МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Экономики и управления на предприятии природопользования»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(бакалаврская работа)
по направлению подготовки 09.03.03 Прикладная информатика
(квалификация – бакалавр)

На тему «Разработка веб - приложения «Дашборд управления личными финансами»»

Исполнитель Пшеничный Вячеслав Александрович

Руководитель к.т.н., доцент Попов Николай Николаевич

«К защите допускаю»

Руководитель кафедры _____

кандидат экономических наук

Майборода Евгений Викторович

«18» 01 2026 г.

Филиал Российского государственного гидрометеорологического университета в г. Туапсе
НОРМОКОНТРОЛЬ ПРОЙДЕН
«12» 01 2026 г.
Мартынова Т.В
ПОДПИСЬ РАСШИФРОВКА ПОДПИСИ

Туапсе
2026

ОГЛАВЛЕНИЕ

Введение

В последнее время финансы являются одним из основных ресурсов для человека, и чтобы добиваться определённых результатов в жизни, нужно уметь управлять ими. Для этого необходимы соответствующие инструменты. Одним из таких инструментов является дашборд.

Дашборд — это интерактивный отчёт об основных показателях деятельности, который подключается к обновляемым источникам данных. Индикаторы отображаются в виде графиков, диаграмм, таблиц и других визуальных элементов [16, с.18].

Несомненно, дашборд помогает отслеживать доходы, расходы, количество сбережённых денег, норму сбережений, оценивать, как растёт расход по каждой из категорий. В зависимости от целей и задач в него могут входить разные параметры, например:

— доходы. Важно учитывать все источники дохода, в том числе дополнительные;

— расходы. Необходимо следить за ними, чтобы дашборд точно отражал, что доступно для сбережений и других целей каждый месяц;

— совокупное состояние. Этот показатель показывает, как меняется общее богатство со временем, что влияет на достижение большинства финансовых целей;

— движение денежных средств. Часто в этот параметр для индивидуальных пользователей входит зарплата, но также можно отслеживать денежные потоки из других источников дохода.

Дашборд для контроля личных финансов - это визуальное представление ключевых финансовых показателей на одном экране. Он помогает видеть общую картину и взаимосвязи между активами, долгами, доходами и тратами человека. Он собирает, структурирует, анализирует и представляет информацию в графиках, диаграммах или таблицах [16, с.33]. В настоящее время в системах мониторинга существует необходимость в осуществлении

распределённого оперативного наблюдения за информацией, представляемой в виде набора данных, которые поступают непрерывно и имеют разнородную структуру.

Таким образом, задача создания веб - приложения, способного осуществлять мониторинг за состоянием личных финансов, является весьма актуальной.

Объектом исследований выпускной квалификационной работы являются процессы учёта, анализа и планирования финансовых операций.

Предметом исследований выпускной квалификационной работы являются функции, структура приложения, технологии реализации и безопасность проектирования веб-приложения по контролю финансового благополучия.

Целью выпускной квалификационной работы является проектирование веб-приложения для управления личными финансами.

Для достижения поставленной цели, был определен ряд задач:

— изучить теоретические аспекты управления персональными финансами;

— проанализировать имеющиеся инструменты задачи создания веб-приложения;

— провести анализ предметной области;

— создать собственный инструмент на основе полученных знаний;

— проверить качество программного продукта;

— оценить экономическую эффективность разработки АИС.

Практическая значимость выпускной квалификационной работы заключается в использовании дашборда для финансового планирования, функциональными задачами которого являются: ведение ежедневного учета трат, с возможностью указывать комментарий к каждой из них; учет долговых операций с предоставлением функциональных операций для своевременного оповещения о текущем состоянии долга; возможность индивидуальной настройки приложения для пользователя под себя.

1 Анализ предметной области

1.1 Обзор рынка приложений для финансового планирования

Согласно мнению экспертов по личным финансам, учет расходов и доходов, а также их анализ — первый шаг к выстраиванию осознанного подхода к деньгам и дальнейшего составления финансовых планов [7, с.38].

Для учета собственных финансов можно использовать различные инструменты, например, фиксацию своих расходов и доходов на электронных или бумажных носителях с целью последующего самостоятельного анализа. Однако, как показывают исследования, использование специализированных приложений позволяет сделать процесс более эффективным и снизить расходы на 7% относительно обычного показателя [7, с.44].

Рынок приложений для финансового планирования расширяется из-за необходимости комплексных инструментов планирования и растущей сложности финансового управления. Предприятия тратят всё больше денег на передовое программное обеспечение для облегчения финансового анализа, прогнозирования и бюджетирования. Некоторые тенденции рынка:

1) Принятие облачных решений. Облачное программное обеспечение для финансового планирования становится всё более популярным благодаря своей доступности, масштабируемости и гибкости.

2) Интеграция автоматизации и ИИ. Эти технологии используются всё больше и больше для повышения точности прогнозирования и ускорения процедур финансового планирования.

3) Обработка данных в реальном времени и аналитика. Подчёркивается финансовый анализ в реальном времени для поддержки быстрого и хорошо информированного принятия решений.

4) Настройка и персонализация. Растущая тенденция к решениям для финансового планирования, которые специально разработаны для удовлетворения требований различных предприятий и организаций.

Сервисы для управления личными финансами (англ. Personal Finance

Management, PFM) – это веб- и мобильные приложения, которые помогают пользователям контролировать свои финансы в едином окне. Цель таких приложений заключается в оказании помощи ведения личного или семейного бюджета, контролировать расходы, анализировать категории трат и управлять сбережениями [21, с.59].

Кроме ключевых преимуществ финансовых приложений можно выделить то, что они напоминают о необходимости вести учет с помощью уведомлений, доступны в любой момент за счет привязки к мобильному устройству, а не к компьютеру или ежедневнику, и берут на себя все задачи по автоматизации и расчетам, оставляя пользователю лишь необходимость вносить в приложение данные.

Среди подобных приложений, популярных на территории постсоветского пространства, можно выделить три условных группы. В первую попадают приложения, для которых характерны минимальное количество настроек и простота учета. К таким приложениям можно отнести следующие: «Тяжеловато», «Monefy» и «Monetal».

Следом можно отметить приложения с более сложным функционалом, позволяющим не только учитывать свои расходы и доходы, но и планировать их, а также получать более сложную статистику. Среди представителей этой группы стоит отметить «CoinKeeper», «Дзен- мани» и «Moneon».

Более сложные приложения такие как «1С: Деньги» или «BudgetBakers», рассчитаны на более профессиональных пользователей и предусматривают такие функции как, например, учет оборота акций и облигаций или помощь в формировании налоговых отчетов [22].

Для дальнейшего анализа аналогов на сегодняшнем рынке выбраны три приложения: «Тяжеловато», «CoinKeeper» и «Monefy». Данное решение обусловлено тем, что на момент выполнения выпускной квалификационной работы они являются единственными подобными приложениями, которые находятся в топ-50 скачиваний российского App Store среди бесплатных финансовых приложений.

Мобильные приложения, несомненно, являются отличными помощниками для построения бюджета, но следить за своими доходами и расходами можно и просто используя приложения интернет-банкинга – они постоянно проходят проверки безопасности от сотрудников банков, регистрируют все операции по картам, а также самостоятельно категоризируют большую часть трат и доходов. Единственный минус – это не очень удобно, если пользователь регулярно применяет карты нескольких банков или вообще предпочитает наличные.

