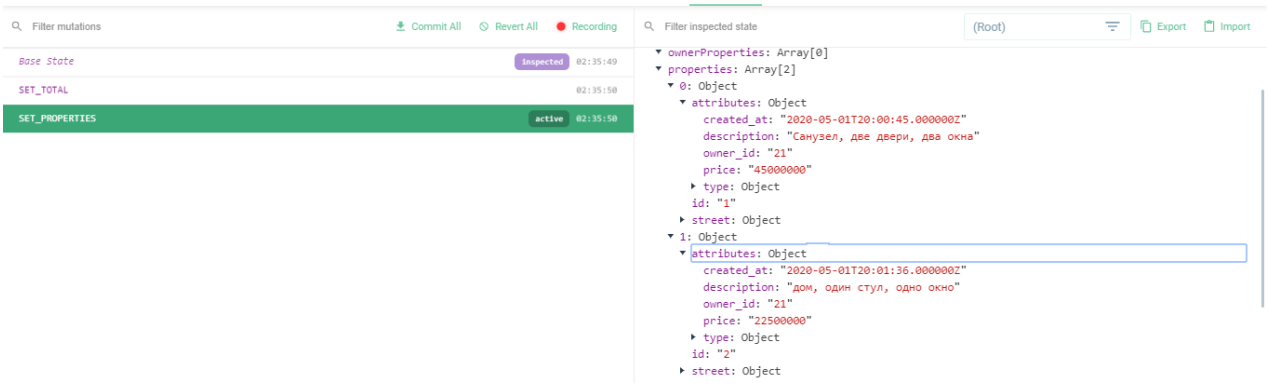

Рисунок 2.42 – Мутации при создании страницы каталога

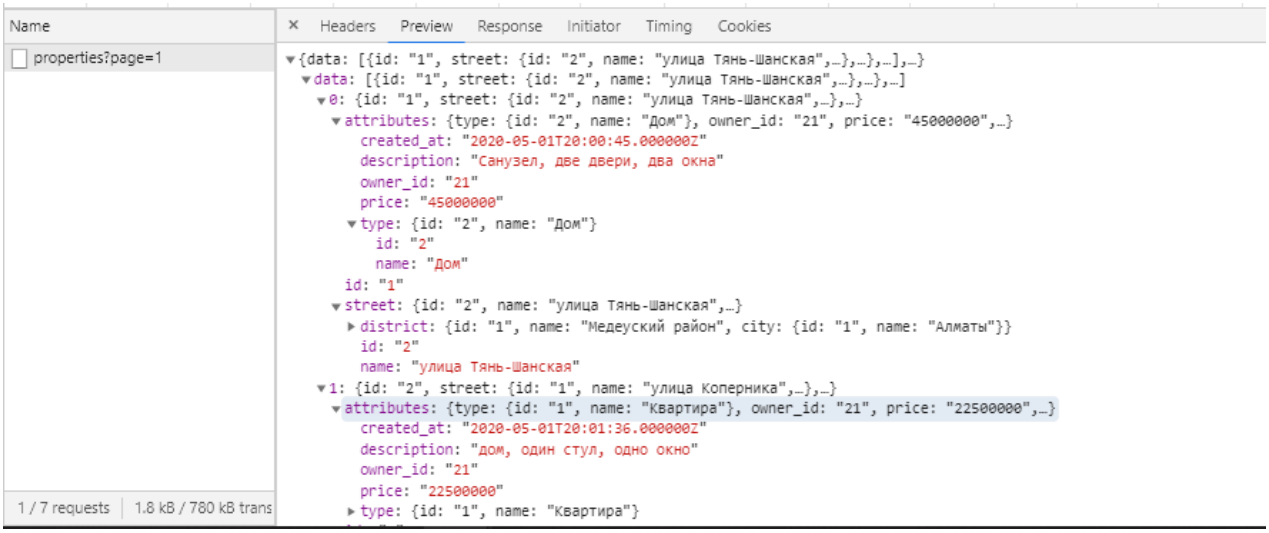

Рисунок 2.43 – Запрос к базе данных

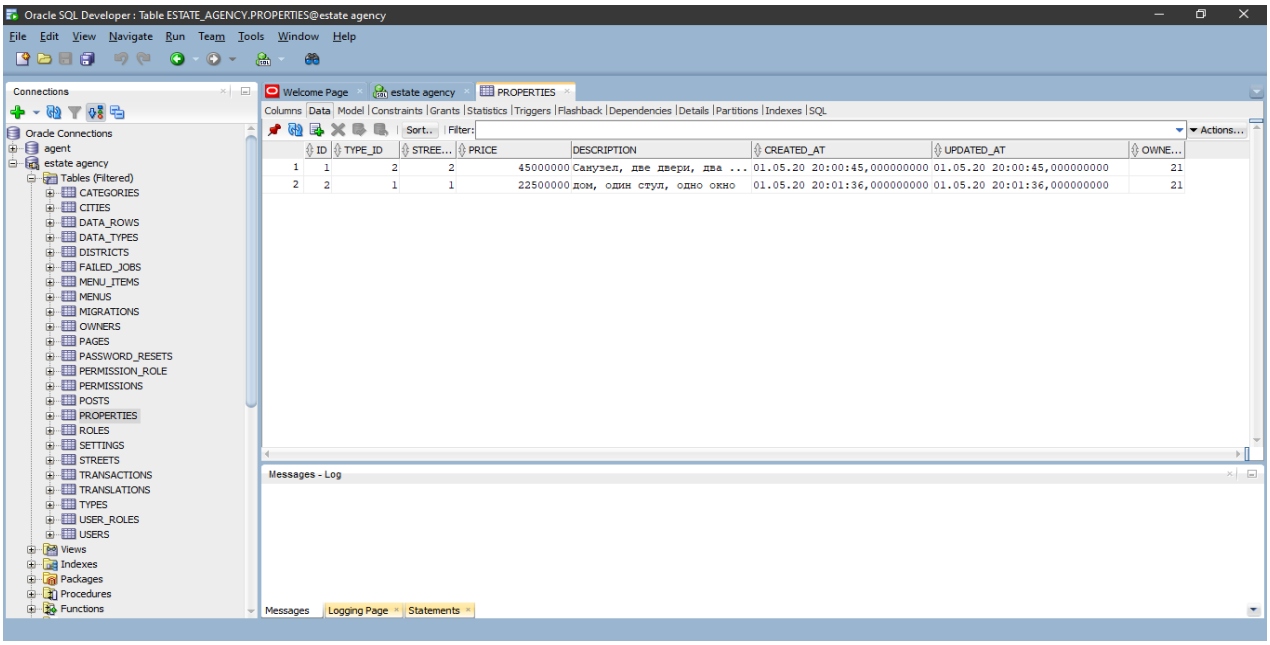

Рисунок 2.44 – Данные таблицы Недвижимости

Как уже было замечено, все входные данные получены, axios отправил запрос на получение недвижимостей из базы и запрос прошел успешно, и теперь рендер страницы принимает более логичный и структурированный характер, рисунок 2.45.

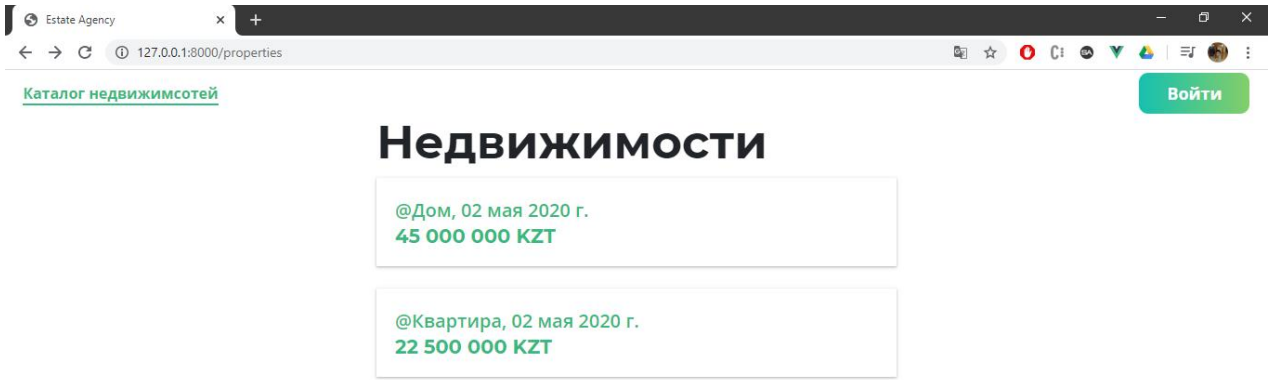

Рисунок 2.45 – Рендер страницы недвижимостей

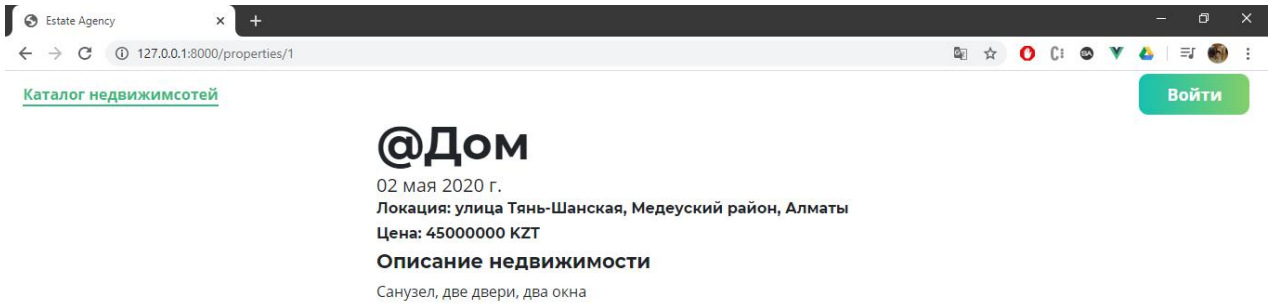

Рисунок 2.46 – Рендер внутренней страницы недвижимости

Все отрисовалось без ошибок и пользователь даже провалился во внутреннюю страницу недвижимости, рисунок 2.46, это означает что контроллеры и API, которые были созданы и скомпилированы ранее, делают свою работу, и теперь техническое задание проекта вовлекает разработку к защите всех потоков информации проходящее через данное веб - приложение.

Самое интересное происходит на странице регистрации, авторизации и личного кабинета.

На странице каталога используется GET запрос, проще говоря запрос на получение данных, на странице регистрации будет использоваться POST запрос, что позволит отправить введенные данные на сервер, серверная часть к созданию пользователя уже создана и готовка к эксплуатации. Следующим шагом, реализуется компонент и выполнение нескольких методов, рисунок 2.47.

```
<template>
    <div class="pt-5">
        <h1 class="title">Регистрация</h1>
        <form @submit.prevent="register">
            <BaseInput label="Введите ваше имя" :class="{ error: $v.name.$error }" @blur="$v.name.$touch()" v-model="name" type="text" placeholder="Имя"
            ></BaseInput>
            <template v-if="$v.name.$error">
                <p v-if="!$v.name.required" class="errorMessage">
                    Это поле обязательно к заполнению
                </p>
            </template>
            <BaseInput label="Введите ваш e-mail" :class="{ error: $v.email.$error }" @blur="$v.email.$touch()" v-model="email" type="text" placeholder="E-mail"
            ></BaseInput>
            <template v-if="$v.email.$error">
                <p v-if="!$v.email.required" class="errorMessage">
                    Это поле обязательно к заполнению
                </p>
            </template>
            <BaseInput label="Введите ваш e-mail" :class="{ error: $v.password.$error }" @blur="$v.password.$touch()" v-model="password" type="password" placeholder="Пароль"
            ></BaseInput>
            <template v-if="$v.password.$error">
                <p v-if="!$v.password.required" class="errorMessage">
                    Это поле обязательно к заполнению
                </p>
            </template>
            <BaseButton :disabled="$v.$anyError" type="submit" buttonClass="button -fill-gradient"
                >Зарегистрироваться</BaseButton
                >
        </form>
    </div>
</template>
```

Рисунок 2.47 – Компонент Регистрации

Это темплейт, формы регистрации, компонента, поля на заполнение которой, будут валидированы и защищены от внедрения туда какого – либо вредоносного кода.

```
methods: {
    register() {
        this.$v.$touch();
        if (!this.$v.$invalid) {
            this.$store
                .dispatch("register", {
                    name: this.name,
                    email: this.email,
                    password: this.password
                })
                .then(() => {
                    window.location.replace(
                        "http://127.0.0.1:8000/profile"
                    );
                });
        }
    }
}
```

Рисунок 2.48 – Компонент Регистрации

В методе на рисунке 2.48 вызывается метод register, который отправляет POST запрос, и вызывает мутацию для записи данных пользователя в состояние, а после успешной записи, перенаправляет пользователя на страницу личного кабинета, где тот может подать объявление и просмотреть имеющиеся у него на продажу недвижимостей. Результат регистрации показан на рисунках 2.49 – 2.51.

