МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

ИНСТИТУТ *ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОМПЬЮТЕРНЫХ НАУК*

КАФЕДРА *АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ И ДИЗАЙНА*
НАПРАВЛЕНИЕ *09.04.02 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ*

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

на тему: Разработка системы безопасного хранения и обработки персональных данных в E-commerce на Next.js

Студент _____
Руководитель работы _____ *Е.Г. Коржов*
Нормоконтроль проведен *А.С. Оганесян*

Проверка на заимствования проведена _____ *А.А. Петрыкина*

Работа рассмотрена кафедрой и допущена к защите в ГЭК

Заведующий кафедрой _____ *Е.Г. Коржов*
Директор института _____ *С.В. Солодов*

Москва 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

УТВЕРЖДАЮ

Институт Информационных технологий и компьютерных наук
Кафедра Автоматизированного проектирования и дизайна — **Зав. Кафедрой** Е.Г. Коржов
Направление 09.04.02 Информационные системы и технологии — «___»_____2025г.

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ МАГИСТРА

Студенту группы _____
(Ф.И.О. полностью)

1. Тема работы Разработка системы безопасного хранения и обработки персональных данных в E-commers на Next.js

2. Цели работы Разработать систему безопасного хранения и обработки персональных данных для e-commerce платформ, построенных на базе фреймворка Next.js для повышения качества и надежности современных веб-приложений

3. Исходные данные Федеральный закон о персональных данных

4. Основная литература, в том числе:

4.1. Монографии, учебники и т.п. 1. Черный Б., Профессиональный TypeScript. Разработка масштабируемых JavaScript-приложений. СПб.: Питер, 2022 — 352 с.; 2. Гойвертс Я., Левитан С., Регулярные выражения: сб. рецептов / пер. с английского СПб.: символ-плюс, 2010 — 608 с.

4.2. Периодическая литература Калягина Л.В., Разумов П.Е., Категория «Данные»: понятие сущность подходы к анализу. Вестник КрасГАУ, № 4, 2014, с. 3-8.

5. Перечень основных этапов исследования и форма промежуточной отчетности по каждому этапу Исследование современных методов аутентификации (Отчет по НИР)

6. Использование информационных технологий при проведении исследований Node.js, Next.js, Drizzle, SQLite, Zustand, Zod, Tailwind CSS, TanStack Query, React Hook Form, Lucia, Vercel, Turso

7. Перечень (примерный) иллюстрированного материала скриншоты, иллюстрации

8. Руководитель диссертации К.т.н., доцент, Коржов Евгений Геннадьевич
(Должность, звание, ф.и.о.)

(подпись)

Дата выдачи задания 03.03.2025

Задание принял к исполнению студент

(подпись)

Аннотация

Выпускная квалификационная работа содержит: 81 страницу, 10 рисунков, 8 таблиц и список литературы, состоящий из 70 источников.

Работа включает в себя анализ современных киберугроз, а также исследование современных способов защиты персональных данных при их непосредственной обработке и хранении.

Выпускная квалификационная работа состоит из пояснительной записки и пояснительной записки и презентации. Пояснительная записка содержит: введение, три главы и заключение.

Annotation

The graduation work contains: 81 pages, 10 tables and list of literature, which consists of 70 sources.

The work includes the analysis of modern cyber threats, as well as the research of modern ways to protect personal data, during their direct processing and storage.

The graduation work consists of an explanatory note and a presentation. The explanatory note contains: an introduction, three chapters and a conclusion.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность работы обусловлена стремительным развитием платформ электронной коммерции, которые так или иначе вынуждены собирать и обрабатывать персональные данные клиентов при проведении транзакций. Помимо данного фактора, в связи с ежегодным ростом общего числа киберпреступлений, правительством РФ, регулярно вводятся новые изменения и требования по обработке персональных данных, требующих внимания со стороны бизнеса. В связи с этим, предприниматели вынуждены не только следить за безопасностью обработки данных, но и за тем, чтобы меры, принимаемые для организации защиты соответствовали законодательству.

На сегодняшний день, фреймворк Next.js, построенный на базе React, активно применяется для разработки интернет-приложений, в том числе и для платформ электронной коммерции. На базе данного фреймворка работают такие приложения как: Spotify, Wallmart, IMDB, Figma и другие. Данная библиотека позволяет разрабатывать надежные приложения, благодаря встроенным механизмам защиты, что делает ее перспективной платформой для реализации системы безопасной обработки и хранения данных.

Объектом исследования являются процессы хранения, обработки, передачи и защиты персональных данных в платформах электронной коммерции на Next.js.

Предмет исследования представляет собой конкретные методы и технологии обеспечения безопасности данных в платформах электронной коммерции, в процессе обработки и хранения данных. Исследование включает такие методы и технологии как: способы контроля, передачи, обработки, шифрования и хранения информации; современные способы аутентификации пользователей; способы защиты от веб уязвимостей.

Целью исследования является разработка безопасной системы хранения и обработки персональных данных для e-commerce платформ, построенных на Next.js. Исследование направлено на повышение качества и надежности

способов обработки персональных данных пользователей интернет-магазинов и платформ.

Исходя из поставленных целей необходимо решить следующий ряд задач:

1) Изучить нормативно-правовые и технические стандарты защиты данных.

2) Провести анализ современных угроз, уязвимостей и способов противодействия им.

3) Разработать безопасную архитектуру приложения с использованием фреймворка Next.js.

4) Провести комплексное тестирование системы на целостность обрабатываемой информации и на устойчивость системы к современным типам угроз.

Практическая значимость работы заключается в том, что разработанная система, может быть, не только внедрена в уже существующие платформы, но и может послужить основой для дальнейших разработок.

Научная новизна исследования заключается в адаптации современных методов защиты информации при обработке персональных данных для платформ электронной коммерции на Next.js, с учетом специфики их работы и ограничений, наложенных законодательством.

Структура работы включает в себя: введение, три главы, заключение и список используемых источников. Первая глава посвящена исследованию теоретических аспектов обработки персональных данных. Во второй главе внимание уделяется проектированию системы и разработке отдельных компонентов обработки и защиты данных. В третьей главе осуществляется непосредственное тестирование как отдельных элементов, так и их комплексного взаимодействия.

Глава 1. Теоретические аспекты обработки персональных данных

1.1 Актуальность разработки средств обработки и хранения персональных данных в платформах электронной коммерции

В настоящее время остро стоит вопрос о защите персональных данных пользователей информационных ресурсов. Количество киберугроз и преступлений в информационном поле постоянно растет. Поэтому так важно знать о современных способах обработки и хранения персональных данных. Понимание механизмов, связанных с хранением и обработкой личных данных пользователей, помогает предотвратить несанкционированный доступ к чувствительной и конфиденциальной информации, предоставляя доступ только для тех пользователей, которые смогли подтвердить свою личность. Такой подход крайне важен в первую очередь для тех компаний, которые хранят персональные данные пользователей или работают непосредственно с их финансами.

Основной целью для злоумышленников, специализирующихся на киберпреступности, являются именно персональные данные пользователей. В период с 2019 по 2020 годы доля киберпреступлений, основной целю которых являлись непосредственно персональные данные пользователей, доходила до 90%. Нужно отметить, что больше половины преступлений, связанных с похищением данных, приходится именно на хакерские атаки и удаленный доступ [46]. По данным Генпрокуратуры РФ, доля преступлений, совершенных с применением информационных технологий, в 2024 году, по сравнению с предыдущим годом, увеличилась с 31.8% до 38.2% [1].

Одной из причин утечки личных данных пользователей может являться неправильно настроенная система обработки или хранения этих самых данных.

Подобные утечки личных данных пользователей, могут привести к серьезным последствиям, как для клиентов, так и для держателей платформ.

Наиболее безобидным вариантом, при попадании личных данных к злоумышленникам, для пользователя будет отправка навязчивой рекламы на его адрес электронной почты. Однако, в современных условиях человек, чьи данные были скомпрометированы, рискует оказаться в куда более опасных ситуациях, например:

1) Стать жертвой мошенника, который зная конфиденциальную информацию, сможет втереться в доверие.

2) Потерять доступ к банковским счетам и накоплениям. Сюда же можно отнести оформление незаконных кредитов на имя потерпевшего и проведение нелегальных сделок с недвижимостью.

3) Если злоумышленник получит чувствительные данные пользователя, то он может попытаться запугать его, с целью вымогания денежных средств или совершения противоправных действий [2, 3].

Если же говорить о бизнесе, то до недавнего времени, подобные утечки могли привести лишь к репутационному ущербу для компаний, но уже в ближайшем бедующем компании не смогут так просто уходить от ответственности за подобные происшествия.

Государственная дума РФ приняла поправки в основной закон по хранению и обработке персональных данных пользователей. Так, уже в 2025 году заработает новая система наказания, предусматривающая введение оборотных штрафов, принудительные работы или лишение свободы на срок до 4 лет [4].

В контексте интернет-платформ, связанных с ведением онлайн торговли, вопрос обработки данных, получаемых от клиентов, имеет критическое значение. В отличие от сайтов-визиток, основная задача которых это информирование посетителей о чем либо, без необходимости сбора персональных данных, интернет-магазины вынуждены постоянно обрабатывать и хранить определённые пользовательские данные.

Классический пример оформления интернет-заказа состоит из нескольких шагов и выглядит следующим образом:

1) Пользователь создает свою персональную страницу на платформе, посредством передачи своих личных данных, которые впоследствии будут необходимы для подтверждения личности пользователя.

2) Добавляет услуги или товары в свою покупательскую корзину.

3) Подтверждает правильность собранного перечня товаров внутри корзины.

4) Вводит дополнительные данные, необходимые для совершения транзакции, такие как: контактная информация, способ доставки, номер банковской карты и т.д.

5) Получает подтверждение оформления заказа.

В данном примере наглядно показан алгоритм действий, которые необходимо совершить и обработать на стороне онлайн сервиса. Помимо обработки полученных данных, их нужно где-то хранить. Накопление и анализ полученной пользовательской базы клиентов — это один из способов выстраивания дальнейшей стратегии ведения бизнеса, с целью получения конкурентного преимущества на рынке. Все это является коммерческой тайной. Анализируя пользовательскую информацию, бизнес может определить как наиболее востребованные товары и услуги, так и наименее популярные, предлагая своим клиентам только самое необходимое.

Нередко подобные хранилища данных становятся целью злоумышленников, так как они предоставляют особую ценность для конкурирующих фирм или лиц, занимающихся противоправной деятельностью. Одной из самых масштабных утечек данных в 2022 году была утечка базы телефонных номеров пользователей WhatsUp, которая содержала список из 478 млн пользователей из 84 стран. В том же году, заместитель председателя правления Сбербанка заявил, что из системы электронного банкинга были украдены данные 65 млн пользователей в России, нанеся общий ущерб в 4,5 млрд рублей [47].

Исходя из всего вышеперечисленного становится понятно, что в современном мире, для хранения и обработки пользовательских данных должны быть использованы современные и надежные средства, которые смогут обезопасить и клиентов интернет-магазинов и держателей бизнеса. Кроме того, подобные системы должны учитывать современные законы о защите персональных данных.

1.2 Основные понятия и определения персональных данных

Данные — это набор фактов, сведений или идей, представленных в виде, пригодном для дальнейшей обработки, передачи или хранения. Данные, которые прошли предварительную обработку называются информацией. Информация возникает в результате решения конкретных задач, например выдача фамилии пользователя по запросу [48].

С точки зрения безопасности, информация должна обладать тремя ключевыми свойствами: доступностью, целостностью и конфиденциальностью. Доступность означает, что информация должна быть предоставлена тем, кто имеет на это право. Конфиденциальность же наоборот запрещает предоставление информации для тех, кто такого права не имеет. А целостность подразумевает, что данные, при хранении или передаче, должны оставаться в неискаженном виде [5].

Согласно Федеральному закону о персональных данных, принятому Государственной Думой 8 июля 2006 года, к персональным данным относится любая информация, прямо или косвенно связанная с определенным человеком [48]. На основании этого можно сделать вывод, что к персональным данным, в контексте платформ электронной коммерции, относятся:

- фамилия, имя, отчество;
- адрес электронной почты, номер телефона;
- пароль, личный идентификатор;
- способ оплаты;

– данные банковских карт;
– списки купленных, понравившихся, добавленных в корзину товаров;
– способ и адрес доставки.

Под обработкой персональных данных подразумевается любое действие или совокупность действий, которые могут совершаться с применением вычислительной техники, над персональными данными. К таким действиям относятся: сбор, запись, хранение, систематизация, обновление, изменение, извлечение, использование, предоставление, блокирование и удаление персональных данных [6].

На основании внесенных изменений в законодательстве РФ, от 25 июля 2011 года, лица, получившие доступ к личным данным пользователей, обязаны не раскрывать полученную информацию третьим лицам, без личного согласия на это пользователя, предоставившего эти данные. Пользователь так же должен иметь возможность в любой момент получить доступ к своим персональным данным [6].

Согласно главе 3, статьи 14, пункта 2 закона о персональных данных, любые сведения, запрошенные их владельцем, должны быть предоставлены в доступной форме, и в них не должно содержаться информации, полученной от других пользователей [6]. Иными словами, каждый клиент компании, должен получать только ту информацию, которая относится непосредственно к нему самому и никому более. Ответственность за обеспечение безопасности при хранении и обработке персональных данных лежит на стороне организации, непосредственно получившей такие данные.

Резюмируя все вышеперечисленное, можно сделать вывод, что разработка современных систем хранения обработки персональных данных должна учитывать не только потребности бизнеса, но и требования, установленные законами РФ, предоставляя пользователям дополнительный функционал возможностей и гарантий безопасности.

1.3 Обзор современных способов обработки и хранения данных

Для сбора и хранения информации в интернет-пространстве используются специализированные базы данных. Данные, хранимые в таких базах, имеют структурированный и упорядоченный вид. Такие базы можно разделить на две большие группы: реляционные и нереляционные.

Характерной чертой для реляционных баз данных является использование Stractured Query Language (SQL), что в переводе означает «Структурированный язык запросов». Реляционные базы данных имеют табличную структуру и представляют собой двумерный массив. Каждая таблица имеет свое уникальное поле или сочетание полей, которые содержат уникальные идентификаторы табличных значений. Примеры реляционных баз данных: SQLite, MySQL, PostgreSQL [49].

В отличие от реляционных баз, у не реляционных нет строгой привязки к SQL или структуре, что делает данную систему более гибкой в плане настройки и изменения. Ключевой особенностью подобных баз данных является именно отсутствие привязке к какой-либо структуре в плане хранения данных. Наиболее простым вариантом хранения информации в таких базах является пара из ключа и значения, которые, как и сама структура, могут не иметь четкой привязки к форме. Например, в нескольких записях по одному ключу могут хранится разные типы данных, что невозможно при использовании реляционного подхода. Такой подход обусловлен тем, что такие модели нацелены в первую очередь на скорость выполнения, а не на целостность и систематизацию. Примеры нереляционных баз данных: MongoDB, Redis, OrientDB. На рисунке 1 представлены структурные различия реляционных и не реляционных баз данных.