Рассмотрим более подробно наиболее востребованные приложения:

1) «Тяжеловато» позиционирует себя как максимально простое приложение, построенное вокруг одного сценария. Функционал приложения заключается в том, что пользователь может добавить в него определенную сумму и указать, на сколько дней планирует ее распределить. Затем приложение рассчитывает ежедневный бюджет и с этого момента пользователю предлагается фиксировать все свои расходы. Если дневной бюджет превышен, то из общей суммы вычитается разница, а затем рассчитывается новая ежедневная сумма. Если же пользователь не выходит за рамки дневного бюджета, сэкономленный остаток предлагается либо оставить на следующий день, либо добавить к общей сумме. Первоначально данное приложение запускалось как веб-приложение, устанавливаемое ярлыком на рабочий экран мобильного устройства. На данный момент приложение функционирует так же, но уже через оболочку App Store [22].

В приложении «Тяжеловато» нет функций, привычных для большинства финансовых приложений, например, возможности распределять расходы по категориям или разделять имеющиеся деньги на несколько счетов. Тем не менее, отсутствие одной из ключевых возможностей других приложений действительно можно рассматривать ни как концептуальную особенность, а как существенный недостаток, так как в приложении нельзя пополнять бюджет. Единственный способ внести в приложение доходы — это сбросить существующий бюджет и историю расходов, чтобы запланировать новый.

Однако, при достаточно базовом функционале, приложение «Тяжеловато» остается одним из наиболее популярных планировщиков бюджета среди молодежи за счет очень простого использования и позиционирования именно на людей среднего возраста.

2) «CoinKeeper», по утверждению разработчиков приложения, является «самым популярным сервисом учета личных финансов в России». По функционалу приложение является классическим финансовым планировщиком, предлагающим пользователям бесплатную и платную версии приложения [22].

Одним из преимуществ приложения является возможность планировать доходы и расходы наперед, а также возможность устанавливать свои финансовые цели. В то же время заметным недостатком можно назвать то, что в отличие от других базовых финансовых приложений, «CoinKeeper» не позволяет добавлять к тратам комментарии. Все остальные примечательные функции «CoinKeeper» предлагает в случае платной подписке: возможность создавать теги операций и анализировать расходы с их помощью, вести совместный бюджет с другими пользователями, иметь доступ к своему аккаунту с нескольких устройств, импортировать транзакции напрямую из ключевых российских онлайн-банков, вести учет долгов и получать напоминания о сроках их возврата, экспортировать данные в Excel.

Таким образом, «CoinKeeper» можно назвать классическим финансовым приложением, предлагающим набор базовых функций по бесплатной подписке и наличием разнообразных премиум-функций.

3) «Monefy», как и многие приложения, предлагает пользователям бесплатную и платную подписки. Бесплатная версия слабо отличается от любых других приложений, если не считать возможности добавлять комментарии ко всем расходам и доходам и планировать будущие доходы и расходы. В отличие от аналогов, в этом приложении нет функции совместного бюджета, доступа с разных устройств, импорта банковской информации и возможности получить статистику. Лимит трат можно устанавливать только общий и на месяц. Функция создания собственных категорий расходов и

доходов доступна только в платной версии, так же как и возможность создать счета в разных валютах и защитить приложение паролем (в остальных приложениях эта функция есть по умолчанию). Зато, в отличие от остальных приложений, в «Monefy» получить экспортированный CSV файл можно бесплатно [22].

В результате сравнения существующих приложений, приведенного в таблице 1.1, можно сделать следующие выводы. Во всех приложениях есть функция установки лимита трат на определенный период или на определенную категорию расходов. В большинстве приложений есть возможность открыть несколько счетов, например, счета с наличными, банковскими картами, вкладами и так далее. В части приложений есть возможность оставить уникальный комментарий к каждой операции, спланировать будущие доходы и расходы и установить в целях безопасности пароль.

Таблица 1.1 – Сравнительный анализ функций приложений

Функции	«Тяжеловато»	«CoinKeeper»	«Monefy»
Комментарии к операциям	нет	есть	есть
Теги к операциям	нет	платно	нет
Личные категории расходов	нет	есть	платно
Личные источники дохода	нет	платно	платно
Статистика по тегам и категориям	нет	платно	нет
Импорт данных из банка	нет	платно	нет
Будущие доходы	нет	нет	есть
Несколько счетов	нет	есть	есть
Валютные счета	нет	нет	платно
Экспорт данных в Excel	нет	нет	есть
Расширенная статистика	нет	платно	нет

Такие функции, как создание тегов для операций и дальнейшая аналитика расходов по ним, импорт данных о транзакциях из онлайн-банков, возможность

вести совместный бюджет с другим пользователем и пользоваться своим аккаунтом с разных устройств, формирование расширенной финансовой статистики и возможность обозначить собственные источники дохода, во всех представленных приложениях являются платными. Кроме того, в приложении «CoinKeeper» есть уникальная функция учета долгов и последующих уведомлений с напоминаниями о необходимости их вернуть.

В результате анализа приложений и определения сравнительных критериев было выявлено, что наиболее отвечающими цели настоящей работы являются следующие критерии:

— возможность оставлять комментарии к операциям, так как это позволяет отслеживать траты более осознанно, нежели приложения, где такой возможности нет, например, «Тяжеловато»;

— возможность иметь доступ к приложению с разных устройств, так как целью работы является разработка веб-приложения, и отсутствие привязки к конкретному устройству является одним из его ключевых преимуществ;

— возможность установить пароль, так как, несмотря на разную сущность паролей в описанных приложениях и проектируемом веб-приложении, будет необходима аутентификация;

— возможность установить лимит трат, так как это одна из наиболее дисциплинирующих функций, потенциально способствующая развитию финансовой грамотности;

— возможность вести учет долгов, так как долги часто сопутствуют финансовой безграмотности, а кроме того, возврат долгов считается первым шагом на пути к накоплениям и осознанности.

Таким образом, видится очевидным, что среди наиболее известных финансовых приложений нет ни одного аналога, выполняющего все функции, предлагаемые в проектируемом приложении, следовательно, ниша на рынке остается пустой.

Тем не менее, в результате анализа существующих решений, их ключевых функций, достоинств и недостатков относительно поставленной

цели, в дальнейшем будет сформулирована цель создания веб-приложения и будут составлены функциональные возможности, которые разрабатываемое приложение будет решать.

1.2 Функциональные требования

В качестве разрабатываемого приложения будет выступать мобильная версия «Дашборд управления личными финансами», в котором необходимо реализовать функциональные требования, позволяющие организовать учет расходов, контролировать текущие долги и давать пользователю возможность точечной настройки приложения.

Основной целью разработки является создание веб-приложения, которое охватит ни только клиентскую и серверную части, но также базу данных, обеспечивающую необходимый функционал для хранения и загрузки данных пользователя.