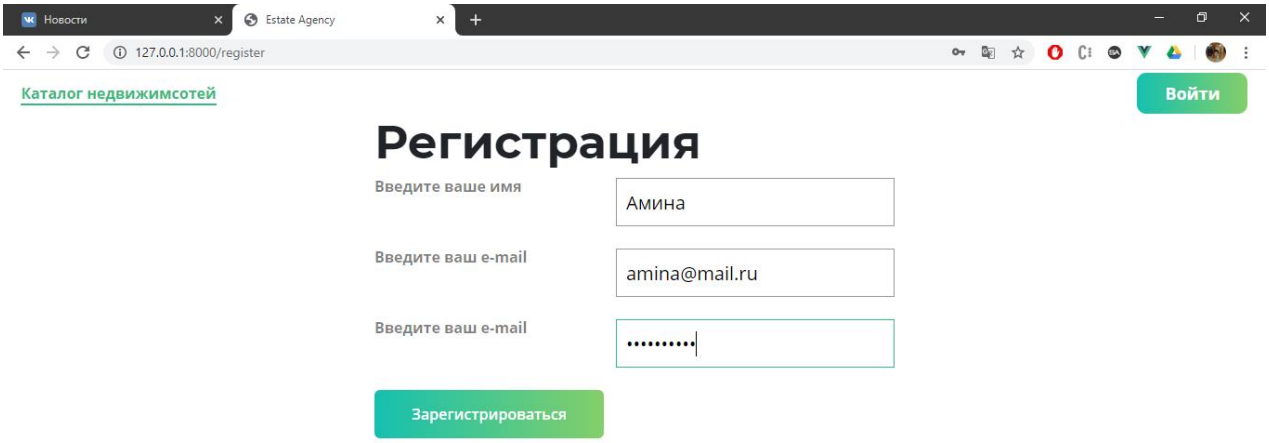

Рисунок 2.49 – Страница регистрации

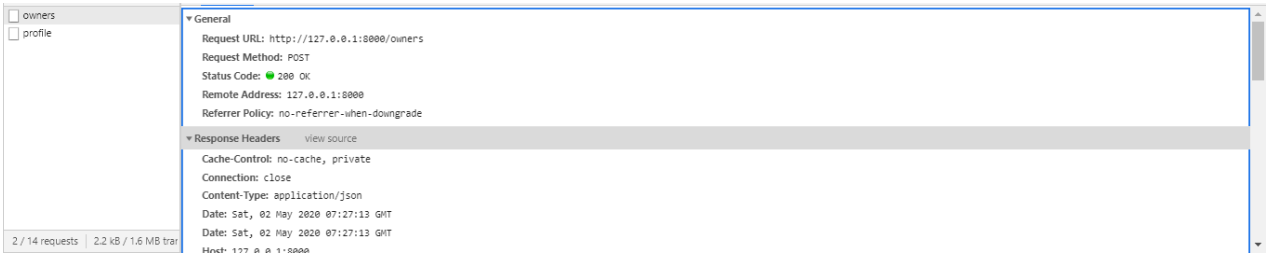

Рисунок 2.50 – Запрос на создание пользователя прошел успешно

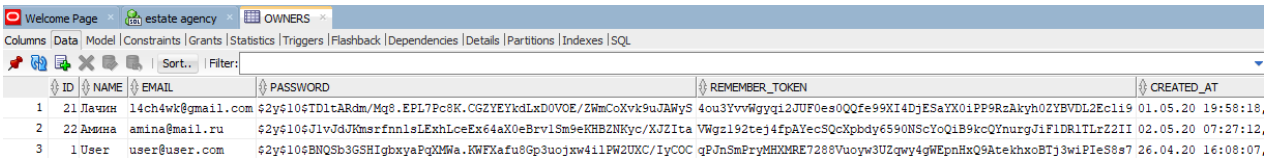

Рисунок 2.51 – Данные пользователя в таблице Owners

Как видно из таблицы Owners базы данных, пароль успешно зашифрован, и пользователю выдается запоминающий токен (CSRF – token), для того чтобы собирать данные по его запросу на странице личного кабинета.

Остается создание и настройка личного кабинета. Структура и механика личного кабинета будет состоять из двух запросов, ну и конечно же компонента.

Первый запрос на получение данных пользователя, т.е. тех недвижимостей, объявлений, которые пользователь выложил.

Второй запрос на создание этих самых объявлений и данных. Это по сути уже отдельная страница, так сказать внутренний компонент на странице личного кабинета, рисунок 2.53. В этой части приложения, будут иметься несколько своих запросов на получение данных, например на вывод всех городов, улиц, районов, чтобы пользователь мог выбрать расположение своей недвижимости.

Структура личного кабинета предоставлена на рисунке 2.52:

```
<template>
  <div class="pt-5">
    <h1>Моя недвижимость</h1>
    <div class="my-3 d-flex justify-content-end">
      <a href="/create">Создать недвижимость</a>
    </div>
    <item-component v-for="property in properties" :key="property.id" :property="property"></item-component>
  </div>
</template>

<script>
import ItemComponent from "./ItemComponent.vue";
import { mapState } from "vuex";
export default {
  components: {
    ItemComponent
  },
  data() {
    return {};
  },
  computed: {
    ...mapState({
      properties: "ownerProperties"
    })
  },
  methods: {},
  created() {
    const userData = JSON.parse(localStorage.getItem("user"));
    if (userData && userData.token) {
      this.$store.dispatch("fetchOwnerProperties", { id: userData.id });
    }
  }
};
</script>
```

Рисунок 2.52 – Структура личного кабинета

В данном компоненте выбираются все данные, которые есть у пользователя в базе. И тут возникают два вопроса:

– как эти данные сформируются и выведутся;

– как приложение поймет, какие данные вывести пользователю.

На первый вопрос ответят отношения в базе данных Oracle, суть в том что по ER – диаграмме показанной ранее, таблица PROPERTIES связана с такими таблицами как STREETS, TYPES, OWNERS один к одному, так что у каждой недвижимости есть своя улица, тип и владелец. Тем временем у каждой улицы есть свой район, а у района есть свой город.

На второй вопрос уже был дан ответ ранее, Laravel после каждой авторизации дает нам CSRF – токен, который будет расположен в виде шестидесятизначных случайных символов, чтобы их трудно было подделать. С помощью этого токена пользователь и авторизовывается и получает нужные данные.

Конечно, существует альтернативный вариант, почему бы просто не возвращать ID пользователя, чтобы данные пользователя получались по этому атрибуту. Но это крайне небезопасно, так как ID можно легко поменять, и совершенно случайным образом, в страничке у пользователя окажутся данные, которые не должны там быть, или ему не принадлежат.

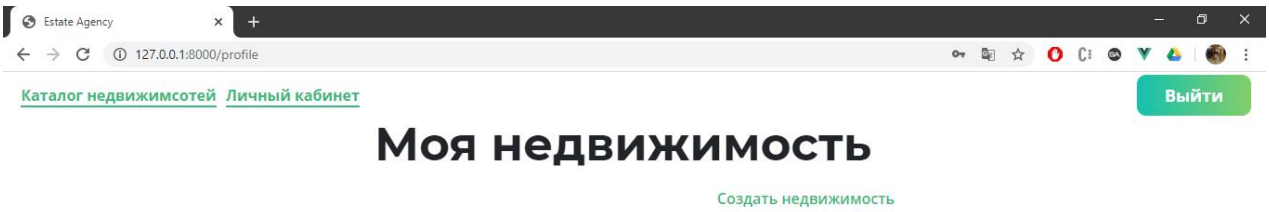

Рисунок 2.53 – Личный кабинет

Рисунок 2.54 – Запрос на данные пользователя

На рисунке 2.54 запрос, отправленный на данные пользователя вернул пустой массив, это значит, что у пользователя нет данных, объявлений. Прежде чем их создать, нужно рассмотреть код на создание недвижимостей, рисунок 2.55:

```
methods: {
  showFilteredDistricts() {
    this.filteredDistricts = this.$store.state.districts.filter(
      district => district.city.id === this.city_id
    );
  },
  showFiltredStreets() {
    this.filtredStreets = this.$store.state.streets.filter(
      street => street.district.id === this.district_id
    );
  },
  createProperty() {
    this.$v.$touch();
    if (!this.$v.$invalid) {
      this.$store.dispatch("createProperty", this.property).then(() => {
        this.property = this.createFreshPropertyObject();
      });
    }
  },
  createFreshPropertyObject() {
    return {
      type_id: null,
      street_id: null,
      owner_id: null,
      price: null,
      description: "",
      owner_id: JSON.parse(localStorage.getItem("user")).id
    };
  }
},
```

Рисунок 2.55 – Методы создание недвижимости

Здесь показываются входные данные и несколько методов, которые вызывают GET запросы на сбор данных с таких таблиц, как CITIES, STREETS, DISTRICTS. И функцию, отправляющую POST запрос на создание недвижимости, которая выполняется при условии если все данные были введены или выбраны. При вызове GET запросов, входные данные обновляются в реальном времени, как и везде в других компонентах, поэтому этот фреймворк и реактивен, а также мутируется состояние Vue, что позволяет безопасно отслеживать и передавать данные на сервер. Результат отражен на рисунках 2.56 и 2.57.