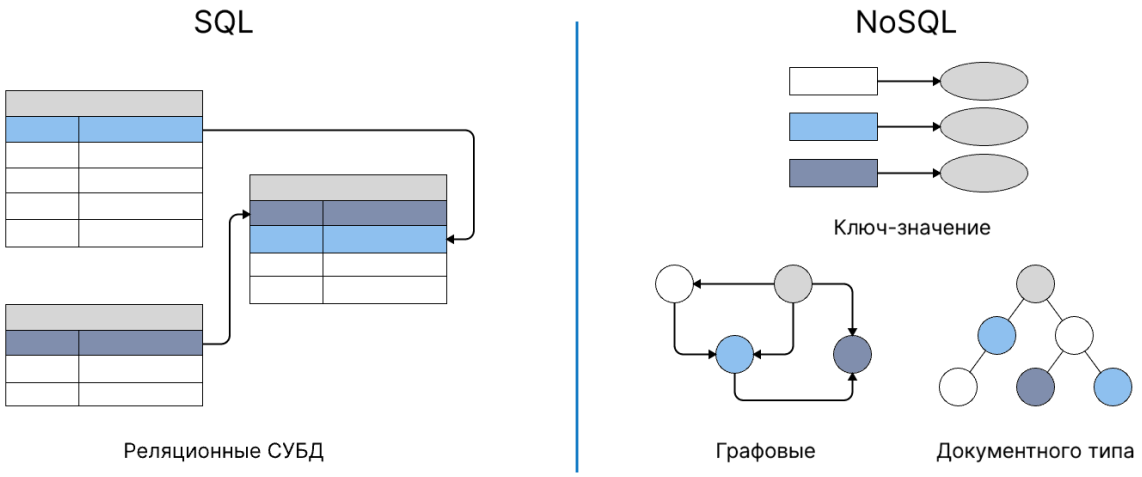

Рисунок 1 – Структуры баз данных

Для работы с базами данных широко используются ORM системы. ORM является посредником между backend-ом приложения и хранилищем данных. Основным преимуществом подобных систем является возможность создавать хранилища данных на том же языке программирования, что и само приложения, что позволяет практически не использовать тот же SQL для управления базой данных. Еще одним плюсом ORM являются встроенные методы, которые позволяют существенно сократить количество кода в программе [50]. Примеры ORM систем: Drizzle, Prisma, TypeORM и другие.

Для взаимодействия с пользователем в интернет-пространстве используются HTML формы, которые являются практически единственным способом получения данных от клиента. HTML расшифровывается как Hyper Text Markup Language, что переводится как «язык разметки гипертекста». Данный язык позволяет выстраивать основной каркас веб-страницы и позволяет взаимодействовать с пользователем. Форма представляет собой совокупность интерактивных элементов управления, которые позволяют сохранять пользовательские изменения [7]. Самым простой пример формы содержит поле ввода и кнопку подтверждения, которая утверждает текущие данные, хранящиеся в поле ввода, и отправляет их на дальнейшую обработку. Для создания подобной структуры в HTML используются теги: form,

отвечающий за границы начала и конца формы; input, устанавливающий определённый тип поля ввода; button, создающий кнопку для взаимодействия с пользователем. Тег input в свою очередь имеет большое количество типов, отличающихся вариантами исполнения, из которых наиболее часто используемыми являются:

1) Text, представляет собой стандартный вариант ввода текста.

2) Number, ограничивает количество допустимых символов ввода до цифр.

3) Email, поле для ввода электронной почты. В таком режиме система автоматически проверяет является ли введенная строка почтой.

4) Password, предназначен для визуального скрытия вводимых символов.

5) Tel, используется для ввода телефонных номеров. При использовании мобильного устройства, добавляет специализированную клавиатуру.

6) Url, ожидает получение интернет-адреса.

7) Color, позволяет выбрать цвет в формате RGB.

8) Range, создает поле с диапазоном значений.

9) Checkbox, специализированное поле, которое может иметь только значения «да» или «нет». Часто используется для составления множественного выбора из списка.

10) Radio, противоположность checkbox. Используется для единичного выбора и списка вариантов.

Однако несмотря на большую поддержку встроенных типов, без дополнительных настроек, такие поля имеют большие проблемы с валидацией данных. Например, поле ввода с типом tel предназначено для ввода номера телефона, что подразумевает использование только цифр или специализированных символов, но на деле не запрещает использование любых доступных знаков, что в свою очередь может привести к неприятным последствиям, особенно, если система на стороне сервера ожидает получение определенного формата.

Исходя из данной особенности обработки полей ввода можно сделать вывод, что необходимо использовать дополнительные способы контроля, которые позволят не только следить за тем, чтобы форма получала необходимые данные, но и совершать определенные действия над ними. Для этих целей используется JavaScript. Пример валидации полей ввода посредством языка JavaScript представлен на рисунке 2.

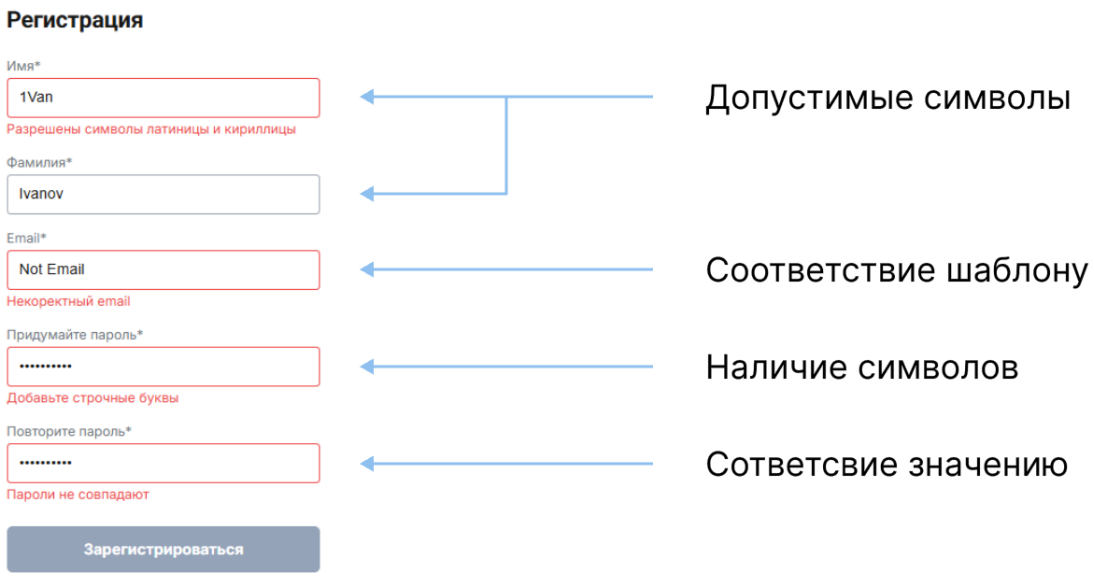

Рисунок 2 – Пример валидации форм

JavaScript — это специализированный язык программирования, который позволяет напрямую взаимодействовать с элементами веб-страниц, расширяя их базовые возможности, предоставляя полный контроль над содержимым HTML документа [8]. До недавнего времени данный язык использовался в основном для работы на стороне клиента, но уже сейчас его можно использовать и для настройки серверной части приложения, что делает его универсальным языком программирования.

Тем не менее, использование JavaScript в чистом виде может привести к серьезным проблемам с безопасностью и производительностью. Являясь языком с динамической типизацией, он определяет тип переменной,

непосредственно в момент выполнения программы [9]. Иными словами, подстраивается под полученные данные. Например, если программа складывает 2 числа, то результатом будет их сумма, но если на месте первого числа окажется строка, то второе число так же превратится в строку, что приведет к соединению двух строк в одну. Такое автоматическое изменение называется «неявным преобразованием», потому что происходит без каких-либо дополнительных инструкций. Подобное поведение может приводить к ошибкам в программе. При этом такие ошибки достаточно сложно искать, так как язык не предоставляет никакой дополнительной информации о правильности написания кода. Поэтому в современной разработке чаще используется TypeScript.

TypeScript — это усовершенствованный вариант JavaScript, который по сути является его надстройкой. В отличие от своего предшественника, данный язык определяет типы получаемых и обрабатываемых данных еще до непосредственного выполнения программы. Такой подход помогает находить ошибки или потенциальные уязвимости еще на стадии разработки программы или ее отдельных частей, что позволяет существенно сократить количество возможных уязвимостей в приложении [51].

Помимо обработки данных на стороне клиента, обрабатывание информации происходит на стороне сервера. Сервер является связующим звеном между клиентом и непосредственно базой данных, где хранятся все необходимые данные приложения. Разработку серверного кода обычно называют «backend» и в нее входят процессы обработки входящих запросов со стороны как клиенткой части, так и со стороны базы данных. Для разработки подобного функционала существует множество разных языков, например: Python, Java, Go, C#, JavaScript и т.д. Каждый язык имеет свои преимущества и недостатки, а его выбор зависит конкретных целей проекта.

Работа серверной части приложения обычно выглядит следующим образом. Клиент посылает запрос, по определенному маршруту, на удаленный сервер, где система должна его принять, обработать и вернуть ответ. Подобные

запросы делятся на конкретные виды, а именно: получение, добавление, изменение или удаление. Исходя из вида запроса, или по-другому метода, сервер осуществляет тот или иной вид операций, а именно:

1) GET — используется для получения каких-либо данных с сервера.

2) POST — предназначен для передачи дополнительной информации на сервер. Чаще всего используется для добавления новых записей в базу данных.

3) PUT — отвечает за полное обновление отдельно взятой записи.

4) PATCH — похож на PUT, но вместо полного обновления используется для частичного изменения записей.

5) DELETE — удаляет запрошенную информацию [10].

Для передачи интернет запросов используются протоколы http и https. Использование первого считается небезопасным, так как, в отличие от https, http не шифрует данные при передаче, что создает брешь в архитектуре безопасности приложения [52]. На рисунке 3 показан пример передачи данных при использовании данных протоколов.

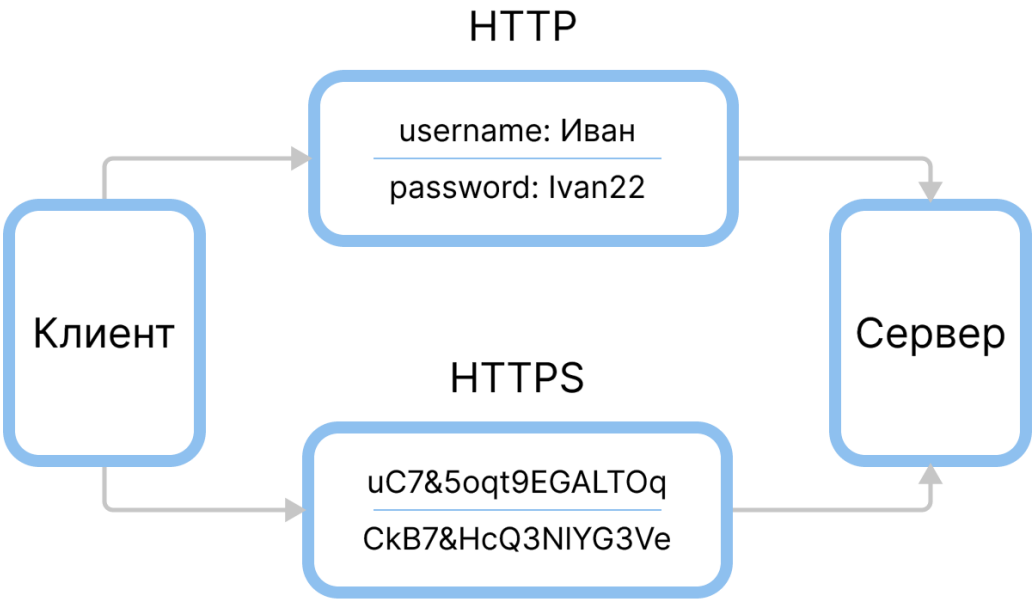

Рисунок 3 – Пример работы протоколов http и https

Вместе с ответом, сервер может также оправлять дополнительную информацию, например «cookie» файлы. Сами файлы могут содержать различного рода информацию, как о пользователе, так и об используемом им устройстве. Такие файлы хранятся на локальном компьютере и могут отправляться вместе с запросами обратно на сервер. Одним из примеров использования подобных файлов является проверка авторизации при помощи уникального идентификатора, который устанавливается в cookie. Данный идентификатор отправляется на сервер каждый раз, когда пользователю нужно подтвердить свою личность. Такой подход позволяет значительно сократить время процесса аутентификации пользователя, но в то же время требует осторожности в настройке, так как зачастую в таких файлах могут передаваться конфиденциальные данные, получив доступ, к которым злоумышленники смогут получить неограниченный доступ к пользовательским данным.

На сегодняшний день, существует несколько способов взаимодействия серверной части приложения с базой данных: классический и с использованием объектно-реляционного преобразования, или по-другому ORM. Классический подход подразумевает прямое использование SQL языка для описания базы данных и сценариев взаимодействия с ней. Но такой подход несет в себе дополнительные сложности, а именно: требуется уверенное владение SQL; изменение наименования переменной в одном месте требует ручного исправления во всех остальных местах; отсутствие встроенной защиты от SQL инъекций предполагает дополнительные проверки входящих данных и т.д. Все эти и другие проблемы решает использование ORM систем. Данная технология позволяет взаимодействовать с базой данных при помощи объектно-ориентированных возможностей современных языков программирования. ORM библиотеки являются посредником между сервером и базой данных приложения. На практике это означает, что у разработчика появляется возможность работать с БД при помощи сторонних языков программирования, например: Python, JavaScript и других. Программа

автоматически переводит сторонний код в SQL формат, причем использует максимально оптимизированный вариант его написания, что значительно сокращает время на разработку и отладку [50].

Исходя из всего вышеперечисленного следует, что разработка надежных, а главное безопасных, систем обработки и хранения данных — это комплексная задача, требующая грамотной настройки как на стороне клиента, так и на стороне сервера. Поэтому выбор технологического стека должен соответствовать всем необходимым требованиям.

1.4 Угрозы безопасности в E-commerce

На сегодняшний день существует значительное число разновидностей киберпреступлений, которые происходят на просторах всемирной паутины. С развитием технологий и мощности вычислительных систем, увеличивается и количество угроз, с которыми приходится сталкиваться как рядовым пользователям, так и держателям интернет-площадок, в том числе интернет-магазинов. Одной из таких угроз являет фишинг.

Фишинг представляет из себя совокупность различных технологий и методов, благодаря которым злоумышленники могут получить доступ к персональным данным пользователей. Отличительной чертой подобных атак является выдача поддельного сайта за настоящий. Атаки, проводимые злоумышленниками, подразумевают заманивание жертвы на фишинговые сайты, путем предоставления поддельных ссылок, которые могут выглядеть как настоящие. Подобные ссылки могут приходить на электронную почту, выдаваться в результатах поиска в интернет-браузерах или быть оставлены в различных социальных сетях. Такие страницы зачастую могут выглядеть идентично с оригинальными и иметь схожий веб адрес, но создаются с единственной целью выманить персональные данные пользователей [56].

В контексте разработки безопасных систем обработки и хранения персональных данных пользователей, данная проблема актуальна тем, что

пользователь, просто переходя по подобной ссылке рискует запустить вредоносный скрипт, который может выманить его персональные данные, даже без ввода какой-либо личной информации. Такой тип угроз называется CSRF-атаками.

CSRF расшифровывается как cross site request forgery и означает межсайтовую подделку запросов. Суть подобной атаки заключается в том, что вредоносный сайт в фоновом режиме отправляет запрос на оригинальный сайт для получение какой-либо информации о пользователе, посредством передачи уникального идентификатора пользователя, который обычно хранится в локальном хранилище на стороне клиента. Получив такой идентификатор, и проверив его, сервис будет уверен, что его прислал настоящий пользователь, которому он принадлежит, в результате будут высланы все запрошенные данные, непосредственно связанные с данным пользователем. Таким образом злоумышленник сможет получить полный доступ к личной информации пользователя, что может быть использовано в дальнейшем для незаконных целей. Важно отметить, что такой вредоносный скрипт могут иметь не только фишинговые сайты, но и вполне безобидные на первый взгляд интернет-ресурсы. Единственным способом защиты от подобного типа атак является грамотная настройка выдачи и хранения идентификаторов доступа на стороне веб-ресурса, непосредственно его предоставившего [57].