В разрабатываемом веб-приложении «Дашборд управления личными финансами» должны присутствовать три функциональных модуля [17, с.88]:

— модуль «Траты», в котором пользователь сможет вносить, удалять и комментировать расходы в течение дня. Также должна присутствовать возможность просмотра остатка финансов на текущий период и просмотра истории трат пользователя за все время;

— модуль «Долги» должен включать в себя возможность вносить долги, выбирать «положительная» или «отрицательная» сумма долга и предоставлять возможность пользователю контролировать время займа. Помимо основной функциональности, должна быть реализована визуализация и интеграция долгов в общую сумму на главной странице веб-приложения;

— модуль «Настройки», в котором сосредоточена вся бизнес-логика в плане индивидуальной настройки приложения под конкретного пользователя. В данном модуле должна быть реализована возможность внесения, изменения и удаления ежемесячных расходов по счетам, а также редактирование профиля и

функция удаления аккаунта.

Разрабатываемое веб-приложение «Дашборд управления личными финансами» подразумевает частое взаимодействие с ним в течение дня, поэтому, основываясь на правилах UI/UX дизайна, необходимо предоставить пользователю интуитивно понятный интерфейс на уровне современных аналогов.

Отличительными особенностями разрабатываемого приложения можно назвать систему справочников с возможностью изменения и дополнения, функции планирования бюджета и возможность перевода средств между счетами пользователя. Это позволяет сделать вывод, что разработка мобильного приложения должна иметь самый эффективный и полный функционал, поэтому приложение конкурентоспособно и может быть выдвинуто на рынок приложений для ведения личного бюджета.

Согласно ГОСТ 34.602-89 в требованиях к автоматизированной системе (АС) должны быть предъявлены требования к входным данным, требования к системе, запрашиваемой самим веб-приложением. Что будет использоваться в качестве входных данных – это то, что необходимо ввести, чтобы приложение работало: потребуется ввод бюджета, которым располагает пользователь и краткое описание потребляемых товаров с указанием их стоимости.

Следует отметить, что пользователь должен иметь возможность единожды записав свои регулярные расходы, выбирать их из предлагаемого списка (рисунок 1.1).

Предположим, пользователь приложения ежедневно выполняет три процедуры: заправляет автомобиль, покупает продукты в магазине и паркует автомобиль около места работы. Таким образом, чтобы не тратить время на ввод наименование статьи расхода, человек может заранее записать его в базу данных, а впоследствии выбирать нужное из списка, дополнительно указав стоимость. Более того, необходимо предусмотреть тот момент, чтобы пользователь имел возможность совершать оплату с минимальными затратами времени и усилий.

Рисунок 1.1 - Контролирование финансов

В качестве функциональных требований необходимо предусмотреть возможность корректировки бюджета, как в положительную, так и отрицательную сторону. Пользователь должен иметь возможность осуществлять контроль над текущим бюджетом, а также вносить коррективы путем его увеличения или уменьшения. Временные данные буду заполняться автоматически в историю трат согласно текущей дате и времени на устройстве (рисунок 1.2).

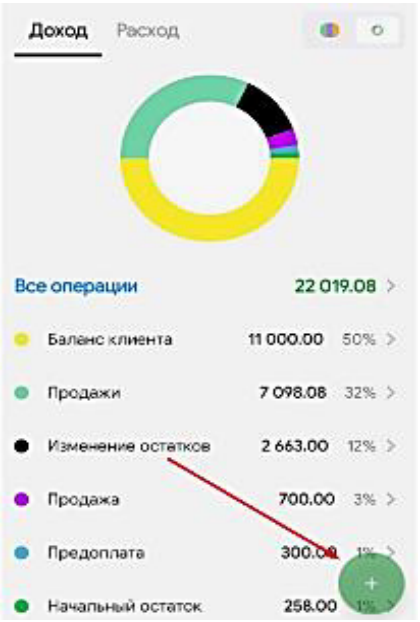

Рисунок 1.2 – Корректировка бюджета

Пользователь должен иметь возможность просматривать историю покупок со всей необходимой информацией. Согласно пункту 2.6.1 ГОСТ 34.602-89 «Требования к системе в целом», 2.6.2 «Требования к функциям (задачам)», 2.6.3 «Требования к видам обеспечения» к разрабатываемому приложению должны предъявляться следующие требования:

— не более 1 Мб оперативной памяти устройства;

— свободное функционирование на устройствах с минимальной мощностью в 400 МГЦ;

— отсутствие аварийного завершения работы;

— продолжение функционирования после сворачивания с целью осуществления другой функции устройства;

— полное и безошибочное осуществление заявленных функций;

— корректный перенос бюджета с одного месяца на другой;

— очистка данных из БД;

— низкий уровень энергозатратности;

— минимальная версия фреймворк, на которой приложение должно свободно функционировать – API 2.2, однако и на API 4.0 приложение должно работать корректно.

Дашборд должен открываться во всех существующих браузерах, поддерживающих текущий стандарт веб-интерфейсов. Интерфейс должен быть адаптирован для отображения в браузере, запущенном как на компьютере, так и на мобильных девайсах. С учетом этих требований предполагается реализовать приложение с использованием Eclipse SDK Android, так как данный инструментарий обеспечивает возможность кроссплатформенной разработки.

2 Проектная часть

2.1 Информационное обеспечение задачи

В качестве методологии для проектирования веб - приложения «Дашборд управления личными финансами» был выбран процесс ICONIX, позволяющий добиться наибольшей эффективности и снизить расхождения процесса разработки и функциональных требований. Данная методология основана на прецедентах. В процессе проектирования для ключевых прецедентов должны быть составлены диаграммы пригодности и диаграммы последовательности. В дальнейшем при анализе диаграмм последовательности должны быть выделены классы и функции, образующие из себя диаграмму классов. Диаграмма классов должна быть использована при написании исходного кода. На основе функциональных требований была спроектирована диаграмма прецедентов и сформулировано текстовое описание для каждого прецедента. Диаграмма прецедентов разрабатываемого приложения приведена на рисунке 2.1.

Рисунок 2.1 – Диаграмма прецедентов

Описание бизнес-прецедента представлено как резюме процесса проектирования в приложении 1.

Нельзя забывать о том, что именно прецеденты в целом помогают описать функционал системы с точки зрения пользователя, а также служат средством коммуникации между экспертами, пользователями и разработчиками. Кроме того, прецеденты позволяют контролировать корректность реализации элемента в течение всей разработки. Прецеденты играют определённую роль в каждом из основных процессов проектирования: разработка требований, анализ и проектирование, выполнение и испытание системы [14, с.39].

И первая диаграмма, которую необходимо рассмотреть будет играть важную роль в информационном обеспечении задачи проектирования веб-приложения, поскольку позволяет визуально отобразить, как пользователи или другие системы будут взаимодействовать с разрабатываемой системой. Речь идет о диаграмме пригодности (Use Case Diagram), которая является одним из типов диаграмм UML, позволяющая визуально отобразить, как пользователи или другие системы будут взаимодействовать с разрабатываемой системой.

Диаграммы пригодности проектируются по отдельности от каждого последующего прецедента. При процессе проектирования данного вида диаграмм, как правило, выделяют наиболее основные три вида объектов:

— граничные объекты, которые непосредственно подразумевают элементы интерфейса;

— объекты-сущности, которые задают существующие вне программы файлы или базу данных;

— объекты-процессы, существующие для дальнейшего преобразования в отдельные классы или их функции.