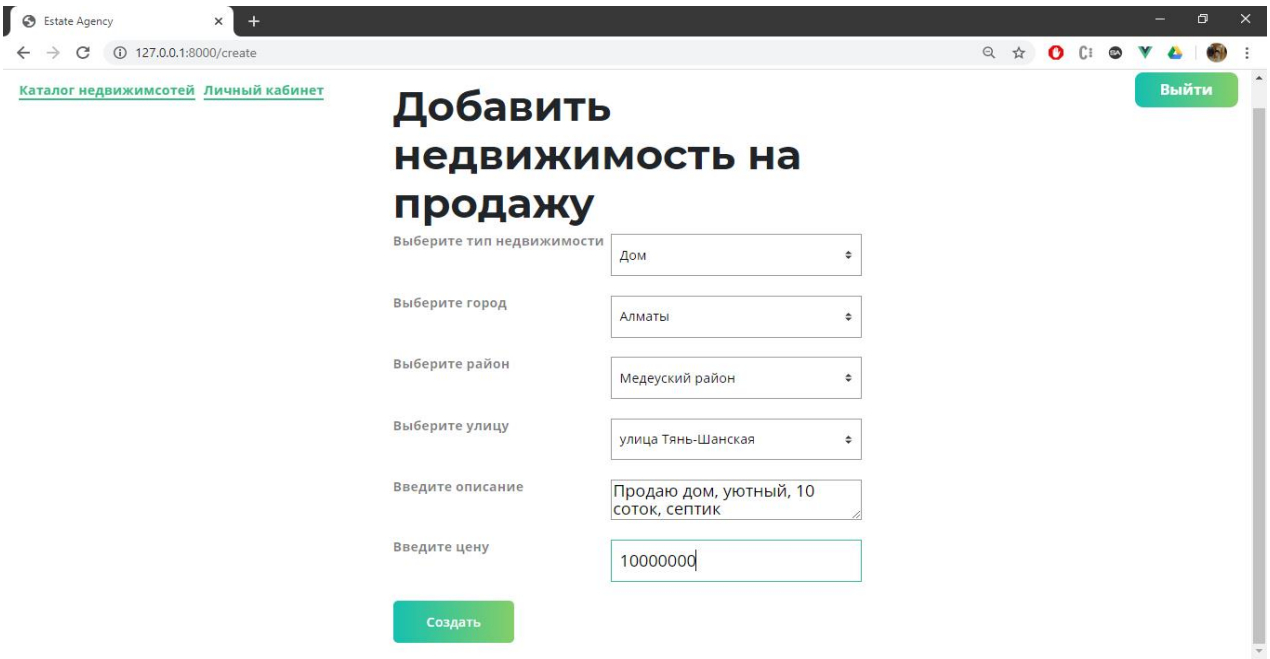

Рисунок 2.56– Страница создания недвижимости

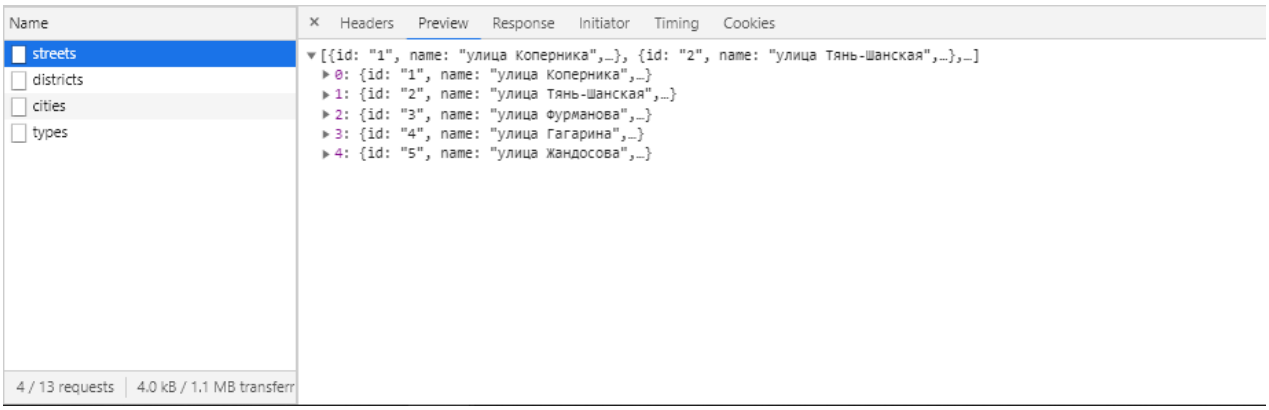

Рисунок 2.57 – GET запросы

После того как пользователь заполнит все поля и выберет все опции, ему остается просто нажать на кнопку создания недвижимости. Результат на рисунках 2.58, 2.59.

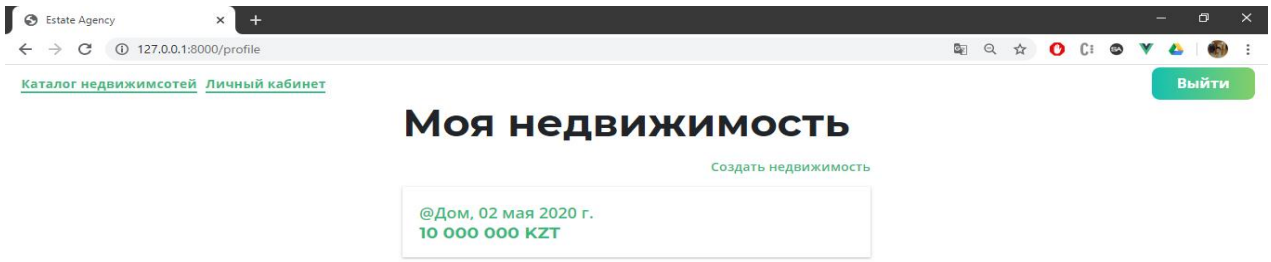

Рисунок 2.58 – Созданная недвижимости

	ID	TYPE_ID	STREE...	PRICE	DESCRIPTION	CREATED_AT	UPDATED_AT
1	3	2	2	10000000	Продаю дом, уютный, 10 соток, септик	02.05.20 10:09:58,000000000	02.05.20 10:09:58,000000000
2	1	2	2	45000000	Санузел, две двери, два окна	01.05.20 20:00:45,000000000	01.05.20 20:00:45,000000000
3	2	1	1	22500000	дом, один стул, одно окно	01.05.20 20:01:36,000000000	01.05.20 20:01:36,000000000

Рисунок 2.59 – Запись успешно сохранилась в базе

Разработка веб – приложения на Vue и Laravel и с помощью базы данных Oracle 11g подошла к эпилогу. Дальше рассматривается как и от каких типов атак он защищен.

2.4 Пример защиты от SQL – инъекций

SQL – инъекции представляют собой внедрения в адресную строку браузера обычного SQL запроса или запросов, в случае если лицо, посещающая веб – сайт является администратором и ему необходимо вывести какую – то информацию о пользователях он пользуется обычными запросами в php, но если лицом является хакер, узнав о содержимом страницы, может модифицировать запрос просто добавив * union select *.

В Laravel это практически невозможно, как уже объяснялось ранее Eloquent ORM от Laravel использует привязку параметров PDO, чтобы избежать внедрения SQL. Привязка параметров гарантирует, что злоумышленники не смогут передать данные запроса, которые могут изменить его намерения. Рассматривается, например, личный кабинет пользователя.

Предполагаемые действия злоумышленника можно расписать следующим образом:

– злоумышленник регистрируется и авторизовывается, тем самым провалившись в личный кабинет;

– злоумышленник отслеживает GET запрос и видит, что запрос проходит по адресу http://127.0.0.1:8000/api/profile с GET параметром ID, рисунок 2.60;

– злоумышленник пробует вписать в адресную сторку браузера другой GET запрос, например, на вывод данных пользователя с каким либо id, рисунок 2.61.

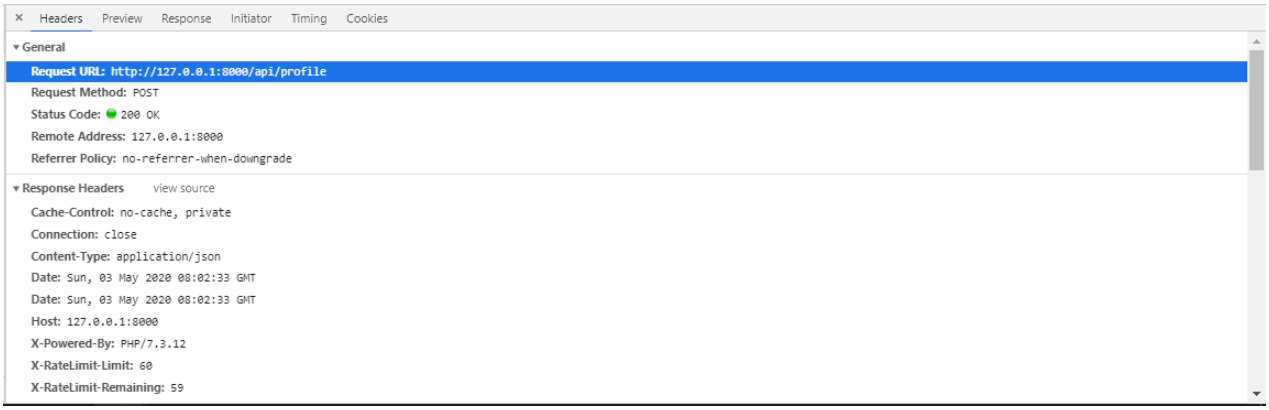

Рисунок 2.60 – GET – запрос на профиль пользователя

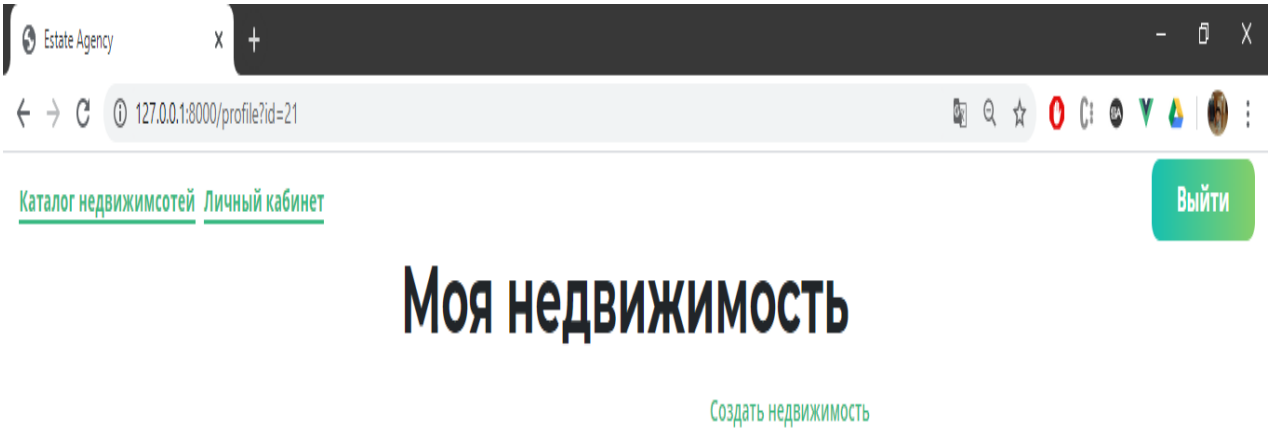

Рисунок 2.61 – Попытка злоумышленника внедрить SQL – инъекцию

К счастью для пользователя, и к большому огорчению для злоумышленника, внедрить SQL запрос ему не позволят, так как Данные выводятся не простым использованием выборки данных, например:

SELECT * FROM Properties WHERE (id = {id})

Это было бы вполне логично смотреться и выполнятся при использовании сырого PHP. Но при использовании фреймворка Laravel, все выглядит куда более структурировано и безопасно, рисунок 2.62.

```php
public function create(Request $request)
{
    $property = new Property;
    $property->description = $request->description;
    $property->price = $request->price;
    $property->type_id = $request->type_id;
    $property->street_id = $request->street_id;
    $property->owner_id = $request->owner_id;
    $property->save();

    return response()->json([
        'message' => 'Ваша недвижимость добавлена'
    ]);
}
/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $property = Property::where('owner_id', '=', $request->id)->get();
    if ($property) {
        return PropertyResource::collection($property);
    }
    if ($property === null) {
        return;
    }
}
```

Рисунок 2.62 – Структура контроллера недвижимостей

Здесь Laravel не обращается к базе напрямую, а к модели существующей таблицы, Property. Приложение пользуется виртуальным посредником базы, так, что оно обращается не к самой базе, а к ее зеркальному отражению, которое прежде чем выводить данные или их изменять, спрашивает, у физической базы, можно ли это сделать. Таким образом, Laravel блокирует все SQL – инъекции.

К тому же злоумышленник не обратил внимание на то, что существует XSRF токен, который является ключевым фактором к выводу информации о пользователе. То есть, помимо того, что невозможно внедрить запросы к базе, в базе данных будет лежать запоминающий токен. Суть этого раздела заключается в том, что нельзя модифицировать GET или POST запрос.

2.5 Пример защиты от XSS – атак

При рассмотрении кода уязвимой страницы, элементы формы, т.е. поля будут иметь следующий вид:

```
<input name="username" value="<? Echo $_GET['username'] ?>">
```

Злоумышленнику достаточно сформировать URL формата:

http://www.server.com/index.php?username="><script>alert(1)</script>

Страница будет уже содержать следующий код:

```
<input name="username" value=""><script>alert(1)</script>">
```

Данный код не причиняет вреда, а лишь демонстрирует уязвимость XSS, выводя диалоговое окно на страницу. Нужно хранить в уме то, что код может быть сформулирован по-другому, став более опасным.

Чтобы защитить веб – приложение от такого типа атаки, нужно для начала разобраться в чем состоит уязвимость и где она может находиться. По предыдущему примеру стало ясно то, что обезопасить нужно формы. В данном проекте есть две формы, на странице авторизации и на странице регистрации.

Ответ на то, что нужно обезопасить получен, теперь возникает вполне логичный вопрос. Как обезопасить формы?