Но это далеко не единственный способ, как злоумышленник может заполучить персональный данные пользователей. Одним из наиболее распространенным видов атак на веб-сайты являются SQL-инъекции. Принцип работы подобных атак основан на внедрении вредоносного кода на сервер, посредством передачи его через запросы к базе данных. Одним из способов внедрения такого скрипта может быть обычное поле ввода на самом сайте. Для получения каких-либо данных от клиента, обычно используются HTML формы, где пользователи ресурса могут указать дополнительные данные о себе. В контексте e-commerce платформ это может быть: адрес доставки, время доставки, комментарии к заказу и т.д. Однако, для

злоумышленников эти же поля могут стать средством передачи вредоносного кода. Вместо конкретной информации в строку ввода прописывается SQL запрос к базе данных, который модифицирует стандартный запрос, хранящийся на сервере, новым. В результате подобной атаки сервер может вернуть все запрошенные со стороны злоумышленника данные, в том числе и конфиденциальную информацию других пользователей [58].

Похожим образом работают и атаки типа XSS, что расшифровывается как атака межсайтового скриптинга. Данный вид атаки также может быть внедрен посредством передачи вредоносного кода через формы взаимодействия с пользователем. Но в отличии от SQL-инъекции, данная атака не запрашивает никаких данных у сервера, а становится его частью и начинает работать непосредственно на стороне клиента в момент загрузки веб-сервиса. Активация данного скрипта может происходить различными способами: автоматически при загрузке сайта, при наведении курсора мышки на определённый объект или после активации обеденного события. Загружая сайт, инфицированный подобным вредоносным скриптом, пользователь рискует скомпрометировать свои персональные данные [26].

Несмотря на то, что общая динамика количества киберпреступлений, связанных со взломом пароля, каждый год уменьшается, данный вид угрозы остается актуален и по сей день. На основании результатов опроса, которые проводил Райффайзен Банк в 2020 году, было выявлено, что более четверти опрошенных отметили, что сталкивались с потерей доступа к личному кабинету в результате взлома персональных страниц. Как итог, преступники смогли получить доступ к банковским счетам некоторых пользователей и пытались снять с них деньги [27]. Для произведения подобных атак, преступники могут использовать один из следующих способов: Dictionary attack и Brute-force. Dictionary attack — это атака с использованием списка наиболее часто используемых паролей. Brute-force же представляет собой перебор всех возможных комбинаций символов. Для проведения последнего

типа атаки часто применяются графические ускорители, которые позволяют значительно увеличить скорость подборки пароля [59].

1.5 Современные способы защиты данных

Один из способов защиты персональных данных — это проведение аутентификации пользователя в системе. Аутентификация является промежуточным шагом по предоставлению конкретных прав пользователю при входе в систему. Процесс входа в систему состоит из трех основных этапов:

1) Идентификация. На данном этапе пользователь представляется системе и сообщить ей кем он является. Например, посредством передачи логина, номера телефона, почты и т.д. Этот шаг позволяет системе получить представление о том, кто пытается получить доступ к данным, если такой пользователь существует.

2) Собственно аутентификация. После того как система получила данные о том, кто пытается войти, ей нужно понять, что это действительно тот самый пользователь, а не кто-то другой. Этот процесс похож на предъявление паспорта в банке, когда клиент должен доказать, что он тот, за кого себя выдает. Только в контексте работы с информационными ресурсами передаются обычно не паспортные данные, а например пароли или одноразовые коды активации, чтобы система могла сравнить их с теми, что хранятся в ее базе данных.

3) Заключительным этапом является авторизация. После того как система успешно подтвердила, что пользователь тот самый, начинается процесс выдачи прав доступа. На основании предоставленных данных программа автоматически выдает те права, на которые у конкретного пользователя есть доступ. Например, для просмотра своих персональных данных или любой сопутствующей информации [48]. Для подтверждения личности пользователя существует большое количество методов аутентификации, а именно [62]:

– по логину и паролю;
– по одноразовому паролю;
– с использованием специальных токенов доступа;
– с использованием сертификатов;
– по уникальным биометрическим данным;
– с подтверждением графического ключа;
– с подтверждением географического положения.

Каждый из методов имеет как свои плюсы, так и недостатки. В контексте разработки e-commerce платформ, наиболее часто встречаемыми являются первые 4 варианта.

Комбинация логина и пароля представляет собой классический подход используемый, для подтверждения личности пользователя. Основной принцип работы, данного метода заключается в создании уникального имени пользователя и пароля, состоящего из секретного сочетания символов. Данный пароль необходимо вводить всякий раз при входе в систему.

Наиболее защищенным вариантом аутентификации по паролю является одноразовый пароль. Данный способ позволяет обезопасить себя от риска потери пароля и заключается в генерации уникального кода, при каждом входе в систему [63]. Вариантом реализации такой системы может быть одноразовый код из SMS.

Аутентификация посредством токена представляет собой упрощенный способ авторизации на информационных ресурсах для уже известных пользователей. После того как сервер получает от клиента подтверждение того, что он действительно тот самый человек, которому принадлежит аккаунт, он высылает специальный токен, своего рода уникальный ключ, который сохраняется на клиентской машине. Такой токен обычно содержит какую-либо сопутствующую информацию о пользователе, например персональный идентификатор, в зашифрованном виде. Получив такой токен обратно, система расшифровывает все сохраненные внутри него данные и на основании этих

параметров принимает решение о выдаче или запрете доступа. Аутентификация по токенам бывает 2-ух видов: с отслеживанием состояния и без. Вариант с отслеживанием состояния считается более безопасным, т.к. не передает никакую дополнительную информацию о клиенте, но требует дополнительного места для хранения и более сложен в настройке [28].

Еще одним способом защиты информации является применение криптографии при передаче или хранении данных. При транспортировке данных обычно используют протоколы шифрования TLS или SSL. Основной принцип работы шифрования заключается в приведении данных в нечитаемый вид, посредством проведения над ними различных манипуляций. Данные протоколы работают по принципу ассиметричного шифрования. Такой вид шифрования использует пару ключей, где один ключ является публичным, а второй закрытым. В ассиметричных алгоритмах шифрования один ключ отвечает за непосредственное шифрование, а второй за расшифровку полученных данных. Публичный ключ, как правило, не является конфиденциальным и допускает свободную передачу сторонним лицам, а закрытый наоборот требует надежного хранения внутри системы [60].

Для хранения наиболее чувствительной информации, такой как пароли доступа, используются алгоритмы однонаправленного шифрования. Такой процесс называется хешированием данных. Хеширование представляет собой преобразование входящей информации любой длинны в зашифрованную строку, имеющую фиксированную длину таким образом, чтобы из конечного варианта невозможно было получить изначальный [61].

Выводы по главе 1

В данной главе, на основании федеральных законов, были проанализированы основные понятия и определения персональных данных, а также выявлены основные свойства, которыми должна обладать информация при ее обработке. Было установлено, что безопасные системы обработки

персональных данных должны соответствовать основным законам и положениям, регулирующим обработку информации. Также было выявлено, что информация, с точки зрения обеспечения информации должна обладать тремя основными свойствами, а именно: доступностью, конфиденциальностью и целостностью.

На основании анализа современных методов обработки и хранения данных в совокупности с результатами исследования современных угроз, представляющих опасность для интернет-приложений, в частности платформ электронной коммерции, была подтверждена актуальность разработки современных безопасных систем обработки и хранения данных. В процессе исследования были рассмотрены и основные средства противодействия актуальных угроз в информационном поле.

Глава 2. Разработка безопасной системы обработки данных

2.1 Выбор технологического стека

Разработка систем безопасного хранения и обработки персональных данных в e-commerce подразумевает обеспечение безопасного взаимодействия между пользователем и сервером, поэтому в данной работе основной фокус внимания будет сконцентрирован на 3-ех основных страницах: регистрации, авторизации и оформлении заказа. Так же в ходе разработки будут использованы следующие условные соглашения: предполагается, что пользователь принял условия хранения и обработки персональных данных заранее; все пользователи, прошедшие регистрацию, подтвердили свой адрес электронной почты; после подтверждения оплаты, платеж прошел успешно.

Технологический стек включает в себя довольно широкий список инструментов, затрагивающих практически все аспекты, связанные с разработкой приложения. Все технологии условно можно разделить на две большие категории: основные и вспомогательные. К основным относятся критически важные элементы, на которых будет строится само приложение, а к вспомогательным наоборот, не оказывающие какого-либо серьезного влияния на работоспособность, но позволяющие ускорить или упростить разработку.

2.1.1 Основные технологии

Для разработки интернет-приложения в первую очередь требуется компилятор, то есть программа, которая будет переводить код, написанный человеком, в понятный для машины. Принимая во внимание то, что язык JavaScript не умеет обращаться к файловым системам компьютера по умолчанию, так как изначально заточен под работу исключительно в браузере, лучшим выбором будет использование платформы Node.js.

Node.js состоит из двух основных частей: компилятора и набора инструментов для работы с различными системами компьютера. Данная платформа позволяет легко и быстро разрабатывать масштабируемые приложения, благодаря встроенной системе пакетного управления, которая позволяет устанавливать, и добавлять в свои наработки сторонние библиотеки других пользователей. На основе Node.js работают такие тяжеловесные приложения как Netflix, Uber, PayPal и другие, что свидетельствует о его надежности и эффективности [53].

Разработка e-commerce платформ подразумевает создание клиентской и серверной части приложения. Поэтому гораздо удобнее использовать фреймворк, который позволяет создавать монолитные приложения, включающие как frontend, так и backend. Фреймворк представляет собой библиотеку готовых инструментов для разработчиков приложений, позволяющих заметно сократить количество строк кода [54]. Одним из таких фреймворков является Next.js.

Next.js предоставляет обширный инструментарий, позволяющий быстро создавать все необходимое для эффективной работы приложения. Данный фреймворк имеет ряд преимуществ:

1) В отличие от большинства библиотек позволяет создавать не только клиентскую, но и серверную часть приложения, что в свою очередь избавляет от необходимости изучать дополнительные языки программирования и связанные с ними библиотеки.

2) Использует упрощенную систему маршрутизации, не требующую дополнительной настройки.

3) Имеет хорошую совместимость с большинством сторонних библиотек и плагинов.

4) Поддерживает работу с TypeScript, что сокращает число возможных ошибок при разработке.

5) Прост в освоении и имеет удобную документацию, где подробно расписаны все основные возможности фреймворка [11].

Разрабатывая интернет-приложение также, нужно помнить о том, что информацию нужно где-то хранить. В контексте e-commerce платформ, решающее значение имеет целостность и систематизация данных, поэтому нереляционные базы данных здесь не подойдут. Лучшим решением для небольшого проекта будет SQLite.

SQLite — это полноценная реляционная база данных с открытым исходным кодом и подробной документацией, сочетающая в себе функциональность и простоту использования. Отличительными преимущественными характеристиками данной базы можно назвать:

1) Легкость настройки. SQLite не требуется установка или дополнительная настройка серверного процесса, который будет отвечать за запуск или остановку БД. Внутри базы отсутствует файлы конфигурации приложения, в том числе те, которые отвечают за аварийное восстановление данных, все работает автоматически.

2) Кроссплатформенность. База поддерживает работу с большим количеством операционных систем, таких как: Windows, Linux, Mac, Android, iOS и т.д. Данная особенность позволяет легко переносить базу данных на другие платформы при необходимости.

3) Небольшой вес. Размер SQLite зависит от используемой операционной системы, архитектуры процессора, используемого компилятора и других параметров, но не превышает 1-го мегабайта памяти. Так, например для операционной системы на базе Mac библиотека займет примерно 750 килобайт памяти, а для систем на базе Linux около 650 килобайт. Поэтому для небольших проектов, где объем возможной памяти для хранения информации ограничен, данный фактор может быть ключевым.

4) Надежность. В основе библиотеки SQLite лежат принципы ACID, что расшифровывается как неделимость, последовательность, изолированность и долговечность. Использование такого подхода означает, что все операции, связанные с изменением информации внутри базы данных, либо происходят полностью, либо не происходят вообще, что защищает целостность данных.

Использование данной библиотеки дает гарантии сохранности информации при: перебоях питания, сбоях операционной системы или сбоях непосредственно внутри программы [12].

Для настройки безопасного взаимодействия между серверной частью и базой данных приложения отлично подходит использование ORM систем, так как они предоставляют дополнительную защиту и инструментарий для написания оптимизированных запросов к хранилищу данных. Исходя из текущего стека, наилучшим вариантом будет применение Drizzle ORM, так как он построен с учетом языка TypeScript и имеет хорошую совместимость с множеством баз данных, в том числе и с SQLite.

Drizzle предоставляет широкий спектр настроек и утилит для настройки и взаимодействия с базами данных. Для удобства разработчиков поддерживается несколько стилей взаимодействия с БД: реляционный и SQL подобный. Реляционный метод работы с хранилищем основывается на использовании заранее заготовленных шаблонов из числа наиболее частых операций, что позволяет избегать чрезмерного написания кода, так как не приходится использовать дополнительные объединения или запросы к другим таблицам. Однако, иногда подобного функционала недостаточно для выполнения той или иной операции. Эту проблему решает SQL подобный способ взаимодействия с БД. Данный метод также предоставляет определенный набор шаблонов, но в отличии от реляционного подхода, использует синтаксис, напоминающий классические SQL запросы. Ключевой же особенностью этого метода является возможность встроить в него непосредственно и сам SQL. Для этого внутри Drizzle существует специальный оператор, который вызывается одной командой, что позволяет использовать не только заранее заготовленные функции, но внедрять нестандартные запросы, как если бы разработка велась на чистом SQL. Важно отметить, что на текущий момент Drizzle является единственной ORM, которая поддерживает несколько способов взаимодействия с реляционными базами данных [13].

2.1.2 Вспомогательные технологии

Разработка любого интернет-приложения не обходится без визуального оформления. Для его настройки используется язык CSS, что расшифровывается как «каскадная таблица стилей». Основное назначение данного языка заключается в настройке визуального стиля для HTML элементов страницы. Однако написание стилей с нуля отнимает достаточно большое количество времени. Поэтому лучшим вариантом настройки визуализации будет использование библиотеки Tailwind CSS.

Tailwind представляет собой универсальную библиотеку уже готовых для работы стилей. Данная технология позволяет писать стили непосредственно в компоненте, где они будут использоваться, а количество символов, требуемое для их объявления, может быть в несколько раз меньше, чем в классическом варианте использования CSS, что позволяет в разы ускорить разработку программы. Однако библиотека не накладывает никаких ограничений, позволяя использовать как свои собственные стили, так и переопределять уже существующие. Tailwind также обладает наглядной и подробной документацией [14]. Для удобства использования данной библиотеки понадобится установить несколько дополнительных утилит, а именно: Tailwind-merge, позволяющую объединять несколько стилей Tailwind в один, без риска создания конфликтов [15]; Clsx, также позволяющую объединять несколько стилей, но в отличии от Tailwind-merge дает возможность применять переменные и условные конструкции внутри для настройки стилей [16].

Для ускорения разработки интерактивных элементов форм с применением готовых стилей потребуется использование заранее заготовленных элементов из HeadlessUI. Это минималистичная библиотека содержит список готовых элементов форм, таких как: диалоговое меню, поля ввода, переключатели, кнопки и т.д. Стоит отменить, что HeadlessUI разработан специально для работы с Tailwind CSS [17].

Для отправки интернет запросов внутри Next.js существует множество заранее заготовленных методов, позволяющих отправлять и принимать данные со стороны сервера или клиента, еще до загрузки основной страницы. Однако использование встроенных функций требует дополнительной ручной настройки, что может занять достаточно много времени. Для решения данной проблемы подойдет библиотека TanStack Query.

TanStack Query — это мощный инструмент специализирующийся на отправке интернет запросов, разработанный для упрощения взаимодействия между клиентом и сервером. TanStack Query позволяет, даже без дополнительной настройки, проводить такие операции как: фоновая загрузка данных, кеширование, обработка устаревшей информации и многое другое. Использование данной библиотеки улучшает читаемость кода, сокращает его количество, повышает производительность и позволяет отслеживать текущее состояние запроса, благодаря встроенным методам. Имеет отличную совместимость с Next.js и TypeScript [18].