Роль прецедента «Войти в приложение» в информационном обеспечении проектирования веб-приложения заключается в том, что он служит основой для функционального тестирования. Пользователь первый раз заходит в приложение, вводит необходимые для регистрации данные, нажимает кнопку «Зарегистрироваться». Приложение проверяет введенные данные на их полноту и корректность.

В случае если данные введены корректно, в базу данных сохраняется созданный аккаунт пользователя. Загружается страница первичной настройки приложения. Диаграмма пригодности прецедента «Войти в приложение» приведена на рисунке 2.2.

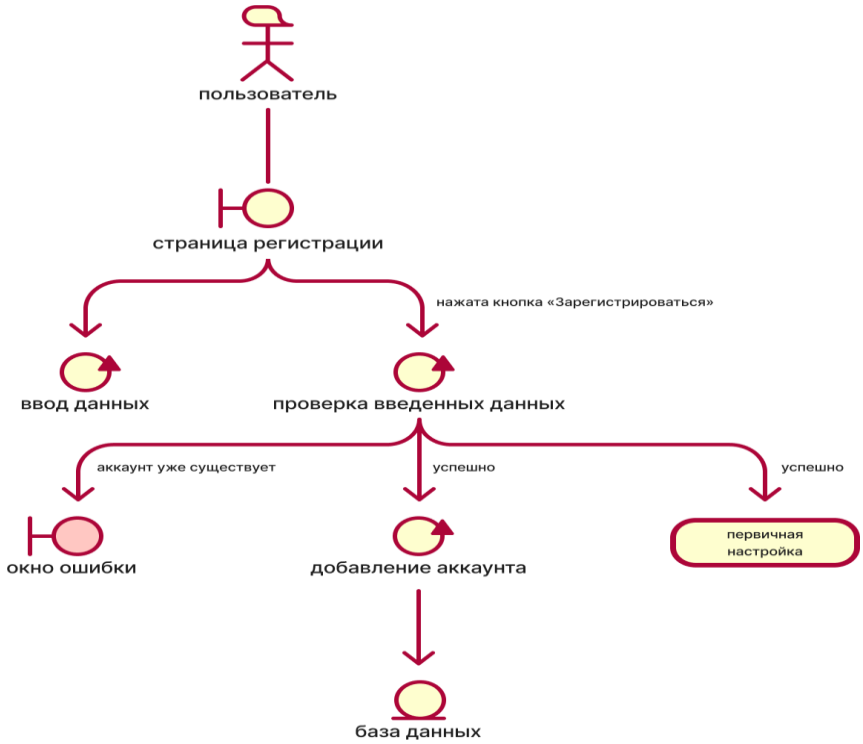

Рисунок 2.2 – Диаграмма пригодности прецедента «Войти в приложение»

Прецедент «Первичная настройка», который будет рассмотрен следующим, помогает в процессе взаимодействия пользователя с приложением, обеспечивая сохранение необходимых данных и информируя пользователя о возможных ошибках при вводе информации. Роль прецедента «Первичная настройка» в информационном обеспечении задачи проектирования веб-приложения заключается в обеспечении первоначальных настроек пользователя после успешной регистрации.

После успешной регистрации, пользователю загружается страница первичной настройки приложения для дальнейшего взаимодействия с ним. Пользователь вводит ежемесячную сумму доходов и желаемый процент накоплений по результатам месяца, нажимает кнопку «Далее». Приложение проверяет введенные данные на их полноту и корректность.

В случае если данные введены корректно, в базу данных сохраняются первоначальные настройки пользователя. Загружается основная страница приложения. Если данные введены некорректно или заполнены не все поля, пользователю сообщается об этом посредством загрузки окна с сообщением об ошибке. Диаграмма пригодности прецедента «Первичная настройка» приведена на рисунке 2.3.

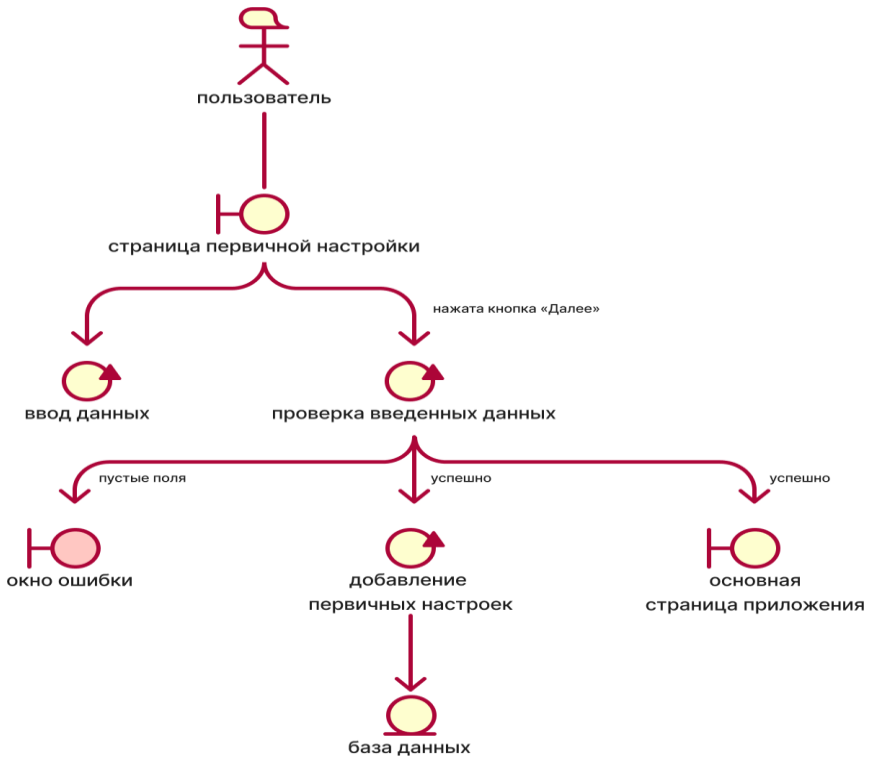

Рисунок 2.3 - Диаграмма пригодности прецедента «Первичная настройка»

Роль прецедентов «Добавить запись о долгах» и «Добавить запись о тратах» заключается в том, что они позволяют пользователю добавлять новые записи о долгах и тратах, что способствует учёту и фиксации финансовых данных. Прецеденты имеют аналогичную последовательность действий и логику работы приложения. Если данные введены некорректно или заполнены не все поля, пользователю сообщается об этом [26].

Поскольку прецеденты «Добавить запись о долгах» и «Добавить запись о тратах» имеют аналогичную последовательность действий и логику работы приложения, рассмотрим только диаграмму пригодности прецедента «Добавить запись о долгах», изображенную на рисунке 2.4.