– первым делом, необходимо установить валидацию входных данных, чтобы например в поле e-mail, не было таких вот скриптовых тегов, а в базе не вызывался скрипт, который мог бы сломать наше приложение;

– затем, необходимо распарсить все запросы, для того чтобы в будущем, в GET параметрах не было, лишних символов и прочих небезопасных для нашего приложения байтов.

Для валидации входных данный при авторизации и регистрации автор будет использовать библиотеку Vuelidate, позволяющая прямо из коробки валидировать все данные которые проходят через наше приложение.

Чтобы подключить библиотеку к данному веб – сайту используется следующая строка кода:

```
Import { required, email } from "vuelidate/lib/validators";
```

А затем устанавливается и внедряется библиотека для валидирования поле e-mail, результат на рисунке 2.63:

```
<template v-if="$v.email.$error"><p v-if="!$v.email.required" class="errorMessage">E-mail введен неправильно</p></template>
```

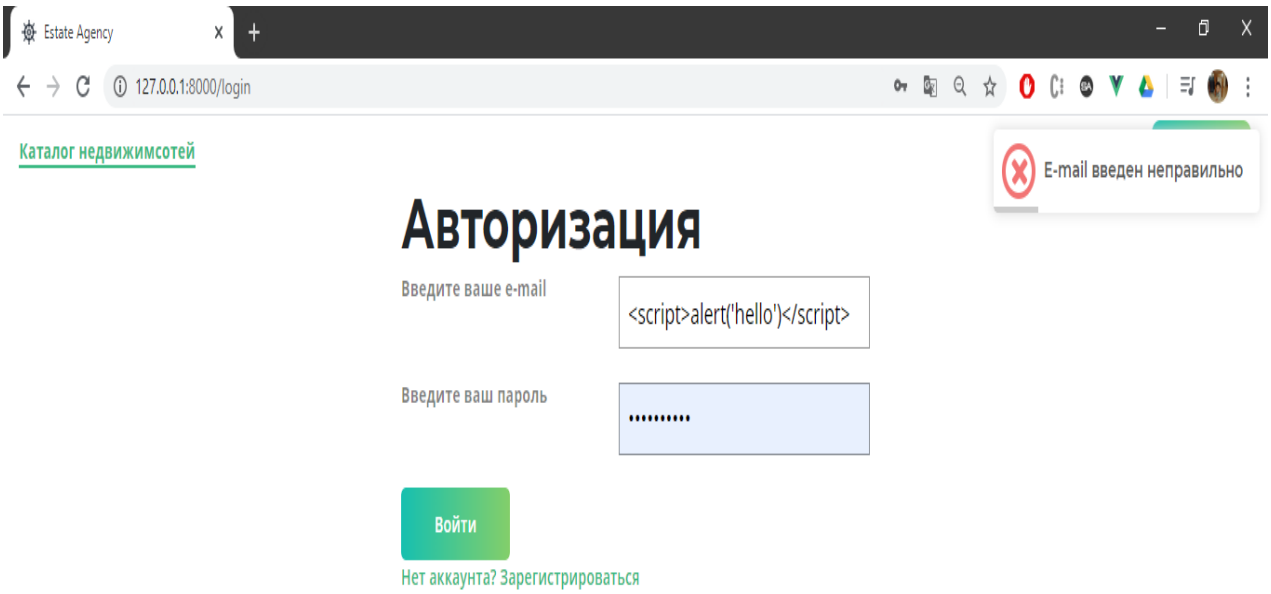

Рисунок 2.63 – Валидация на странице авторизации

Для парса запросов, стоит создать новый контроллер, в котором и настроится каким образом будут обрабатываться запросы, которые содержат в себе сомнительного характера контент. Для этого создаются три функции, рисунок 2.64

```
public function __construct(Request $request)
{
    $this->request = $request;
}

/**
 * @param Builder $builder
 */
public function apply(Builder $builder)
{
    $this->builder = $builder;

    foreach ($this->fields() as $field => $value) {
        $method = camel_case($field);
        if (method_exists($this, $method)) {
            call_user_func_array([$this, $method], (array) $value);
        }
    }
}

/**
 * @return array
 */
protected function fields(): array
{
    return array_filter(
        array_map('trim', $this->request->all())
    );
}
```

Рисунок 2.64 – Контроллер обработки запросов

Первая функция отвечает за просмотр запроса и последующую передачу параметров следующим. Функция apply пропускает те запросы, которые не

содержать в себе, символы вроде “<”, “>” или символов в форме camelCase, верблюжий стиль написания, обычно свойственен для скриптовых функций.

В противном случае, последняя будет срезать и парсить запрос, рисунок 2.65, 2.66

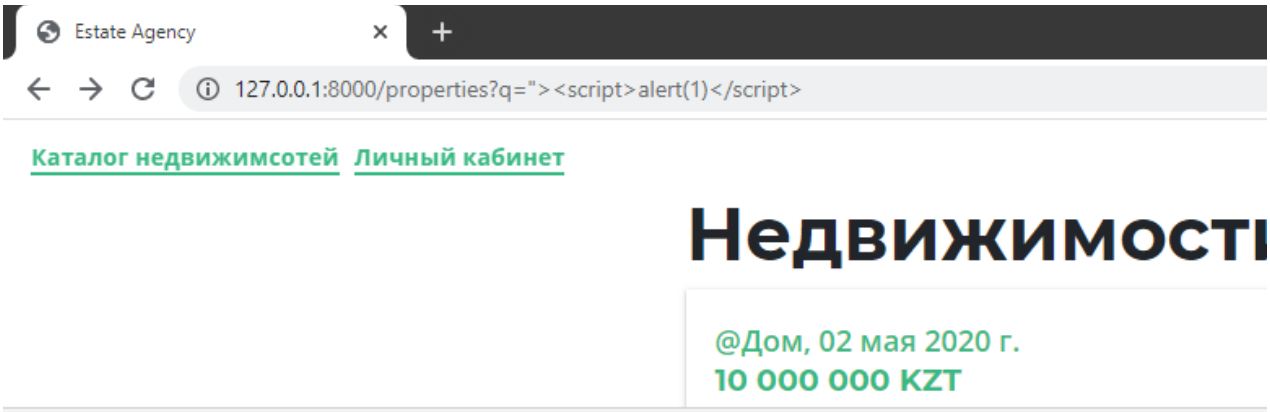

Рисунок 2.65 – Пример XSS внедрения

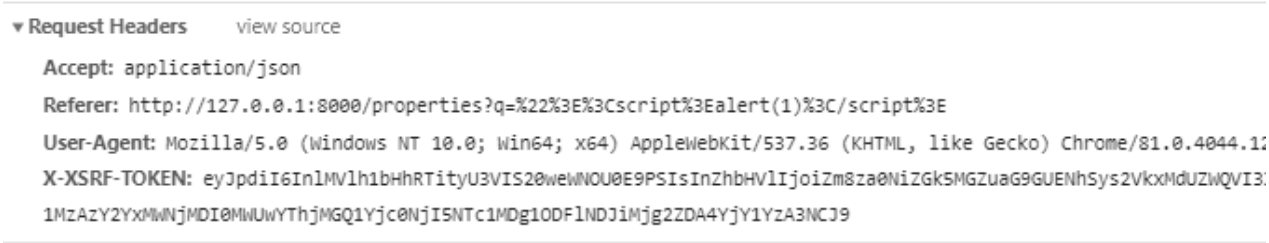

Рисунок 2.66 – Успешный парсинг GET – параметра

3 Безопасность жизнедеятельности

В данной дипломной работе рассматривается использование и разработка веб – приложения и работы с недвижимостью на удаленной основе, т.е. онлайн консультирование. Разработанный комплекс будет располагаться в офисе.

3.1 Анализ условий труда сотрудников офиса

Офис находится на 2 этаже состоит из 6 помещений в состав которых входит:

– кухня;

– кабинет директора;

– офисный отдел;

– уборная;

– переговорная;

– место хранения техники;

– график работы в будние дни с 09-00 до 18-00, без учетов праздников и вынужденного выхода на работу по разным обстоятельствам.

В офисе работают 20 человек и для создания условий для безопасной, и продуктивной работы необходимо учитывать следующие требования:

– требования к видеодисплейным терминалам и персональным электронно-вычислительным машинам;

– требования к помещениям для эксплуатации ВДТ и ПЭМ;

– требования к освещению помещений и рабочих мест;

– требования к шуму и вибрации;

– требования к организации рабочего места;

– требования к организации рабочего места профессионального пользователя;

– требования к организации режима работы с ВДТ и ПЭВМ студентов высших учебных заведений.

На все эти требования предусмотренные нормативы утвержденные СанПин № 1.01.004.01 [8].

Настоящие санитарные правила и нормы (в дальнейшем Правила) "Гигиенические требования к организации и условиям работы с видеодисплейными терминалами и персональными электронно-вычислительными машина-мл" регламентируют гигиенические требования к проектированию, размещению и эксплуатации отечественных и импортных видеодисплейных терминалов (ВДТ) с электронно-лучевыми, жидкокристаллическими и плазменными трубками, используемыми во всех типах электронно-вычислительных машин, в производственном оборудовании и игровых комплексах на базе персональных электронно-вычислительных машин (ПЭВМ), персональных компьютеров (ПК) и электронных систем (ЭС) на их основе, принадлежащих министерствам, ведомствам, хозяйствующим субъектам независимо от подчиненности и форм собственности [8].

Государственный санитарно-эпидемиологический надзор за соблюдением настоящих Правил государственными органами, а также всеми предприятиями, организациями и учреждениями, должностными лицами и отдельными гражданами возлагается на органы и учреждения Государственной санитарно-эпидемиологической службы Министерства здравоохранения Республики Казахстан (ст. 16 Закона Республики Казахстан "О санитарно-эпидемиологическом благополучии населения", "Положение о Государственной санитарно-эпидемиологической службе Республики Казахстан", утвержденное Постановлением Кабинета Министров Республики Казахстан 25.04.95т. N 547) [9].

Санитарные правила и нормы являются республиканским нормативным документом и обязательны для соблюдения министерствами, ведомствами, учреждениями, организациями, другими юридическими» физическими лицами, независимо от подчиненности и форм собственности, проектирующими, эксплуатирующими и ремонтирующими БДТ и ПЭВМ (ст. 18 Закона РК "О санитарно-эпидемиологическом благополучии населения").

Нарушение санитарных правил и норм влечет за собой дисциплинарную, административную или уголовную ответственность, должностных лиц, специалистов и работников организации и учреждении, ученных заведений физических лиц, осуществляющих разработку, производство, закупку, реализацию и применение ВДТ, ПЭВМ, ПК и ЭС, независимо от их ведомственной принадлежности, в соответствии с законодательством Республики Казахстан [10].

С вводом в действие настоящих Правил утрачивают силу "Санитарные правила и нормы для работников вычислительных центров - N1.10.076-94 и Санитарно-гигиенические нормы и правила устройства, оборудования, содержания и режима работы на персональных машинах и видеодисплейных терминалах в кабинетах вычислительной техникой и дисплейных классах всех типов средних учебных заведений" N1.10.077-44" утвержденные Минздравом РК 22.08.94 года [8].

Лица, прошедшие медицинское обследование, начальный инструктаж, а также знакомые с требованиями безопасности труда группы по электробезопасности, могут работать на персональном компьютере (ПК).

При работе на персональном компьютере сотрудник обязан:

– выполнять только ту работу, которая определена его должностной инструкцией;

– соблюдать правила внутреннего трудового распорядка;

– соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности;

– правильно использовать средства индивидуальной и коллективной защиты;

– соблюдать требования по охране труда;

– немедленно уведомлять своего непосредственного руководителя или руководителя высшего звена о любой ситуации, которая угрожает жизни и здоровью людей, обо всех несчастных случаях на рабочем месте или об ухудшении его состояния здоровья, включая признаки острого профессионального заболевания (отравления);

– обучаться безопасным методам и методам выполнения работы и оказания первой помощи пострадавшим на работе, инструктажу по охране труда, проверке знаний требований охраны труда;

– сдать обязательные периодические (во время работы) медицинские осмотры, а также проходить внеочередные медицинские осмотры по указанию работодателя в случаях, предусмотренных Трудовым кодексом и другими федеральными законами;

– уметь оказывать первую помощь жертвам электрического тока и других несчастных случаев;

– уметь применять первичные средства пожаротушения.