Работа с данными в информационном пространстве означает обработку интернет-форм, но стандартные методы взаимодействия с формами не предоставляют для этого практически никакого функционала. Разработка же подобного инструментария с нуля требует значительных средств и времени. React Hook Form решает данную проблему.

React Hook Form — это библиотека специализирующаяся на взаимодействии с HTML формами внутри приложения. Как говорилось ранее стандартные элементы форм, несмотря на установку типа вводимых данных, допускают ввод недопустимых символов. Использование React Hook Form позволяет следить за: типами вводимых данных, допустимыми символами, валидностью, как отдельных полей, так и формы целиком. Большим плюсом является то, что библиотека работает не только с классическими HTML полями, но и с большим количеством сторонних библиотек, в том числе и с HeadlessUI [19].

Для валидации полей, в React Hook Form требуется настроить типы получаемых данных и шаблоны заполнения. Все это можно прописать с нуля, но гораздо быстрее и эффективнее воспользоваться минималистичной библиотекой Zod. Zod позволяет быстро создавать сложные типы валидации данных, благодаря простому синтаксису и обширному инструментарию, включающему возможности настройки: длины, типа данных, допустимых символов, соответствия определенной схеме и специальных дополнительных проверок, расширяющий стандартные способы валидации [20]. Данную библиотеку удобно использовать не только для валидации данных на стороне клиента, но и на стороне сервера проекта. Это позволяет следить за целостностью отправленных данных, или блокировать невалидные запросы к серверу, что укрепляет безопасность системы.

В некоторых случаях может потребоваться отслеживания статуса авторизации на стороне клиента, чтобы вывести какую-либо дополнительную информацию о пользователе или наоборот скрывать, если он не авторизирован в системе. Для этого обычно используется локальное хранилище данных. Данный способ помогает сокращать количество запросов и передаваемой информации между серверной частью и клиентской частью приложения. Классические варианты локальных хранилищ довольно громоздкие и требуют большого количества времени для изучения. Поэтому наиболее подходящим вариантом в данной ситуации будет использование Zustand.

Zustand представляет собой небольшую библиотеку, которая позволяет хранить и изменять определенные состояния компонентов. В отличии от своих конкурентов, Zustand не оборачивает приложение целиком и не требует прописывания сложных схем взаимодействия. Идеально подходит для небольших проектов [21].

Для разработки механизмов регистрации и аутентификации пользователей также потребуется добавить несколько дополнительных утилит, таких как: Lucia, Arctic и bcrypt.

Lucia представляет собой небольшое приложение, позволяющее легко и быстро настраивать системы аутентификации пользователей посредством создания уникального, персонального идентификатора сессий. Поддерживает интеграцию с аутентификацией через сторонние сервисы, так называемая OAuth аутентификация [22].

Для упрощения процесса подтверждения пользователя через сторонние сервисы, используется сервис Arctic. Arctic является удобным сборником, содержащим стандартные настройки, необходимые для OAuth аутентификации, наиболее популярных компаний, таких как: Yandex, VK, Zoom, GitHub, Google и другие [23].

Bcrypt же является библиотекой, специализирующейся на шифровании и хешировании данных, и использующей одноименный алгоритм. Данный алгоритм шифрования широко используется с 1999 года и хорошо зарекомендовал себя. Несмотря на возрастающие вычислительные мощности, bcrypt все еще остается актуальным в применении, так как использует настраиваемую сложность шифрования данных, включающую: стоимость операций, дополнительный символы (соли) и специальные ключи. Такой подход обеспечивает надежную защиту данных [55].

Также для взаимодействия с пользователем понадобятся еще пара дополнительных инструментов: Luxon и react-input mask. Первая представляет собой небольшой, но эффективный инструмент для работы с датами и временем, предоставляя удобные возможности по настройкам: формата дат, часового пояса, сложения, вычитания и т.д. А вторая позволяет настраивать так называемые маски, для полей ввода. Использование масок позволяет улучшить пользовательский опыт и унифицировать данные, которые приходят со стороны пользователей. Например, для ввода номера телефона существует множество разных шаблонов и вариантов, но используя маску ввода, для всех пользователей автоматически будет установлен один единственный вариант отправки данных на сервер. Это систематизирует собираемую информацию, и позволит лучше ее обрабатывать [24, 25].

2.2 Проектирование и разработка базы данных

Разработка базы дынных является фундаментальной основной для платформ электронной коммерции, так как именно в ней хранятся и анализируются не только списки доступных товаров и услуг, но и личные пользовательские данные. Разработка подобной системы происходит в несколько этапов:

– определение сущностей и хранимых параметров;

– построение визуальной блок схемы;

– перенос всех параметров и связей в код.

Использование данного подхода при проектировании базы данных дает ряд преимуществ, а именно: позволяет избежать дублирование таблиц или параметров, уменьшает количество потенциальных ошибок при написании кода, наглядность [64]. Проектная схема базы данных представлена на рисунке 4.

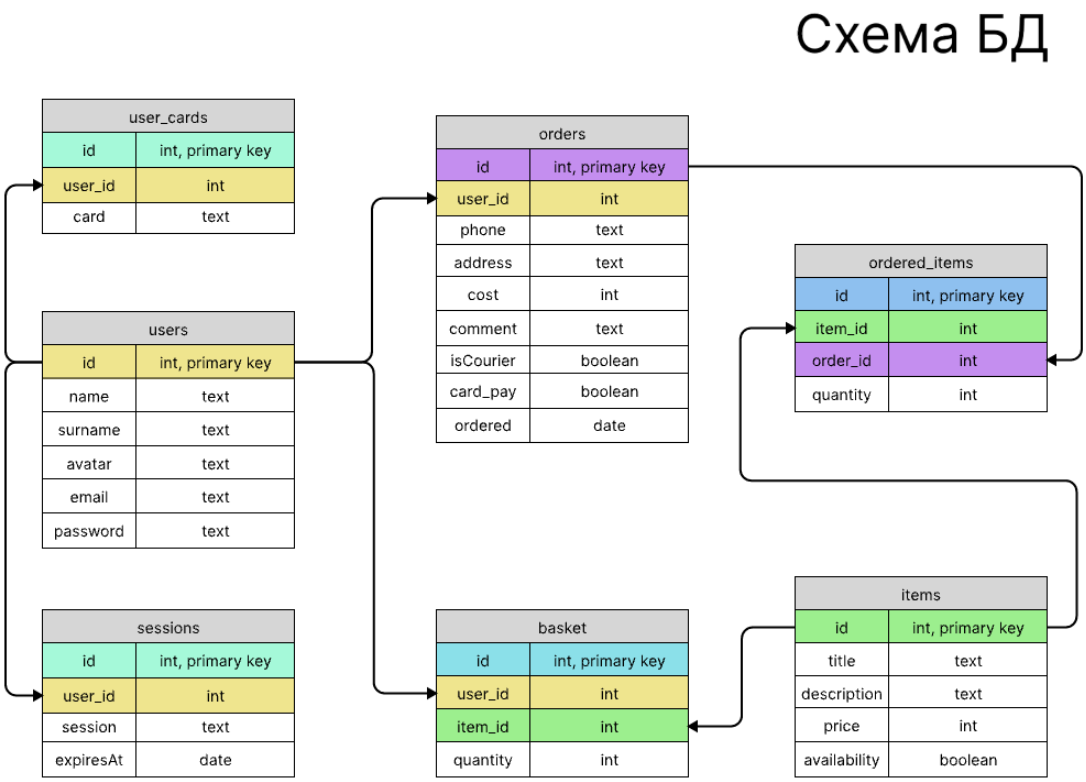

Рисунок 4 – Блок схема базы данных

В соответствии с данным планом, определим перечень хранимых сущностей и параметров. В контексте платформ электронной коммерции, основными сущностями являются: товары и пользователи. Данные модели являются ключевыми, так как все взаимодействие строится вокруг них. К дополнительным моделям относятся: банковские карты, корзины товаров, заказы и истории заказов и т.д.

При проектировании сущности пользователя необходимо учесть то, что некоторые поля должны быть уникальными. К таким полям можно отнести: логин, почту, номер телефона, паспортные данные, идентификационный номер и т.д. Сама возможность сохранить несколько одинаковых значений, связанных с идентификацией пользователя, внутри базы создает серьезную брешь в безопасности. Для создания уникальных полей в SQLite при помощи Drizzle существует несколько команд: primaryKey() и unique(). Обе команды создают уникальные поля, которые не позволят внести такую же запись в таблицу снова. Такой подход гарантирует, что в базе данных невозможно будет создать 2-ух пользователей с одинаковой электронной почтой или одинаковым идентификатором. Команда primaryKey(), помимо уникальности, создает так называемый первичный ключ, который гарантирует, что значение также не будет пустым. На первичный ключ могут ссылаться записи в других таблицах, если к нему привязан внешний ключ.

Для обеспечения целостности и последовательности данных между таблицами, в первичных ключах устанавливаются специальные маркеры, которые определяют, что нужно делать с другими записями в БД, которые были связаны с текущей при удалении или изменении. Существует 4 варианта действия, а именно:

1) CASCADE — повторяет текущую операцию для всех, связанных с текущими данными записями.

2) NO ACTION — не применяет никаких действий.

3) SET NULL — устанавливает NULL для всех зависимых записей.

4) SET DEFAULT — устанавливает значение по умолчанию [31].

Использование данных команд позволяет существенно сократить количество операций, необходимых для внесения изменений в таблицу, но что более важно, предотвращает искажение хранимой информации.

Взаимодействие между таблицами происходит посредством объединения нескольких полей из разных таблиц при помощи одного из видов связи. Связи бывают трех видов: один к одному, один ко многим, многие ко многим. Первый тип связи используется достаточно и редко и предназначен в основном для вынесений отдельных данных одной таблицы в другую, с сохранением привязки к определенной записи. Связь вида один ко многим позволяется привязать к одной записи несколько значений из другой таблицы. Последний вид связи позволяет работает схожим образом, что и один ко многим, но работает в оба направления. Для создания связи типа многие ко многим, в отличии от первых двух, используется дополнительная таблица, связывающая ключи из двух других. Важно отметить, что неправильное прокладывание связей может привести к несанкционированному доступу к чужим персональным данным или утечкам баз данных.

Таким образом, определив основные сущности и проложив между ними все необходимые связи, получилось создать устойчивую базу данных, с возможностью дальнейшего масштабирования, путем добавления новых сущностей или декомпозиции уже существующих.

2.3 Разработка backend приложения

Backend или по-другому серверная часть приложения отвечает за обработку клиентских запросов, с предоставлением результатов обработки этих самых запросов. Backend является связующим звеном между frontend приложения и базой данных, поэтому требует особого внимания при разработке. Ошибки, допущенные при проектировании серверной части,

могут приводить к серьезным проблемам, как с работоспособностью приложения, так и с его безопасностью. Для минимизации возникновения подобных проблем применяются определенные архитектурные стили и паттерны. Одним из таких паттернов является REST. Данная аббревиатура расшифровывается как Representational State Transfer, что переводится как транспортировка представленного состояния [29].

Стиль REST представляет собой набор из 6 пунктов, которым нужно придерживаться при разработке веб-приложения, а именно: клиент-сервер, отсутствие состояния, единый интерфейс, многоуровневая система, кешируемость и предоставление кода по требованию. Важно отметить, что данные пункты являются именно рекомендательными, а не строгими правилами, что дает возможность адаптировать приложение исходя из текущих потребностей.

Принцип построения приложения через клиент-сервер означает разделение обязанностей между клиенткой частью приложения и серверной. Согласно данному принципу, серверная часть должна отвечать только за прием, обработку и выдачу данных, в то время как за клиентской частью закрепляется задача отображения этих самых данных. Благодаря такому подходу, клиент получает возможность перерисовывать не всю страницу целиком, а только ту часть, данные которой изменились, что значительно ускоряет работу системы в целом.

Отсутствие состояния в REST означает то, что сервер не хранит никакой информации о состоянии со стороны клиента, и каждый новый запрос должен обрабатываться независимо от предыдущего. Реализация данного пункта предполагает, что все запросы, поступаемые на сервер, содержат всю необходимую, для их обработки, информацию [29].

Принцип единого интерфейса подразумевает использование единого шаблона для взаимодействия клиента и сервера. Иными словами, все маршруты обработки данных, поступающих со стороны клиента, должны иметь: четкую структуру, унифицированный формат передачи данных и

методы обработки. Таким образом, все операции, связанные с пользователем, должны проводится по пути нахождения пользовательского обработчика данных, а операции связанные с товарами по пути обработки товаров соответственно.

Многоуровневость в REST дает возможность соединять несколько API приложений в цепочку запросов. Например, если клиент запрашивает прогноз погоды, то при обработке такого запроса, сервер может отправить данный запрос на другой ресурс и так далее, пока тот не будет обработан в конечной точке и не вернет результат, посредством обратной передачи данных среди всех участников цепи [30].

Кешируемость в контексте разработки REST API, означает возможность сохранять часть данных на стороне клиента. Такой подход позволяет существенно снизить нагрузку как на стороне клиента, позволяя не перерисовывать страницу при отсутствии изменений в результате полученного ответа, так и на стороне сервера, путем сокращения общего числа запросов к нему.

Принцип кода по требованию означает возможность выполнения скриптовых сценариев на стороне клиента, посредством передачи его сервером. Данный принцип предназначен в первую очередь снижения нагрузки на сервер, и позволяет делегировать часть функционала, например для валидации или модификации данных, клиенту [30]. Визуальное представление приложения, построенного по архитектуре REST представлено на рисунке 5.

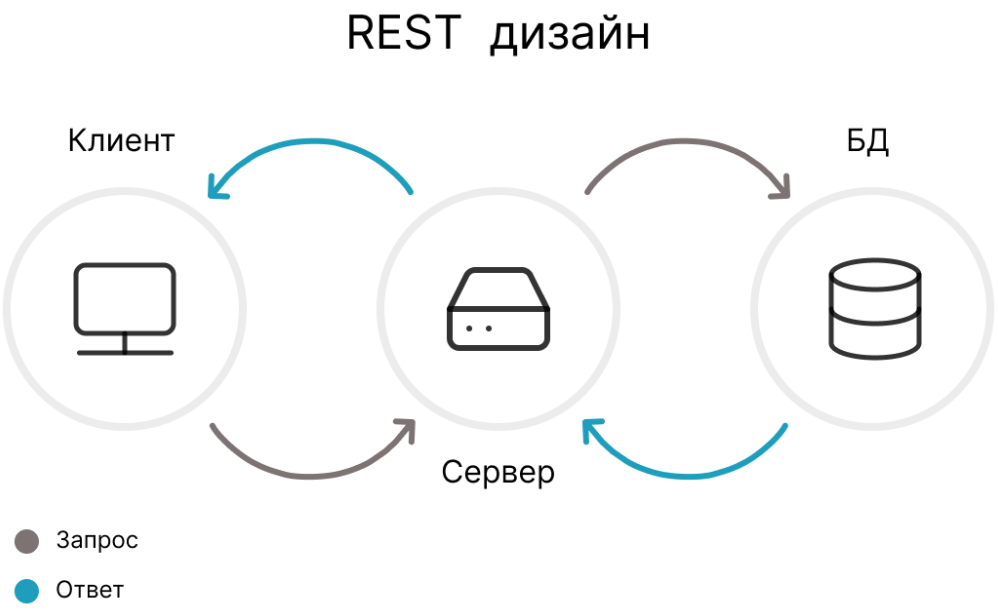

Рисунок 5 – Методология REST API

Приложение, построенное с использованием REST архитектуры, имеет хорошую совместимость с протоколами передачи данных HTTP и HTTPS. Это объясняется тем, что Рой Филдинг, сформулировавший основные принципы архитектурного стиля REST приложений, также является и разработчиком протокола HTTP [30]. Данная особенность позволяет быстро и в короткие сроки внедрять CRUD операции в сервис.