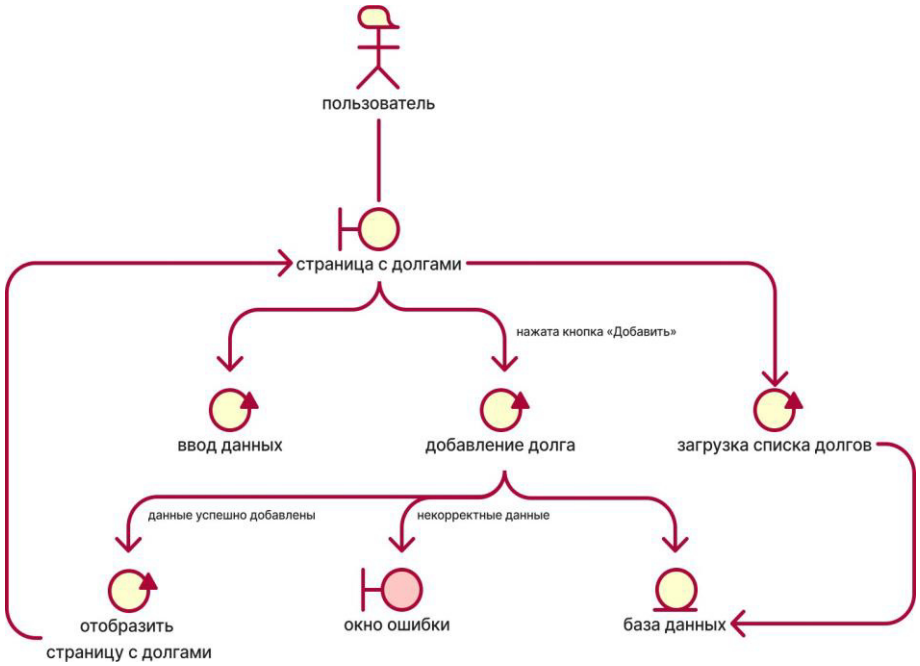

Рисунок 2.4 - Диаграмма пригодности прецедента «Добавить запись о долгах»

Таким образом, прецедент «Добавить запись о долгах» важен для обеспечения функциональности приложения, связанной с финансовым планированием и учётом расходов и доходов пользователя.

Пользователь открывает страницу с долгами, заполняет необходимые поля и выбирает дату, когда долг должен быть возвращен, нажимает кнопку «Добавить». Далее приложение проверяет введенные данные на их полноту и корректность. В случае если данные введены корректно, в базу данных сохраняется запись о долге. На странице с долгами появляется созданная пользователем запись.

Проектирование веб-приложения для финансового планирования личного бюджета всецело обеспечивает возможность изменения записей о тратах. В рамках этого прецедента пользователь может нажать на существующую запись, отредактировать необходимые поля и сохранить изменения.

При правильном редактировании запись обновляется в базе данных, а на основной странице приложения появляется отредактированная запись. Если изменённые данные содержат ошибку, пользователя оповещают об этом, и отредактированная запись не обновляется в базе данных [26].

Получается, что прецедент «Редактировать запись о тратах» позволяет пользователям управлять записями о расходах, что является важным функционалом для приложения, связанного с финансовым планированием. Диаграмма пригодности прецедента «Редактировать запись о тратах», проиллюстрирована на рисунке 2.5.

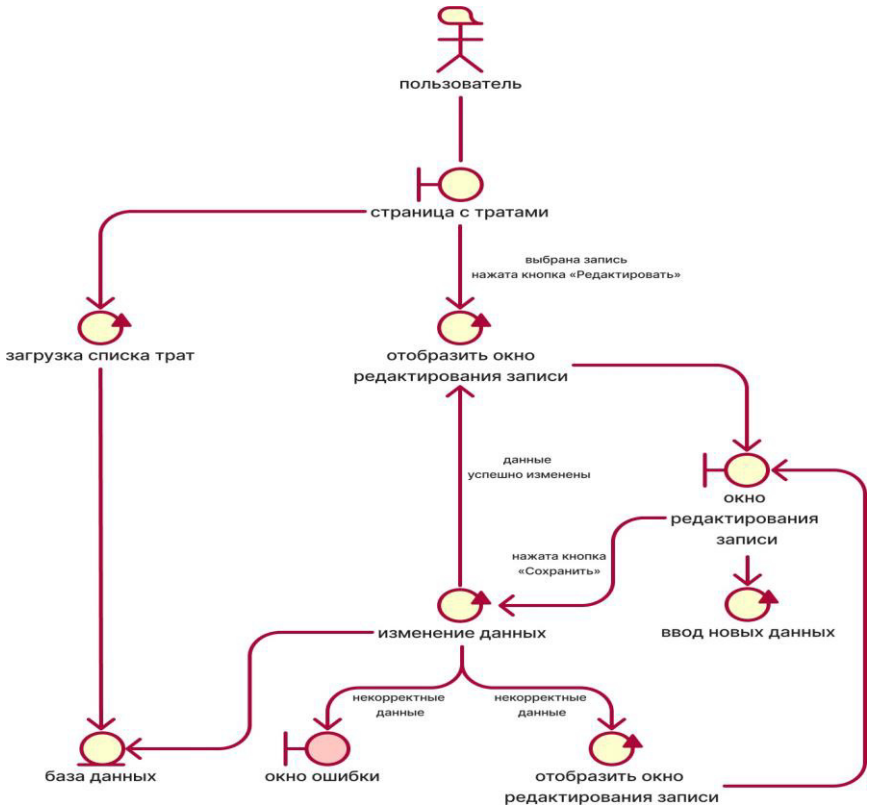

Рисунок 2.5 - Диаграмма пригодности прецедента «Редактировать запись о тратах»

Прецедент «Удалить запись о тратах» в контексте проектирования веб-приложения для финансового планирования обеспечивает возможность удаления записи о тратах и связанных с ней данных из базы данных. После выполнения прецедента на основной странице приложения обновляется список записей о тратах, исключая удалённую запись.

Таким образом, прецедент «Удалить запись о тратах» входит в функционал приложения, позволяя пользователям управлять записями и поддерживать актуальность данных. На основной странице приложения обновляется список записей о тратах без удаленной записи.

Поскольку прецеденты удаления имеют аналогичную логику работы, рассмотрим только диаграмму пригодности прецедента «Удалить запись о тратах», изображенную на рисунке 2.6.

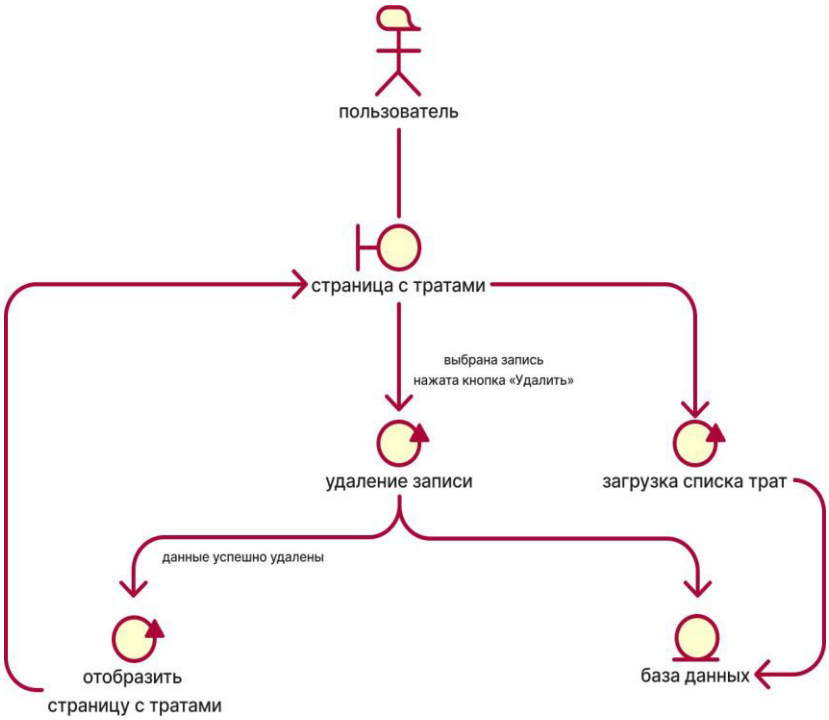

Рисунок 2.6 - Диаграмма пригодности прецедента «Удалить запись о тратах»

При разработке диаграмм последовательности за основу берутся ключевые объекты диаграмм пригодности, а именно — сущностные и граничные объекты, а также акторы.