При работе с персональным компьютером на работника могут влиять следующие опасные и вредные производственные факторы:

– повышенный уровень электромагнитного излучения;

– повышенный уровень статического электричества;

– низкая ионизация воздуха;

– статическая, физическая перегрузка;

– перенапряжение визуальных анализаторов;

– недостаточная освещенность на рабочем месте.

Конструкция ПК должна обеспечивать возможность вращения корпуса в горизонтальной и вертикальной плоскости с фиксацией в указанном положении для обеспечения наблюдения за экраном ВДТ на переднем крае. Конструкция ПК должна предусматривать окраску корпуса в спокойных мягких тонах с рассеянным рассеиванием света. Корпус ПК, клавиатура и другие блоки и устройства ПК должны иметь матовую поверхность с коэффициентом отражения 0,4–0,6 и не иметь блестящих деталей, которые могут создавать блики.

Конструкция ВДТ должна предусматривать регулировку яркости и контрастности.

Площадь одной рабочей станции для пользователей ПК с ВДТ на основе электронно-лучевой трубки (ЭЛТ) должна составлять не менее 6 м2, в помещениях культурно-развлекательных учреждений и с ВДТ на основе плоских дискретных экранов (жидкокристаллических, плазма) - 4,5 м2.

При использовании ELEM с ЭЛТ на основе ЭЛТ (без вспомогательных устройств - принтера, сканера и т. д.), Которые отвечают требованиям международных стандартов компьютерной безопасности, с рабочим временем менее 4 часов в день, минимум допускается площадь 4,5 м2 на рабочую станцию (взрослый и студент высшего профессионального образования).

Помещения, где расположены рабочие места с помощью ПК, должны быть оснащены защитным заземлением в соответствии с техническими требованиями по эксплуатации.

Рабочие станции с компьютерами должны быть расположены таким образом, чтобы расстояние от экрана одного видеомонитора до задней части другого составляло не менее 2 м, а расстояние между боковыми поверхностями видеомонитора не менее 1,2 м.

Рабочие столы должны быть расположены таким образом, чтобы терминалы видеодисплея были расположены рядом со световыми проемами, чтобы естественный свет падал в основном влево.

Оконные проемы в помещениях, где используются персональные компьютеры, должны быть оснащены регулируемыми устройствами, такими как: жалюзи, шторы, внешние козырьки и т. д.

Искусственное освещение помещений для работы персонального компьютера должно осуществляться системой общего равномерного освещения. В производственных и административно-общественных помещениях, в случае первичной работы с документами, должны использоваться комбинированные системы освещения (в дополнение к общему освещению также устанавливаются локальные светильники для размещения документов).

Экран видеомонитора должен находиться от глаз пользователя на расстоянии от 600 до 700 мм, но не ближе 500 мм, с учетом размера буквенно-цифровых символов и символов.

Рабочая мебель для пользователей компьютерной техники должна соответствовать следующим требованиям:

– высота рабочей поверхности стола должна быть отрегулирована в пределах 680-800 мм; при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм;

– рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной не менее 500 мм, глубиной не менее 450 мм на уровне колен и не менее 650 мм на уровне вытянутых ног;

– рабочее кресло должно быть подъемным, регулироваться по высоте и углу наклона сиденья и спинки, а также по расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легкой осуществляемой и иметь надежную фиксацию;

– рабочее место должно быть оборудовано подставкой для ног шириной не менее 300 мм, глубиной не менее 400 мм, регулировка высоты в пределах 150 мм и углом наклона опорной поверхности подложки до 20 °; Поверхность подставки должна быть рифленой и иметь кромку высотой 10 мм вдоль передней кромки;

– клавиатуру следует размещать на поверхности стола на расстоянии 100–300 мм от края, обращенного к пользователю, или на специальной регулируемой по высоте рабочей поверхности, отделенной от основной столешницы.

В помещениях, оборудованных ПК, проводить ежедневную влажную уборку и систематическую вентиляцию после каждого часа работы на ПК.

Женщин с момента беременности переводят на работу, не связанную с использованием персональных компьютеров, или для них, ограниченную временем работы с ПК (не более 3 часов в рабочую смену).

В случае травмы или недомогания необходимо прекратить работу, уведомить руководителя о работе и обратиться в медицинское учреждение.

Требования по безопасности жизнедеятельности**:**

– подготовьте рабочее место;

– отрегулируйте освещение на рабочем месте, убедитесь, что на экране нет бликов;

– убедитесь, что оборудование правильно подключено к электросети;

– Проверьте провода питания и оголенные провода;

– убедитесь, что системный блок, монитор и защитный экран заземлены;

– протрите поверхность экрана монитора и защитного экрана антистатической тканью;

– проверьте, правильно ли установлены стол, стул, подставка для ног, подставка для колонки, угол экрана, положение клавиатуры, положение мыши на специальном коврике, при необходимости отрегулируйте рабочий стол и стулья, а также расположение компьютерных компонентов в соответствии с Требования эргономики и с целью исключения неудобных поз и длительных нагрузок на организм.

При работе на ПК сотруднику запрещается:

– прикасаться к задней панели системного блока (процессора) при включенном питании;

– включать разъемы интерфейсных кабелей периферийных устройств при включенном питании;

– позволять влаге проникать на поверхность системного блока (процессора), монитора, рабочей поверхности клавиатуры, дисководов, принтеров и других устройств;

– производить самостоятельное вскрытие и ремонт оборудования;

– работать на компьютере со снятым корпусом;

– отключите оборудование от сети и вытащите вилку из розетки, держась за шнур.

Во время регламентированных перерывов в целях снижения нервно-эмоционального напряжения, утомления зрительного анализатора, устранения влияния гиподинамии и гипокинезии, предотвращения развития познотонической усталости при выполнении комплексов упражнений.

Во всех случаях обрыва проводов электропитания, неисправности заземления и других повреждений, появления ожогов, немедленно отключите питание и сообщите о чрезвычайной ситуации руководителю.

Не запускайте устройство до устранения неисправностей.

В случае пожара:

– немедленно сообщить «01» пожарной команде, уведомить рабочих, уведомить начальника подразделения, сообщить о пожаре на пост охраны;

– откройте аварийные выходы из здания, отключите питание, закройте окна и закройте двери;

– начните тушить пожар первичными средствами пожаротушения, если это не представляет опасности для жизни;

– организовать собрание пожарной команды;

– выйдите из здания и окажитесь в зоне эвакуации.

В случае аварии:

– немедленно организовать первую помощь пострадавшему и, при необходимости, доставить его в медицинскую организацию;

– принять срочные меры для предотвращения развития чрезвычайной или другой чрезвычайной ситуации и воздействия травмирующих факторов на других людей.

3.2 Расчеты

Воздухообменом называют замену загрязненного воздуха чистым. Воздухообмен делят на естественный и искусственный. Естественный происходит вследствие разности и перепада давления воздуха внутри помещения и снаружи. Осуществляется он с помощью периодического открывания форточек, фрамуг, окон (аэрация), а также через щели стен, окон, двери (инфильтрация).

Искусственный воздухообмен осуществляется путем использования различных систем механической вентиляции и кондиционирования.

Кратность воздухообмена определяет, сколько раз в час необходимо менять весь воздух помещения, чтобы очистить его до предела допустимой концентрации загрязнения (ПДК).

Кратность воздухообмена N задается формулой:

$$N = \frac{V}{W}, \tag{3.1}$$

где V(м^3/ч) – необходимое количество чистого воздуха, поступающего в помещение в течение 1 часа;

W(м^3) – объем помещения.

Путем естественной аэрации обычно достигают трех – четырехкратного воздухообмена, а при необходимости большей кратности прибегают к механической вентиляции.

Объем чистого приточного воздуха, который должен разбавлять вредные газы до предельно допустимой концентрации, определяется по формуле:

$$V = \frac{B}{p_b - p_o}, \tag{3.2}$$

где В– количество вредного вещества (газа), поступающего в помещение в 1 час, мг/ч;

ρ_B - ПДК вредного вещества в воздухе рабочего помещения, мг/м^3;

ρ_0 – концентрация того же вредного вещества в приточном наружном воздухе, мг/м^3.

Количество вредных газов В, находящихся в воздухе рабочего помещения, можно определить несколькими способами:

Измерением концентрации газа на единицу объема b с помощью газоанализатора. Тогда количество вредного вещества определяется по формуле:

$$B = a * b * W, \quad (3.3)$$

где a – коэффициент инфильтрации (для камеральных цехов a=1, для гаражей a=2);

b – концентрация вредного вещества в воздухе (мг/м^3 в 1 час);

W (м^3) – кубатура рабочего помещения.

Определением расхода вредного вещества всеми работающими за смену (8 часов) в одном рабочем помещении:

$$B = i * \frac{b_n}{8}, \quad (3.4)$$

где b_n – количество материала, содержащего вредное вещество, расходуемое всеми работающими в данном помещении, мг.

С учетом выделения углекислого газа (CO_2) в процессе дыхания человека в объеме 22,6 литров в 1 час. Тогда:

$$B = 22{,}6 * n, \quad (3.5)$$

где n – число работающих в помещении.

Определить кратность воздухообмена офиса высотой h = 3,5 м, в котором работают 20 человек, на каждого человека приходится 4,5 м^2 площади. Загрязнение воздуха происходит за счет выдыхаемого углекислого газа. Принудительная вентиляция отсутствует.

Количество вредного вещества В, поступающего в помещение в 1 час, задается формулой:

$$B = 22{,}6 * n \quad (3.6)$$

Предельно допустимая концентрация CO_2 составляет 0,1 % или ρ_B = 1 л/м^3. В атмосферном воздухе углекислого газа содержится 0,035 %, т. е. ρ_o =

0,35 л/м^3. Тогда объем чистого воздуха V, необходимого для n человек, согласно формуле (2), составит:

$$V = \frac{22,6*n}{1-0,35} \approx 34,8 * n \quad (3.7)$$

Кратность воздухообмена определяется по формуле :

$$N = \frac{34,8*n}{W} \quad (3.8)$$

Для рассматриваемого производственного помещения n = 20 человек, объем $W = 4,5 * n * h = 4,5 * 20 * 3,5 = 315$ м3

$$N = \frac{34,8*20}{315} \approx 2,2 \quad (3.9)$$

Следовательно, если 3 раза в 1 час производить замену загрязненного воздуха помещения чистым воздухом, концентрация углекислого газа в помещении будет ниже предельно допустимой.

Производятся расчеты на основе, что имеется офис, находящийся на пятом этаже бизнес центра. Офис используется web-студией A-lux & Zoom Apps, которая активно использует техническое оборудование потребляющие много электроэнергии.

В офисе используется техника, подключенная в сеть электропитания. Следовательно, должны соблюдаться меры предосторожности по использованию техники. Персонал должен быть хорошо осведомлен о технических характеристиках техники и правилах ее использования.

Защитное заземление должно обеспечить безопасность людей от поражения электрическим током при взаимодействии с проводниками электрического тока.

Защитное заземление или зануление электроустановок следует выполнять:

– при номинальном напряжении 380 В и выше переменного тока 440 В, и выше постоянного тока - во всех случаях;

– при номинальном напряжении от 42 В до 380 В переменного тока и от 110 В до 440 В постоянного тока;

– при работах в условиях с повышенной опасностью и особо опасных по ГОСТ 12.1.013-78.

Материал, конструкция заземляющих защитных проводников должны обеспечить устойчивость к механическим, химическим и термическим воздействиям

Для питания аппаратуры на предприятиях используется трехфазный ток напряжением 380-220В. В таком случае потенциал не должен превышать 125В.

Сопротивление высчитывается по формуле:

$$R_з = 125/I_з, \qquad (3,10)$$

где $R_з$- сопротивление заземлителя;

$I_з$ – ток замыкания на землю.

По этой норме в эффективно заземленных сетях электробезопасность считается обеспеченной, если $\varphi \leq 10$ кВ и напряжение прикосновения и шага в любое время года не превышает допустимых значений ГОСТ 12.1.019 – 2017.