CRUD представляет собой акроним, обозначающий четыре вида основных операции, производимых над данными, а именно:

1) Create — отвечает за добавление новых данных.

2) Read — предоставляет доступ к хранимой информации.

3) Update — обновляет хранимые данные.

4) Delete — производит удаление записи из базы данных [65].

CRUD операции являются основой для большинства интернет-приложений, в том числе и платформ электронной коммерции. Стандартные HTTP методы, идеально сочетаются с подобной архитектурой, так как собственные встроенные методы протокола реализуют схожий

функционал. Примерами реализации подобного сервиса могут быть процессы добавления пользователя в базу данных, получение имени пользователя, обновление пароля, удаление пользователя из системы.

В соответствии с принципом безопасности об обеспечения целостности данных, при осуществлении подобных манипуляций, требуется обеспечить их сохранность и последовательность. Специально для таких целей используются транзакции. Транзакция представляет собой последовательность из нескольких операций, проводимых над данными, которая либо выполнится полностью, либо не выполнится вообще [67]. Применение транзакций при обработке информации, помогает избежать ситуаций, при которых часть данных изменилась, а часть нет. Примером такой ситуации в контексте e-commerce платформы может быть процесс оплаты товаров и услуг, когда из-за потери интернет-соединения или перебоях с электричеством, часть товаров или даже все, могут оказаться неоплаченными, даже если у клиента списались деньги со счета. Использование же транзакций делает такую ситуацию невозможной, так как заменяет несколько операций на одну, которая либо выполнится, либо нет, что обеспечивает целостность и сохранность данных.

Для реализации механизмов защиты на стороне сервера, в e-commerce, при выдаче персональной информации используются дополнительные инструменты, позволяющие скрывать особо чувствительные данные. К таким данным относятся: пароль, логин, номер банковской карты и т.д. Стоит отметить, что в зависимости от типа хранимых данных стоит применять различные подходы. Например, для обеспечения безопасности пользовательских данных, пароль, который хранится в базе данных, никогда и ни при каких обстоятельствах не должен возвращаться при запросе какой-либо информации о пользователе, так как существует угроза его раскрытия третьим лицам.

В тоже время, такие данные как электронная почта или номер банковской карты, несмотря на свою значимость, могут потребоваться для сверки каких-

либо данных. Например, у пользователя, внутри базы данных, хранится несколько банковских карт и нужно выбрать только одну конкретную, или требуется удостовериться, что к приложению привязана конкретная почта. Современные интернет-ресурсы, в частности платформы электронной коммерции, запускаются на большом количестве разнообразных устройств от персональных компьютеров, до мобильных телефонов. В связи с этим локации, где может находится клиент, могут быть достаточно многолюдными, поэтому возникает риск раскрытия персональных, когда злоумышленник их может просто подсмотреть со стороны [66]. Наиболее эффективным решением в данной ситуации будет целенаправленное сокрытие части информации. В таком случае возможными данными, которые отправляются с сервера могут быть: последние 4 цифры номера карты или показ только первых нескольких символов адреса электронной почты вместе с доменом.

При реализации такого метода обработки персональных данных, клиент будет получать достаточное количество информации, необходимое для совершения или подтверждения како-либо операции, а злоумышленник, даже если сможет получить доступ к устройству отображения данных, не сможет получить к ней полного доступа, что дает гарантии от ее дальнейшего неправомерного использования. Пример реализации данного механизма представлен на рисунке 6.

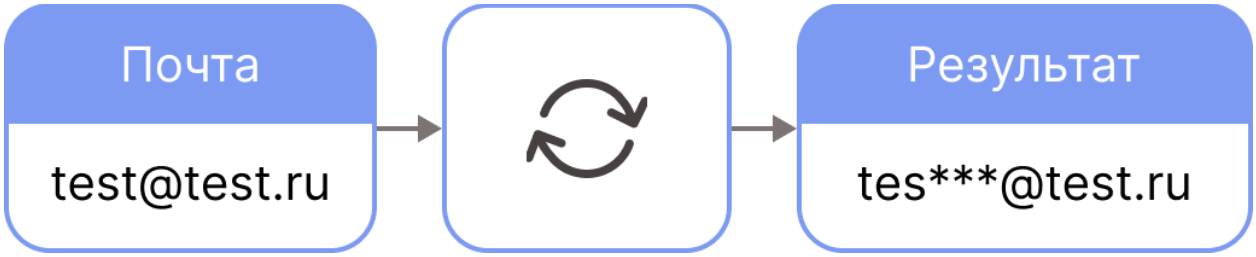

Рисунок 6 – пример работы функции частичного сокрытия информации

2.4 Разработка frontend приложения

Next.js, используемый для разработки приложения, разработан на основе фреймворка React, который позволяет создавать отдельные, независимые друг от друга компоненты и элементы для веб-страниц [32]. Данный подход позволяет собирать интернет-страницы из отдельных блоков, которые условно можно разделить на обязательные и дополнительные. К обязательным блокам можно отнести элементы, которые являются неизменными при переходе от страницы к странице. А к дополнительным наоборот, постоянно изменяющие свое состояние. Такая особенность позволяет создавать универсальные одностраничные сайты или по-другому SPA приложения [68].

SPA расшифровывается как Single Page Application (одностраничное приложение). Данная методология построения веб-ресурсов основывается на частичном обновлении содержимого страницы вместо классического перерисовывания всего содержимого с нуля. Использование SPA дает ряд преимуществ, а именно:

1) Улучшает скорость загрузки страниц, за счет сокращения количества информации, требуемой со стороны сервера, и количества обновляемых элементов.

2) Позволяет кешировать, то есть запоминать последнее состояния ранее отрисованных элементов, что позволяет снижать нагрузку системы на стороне клиента, при повторной отрисовке компонентов.

3) Ускоряет разработку приложения, так как дает возможность для разработчиков работать над отдельным частями приложения независимо друг от друга, а также делится общими наработками.

4) Позволяет создавать мультиплатформенные приложения, благодаря использованию единой кодовой базы и возможности создавать адаптивные элементы веб-интерфейса [33].

Пример работы SPA приложения представлен на рисунке 7.

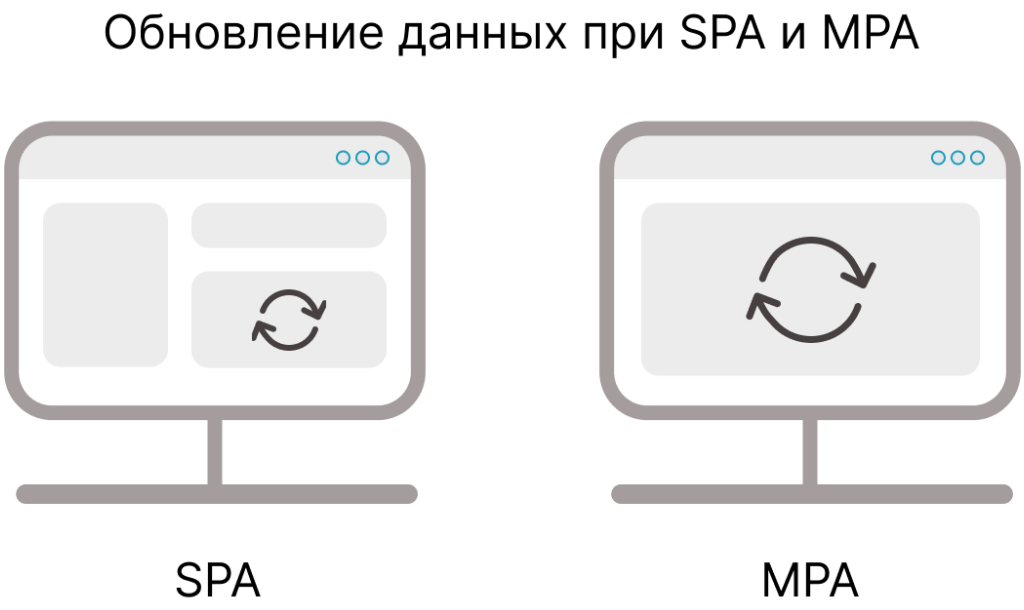

Рисунок 7 – Пример работы SPA приложения

Архитектура построения SPA приложения выглядит следующим образом: предоставление клиенту всей, необходимой для работы frontend приложения, кодовой базы; разработки и настройки маршрутов, отвечающих за изменение отдельных элементов веб-приложения; разделение элементов на независимые друг от друга блоки; настройка локальных хранилищ и состояний, необходимых для хранения временных данных на стороне клиента; использование асинхронных фоновых запросов к серверу, для отрисовки новых компонентов [34].

Разработка элементов хранения и обработки данных на стороне клиента, в соответствии с концепцией SPA, будет включать в себя создание: маршрутов, локальных хранилищ, инструментов для взаимодействия сервера с клиентом, систем проверки полученных данных.

При создании маршрутов, по которым страница будет динамически перерисовываться, нужно учесть, что маршрут может быть: статическим, динамическим, защищенным и незащищенным. Статические маршруты, в отличии от динамических заранее определены и никогда не меняются,

например адрес главной страницы. Динамические же отрисовываются в зависимости от переданных параметров в адресной строке, в контексте e-commerce платформ это может быть уникальный идентификатор пользователя или товара [35].

Если говорить про страницу пользователя, то информация, содержащаяся на ней, может быть конфиденциальна, поэтому отображение данной страницы должно происходить только при подтверждении личности пользователя. Для этого существует механизм разделения на защищенные и незащищенные маршруты. Защищенные маршруты используются для запрещения доступа пользователя к информации, которая может содержать личные данные пользователя. Помимо защиты данных, это предотвращает возникновение ошибок в отрисовке элементов страниц и противодействует введению пользователя в заблуждение. Незащищенный же маршрут напротив, никак не препятствует взаимодействию с клиентом. Информация, хранимая на таких страницах, является публичной и не несет в себе никаких рисков раскрытия.

Однако несмотря на то, что страница может быть публичной, на ней все еще может потребоваться сокрытие определенной информации, например, если говорить про разработку интернет-магазина, это могут быть элементы добавления товаров в корзину, или отображение личной информации о пользователя. Именно для этих целей используются локальные хранилища. По своей структуре они напоминают набор глобальных, то есть доступных в любой части приложения, переменных, хранимых на стороне пользователя. Данный механизм используется в первую очередь для сокращения общего числа запросов и информации, передаваемой между клиентом и сервером. Данные, находящиеся в таких хранилищах, как правило не несут серьезной угрозы для безопасности пользователей и могут является общедоступными, например фамилия и имя пользователя [36].

Используя подобные хранилища, можно также отслеживать авторизирован ли пользователь внутри системы или нет. Нужно отметить, что

такой параметр обычно является просто идентификатором, который, при положительном значении, отправляет запрос, подтверждающий авторизацию на сервере.

Для реализации фоновых запросов, в соответствии с концепцией SPA, используется несколько способов для их отправки, а именно контролируемый и неконтролируемый со стороны пользователя. Неконтролируемые способы основаны на отправки запросов непосредственно в момент отрисовки страницы и имеют весь необходимый для этого набор данных. К таким запросам относятся: проверки доступа, получение списков данных для отображения на странице и т.д. Контролируемые же происходят только в момент определённых действий пользователя, при взаимодействии с интерфейсом приложения.

Основным элементом, позволяющим проводить контролируемые запросы к серверу, являются HTML формы. В контексте разработки платформ электронной коммерции, примерами таких форм могут быть: регистрация, авторизация, форма добавления банковских карт или форма подтверждения заказа. Ключевым этапом при разработке подобного функционала является валидация, так как сервер ожидает, что полученные данные, придут в определенном виде или формате. Иными словами, если сервер ожидает увидеть в запросе адрес электронной почты, то клиент не должен посылать ничего кроме правильно написанного адреса почты, иначе это может нарушить общую логику работы всего приложения.

Для обеспечения валидности входных данных могут применяться различные способы, например: проверка на соответствие шаблону, ограничение количества символов, запрет на использование определенных символов, проверки на пустые значения и т.д. Одним из наиболее распространенных способов проверки введенных данных являются регулярные выражения.

Регулярные выражения представляют собой набор команд, при комбинировании которых можно собрать определенный шаблоны для работы с текстовыми данными. Данные шаблоны могут применяться для:

- поиска определенной строки или символов в тексте;
- полного или частичного соответствия введенных данных;
- парсинга, или по-другому разбиения данных [69].

Использование регулярных выражений для валидации HTML форм, позволяет повысить общую архитектуру безопасности приложения, так как именно они могут быть источником внедрения вредоносного кода посредством SQL инъекций или XSS атак. Для усиления защиты от подобных атак также требуется настроить политику защиты контента CSP. Принцип работы данного метода состоит в том, чтобы добавлять ко всем ответам с сервера определенный заголовок, а именно Content-Security-Policy. Такой заголовок содержит определенный набор правил и инструкций, по которым сайт будет работать, например: разрешить или запретить выполнение определенных скриптов, настроить адреса допустимых изображений, запретить встраивание содержимого сайта на других площадках и т.д. [40].

2.5 Проектирование систем аутентификации

Наиболее важным этапом в разработке интернет-приложения, в том числе и e-commerce платформ являются системы подтверждения личности пользователей. Основная задача таких инструментов — это обеспечение безопасности в системе. Приложения интернет-магазинов часто хранят внутри себя большой объем данных, который может включать в себя: данные банковских карт, состояние личного счета, историю покупок, персональные скидки и т.д. Поэтому доступ к подобным данным должен быть разрешен только для санкционированных пользователей. В рамках данной работы будут рассмотрены несколько вариантов реализации механизмов аутентификации: внутри самого приложения и с использованием внешних сервисов.

2.5.1 Аутентификация на стороне сервера

Для реализации данных механизмов на стороне сервера, требуется разработать несколько способов подтверждения личности, а именно: первичный и вторичный. Первичный способ представляет собой классический вариант ввод логина и пароля на стороне клиента и последующим подтверждением их на стороне сервера. Для реализации надежного метода аутентификации подобным образом, необходимо произвести первоначальные настройки как на стороне клиента, так и на стороне сервера.

Основываясь на данных, полученных при исследовании надежности паролей, проводимого Уральским экономическим университетом в 2022 году, наиболее важным параметрами при составлении пароля является: его длина и разнообразие используемых символов [70]. Используя эти данные, на стороне клиента, при составлении пароля, необходимо проводить следующие проверки: длинна не менее 8 символов, наличие букв в нижнем регистре, наличие букв в верхнем регистре, наличие специальных символов. Наиболее эффективными, с точки зрения защиты будут пароли длинной более 11-ти символов, однако использование подобного ограничения может дать явные подсказки для злоумышленников, позволив сократить диапазон проверяемых значений.

Хранение же паролей на стороне сервера должна осуществляться с использований криптографических технологий. Используемый, а данной работе алгоритм bcrypt имеет хорошие показатели устойчивости к переборам данных. Так, на расшифровку пароля длинной в 4 символа, при значении параметра стоимости в 10 единиц, использующего только буквы нижнего регистра, понадобится 4 дня [55]. Использование данного алгоритма, в сочетании с устойчивым ко взломам паролем, дает гарантии того, что даже при утечке хеша пароля в открытый доступ, злоумышленнику потребуется значительное время и ресурсы для расшифровки пароля. Пример работы данного алгоритма представлен на рисунке 8.