2.2 Структура приложения

Структура (архитектура) приложения представляет собой, ничто иное, как совокупность компонентов, которые определяют внешний вид и бизнес-логику продукта. Она помогает разработчикам организовать код и сделать его более понятным. В этой связи необходимо выделить основные структурные элементы приложения [7, с.108]:

— интерфейс (UI) - это то, что видит пользователь: экраны, кнопки, меню;

— бизнес-логику составляют правила и алгоритмы, определяющие, как приложение будет работать;

— хранение данных - процесс, при котором записывается информация о пользователях, заказах, меню. Это может быть база данных или файл на устройстве пользователя;

— сетевые запросы, охватывающие связь приложения с сервером, где хранятся данные, и обрабатывается бизнес-логика.

Веб-приложение представляет собой динамическую систему, которая работает с большим количеством данных. Рассмотрим схему работы сайта, с помощью которой пользователь взаимодействует с веб-приложением. Многостраничные сайты работают через полную перезагрузку HTML-страницы. При каждом новом запросе серверу необходимо отправлять клиенту новую HTML-страницу целиком. При этом для каждого нового запроса при переходе на другую страницу сайта, серверу необходимо сгенерировать новую страницу для каждого пользователя. Такой подход значительно замедляет работу пользователя и нагружает сеть. Схема описанного взаимодействия клиента и сервера представлена на рисунке 2.7.

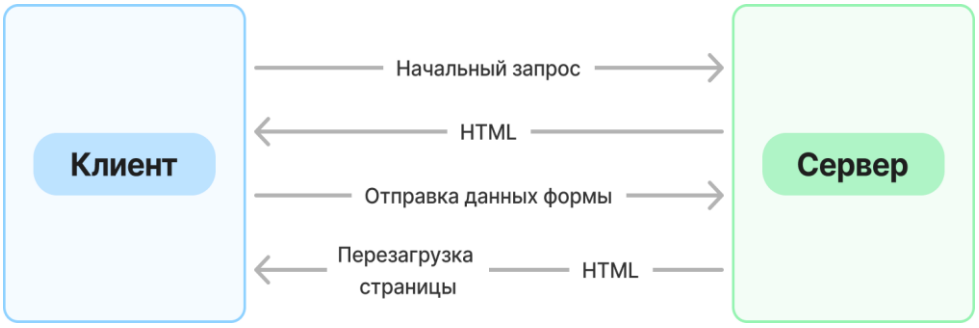

Рисунок 2.7 - Схема работы стандартного многостраничного сайта

На сегодняшний день наиболее распространены сайты, имеющие структуру SPA. SPA-приложения имеют отличную от многостраничных сайтов парадигму работы. Сервер при первом запросе передает HTML - страницу на клиент. Дальнейшие запросы к серверу и обновления страницы осуществляются с помощью AJAX запросов и ответов в формате JSON соответственно [1, с.22].

Благодаря такому подходу к проектированию приложения, удается уменьшить сложность и объем серверной части, что позволит снизить нагрузку на сервер, так как большая часть логики будет выполняться на стороне клиента.

Схема работы SPA-приложения представлена на рисунке 2.8.

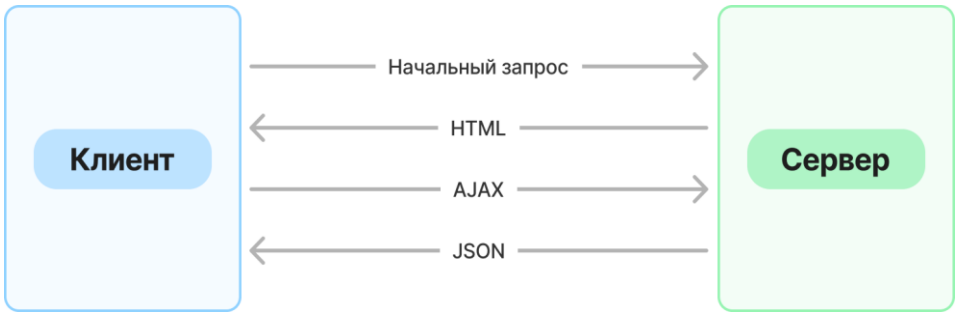

Рисунок 2.8 – Схема работы SPA - приложения

Проектируемое веб-приложение будет реализовано как одностраничное по причине того, что сам функционал подразумевает его активное переключение среди информационных потоков, а частая перезагрузка страницы отразиться, несомненно, на пользовательском опыте.

Коммуникацию между клиентом и сервером необходимо спроектировать в едином стиле. Добиться этого можно с помощью применения AJAX. Для реализации взаимодействия клиента и сервера был выбран современный метод Axios, который поддерживается всеми браузерами и по этой причине заслужил немалую популярность, благодаря функции автоматического преобразования JSON-данных [3, с.57].

Поскольку данное веб-приложение будет спроектировано как SPA, логичным будет разбить на компоненты интерфейс клиентской части, что, конечно же, позволит намного облегчить последующее поддержание работоспособности самого дашборда и упростит возможное масштабирование в дальнейшем.

Схема архитектуры клиентской части приложения приведена на рисунке 2.9. Бизнес-логика веб-приложения не содержит в себе ресурсоемких задач и долгих процедур. По большому счету, вся бизнес-логика состоит из чтения и записи информации в базу данных по тем или иным правилам.

Исключением является задача валидации данных, полученных от клиента или базы данных. Исходя из этого, сервер должен достаточно быстро выполнять задачи чтения и записи.