Оборудование использует напряжение 380/220В, следовательно, $R_з \leq 4$Ом. ГОСТ 12.1.038 – 83.

Значения сопротивления растекания заземлителя определяется путем инструментальных замеров примем $R_е$ = 19 Ом.

Значение растекания искусственного заземлителя высчитывается по формуле:

$$R_и=(R_е*R_з)/(R_е-R) \qquad (3.11)$$

Зная, что $R_з$ = 4 Ом и $R_е$ = 19 Ом, получаем:

$R_и$=(19*4)/(19–4)=5,1Ом

Далее определим удельное сопротивление почвы согласно ГОСТ 12.1.030-81 для вертикальных заземлителей по формуле:

$$\rho_{расч} = \rho_{изм} * \psi, \qquad (3.12)$$

где ψ – коэффициент сезонности равный 1,3;

$\rho_{изм}$ – сопротивление грунта (смешанный грунт) равен 100 Ом*м.

Используя данные значения произведем расчет удельного сопротивления грунта:

$\rho_{расч}$ = 100 * 1,3= 130 Ом*м.

Произведя расчет для вертикальных заземлителей по аналогии рассчитаем сопротивление горизонтальных заземлителей, из таблицы берем ψ = 2,3:

$\rho_{расч}$ = 100 * 2,3= 230 Ом*м

Далее определяется с типом заземлителя. В данном расчете будут использоваться стержневой электрод длиной l=2,1 м, диаметром d=0,1 м и глубиной заложения t= 0,7 метра. Верхние концы соединены с помощью горизонтального электрода – стальной полосы сечением 4х65 мм.

Далее определяется сопротивление одиночного вертикального заземлителя. Произведится расчет растекания сопротивления электродов для стержневого заземлителя круглого сечения в земле по формуле:

$$R_в = (p/2\pi l)(\ln 21/d+(\ln(4t+l)/(4t-l))/2, \quad (3.13)$$

где ρ - удельное сопротивление грунта при вертикальном заземлителе ρ=130 Ом.

$R_в$ = (130/2x3,14*2,1)(ln2*2,1/0,1+(ln(4*0,7+2,1)/(4*0,7-2,1))/2) ~81,7 Ом

Далее произведятся расчеты сопротивления растеканию электродов для стержневого заземлителя в земле по формуле:

$$R_г = (\rho/2\pi L)(\ln L2 /bt), \quad (3.14)$$

где L – длина стальной ленты, которая укладывается на расстоянии 1,4 м.

Rг = (230 / 2 * 3,14 * 45,5)(ln 45,5*2 / 0,1 * 0,7) = 88 Ом.

При размещении электродов по периметру на расстоянии 1,4 м от каркаса здания, количество вертикальных электродов составляет n = 32 шт. на расстоянии 3 м друг от друга. Коэффициенты использования электродов составляют – для вертикального ηв = 0,73 для горизонтального - $\eta_г$ = 0,62.

Сопротивление растекания группового заземлителя по формуле:

$$R = R_в R_г/(R_в \eta_г+R_г \eta_в n) \quad (3.15)$$

R = 81,7 * 88/ (81,7 * 0,62+88 * 0,73 * 32) = 3,4 Ом.

В конце проверяется соблюдение необходимого условия $R_з \geq R$, следовательно, 4 ≥ 3, 4, что дает нам возможность утверждать, что необходимое условие электробезопасности выполняется.

3.3 Выводы

В первом случае производится расчет кратности воздухообмена. Выводом которого, устанавливается, что аэрацию помещения, объемом 315 $м^3$, где работают 20 человек, можно осуществлять с помощью постоянно открытой форточки, площадью 0,1 $м^2$. Возможно также периодическое, через каждые 20 минут, проветривание помещения с помощью открывания на 2 минуты фрамуги, площадью 1 $м^2$.

В результате произведенного расчета по безопасности от поражения электрическим током в офисе, сделан вывод, что удовлетворяется условие,

сопротивление растекания группового заземлителя R=3,4Ом не превышает сопротивление заземлителя $R_з$ = 4 Ом. Важно определиться с типом используемого заземлителя и его размерами. Также стоит учесть количество используемых заземлителей по периметру здания. В итоге нужно отметить важность и точность произведения расчетов, так как от них напрямую зависит жизнь человека.

4 Анализ рисков

В этой части дипломного проекта используется методика расчета по двум параметрам. В качестве параметров, над которыми будут производится расчеты, используются:

– коэффициенты «Конфиденциальности», «Целостности», «Доступности»;

– коэффициент показателя степени уязвимости;

– коэффициент уровня угрозы.

Согласно стандарту ISO/IEC 27005[11] уровень риска вычисляется с учетом трех показателей – ценности ресурса, уровня угрозы и степени уязвимости.

$$R = S * L(t) * L(v), \tag{4.1}$$

где R - значение риска;

S - ценность актива/ресурса;

L(t) - уровень угрозы;

L(v) - уровень/степень уязвимости.

После расчета показателей уровня риска, производится введение мер, препятствующих реализации риска, в данном случае этими мерами должно служить введение программно-аппаратного комплекса. Перерасчет остаточного показателя риска после введения мер осуществляется по формуле ниже:

$$R_{ост} = S * L(t)_{ост} * L(v)_{ост}, \tag{4.2}$$

где $R_{ост}$ - остаточный риск;

S - ценность актива/ресурса;

$L(t)_{ост}$ - уровень угрозы после введения мер;

$L(v)_{ост}$ - уровень/степень уязвимости после введения мер;

4.1 Расчет рисков

В качестве среды, для которой будут производится расчеты рисков используется агентство недвижимости, в которой имеется головной офис. Агентство пользуется услугами интернет провайдера, и хостинга. Сотрудники агентства работают за персональными компьютерами под управлением ОС Windows. Глава агентства недвижимости решает произвести внедрение в головной офис разработанное веб - приложение, предварительно загрузив его на хостинг. Ставится задача, произвести расчеты рисков всех активов, которые так или иначе связаны с работой приложения. Расчеты требуется произвести до и после внедрения в рабочую среду.

Первым делом производится описание присутствующих в организации основных активов. Выполняется построение диаграмм в программе Coras.

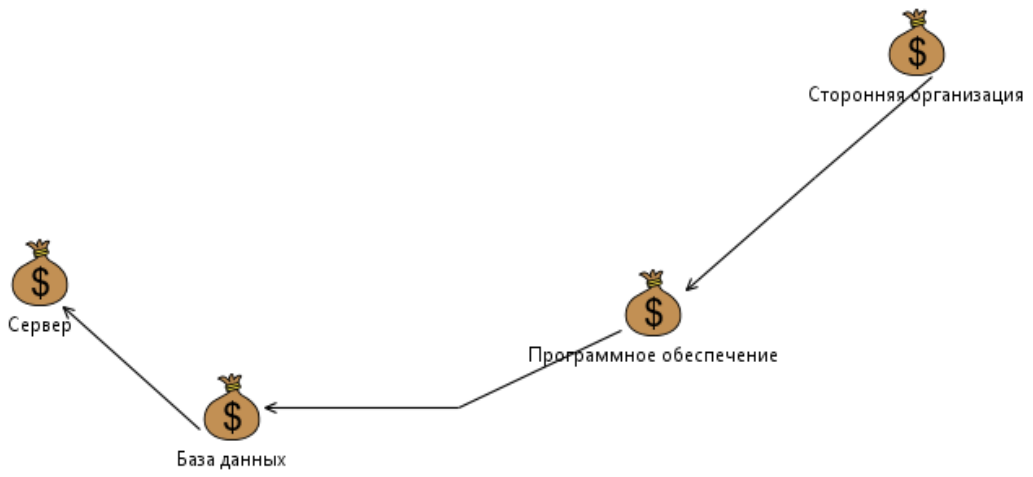

Рисунок 4.1 – активы

На рисунке 4.1 представлены активы агентства недвижимости. Стоит выделить актив “Сторонняя организация” – это актив, который предоставляет данные для анализа и поиска инцидентов, которой никак не возможно будет проанализировать на предмет рисков, так как этот актив совмещает в себе любую организацию и рассматриваться не будет. “База данных” является непосредственным атрибутом сервера агентства, база будет содержать записи и персональные данные пользователей и сотрудников агентства, использующих данное веб – приложение. После описания перечня активов, можно приступать к рассмотрению списка угроз и уязвимостей, которые могут возникать для выбранных активов. “Программное обеспечение” – это и есть веб – прилоложение, которое используется сервером.

Рисунок 4.2 описывает пары угроз и уязвимостей, а также к каким активам, по какому сценарию и кем реализуются данные пары. Из диаграммы видно, что основными факторами угроз и уязвимостей являются злоумышленники, сотрудники и системные сбои. Стоить заметить, что угроза одного актива несёт влияние и на другой. К примеру, отключение Brutforce атака сказывается и на сервере и на ПО. Также стоит учитывать и человеческий фактор, который невозможно исключить, но его угрозу возникновения можно понизить. Список угроз и уязвимостей:

– открытый доступ к рабочему месту;
– доступ к портам сервера;
– подбора пароля;
– DoS атака;
– SQL – инъекции;
– некомпетентность сотрудников;
– человеческий фактор;
– отсутствие обновлений.

После описания пар угроз и уязвимостей, можно перейти к построению диаграммы рисков, возникающих в организации относительно выбранных

активов. Рисунок 4.3 как раз и отображает данную диаграмму, описывая основные риски, активы к которым они относятся, а также источники возникновения данных рисков. Из диаграммы рисков видно, кто является причиной возникновения рисков и на что он влияет. Список рисков включается в себя:

– физический контакт;
– перехват трафика;
– Bruteforce атака;
– отказ в обслуживание;
– дыры в базе;
– нестабильность работы;
– утеря учетных данных;
– загрузка системы со сторонних носителей.

Как видно из рисунка 4.4 каждому риску выставляется определенный числовой показатель, сопровождаемый параметром приемлемости риска.

Когда произведен первичный расчет рисков, можно переходить к процессу выбора мер по обработке риска. Рисунок 4.5 отображает диаграмму 4.2 с наложенными на нее мерами по обработке рисков. Список применённых мер:

– 2-х факторная аутентификация;
– использование сетевого монитора;
– установка сложного пароля;
– обновление оборудования;
– обновление BIOS.

Рисунок 4.6 аналогичен 4.4, как и в случае с рисунком 6.5 на него наложены меры, которые помогают уменьшить показатель, получаемый при повторном расчете. Список применённых мер:

– использование веб – сокетов;
– двухфакторная аутентификация;
– проведение аудита БД;
– использование политик учетной записи;
– ограничение максимального количества запросов на сервер;
– отключение портов и обновление BIOS;
– обновление оборудования.

В ходе проведения расчетов получена итоговая таблица расчетов (таблица 4.1).

Таблица 4.1– Таблица рисков.