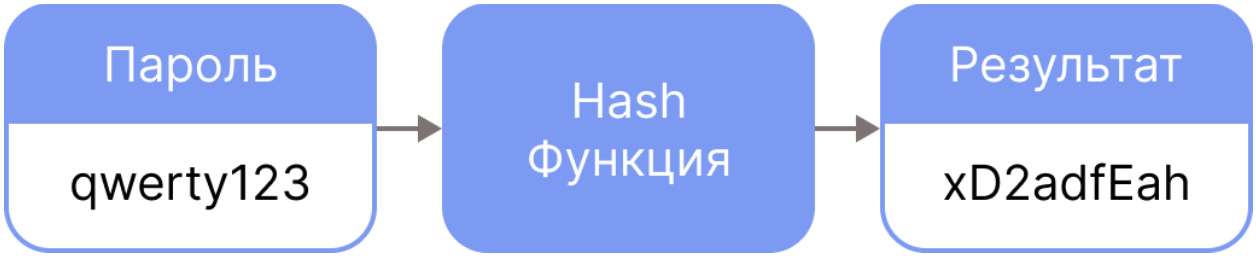

Рисунок 8 – Хеширование паролей

Естественным минусом первичной аутентификации является ее трудозатратность, так как интернет-приложения могут содержать множество страниц, которые содержат конфиденциальную информацию, и если оставить только ее, то данные манипуляции придется проводить при каждом новом запросе персональных данных. Поэтому обязательно нужно предусмотреть вторичный способ подтверждения личности пользователя, например с использованием токенов. Токен представляет собор уникальную комбинацию символов, которая выдается пользователю после прохождения успешной первичной аутентификации. Наличие данного токена в запросе и будет являться доказательством того, что запрос поступает от проверенного клиента. Пример аутентификации по токенам представлен на рисунке 9.

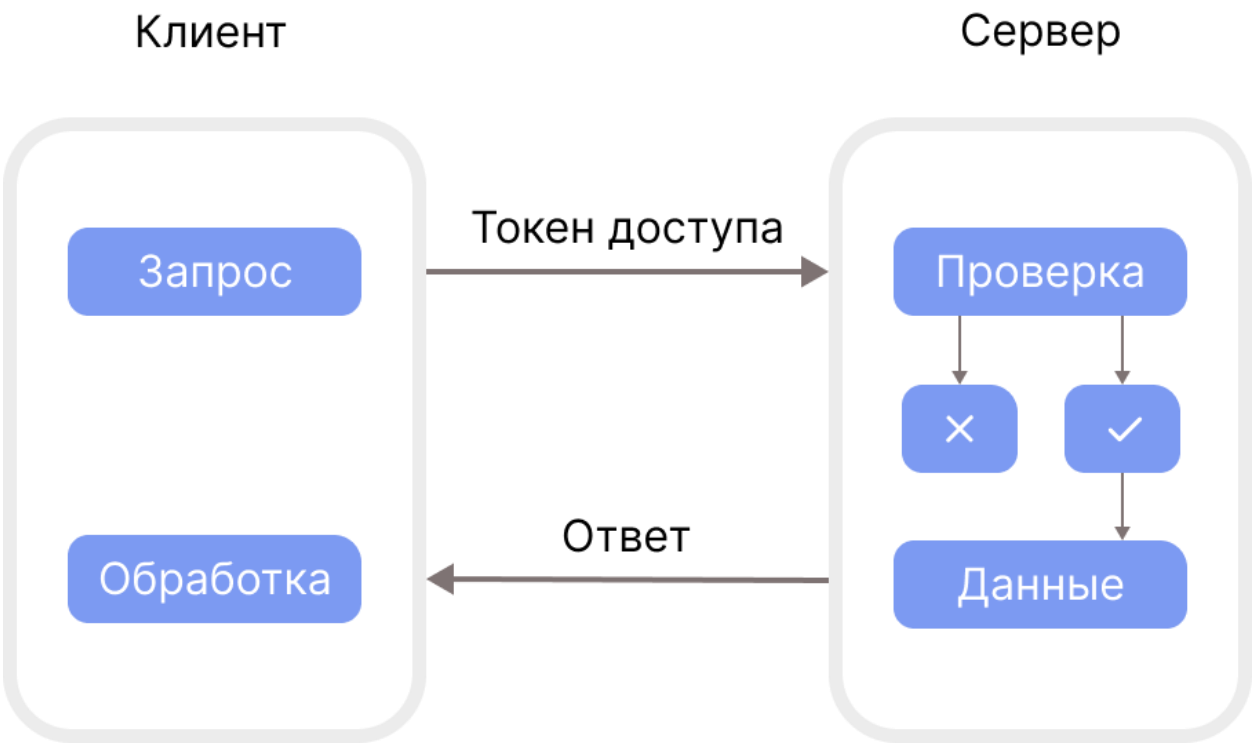

Рисунок 9 – Аутентификация посредством токенов

В контексте разработки REST приложений, наиболее подходящей реализацией технологией с использованием токенов доступа будет JWT. Однако, именно в этом месте стоит отойти от общих принципов построения приложения по архитектуре REST. Благодаря гибкости выбранного архитектурного стиля это можно легко реализовать. Дело в том, что при использовании концепции отсутствия состояния, которого придерживается JWT, система не имеет контроля над текущим количеством устройств, которое имеет доступ к системе. Помимо этого, JWT токен, должен передавать внутри себя определенный набор данных, который связан с пользователем, например его уникальный идентификатор. Еще одним ощутимым недостатком данного метода является невозможность аннулировать подобный токен, например при запросе от пользователя, до тех пор, пока не истечет срок его действия. Все эти проблемы решает механизм отслеживаемых сессий [37].

Отслеживаемые сессии требуют дополнительных ресурсов, для хранения и обработки данных токена, однако в отличии от сессий, основанных

на JWT, дают полный контроль над ними на стороне сервера. Использование подобного механизма позволяет:

- ограничивать количество устройств входа;
- принудительно завершать открытые сессии;
- сокращает общее количество информации, передаваемое сервером.

Для хранения токена на стороне клиента не подойдет использование локального хранилища, так как оно может подвергнуться атаке со стороны. Наиболее надежным местом для его хранения будут cookie файлы. Куки-файлы представляют собой набор символов или текстовых данных, которые непосредственно хранятся в браузере клиента [38]. Однако сами по себе cookie никак предварительно не защищены и требуют дополнительной настройки параметров. Для работы с файлами куки важно настроить следующий набор правил:

1) Secure — разрешает отправку файлов только через HTTPS.

2) SameSite — определяет, как браузер взаимодействует с куки-файлами при запросах с других сайтов. При правильной настройке блокирует CSRF атаки.

3) HttpOnly — отключает возможность взаимодействовать с cookie через JavaScript.

4) Max-Age — устанавливает время жизни файла [39].

Использование данных параметров при настройке cookie файлов позволяет существенно повысить безопасность взаимодействия клиента и сервера, значительно снижая риски утечки персональных данных пользователей.

2.5.2 OAauth аутентификация с использованием внешних сервисов

Благодаря развитию современных технологий, на сегодняшний день существует возможность произвести первичную аутентификацию в 1 клик, без использования связки логина и пароля. Специально для этого, был разработан

протокол аутентификации OAuth. Суть данного протокола заключается в делегировании полномочий по подтверждению личности пользователя сторонним ресурсам, например: Yandex, Google, VK и т.д.

Разработка и внедрение данного способа подтверждения личности состоит из нескольких шагов. Первый шаг самый простой и включает в себя регистрацию своего веб-сервиса на выбранной платформе, например на сервисе Яндекса. Это необходимо, для получения уникальных ключей, которые позволят взаимодействовать с платформой в дальнейшем, а также для настройки перечня запрашиваемых у ресурса персональных данных, например имени пользователя или адреса электронной почты. Затем необходимо разработать механизмы взаимодействия между сервисами на стороне приложения. Схематичное представление работы протокола представлено на рисунке 10.

Рисунок 10 – Принцип работы протокола OAuth

OAuth протокол работает следующим образом:

1) Приложение перенаправляет пользователя на сторонний сервис, где он должен успешно пройти процедуру авторизации.

2) OAuth-сервер выдает специальный токен доступа и перенаправляет пользователя обратно.

3) Сервер приложения обменивает токен доступа на информацию о пользователе [41].

Использование данного протокола позволяет ускорить процесс входа в систему, а также повышает безопасность веб-ресурса, позволяя не использовать пароль и нигде его не хранить, что предотвращает такие типы атак как Brute-force и Dictionary Attack.

Выводы по главе 2

В данной главе были рассмотрены и определены основные технологии, которые позволили разработать комплекс взаимосвязанных инструментов и механизмов для непосредственной обработки и хранения персональных данных пользователей. Было установлено, что фреймворк Next.js позволяет создавать надежные средства обработки данных не только на стороне клиента, но и сервера, благодаря: автоматическому экранированию опасных символов, возможности гибкой настройки для cookie файлов, дополнительной настройке общих политик безопасности и хорошей совместимости с другими библиотеками.

Также было установлено, что SQLite является надежной базой данных для непосредственного хранения личных данных пользователей, благодаря принципам ACID, которые обеспечивают целостность информации при ее обработке и хранении.

На основании анализа современных угроз и методов защиты, был разработан современный комплекс мер и инструментов, необходимый для защиты пользовательских данных, а именно: валидация данных на стороне клиента и сервера, аутентификация пользователей посредством паролей и токенов доступа, запрет на использование простых паролей, хеширование

паролей при хранении в базе, защита токенов от внешних атак, отслеживание состояния на сервере и частичное сокрытие чувствительной информации.

Для разработки также был выбран архитектурный стиль REST, который дает возможность создавать гибкие и масштабируемые приложения. Именно данная методология позволила наладить надежное взаимодействие между клиентом, сервером и базой данных.

Глава 3. Тестирование

3.1 Методы тестирования

Тестирование веб-приложения или отдельных его элементов является неотъемлемой частью разработки, так как позволяет отловить и устранить большую часть ошибок, еще до окончания работы над приложением. Основная цель проведения тестирования — это выявление, подтверждение, исключение рисков возникновения ошибок в системе, а также оценка показателей тестируемых компонентов, необходимая для дальнейших исследований.

Тестирование бывает ручное и машинное. Машинное тестирование выполняется с применением средств вычислительной техники и часто используется для проверки работоспособности устоявшегося функционала системы после обновления. Очевидным плюсом данного подхода является возможность проведения множественного тестирования функционала или компонентов системы в короткие сроки. Однако данный способ требует значительных затрат сил на подготовку, внедрение и поддержание, что особенно сложно в периоды активной разработки, когда весь функционал может перестраиваться по несколько раз. Ручное же тестирование гораздо чаще применятся при первичном анализе компонентов системы. Такое тестирование практически ничем не отличается от машинного в плане качества тестов, однако значительно уступает в скорости проведения повторных замеров из-за того, что производится живым человеком.

Тестирование делится на 2 основных направления: функциональное и нефункциональное. Функциональное тестирование используется для того, чтобы определить, как работают конкретные элементы или интерфейсы внутри приложения. В контексте разработки e-commerce платформ это может быть: проверка корректности введенных данных, правильность рассчитывания суммы всех товаров в корзине пользователя, возможность зарегистрироваться, авторизироваться и т.д. Функциональное тестирование также разделяется на

несколько основных веток: на основе структуры (методом «белого ящика») и на основе спецификации (методом «черного» ящика). Проверки на основании структуры осуществляются с учетом анализа результата отработки кодовой базы, в то время как проверки на основании спецификации производятся исключительно на основании входных и выходных данных, без учета логики работы системы в целом [42]. Проверки также осуществляются с учетом уровня глубины:

1) Верхний, имитирующий реальную работу приложения.

2) Средний, исключающий тестирование клиентской части.

3) Нижний, проверяющий логику работы баз данных.

4) Интеграционный, взаимодействующий с внешними сервисами [81].

При проведении функционального тестирования применяются ряд методологий, необходимых для эффективного поиска возможных ошибок и уязвимостей, а именно: позитивные и негативные проверки. Позитивные проверки используются для подтверждения того, что система работает так, как было изначально задумано. Например, если ожидается что пользователь, нажимая на кнопку, сможет добавить товар в корзину, то успешным результатом теста будет добавление товара в корзину. Негативные же проверки работают наоборот, и нужны для исключения того, что система отработает непредсказуемым образом. Успешным результатом в данном случае будет считаться, например, невозможность добавить товар в корзину будучи неавторизированным в системе.

Для проведения подобных проверок используются граничные значения и классы эквивалентности. Классы эквивалентности представляют собой диапазон чисел или набор определенных символов, которые отрабатывают в системе определённым образом. Например, для поля, ожидающего получить месяц в виде числа, при проведении позитивных проверок таким диапазоном будет являться интервал чисел от 1 до 12 включительно.

Наиболее эффективным способом использования классов эквивалентности, при тестировании приложения, будет метод граничных

значений. Граничные значения — это такие значения, которые являются переходными от одного диапазона класса эквивалентности к другому. На примере того же поля ввода для числа месяца, такими значениями будут числа 0, 1, 2, 11, 12 и 13, так как именно они являются начальными и конечными значениями в своих диапазонах [81].

В данной работе функциональное тестирование будет применяться для проверки работы frontend и backend приложения, базы данных и интеграции со сторонним сервисом авторизации. В ходе работы будут проверены: локальные хранилища, поля ввода, системы проверки входных данных, целостность хранимой информации и результаты обработки данных.

Нефункциональное тестирование направленно в первую очередь на проверку эффективности работы приложения или его отдельных элементов в различных условиях. Примерами подобного тестирования могут быть:

1) Нагрузочные, анализирующее эффективность работы в часы пик.

2) Проверяющие совместимость с различными типами браузеров.

3) Анализирующие работу оборудования.

4) Тестирующие удобство пользовательского опыта.

5) Проверяющие безопасность системы в целом [42].

В контексте данной работы, нефункциональное тестирование будет использовано для проверок безопасности хранения и обработки персональных данных, с учетом возможных угроз надежности. В ходе тестирования безопасности будут отработаны различные сценарии возможных угроз со стороны. Согласно «Национальному Стандарту Российской Федерации, а именно ГОСТ Р 56920-2016/ISO/IEC/IEEE 29119-1:2013 Системная и программная инженерия. Тестирование программного обеспечения», проведение исчерпывающей проверки, иными словами, исключающей абсолютно любой риск, невозможно, поэтому она производится выборочными действиями [42]. На основании данного стандарта будет определен ряд выборочных тестов, демонстрирующих поведение системы при попытках атак со стороны.

3.2 Подготовка тестового окружения и инструментов

Для проведения всеобъемлющего тестирования понадобится подготовить специализированный тестовый стенд. Суть данного стенда заключается в симуляции условий, которые будут максимально приближены к тем, с которыми веб-приложение сталкивается на этапе коммерческой эксплуатации. Моделирование условий работы включает в себя хостинг приложения и базы данных на удаленных серверах. Хостинг представляет собой процесс, при котором вся кодовая, и не только, база проекта размещается и хранится на определенной машине, которая имеет бесперебойный выход в интернет, и предоставляет возможность взаимодействовать с проектом в интернет-пространстве.

Учитывая то, что frontend и backend приложения разрабатывался на Next.js, идеальным решения для разворачивания проекта будет платформа Vercel. Данная платформа разработана непосредственно разработчиками фреймворка Next.js, поэтому имеет отличную совместимость со всеми особенностями данного инструмента. Основными плюсами данной платформы являются:

1) Простота. Vercel позволяет быстро и легко разместить приложение в сети.

2) Шифрование трафика по сертификатам. Из коробки имеет встроенную поддержку HTTPS соединения.

3) Автоматизирование производительности веб-сервиса.

4) Балансировка входного трафика при высоких нагрузках.

5) Надежная работа сервиса из любой точки мира.

6) Бесплатный тариф. Vercel дает возможность бесплатного размещения ресурсов, несильно ограничивая возможности настроек системы [43].