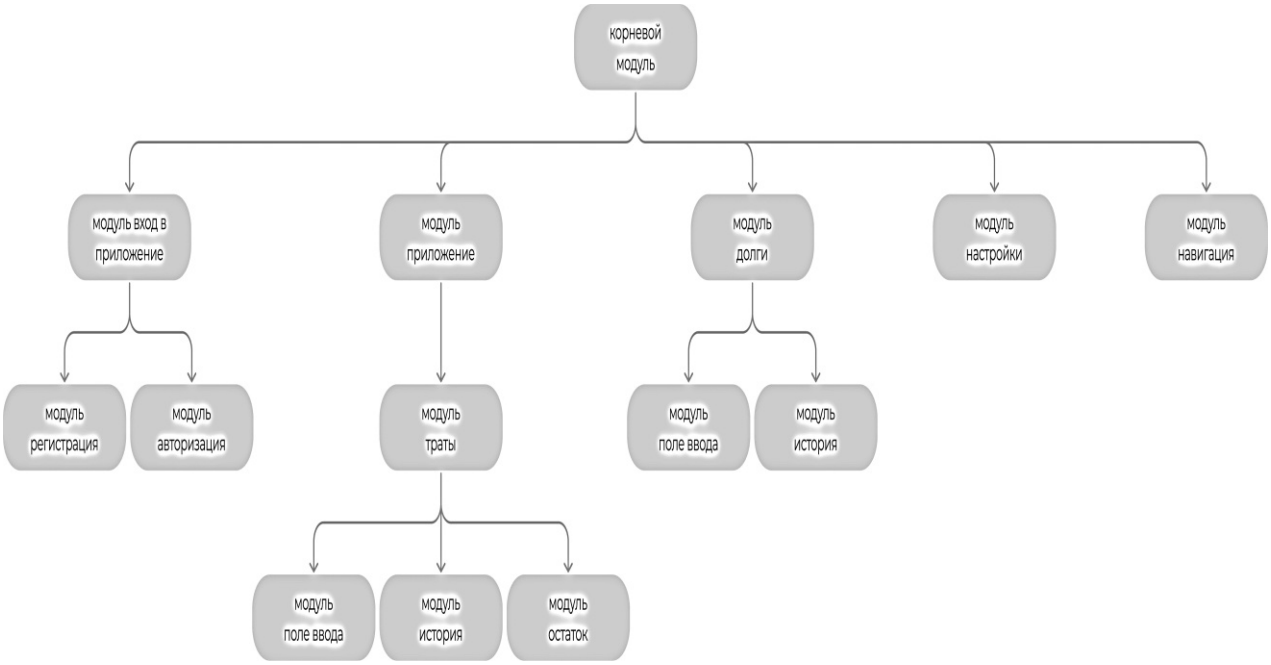

Рисунок 2.9 - Схема интерфейса клиентской части приложения

Концепция веб-приложения «Дашборд управления личными финансами» приводит к выводу, что данные для его функционирования не будут часто обновляться, имеют простую структуру и низкий уровень связности между друг другом. Поскольку в NoSQL базе данных применяется JSON-подобный формат хранения данных, при выборе между базами NoSQL и SQL, более оптимальным вариантом является использование NoSQL базы данных [18, с.188].

2.3 Выбор средств разработки

Поскольку клиентская часть должна быть реализована в виде SPA-приложения, необходимо проанализировать соответствующие средства разработки и подобрать оптимальный инструмент. На сегодняшний день в индустрии веб-технологий, предназначенных для создания SPA-приложений, тройкой лидеров являются три фреймворка.

React представляет из себя JavaScript-фреймворк и был создан для реализации одностраничных приложений [20,с.207].

С помощью данного фреймворка можно разрабатывать мобильные приложения, а с последним обновлением появилась возможность реализовывать и серверную часть. Целью данной библиотеки является обеспечение высокой скорости, простоты и масштабируемости.

Vue.js представляет из себя JavaScript-фреймворк для разработки пользовательского интерфейса. Хорошо показывает себя в ситуациях интеграции с другими JavaScript-библиотеками. Имеет большое сходство с

React, однако, обладает некоторыми внутренними структурными отличиями. Данный фреймворк является самым молодым в тройке рассматриваемых инструментов, однако, Vue.js прогнозируют большое будущее [14, с.98].

Angular представляет из себя JavaScript-фреймворк и обладает полноценной инфраструктурой для разработки больших приложений, включающих в себя большое количество компонентов. Angular - очень мощный инструмент, однако является фреймворком со строгой архитектурой, что значительно снижает его гибкость.

Для разработки пользовательского интерфейса было принято решение использовать библиотеку React. При выборе библиотеки главными критериями являлись большое количество готовых модулей, популярность инструмента в мировом сообществе и проверенность на проектах, которые были реализованы мною ранее посредством выбранной библиотеки [10].

Для сборки кода в один файл применяется инструмент Webpack, в котором присутствует конфигурационный файл, где описано, как обрабатывать определенные типы файлов, а также как обработать полученный JavaScript файл. Как правило, React используется совместно с другими библиотеками для управления состоянием приложения.

Поэтому рассмотрим библиотеки данного типа для выбора подходящей.

Redux библиотека применяется для разработки клиентской части веб-

приложений. Содержит в себе ряд инструментов, благодаря которым удается упростить передачу данных хранилища, содержащего в себе информацию о состоянии приложения, посредством контекста. Имеет ряд утилит, облегчающих разработку и предоставляющих удобство отладки тестирования веб-приложения [10].

Mobx - схожая по своей сути с React библиотека. Однако, имеющая ряд архитектурных отличий. Во-первых, по сравнению с возможностью создавать единственное хранилище в Redux, Mobx предоставляет возможность создавать несколько хранилищ для приложений. Во-вторых, Mobx более ориентирован на создание небольших приложений, и с его помощью труднее поддерживать масштабируемость при более высокой производительности.

React Context API представляет собой набор хуков, применение которых дает возможность управлять состоянием приложения. React Context API не является отдельной библиотекой, поэтому позволяет осуществлять процесс интеграции в приложение без особых сложностей. Однако, из-за присутствия большого состояния и малого количества информации могут возникнуть сложности в работе с приложением.

Для разработки приложения в связке с React была выбрана библиотека Redux. Выбор обуславливается тем, что React-redux позволяет хранить объект состояния приложения в контексте React приложения. React-redux библиотека предоставляет функцию connect, с помощью которой можно получить доступ к данным, хранящимся в объекте состояния. Это можно представить в виде объекта. Если же необходимо внести какие-либо изменения, отправляется действие. В данном случае действие представляет из себя объект, у которого есть обязательно поле type, также могут присутствовать и другие поля. Процесс изменения состояния происходит в функции-редукторе, в которую, в свою очередь, передаются текущее состояние и действие.

Исходя из всего вышеперечисленного, можно сделать вывод, что связка React-redux полностью отвечает требованиям разрабатываемого веб-приложения.

Поскольку Redux не поддерживает асинхронность, а это необходимое свойство клиент-серверного веб-приложения с отзывчивым интерфейсом, необходимо использование Redux-thunk.

Данный «middleware» позволяет отправлять действие с задержкой или при выполнении необходимого условия. Среди доступных вариантов для написания логики серверной части приложения наиболее часто используются PHP, Python и Node.js [2, с.138].

PHP - язык программирования, который полностью основан на сценариях. Является динамическим языком, поэтому довольно часто применяется в веб- разработке. Хорошо показывает себя при работе с СУБД, за счет большого количества написанных для этого инструментов [29, с.40].

Python - язык программирования, стремительно набирающий популярность. Обладает средой разработки на всех современный операционных системах. Существенным недостатком считается сравнительно низкая скорость написания кода программного продукта, что определено его интерпретируемостью. Исходя из этого, Python уверенно занял нишу автоматизаций, в которых не является критичным время выполнения программы [30].