Код актива	Угрозы	Уязвимости	Максимальный уровень риска	Меры по обработке риска	Остаточный уровень риска	Дата
А. Сервер						
A.1	Brutforce атака	Несанкционированный доступ	26	Использование политик блокировки учетной записи	12	27.06.2020
A.2	Выполнение произвольного кода путем посылки специального потока UDP – пакетов на закрытый порт	Возможность доступа к портам сервера	48	Использование двухфакторной аутентификации	12	28.06.2020
A.3	Нестабильность работы сервера	Открытый доступ к рабочему пространству	48	Обновление оборудования	8	29.06.2020
B. Программное обеспечение						
B.1	Отказ в обслуживании	DoS-атака	16	Ограничение числа запросов	8	30.06.2020
B.2	Подбро пароля к учетным записям пользователей	BrutForce -атака	20	Использование политик блокировки учетной записи	6	01.07.2020
C. База данных						
C.1	Дыры в безопасности БД	SQL - инъекции	46	Проведение аудита, и ограничение возможностей работы с базой с учетом IP адреса машины.	12	30.06.2020
C.2	Загрузка системы со сторонних носителей	Отсутствие обновлений	16	Обновление BIOS, отключение портов	9	01.07.2020
C.3	Утеря данных БД	Человеческий фактор	18	Использование двухфакторной аутентификации	6	01.07.2020