Для моделирования условий реального взаимодействия между приложением и базой данных, когда сервер веб-сайта находится в одном географическом положении, а база данных в другом, будет использоваться

сервис Turso. Turso представляет собой облачное хранилище, позволяющее удаленно разворачивать базы данных, в основе которых лежит SQLite. Использование данной технологии, позволит провести тестирование удаленного хранения, передачи и целостности данных, посредством передачи трафика между точками, находящимися на значительном расстоянии друг от друга. Основными преимуществами использования Turso являются:

1) Легкость интеграции. Turso разработан на основе SQLite, поэтому имеет отличную совместимость с данной базой.

2) Низкая задержка. Сервис хорошо оптимизирован для обработки большого количества запросов.

3) Бесплатный тариф. Turso, так же, как и Vercel, дает возможность бесплатного размещения ресурсов на удаленных серверах, с незначительными ограничениями.

4) Графический интерфейс. Позволяет взаимодействовать с базой, при помощи встроенных графических функций вывода информации на экран. Данная особенность позволяет получать состояние данных в таблицах, без ввода дополнительных SQL команд [44].

Тестирование frontend проекта будет проходить с применением браузера Google Chrome версии 136, с использованием операционной системы Windows 10. Для проверки локальных хранилищ, cookie и взаимодействия клиента с сервером используется встроенная в браузер панель разработчика DevTools, а именно вкладки: application (приложение), для проверки локальных данных; network (сеть), для отслеживания отправляемых запросов.

Для тестирования корректности обработки входных данных на backend приложения используется Postman. Postman представляет из себя инструмент, разработанный специально для тестирования серверной части приложения, посредством отправки HTTP или HTTPS запросов к приложению. Postman поддерживает использование наиболее популярных методов запросов, таких как: GET, POST, PUT, PATCH, DELETE и других. Запросы, создаваемые при помощи данной программы, имеют гибкие параметры настройки,

позволяющие передавать такие параметры как: тело запроса, заголовки, cookie файлы и специальные параметры адресной строки [45].

3.3 Тестирование frontend приложения

Проверка работоспособности клиентской части платформы электронной коммерции должна учитывать результаты, в том числе, и для различных размеров экрана. Как говорилось ранее, современные интернет-приложения могут запускаться не только на персональных компьютерах, но и на планшетах и смартфонах. Поэтому необходимо удостовериться в том, что приложение будет работать одинаково при различных размерах дисплея. Для тестирования функциональности будут использоваться следующие размеры экрана: 1920×1080, 1024×768 и 320×568. Проверки будут включать валидацию полей ввода, а также регистрацию, аутентификацию, защищенные маршруты и локальные хранилища.

Для регистрации нового пользователя используется следующий шаблон: имя, фамилия допускают ввод только латиницы и кириллицы; почта должна соответствовать шаблону и быть уникальной; пароль должен быть не меньше 8 символов, иметь буквы верхнего и нижнего регистров, а также цифры и специальные символы; поле с подтверждением пароля должно полностью совпадать с введенным паролем. Тестовые данные представлены в таблице 1.

Таблица 1 – Тестовые данные для формы регистрации

Тест №	Имя	Фамилия	Почта	Пароль	Подтверждение пароля
1	Ivan	Leonov	test@test.ru	Fk$f21Dw	Fk$f21Dw
2	Ivan	12345	ivanov@test.ru	Myp@ssw0rd	Myp@ssw0rd
3	Иван	Леонов	marin@testru	xD%66datE	xD%66datE

Продолжение таблицы 1

Тест №	Имя	Фамилия	Почта	Пароль	Подтверждение пароля
5	Petr	Лимонов	Af2fd_d@test.ru	Qwerty123!	qwerty123
6	_	Ldoaf	Dddd.rd	12345678	12345678
7	Kirill	Vold	test@test.ru	Kvrndr^!!42	Kvrndr^!!42

Результаты работы программы представлены в таблице 2.

Таблица 2 – Регистрация нового пользователя, frontend

Тест №	Наличие ошибки	Ожидаемый результат	Фактический результат
1	Нет	Успешно	Успешно
2	Да	Ошибка	Ошибка
3	Да	Ошибка	Ошибка
4	Да	Ошибка	Ошибка
5	Да	Ошибка	Ошибка
6	Да	Ошибка	Ошибка
7	Нет	Ошибка	Ошибка

Как видно из результатов, система не дает возможности создать несколько пользователей с одинаковой почтой или зарегистрировать недопустимый тип данных. В настоящий момент в системе зарегистрирован только 1 пользователь. Добавим еще несколько, исправив невалидные данные для пользователей 2-5 из таблицы 1. Предполагается, что данные, хранимые в локальных хранилищах, будут уникальными для каждого пользователя, а также содержать такие данные как: имя, фамилия, скрытая почта и статус

авторизации. Проверим это запросив личные данные каждого пользователя. Результаты представлены в таблице 3.

Таблица 3 – Данные хранимые в локальных хранилищах

Тест №	Имя	Фамилия	Почта	Авторизирован
1	Ivan	Leonov	tes****@test.ru	Да
2	Ivan	Bileev	Iva****@test.ru	Да
3	Иван	Леонов	mar****@test.ru	Да
4	Катя	Petrova	pet****@test.ru	Да
5	Petr	Лимонов	Af2****@test.ru	Да

Главным элементом, обеспечивающим безопасность системы на стороне клиента, является токен доступа, который выдается каждому пользователю, при входе в систему. Именно токен дает право доступа к конфиденциальной информации и к защищенным маршрутам, обрисовывающим эту самую информацию. Тестирование токенов на стороне клиента представлено в таблице 4.

Таблица 4 – Тестирование токенов на стороне клиента

Тест №	Токен	Действителен	Доступ к данным
1	4cda45f9d6299bd753940a335be3a82a73492e41b5bcc2f4fc06531eaf01ef95	Да	Разрешено
2	66c7ffe04514cf31e1c9ab610da2cad5f5accbc7d266dc03a18e93ed1ce5df27	Да	Разрешено

Продолжение таблицы 4

Тест №	Токен	Действителен	Доступ к данным
3	69e7ea999baa825a2b241e2d052743d74c35816a467da5171115c7c05f7dcbb0	Нет	Запрещено
4	6b851986cf2373d3071032d438d45ce69ad1c0c0b552c6d1d2ff3832f78cfb4c	Да	Разрешено
5	ae088cff659a55396fbcb80c55b346a19bc8f7cc15c33b8dde39cf06ea53c217	Нет	Запрещено
6	1111	Да	Запрещено

Как видно из результатов, токен доступа является уникальным для каждого пользователя. Кроме того, были протестированы попытки входа систему с токеном, у которого истек срок годности, и попытка отправки не валидной строки, имитирующей токен. Доступ к данным, смогли получить только те клиенты, у которых был действительный токен, что помогло защитить страницы и информацию от неправомерного доступа. Аналогичные результаты были получены и при аутентификации через сторонний сервис.

3.4 Тестирование backend приложения

Тестирование backend приложения включает в себя комплексную проверку не только того, как отрабатывают конкретные функции обработки информации, например парсинг строки, но и результаты работы базы данных. Основная цель — проверка работоспособности всех критически важных функций, и целостность передачи данных. Так как backend приложения является полностью автономным, то существует вероятность того, что информация, которую он получит может быть отправлена не с доверенного клиента, поэтому требуется удостовериться в том, что данные являются валидными. Для тестирования полностью очистим базу данных и попробуем

еще раз зарегистрировать пользователей из таблицы 1, при помощи Postman. Результаты представлены в таблице 5.

Таблица 5 – Регистрация нового пользователя, backend

Тест №	Наличие ошибки	Ожидаемый результат	Фактический результат
1	нет	Успешно	Успешно
2	да	Ошибка	Ошибка
3	да	Ошибка	Ошибка
4	да	Ошибка	Ошибка
5	да	Ошибка	Ошибка
6	да	Ошибка	Ошибка
7	нет	Ошибка	Ошибка

Как видно из таблицы, результаты идентичны тем, что были при регистрации нового пользователя на клиентской части приложения. Таким образом, можно сделать выводы о том, что даже если пользователь сможет послать невалидные данные на сервер, то их не получится обработать, так как сервер использует те же проверки информации, что и клиент.

Наиболее важным этапом проверки backend приложения является функции хеширования пароля. Данная функция предназначена для хранения наиболее чувствительной информации, такой как пароль пользователя. Основным критерием проверки данной функции является оригинальность, так как хеш-функция должна формировать уникальный результат для каждого пароля, даже если они идентичны. Для проведения данного тестирования создадим новых пользователей, с паролями различной сложности. Результаты представлены в таблице 6.

Таблица 6 – Хеширование паролей

Тест №	Пароль	Хеш
1	a8r*!7mJ	$2b$10$YFIdtrSkffYMlHtgcBgjW.0I3QVxdVYN/8x4Zw/gDnqpvm/s8C9wa
2	a8r*!7mJ	$2b$10$JZsROa7g4ye9HXPL6BQeXOW/jMH1T8.6Vwd1SMYQMQliH/HM4MzCi
3	uWG3OvMuf&r45uW	$2b$10$DCghtEBOj5dKo1RnQ5A9N.z1X30KAor8M2lxnql03qB3oZGZjb7bC
4	uWG3OvMuf&r45uW	$2b$10$I3F8YPBH60OSxA1QMEoHBu6CXNR6S3HWDPg78buctGwz2zZM48wCC
5	2m$llyjmJM	$2b$10$uC2u8EawVyyiIMuBdOfUB.E.oRjCUevqhWJ6o.QMQtpkkhpPlmvE.
6	oxjlYY9(DE	$2b$10$3es89Z2hvWbaP1VyXrMtx.TCs8exNtQA3g/iJHP6b6hF7.GQscPFK

Результаты наглядно демонстрируют, что хеш, генерируемый алгоритмом bcrypt, является уникальным для каждого введенного пароля, в том числе и для одинаковых значений. Итоговое значение всегда имеет фиксированную длину и уникальный порядок символов. Таким образом получаем, что даже если злоумышленник получит доступ к таблице паролей пользователей, он не сможет воспользоваться ими, так как расшифровка хеша на базе выбранного алгоритма занимает достаточно большое количество времени.

Тестирование базы данных включает в себя: проверку добавления записей, удаления, изменения, а также целостность последовательных операций, при проведении транзакций. Тестирования данного функционала будет происходить посредством взаимодействия пользователя с товарами магазина. Алгоритм выглядит следующим образом:

1) Пользователь добавляет товар в корзину.
2) Изменяет количество товара в корзине.
3) Формирует заказ.

Ожидается, что: товар успешно добавится в корзину, количество будет успешно изменено, товар будет удален из корзины и перенесен в таблицу оплаченных покупок. При тестировании формирования заказа, в некоторых тест-кейсах будут имитироваться системные ошибки, для проверки целостности обрабатываемых данных, при внештатных ситуациях. Результаты представлены в таблице 7.

Таблица 7 – Оформление заказа

Тест №	Товаров в начале	Добавлено товаров	Тип Ошибки	Результат
1	1	2	Отсутствует	Оплачено 3 товара
2	2	3	Пользователь не найден	Операция отменена
3	6	4	Ошибка после добавления в таблицу заказа	Операция отменена
4	5	3	Ошибка при удалении товаров из корзины	Операция отменена
5	4	7	Отсутствует	Оплачено 11 товаров

По результатам тестирования было установлено, что приложение обеспечивает целостность данных, так как при любом сбое, в ходе обработки персональных данных, программа автоматически откатывает изменения без внесения поврежденных или частичных данных в систему. Также было

установлено, что система корректно отрабатывает: добавление, удаление, изменение и последовательные операции.

3.5 Тестирование безопасности приложения

Процесс тестирования безопасности приложения строится на моделировании реальных киберугроз, с которыми система может столкнуться во время работы. Соответственно, в рамках данной работы необходимо смоделировать следующие угрозы: SQL-инъекция, XSS и SCRF атаки.

SQL-инъекция может быть внедрена в приложение посредством нескольких способов: через параметры адресной строки и непосредственно через поля ввода в HTML формах. Предполагается, что система устойчива к атакам подобного рода. Для тестирования будет использоваться форма аутентификации и динамические маршруты товаров.

При тестировании уязвимости адресной строки будет смоделирована ситуация, при которой в строку будет внедрен SQL код, запрашивающий определенный товар. Пример подобных тест кейсов представлен в таблице 8.

Таблица 8 – Тестирование SQL-инъекций в адресной строке

Тест №	Адресная строка	Ожидаемый результат	Фактический результат
1	… /products/3 and 1=1	Ошибка	Ошибка
2	…/products/999 or 2	Ошибка	Ошибка
3	…/products/' or 2	Ошибка	Ошибка
4	… /products/3	Успешно	Успешно

Аналогичные результаты дает и тестирование формы аутентификации. Хотя базовая защита не позволяет внедрить SQL код, так как данные,

полученные сервером, проходят строгую систему валидации, его все еще можно вставить напрямую в исходный код. Как и в случае с адресной строкой форма аутентификации вернула успешный результат только при введении чистых данных, не содержащих SQL примесей, таким образом удалось доказать, что Drizzle в сочетании с валидацией полей ввода дает хорошую базовую защиту от SQL атак.

Симуляция XSS атак будет происходить похожим образом. Для проверки устойчивости, вредоносный код будет внедрен: в валидируемое поле, в не валидируемое поле и напрямую в код программы. Во всех трех случаях будет имитироваться попытка добавления вредоносного кода, который будет пытаться отправить GET запрос на сторонний ресурс.

При тестировании валидируемых полей результат оказался отрицательным, так как система ожидает определенных шаблонных выражений, непохожих на внедряемый код. Приложение либо не дает ввести подобные символы, либо запрещает дальнейшее выполнение. Результат идентичен как на клиенте, так и на сервере.

Использование полей, которые никак не защищены от ввода опасных символов также не дало никаких результатов. Все дело в том, что внедряемый код сталкивается со встроенной защитой Next.js, которая автоматически экранирует все символы, представляющие опасность для программного обеспечения. Таким образом, даже если разработчик по каким-то причинам не добавит проверку допустимых символов, система все равно не даст внедрить вредоносный код.

При добавлении кода напрямую в программное обеспечение, также не удалось запустить опасные процессы, которые смогли бы хоть как-то навредить пользователю. Несмотря на то, что код вставлен максимально небезопасным способом, политики безопасности, настроенные во время разработки магазина, не разрешают отправлять запросы на сторонние сервисы и ресурсы. Получается, что даже если злоумышленник сможет преодолеть

первые несколько линий обороны, он все равно не сможет внедрить вредоносный код в приложение.

Для тестирования последнего типа атак, направленного на использование файлов куки, понадобится подготовить так называемый фишинговый сайт, который будет развернут на другом домене. Внутри данного сайта также необходимо настроить фоновую отправку запроса к серверу оригинального приложения, которая будет пересылать хранящийся в cookie-файлах токен доступа. Предполагается, что настройки хранения токена не позволят отправить его с поддельного сайта и получить тем самым доступ к персональной информации клиента.

После небольшой подготовки, проверим как работает защита. Сначала авторизируемся на основном сайте, чтобы получить токен доступа. После этого перейдем на вредоносный сайт и откроем панель разработчика на вкладке сеть, демонстрирующую результаты запроса. Сервер постоянно возвращает ошибку, так как браузер не отправляет куки с пометкой SameSite из сторонних источников. Чтобы убедится, в работоспособности сервера, вернемся к основному клиенту и попробуем сделать идентичный запрос. В панели разработчика сразу отображается статус успешного выполнения. Таким образом было установлено, что настройки cookie-файлов, используемые при разработке системы, предоставляют надежную защиту от CSRF атак, посредством блокировки передачи данных пользователя.

Выводы по главе 3

В третьей главе было проведено всестороннее исследование и тестирование надежности разработанного приложения. Тестирование проводилось согласно Национальному Стандарту Российской Федерации, а именно ГОСТ Р 56920-2016/ISO/IEC/IEEE 29119-1:2013 «Системная и программная инженерия. Тестирование программного обеспечения».

Тестирование проводилось на основании спецификации, структуры и с моделированием ситуаций реальных кибератак.

В ходе исследования было развернуто специальное тестовое окружение, которое было максимально приближено к реальным условиям работы интернет-приложения. Система была развернута на удаленных серверах Vercel и Turso, что позволило протестировать работоспособность приложения в условиях реального хостинга.

В ходе тестирования было установлено, что система обеспечивает целостность и безопасность передачи информации, не позволяя пользователям вводить некорректные данные. Система также показала хорошие результаты при аутентификации пользователей, блокируя любое действие, к которому клиент не имеет доступа. Таким образом получилось добиться того, чтобы информация обладала тремя ключевыми свойствами: целостностью, доступностью и конфиденциальность.

Также, по результатам тестирования, было выявлено что система обладает хорошей устойчивостью к современным киберугрозам, благодаря хорошо настроенной системе защиты от конкретных видов атак.

Заключение

В данной работе была спроектирована и разработана безопасная система обработки и хранения персональных данных на основе фреймворка Next.js. Характерной чертой данной системы является ее комплексность, так как она включает в себя проектирование клиентской, серверной частей и базы данных. В ходе работы были налажены основные процессы по получению, обработке, транспортировке, хранению и защиты информации от современных типов угроз. Данная система соответствует основным требованиям и характеристикам, предъявляемым при обработке персональных данных.

В ходе работы, работы были изучены основные нормативно-правовые требования по защите персональных данных, а именно «152-ФЗ о персональных данных», «261-ФЗ о внесении изменений в основной закон о персональных данных» и «Национальный Стандарт Российской Федерации о методах и средствах обеспечения безопасности Гост Р ИСО/МЭК 27001-2006». Также был проведен анализ современных угроз и потенциальных уязвимостей, с которыми может столкнуться интернет-приложение. На основании данного анализа был разработан комплекс мер, необходимый для обеспечения безопасной обработки и хранения данных. А также произведено комплексное тестирование всего приложения.

Было установлено, что современные безопасные системы обработки данных для E-commerce должны: соответствовать законодательству, быть устойчивыми к современным угрозам, обеспечивать целостность хранения, обработки и передачи информации, использовать надежные способы работы с токенами доступа, валидировать данные не только на стороне клиента, но и сервера.

По результатам тестирования, приложение, разработанное на базе фреймворка Next.js, имеет хорошую устойчивость современным типам угроз, включающим такие атаки как SQL-инъекции, XSS и SCRF. Использование алгоритма bcrypt позволяет надежно хешировать пароли, что предотвращает

их раскрытие третьим лицам. Также было установлено, что валидация полей HTML форм является неотъемлемой частью разработки безопасных систем обработки данных, так как именно они представляют потенциальную угрозу проникновения в систему, путем внедрения вредоносного кода в программу. Контролирование получаемой информации позволяет не только обезопасить приложение, но и дает возможность систематизировать и унифицировать персональные данные пользователей.

Перспективы дальнейшего развития системы включает в себя широкий выбор направлений, таких как: интеграция с технологиями машинного обучения путем анализа данных или действий пользователя; внедрение более надежных методов аутентификации пользователей, например биометрических; исследование более надежных и быстрых алгоритмов хеширования и шифрования данных.

Резюмируя все вышесказанное можно сделать вывод о том, что Next.js имеет хорошие перспективы развития а также позволяет разрабатывать безопасные приложения и системы обработки персональных данных в платформах электронной коммерции благодаря: автоматическому экранированию входных данных, возможности гибкой настройки для cookie файлов, возможности дополнительной настройки общих политик безопасности приложения, хорошей совместимости с другими библиотеками.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ТАСС [Электронный ресурс]. – Режим доступа: https://tass.ru/obschestvo/20933305 (Дата обращения: 13.03.2025).

2. Kaspersky, Утечка персональных данных [Электронный ресурс]. – Режим доступа: https://www.kaspersky.ru/resource-center/definitions/data-breach (Дата обращения: 14.04.2025).

3. Сбербанк, Угрозы и риски утечки персональных данных [Электронный ресурс]. – Режим доступа: http://www.sberbank.ru/ru/person/kibrary/articles/ugrozy_i_riski_utechki_personalnyh_dannyh (Дата обращения: 08.04.2025).

4. СберКорус, Изменения в обработке и защите персональных данных в 2025 году [Электронный ресурс]. – Режим доступа: https://www.esphere.ru/blog/izmeneniya-v-obrabotke-i-zashchite-personalnykh-dannykh-v-2025-godu/ (Дата обращения: 04.03.2025).

5. Электронный фонд правовых и нормативно-технических документов. Национальный стандарт Российской Федерации ГОСТ Р ИСО/МЭК 27001-2006 [Электронный ресурс]. – Режим доступа: https://docs.cntd.ru/document/1200058325 (Дата обращения: 10.04.2025).

6. Управление Президента Российской Федерации. Федеральный закон «О персональных данных» [Электронный ресурс]. – Режим доступа: http://letters.kremlin.ru/info-service/acts/9 (Дата обращения: 9.03.2025).

7. MDN, The Form element [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/form (Дата обращения: 3.03.2025).

8. MDN, What is JavaScript? [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/What_is_JavaScript (Дата обращения: 4.03.2025).

9. MDN, Dynamic typing [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing (Дата обращения: 5.03.2025).

10. Postman, What are HTTP methods? [Электронный ресурс]. – Режим доступа: https://blog.postman.com/what-are-http-methods/ (Дата обращения: 17.03.2025).

11. Next.js [Электронный ресурс]. – Режим доступа: https://nextjs.org/docs (Дата обращения: 02.04.2025).

12. SQLite [Электронный ресурс]. – Режим доступа: https://sqlite.org/about.html (Дата обращения: 02.04.2025).

13. Drizzle [Электронный ресурс]. – Режим доступа: https://orm.drizzle.team/docs/overview (Дата обращения: 02.04.2025).

14. Tailwind CSS [Электронный ресурс]. – Режим доступа: https://v3.tailwindcss.com/ (Дата обращения: 03.04.2025).

15. Npm. Tailwind-merge [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/tailwind-merge (Дата обращения: 03.04.2025).

[16. Npm. Clsx [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/clsx (Дата обращения: 03.04.2025).

17. HeadlessUI [Электронный ресурс]. – Режим доступа: https://headlessui.com/ (Дата обращения: 03.04.2025).

18. TanStack Query [Электронный ресурс]. – Режим доступа: https://tanstack.com/query/latest (Дата обращения: 04.04.2025).

19. React Hook Form [Электронный ресурс]. – Режим доступа: https://react-hook-form.com/ (Дата обращения: 04.04.2025).

20. Zod [Электронный ресурс]. – Режим доступа: https://zod.dev/?id=introduction (Дата обращения: 04.04.2025).

21. GitHub. Zustand [Электронный ресурс]. – Режим доступа: https://github.com/pmndrs/zustand (Дата обращения: 04.04.2025).

22. Lucia [Электронный ресурс]. – Режим доступа: https://lucia-auth.com/ (Дата обращения: 05.04.2025).

23. Arctic [Электронный ресурс]. – Режим доступа: https://arcticjs.dev/ (Дата обращения: 05.04.2025).

24. Luxon [Электронный ресурс]. – Режим доступа: https://moment.github.io/luxon/#/?id=luxon (Дата обращения: 06.04.2025).

25. Npm. React input mask [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/@react-input/mask (Дата обращения: 06.04.2025).

26. Kaspersky. Что такое атака с использованием межсайтового скриптинга? Определение и описание. [Электронный ресурс]. – Режим доступа: https://www.kaspersky.ru/resource-center/definitions/what-is-a-cross-site-scripting-attack (Дата обращения: 16.04.2025).

27. Kaspersky. Каждый третий россиянин чуть не лишился денег после взлома личного кабинета [Электронный ресурс]. – Режим доступа: https://www.kaspersky.ru/about/press-releases/kazhdii-tretii-rossiyanin-chut-ne-lishilsya-deneg-posle-vzloma-lichnogo-kabineta дата обращения (07.03.2025).

28. Auth.js. Session strategies [Электронный ресурс]. – Режим доступа: https://authjs.dev/concepts/session-strategies (Дата обращения: 13.03.2025).

29. Яндекс. REST API: что это такое и как работает [Электронный ресурс]. – Режим доступа: https://practicum.yandex.ru/blog/chto-takoe-rest-api-i-kak-rabotaet/ (Дата обращения: 19.04.2025).

30. Readme. The History of REST APIs [Электронный ресурс]. – Режим доступа: https://readme.com/resources/the-history-of-rest-apis (Дата обращения: 19.04.2025).

31. Metanit. Внешние ключи [Электронный ресурс]. – Режим доступа: https://metanit.com/sql/sqlserver/3.5.php (Дата обращения: 10.04.2025).

32. React [Электронный ресурс]. – Режим доступа: https://react.dev/learn (Дата обращения: 18.04.2025).

33. Adobe for Business. Single-page application (SPAs) – what they are and how they work [Электронный ресурс]. – Режим доступа:

https://business.adobe.com/blog/basics/learn-the-benefits-of-single-page-apps-spa (Дата обращения: 27.03.2025).

34. The Single Page Interface Manifesto [Электронный ресурс]. – Режим доступа: https://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php (Дата обращения: 04.03.2025).

35. Next.js. Pages and Layouts [Электронный ресурс]. – Режим доступа: https://nextjs.org/docs/pages/building-your-application/routing/pages-and-layouts (Дата обращения: 09.03.2025).

36. SearchInform Layouts [Электронный ресурс]. – Режим доступа: https://searchinform.ru/resheniya/biznes-zadachi/zaschita-personalnykh-dannykh/kakie-personalnye-dannye-yavlyayutsya-obshchedostupnymi/ (Дата обращения: 19.03.2025).

37. Next.js. How to implement authentication in Next.js [Электронный ресурс]. – Режим доступа: https://nextjs.org/docs/app/guides/authentication (Дата обращения: 15.03.2025).

38. Kaspersky. Что такое cookie [Электронный ресурс]. – Режим доступа: https://www.kaspersky.ru/resource-center/definitions/cookies (Дата обращения: 07.04.2025).

39. MDN. Using HTTP cookies [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies (Дата обращения: 19.04.2025).

40. Next.js. How to set a Content Security Policy (CSP) for Next.js application [Электронный ресурс]. – Режим доступа: https://nextjs.org/docs/app/guides/content-security-policy (Дата обращения: 11.04.2025).

41. Яндекс. Реализация OAuth в Яндексе [Электронный ресурс]. – Режим доступа: https://yandex.ru/dev/id/doc/ru/concepts/ya-oauth-intro (Дата обращения: 22.03.2025).

42. Электронный фонд правовых и нормативно-технических документов. Национальный стандарт Российской Федерации ГОСТ Р 56920-

2016/ISO/IEC/IEEE 29119-1:2013 [Электронный ресурс]. – Режим доступа: https://docs.cntd.ru/document/1200134996 (Дата обращения: 09.03.2025).

43. Vercel [Электронный ресурс]. – Режим доступа: https://vercel.com/frameworks/nextjs (Дата обращения: 14.03.2025).

44. Turso [Электронный ресурс]. – Режим доступа: https://docs.turso.tech/introduction (Дата обращения: 18.04.2025).

45. Postman [Электронный ресурс]. – Режим доступа: https://www.postman.com (Дата обращения: 24.04.2025).

46. Ширяев П.С., Утечки конфиденциальных данных: главный враг внутри. Государственное управление. Электронный вестник, № 91, 2022, с. 226-240.

47. Страхов А.А., Дубинина Н.М., Об утечке данных и DLP-системах. Естественные науки. Компьютерные науки и информатика, № 4, 2022, с. 226-232.

48. Калягина Л.В., Разумов П.Е., Категория «Данные»: понятие, сущность подходы к анализу. Вестник КрасГАУ, № 4, 2014, с. 3-8.

49. Жалолов О.И., Хаятов Х.У., Понятие SQL и реляционной базы данных. Universum: технические науки, № 6 (75), 2020, с. 26-29.

50. Романов С.С., Достоинства, недостатки и альтернативы объектно-реляционного отображения (ORM), № 12 (17), 2016, часть 2, с. 147-149.

51. Черный Б., Профессиональный TypeScript. Разработка масштабируемых JavaScript-приложений. СПб.: Питер, 2022 — 352 с.

52. Изюмов А.Е., Исследование безопасности протокола http. Научно-технический вестник информационных технологий, механики и оптики, № 3 (19), 2005, с. 161-165.

53. Чепегин И.Д., Серверный JavaScript – преимущества и недостатки Node.js. Вестник науки и образования, № 12 (90), Часть 1, 2020, с. 18-20.

54. Байдыбеков А.А., Гильванов Р.Г., Молодкин, И.А., Современные фреймворки для разработки web-приложений. Интеллектуальные технологии на транспорте, № 4, 2020, с. 23-29.

55. Toras P.B., Syahril E., Erna B.N., Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force. Journal of Physics Conference Series, Vol. 1811, Iss. 1, 2021, p. 1-7.

56. Завьялов А.Н., Интернет-мошенничество (фишинг): проблемы противодействия и предупреждения. Baikal Research Journal, № 2, Т. 13, 2022, с. 36.

57. Барабанов А.В., Лавров А.И., Марков А.С., Полотнянщиков И.А., Исследование атак типа «Межсайтовая подделка запросов». Вопросы кибербезопасности № 5 (18), 2016, с. 43-49.

58. Беликов Г.В., Крылов И.Д., Селищев В.А., SQL-инъекция как способ обхода авторизации. Известия ТулГУ. Технические науки, № 12, 2021, с. 217-221.

59. Снегуров А.В., Чакрян В.Х., Анализ устойчивости к взлому современных механизмов парольной защиты операционных систем. Восточно-Европейский журнал передовых технологий, № 50, 2011, с. 27-29.

60. Муратов Г.А., Особенности работы протокола TLS/SSL. Молодой исследователь Дона, № 3 (30), 2021, с. 67-70.

61. Шамухамедов Г.Х., Хадыров Н.К., Союнова О.М., Анализ современных методов хеширования. Science Time, № 6 (18), 2015, с. 590-593.

62. Nilesh A.L., A Review Of Authentication Methods. International journal of scientific & technology research, Vol. 5, Iss. 11, 2016, p. 246-249.

63. Бочнев Н.А., Харисов А.Р., Надежные методы аутентификации в государственных и корпоративных информационных системах: принципы, технологии и перспективы. Международный научный журнал «ВЕСТНИК НАУКИ» № 12 (81), Том 5 ч. 1, 2024, с. 642-648.

64. Убалехт И.П., Построение схем баз данных с учетом связей между атрибутами сущностей инфологической модели. Динамика систем, механизмов и машин, № 1, 2012, с. 285-288.

65. Минакова О.В., Акамсина Н.В., Курипта О.В., Разработка программных инструментов на базе расширяемых платформ с открытым

исходным кодом. Вестник Воронежского государственного технического университета, № 4, Том 18, 2022, с. 56-62.

66. Лехтман В.Я., Дядоян К.А., Трегуб А.Д., Методы хакинга и борьба с ними. Международный научный журнал «Символ науки», № 3, 2016, с. 59-61.

67. Амбросенко Р.Н., Транзакционный подход к повышению качества обработки информации в образовательных средах дистанционного обучения. Вестник КрасГАУ, № 3, 2007, с. 62-65.

68. Газизуллин Н.И., Плещинская И.Е., Разработка клиентской части веб-приложения с использованием технологий SPA. Научно-образовательный журнал для студентов и преподавателей «StudNet», № 8, 2020, с. 104-109.

69. Гойвертс Я., Левитан С., Регулярные выражения: сб. рецептов / пер. с англ. СПб.: Символ-плюс, 2010 — 608 с.

70. Назарова А.Д., Анализ надежности паролей для защиты данных. Умная цифровая экономика, № 4, Том 2, 2022, с. 41-46.