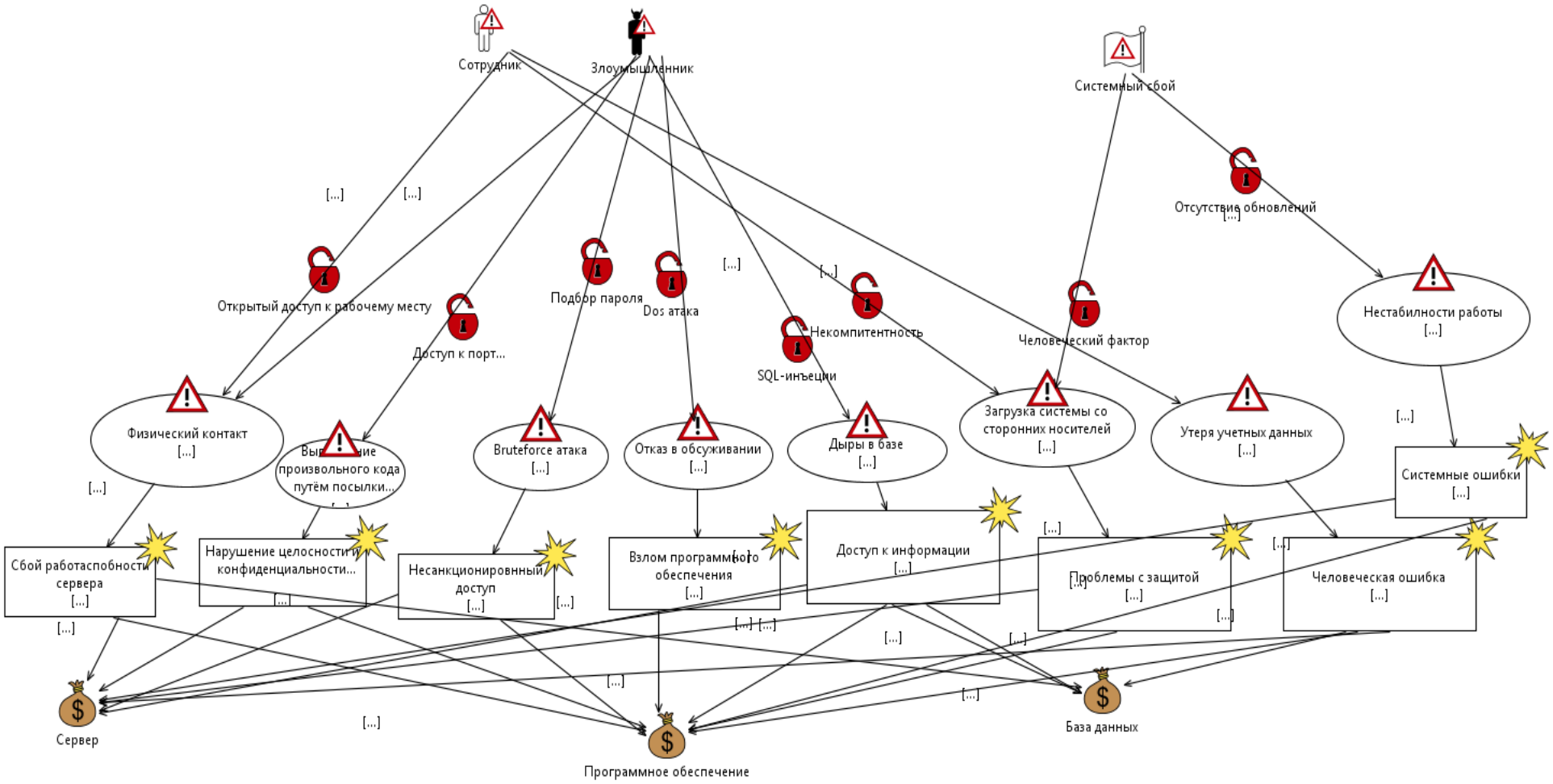

Рисунок 4.2 – Угрозы и уязвимости

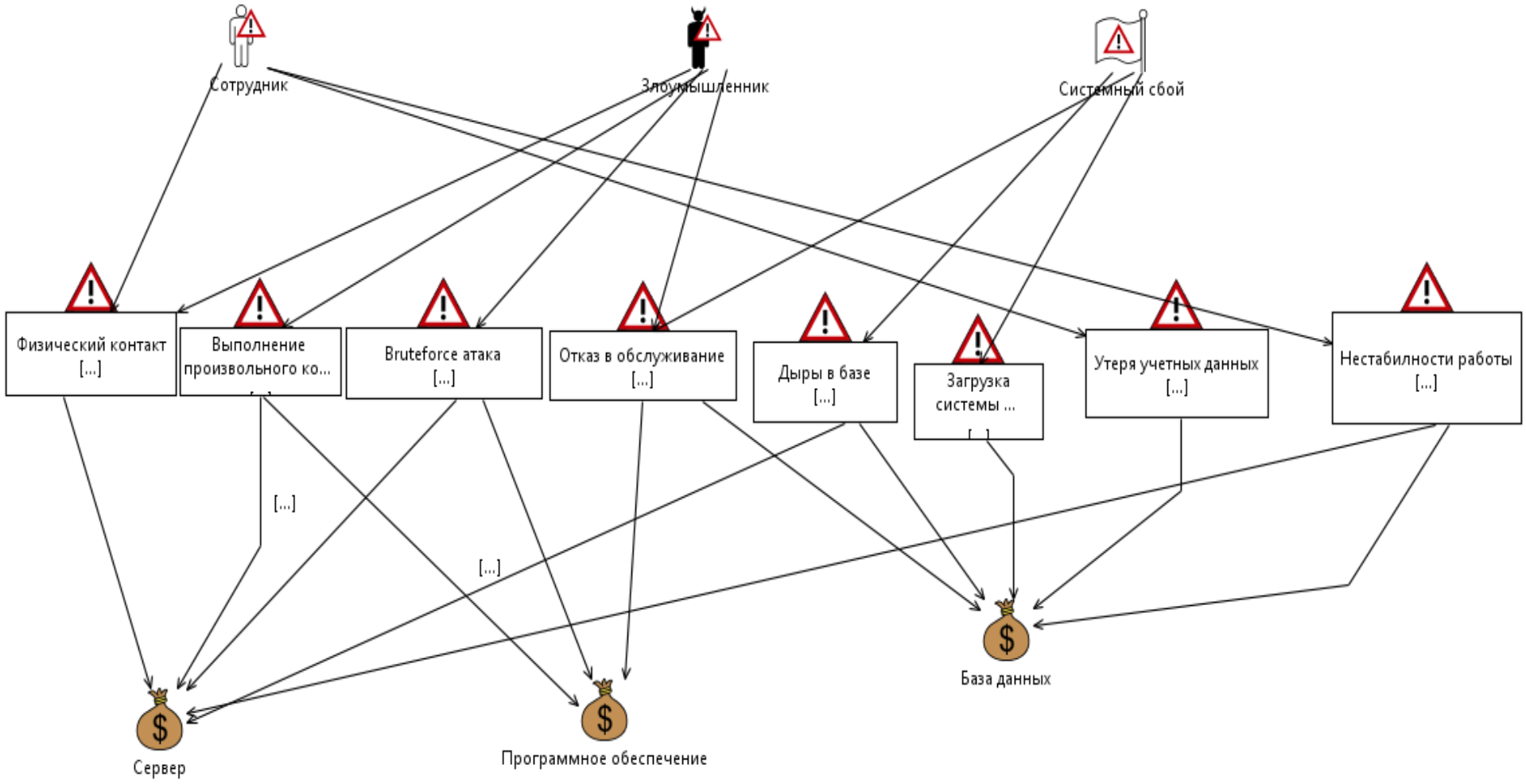

Рисунок 4.3 – Диаграмма рисков

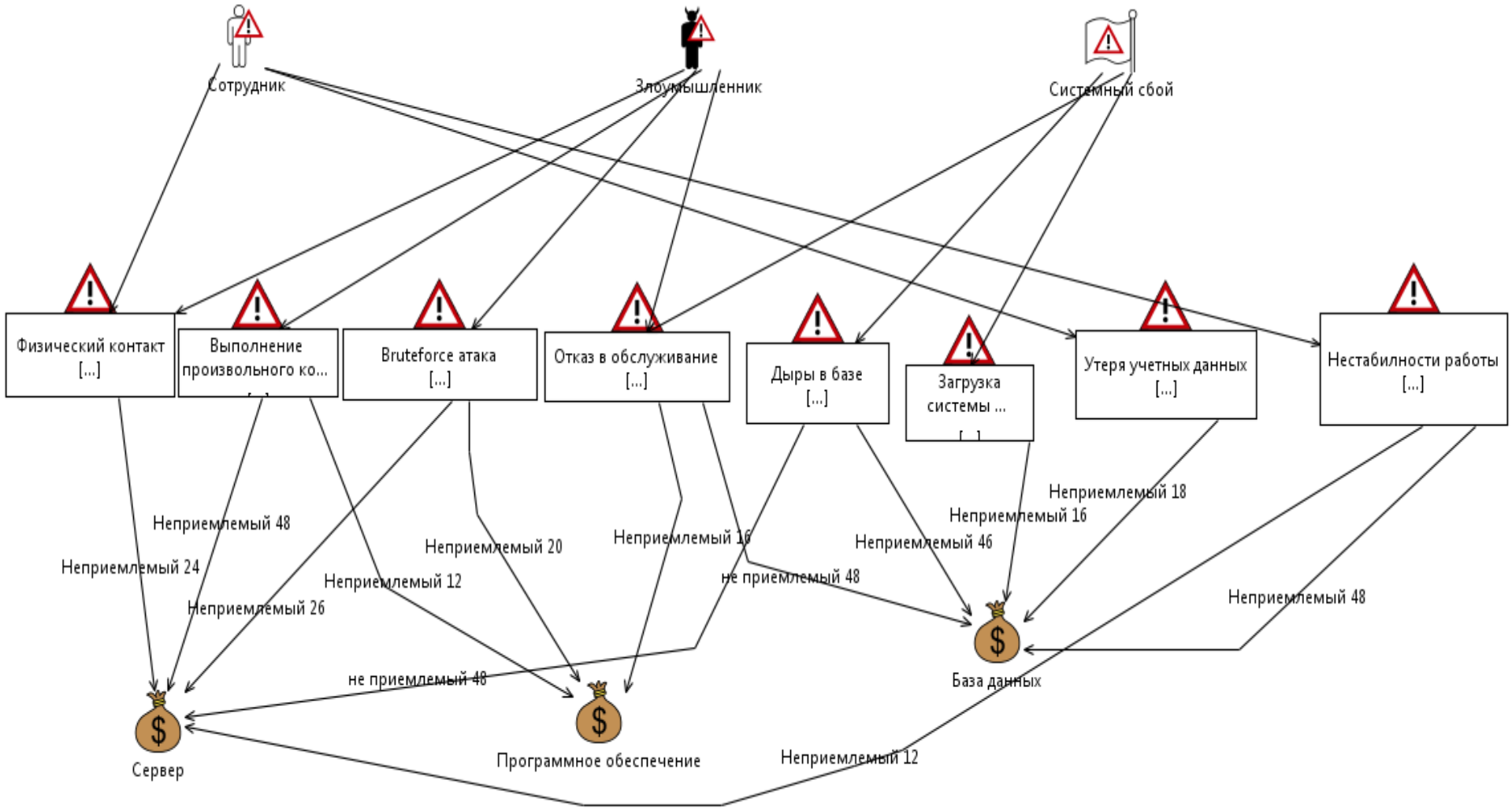

Рисунок 4.4 – Диаграмма рисков с оценкой

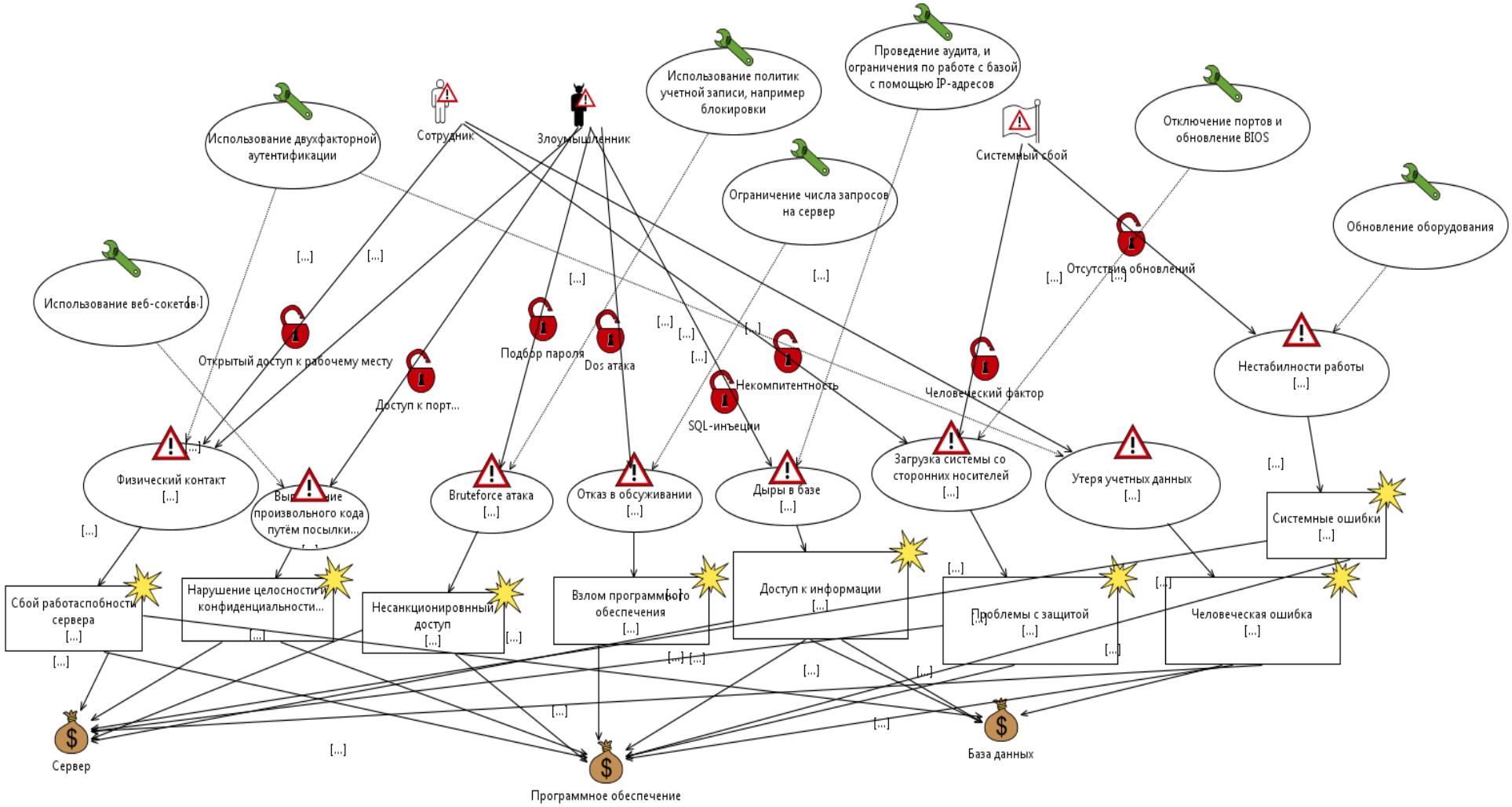

Рисунок 4.5 – Включение мер в схему с угрозами и уязвимостями

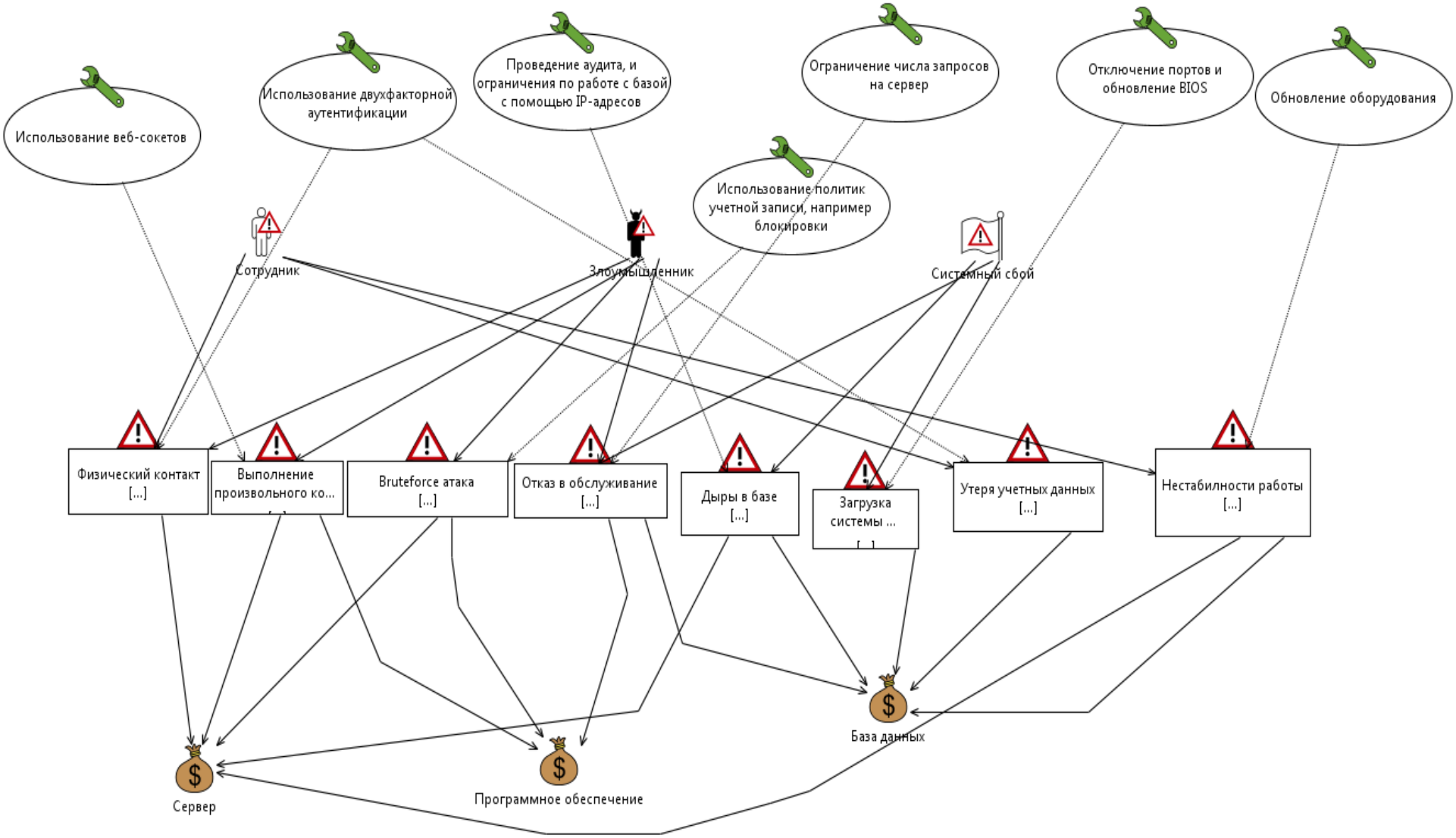

Рисунок 4.6 – Диаграмма рисков с мерами по их исключению

4.2. Вывод по анализу рисков

По итогам данной главы были проведены расчеты рисков, возникающих в агенстве недвижимости и хостинг провайдере относительно активов на риски которых может повлиять введение в эксплуатацию веб – приложения, разрабатываемого в дипломном проекте. По итогам, введение в организацию проекта позволяет понизить риски и закрыть уязвимости. После введение мер, большинство рисков удалось свести к их минимальному показателю. Стоит выделить угрозы с самыми большими уровнями риска, ими являются:

– выполнение произвольного кода - максимальный уровень риска 48;

– нестабильность работы сервера - максимальный уровень риска 48;

– дыры в безопасности БД - максимальный уровень риска 46.

Выполнение произвольного кода удалось уменьшить в 4 раза путем использования двухфакторной аутентификации. Остаточный уровень риска составляет 12. Нестабильность работы сервера снизился в 6 раз, в следствии обновления оборудования. Остаточный уровень риска составляет 8. Дыры в безопасности БД, проявляющиеся путем проведения злоумышленниками SQL - инъекций, была снижена в 3 раз, путем проведения аудита, и ограничения возможностей работы с базой с учетом IP адреса машины.

Заключение

В данном дипломном проекте были спроектированы веб – приложение на базе Oracle Database 11g и комплексные меры защиты к этому приложению. Эти меры были реализованы путем детального анализа уязвимостей и проблем, с которыми могут столкнуться приложение и база данных.

В процессе разработки данного проекта были использованы передовые технологии, перечисленные ниже:

– php фреймворк – Laravel;

– javascript фреймворк –Vue;

– css фреймворк – Scss;

– база данных Oracle Database 11g;

– собственная система управления веб – сайтом Voyager.

Во время работы над данным проектом возникли такие проблемы, как SQL – инъекции, XSS и CSRF атаки , самые распространенные типы атак на веб – приложения. Чтобы их решить были использованы:

– CSRFProtection, механизм для защиты от кросс сайтовой подделки запроса;

– Eloquent ORM, включенный в Laravel, использует операторы, которые избегают любого пользовательского ввода, который может прийти через формы;

– Vuelidate – Vue библиотека для валидирования входных данных пользователя для предотвращения XSS – атак.

Таким образом, дипломный

Список литературы

1. Полная документация к СУБД Oracle Database, URL: https://docs.oracle.com/en/ (дата обращения 24.03.2020).
2. Полная документация к фреймворку Laravel, URL: https://laravel.com/docs/7.x/ (дата обращения 28.03.2020).
3. Полная документация к административной панели Voyager, URL: https://voyager-docs.devdojo.com/ (дата обращения 31.03.2020).
4. Полная документация к фреймворку VueJs, URL: https://ru.vuejs.org/v2/guide/index.html (дата обращения 02.04.2020).
5. Полная документация к фреймоврку SCSS, URL: https://sass-scss.ru/ (дата обращения 03.04.2020).
6. HTTP-клиент на основе обещаний для браузера и node.js, URL: https://github.com/axios/axios (дата обращения 02.04.2020).
7. Централизованное государственное управление для Vue.js. URL: https://vuex.vuejs.org (дата обращения 03.04.2020).
8. СанПиН 2.2.2.542-96. «Гигиенические требования к видеодисплейным терминалам, персональным электронно-вычислительным машинам и организация работы [Текст] / Санитарные правила и нормы. – М.: Информационно-издательский центр Минздрава Казахстана. 2001. -20 с.
9. ГОСТ Р 50571ю3-2009 (МЭК 60363-4-41:2005. Электроустановки низковольтные. Требования для обеспечения безопасности. Защита от поражения электрическим током / М.: Издательский центр Стандартинформ, 2009.-70 с.
10. СанПиН 2.2.4.3359-16. Санитарно-эпидемиологические требования к физическим факторам на рабочих местах / М.: Информационно-издательский центр Минздрава Казахстана. 2016 – 47с.
11. ГОСТ ISO/IEC 27005. Информационная технология. Методы и средства обеспечения безопасности. Менеджмент риска информационной безопасности / Международный стандарт. 2011 – 94с.
12. Ефремова О.С. Документация по охране труда в организации. [Текст] Практическое пособие /О.С. Ефремова 5-е изд. перераб. и доп. -М.: Изд. “Альфа-Пресс”,2015 г. – 152 с\
13. Ефремов О.С. Охрана труда в организации в схемах и таблицах. [Текст] / О.С. Ефремова 7-е изд. перераб. и доп. -М.: Изд. “Альфа-Пресс”,2018 г. – 124 с.
14. СН 2.2.4/2.1.8.562-96. Санитарные нормы. Шум на рабочих местах, в жилых помещениях, общественных зданиях и на территории жилой застройки [Текст] / Санитарные нормы. -М.: Издательский центр Минздрава РФ. 1996. – 25 с.