Node.js - платформа, которая преобразует JavaScript в машинный код и позволяет реализовать серверную часть веб-приложений. Свою популярность Node.js получил благодаря большому количеству NPM-пакетов, за счет которых значительно расширился функционал данной платформы. Node.js применяет тот же язык, что и клиентские библиотеки JavaScript. За счет этого можно добиться значительного прироста в скорости выполнения приложения.

Исходя из рассмотренных вариантов, было принято решение выбрать Node.js в качестве платформы для разработки серверной логики. Немаловажным фактором при выборе является наличие практического опыта применения данного инструмента. Как было описано выше, Node.js обладает большим количеством NPM- пакетов, позволяющих упростить процесс разработки.

Для создания веб-сервера приложения нам также понадобится выбрать подходящий пакет. Рассмотрим наиболее популярные пакеты для данной платформы [28, с.19].

Express предназначен для создания веб-серверов приложений.

Обладает минималистичным API и хорошо показывает себя при высокой нагрузке на сервер. Является наиболее востребованным пакетом разработки, поэтому имеет хорошо описанную и подробную документацию. Минусом данного пакета является отсутствие рекомендованного метода организации кода, что при первом знакомстве может увеличить процесс проектирования серверной части приложения.

Hapi рассчитан на проектирование веб-сервера, но в отличие от пакета Express имеет более сложную структуру API. Однако, в данном пакете присутствует довольно мощная система плагинов, благодаря которой можно ускорить разработку и сократить расходы на поддержание готового решения.

Koa представляет доработанную версию пакета Express от той же команды разработчиков. Данный пакет является более производительным и имеет понятную структуру кода. Однако, Koa представили совсем недавно и вследствие относительной молодости возникают трудности интеграции с другими NPM-пакетами.

Таким образом, для реализации веб-сервера был выбран NPM-пакет Express, поскольку он сочетает в себе скорость и стабильность текущей версии. HTTP-запросы будут обрабатываться посредством Router, что позволит декомпозировать логику обработки запросов и направить их в нужный класс в зависимости от того, на какой URL пришел запрос в конкретном случае.

В процессе проектирования возникает вопрос, какую систему управления базами данных (СУБД) выбрать. От выбора СУБД будет зависеть скорость работы приложения, а также последующая масштабируемость. Система управления базами данных (СУБД) играет важную роль в проектировании веб-приложения, так как позволяет хранить, организовывать и извлекать информацию для различных типов приложений.

Изучим ряд наиболее актуальных СУБД для проектирования дашборда [27, с.81]:

MongoDB - данная СУБД классифицируется как NoSQL, в качестве формата данных применяются JSON-документы. Система построена на кластерах и применении сегментированных объектов БД.

Данная БД отлично интегрируется в процесс разработки на JavaScript и различных фреймворках, основанных на данном языке программирования.

MySQL - реляционная СУБД, применяется в разработке большого количества приложений. Использует табличное представление данных.

Рассмотренные выше СУБД представляют данные каждая в своем стиле. В MySQL используется табличное представление, тогда как MongoDB создает коллекции и документы. В разрабатываемом веб-приложении «Дашборд управления личными финансами» практически вся информация будет обрабатываться и передаваться в формате JSON. Отсюда следует, что использование таблиц для хранения информации неудобно для данного веб-приложения. К тому же, JSON - формат обладает большой гибкостью и масштабируемостью по сравнению с табличным представлением. Исходя из этого, можно сделать вывод, что именно использование MongoDB будет правильным выбором [24, с.55].

2.3.1 Общая архитектура системы с выбранными инструментами разработки

Общая архитектура разрабатываемой системы (веб-приложения) с применением выбранного набора инструментов (компонентов, шаблонов, инструментов) предназначена для обеспечения функциональности, надёжности и удобства использования веб-сервисов.

Архитектура определяет структуру приложения, его компоненты и взаимодействия между ними. Цель - обеспечить правильную работу всех элементов вместе, чтобы приложение было гибким, масштабируемым,

производительным и безопасным Общая архитектура разрабатываемой системы с применением выбранного стека технологий включает несколько слоёв, каждый из которых отвечает за определённую функцию [23,с.73].

Стек технологий - это набор инструментов и технологий, с помощью которых создают IT-продукт.

Включает языки программирования, фреймворки, базы данных, среды исполнения, библиотеки и другие компоненты. Общая архитектура разрабатываемой системы с применением выбранного стека технологий представлена на рисунке 2.10. Веб-приложение включает в себя две составляющие: клиент и сервер.

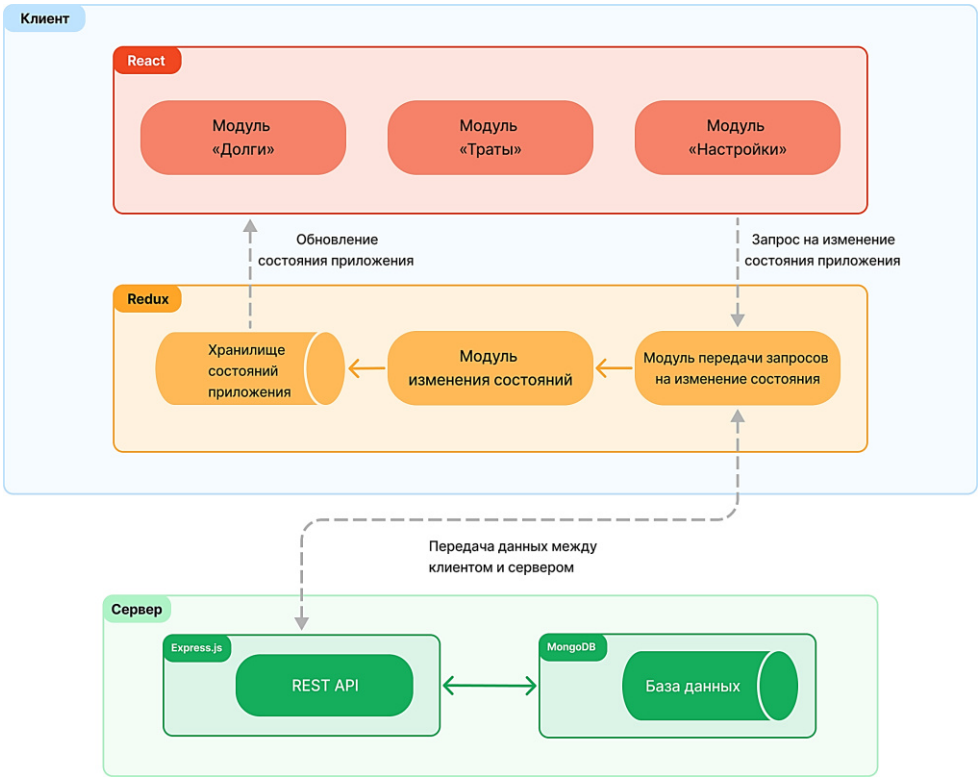

Рисунок 2.10 - Общая архитектура системы

Реализация пользовательского интерфейса будет выполнена посредством библиотеки React. Основные элементы приложения будут разбиты на отдельные модули, называемые компонентами. Отличительной особенностью React является использование состояний компонентов приложения, благодаря которым обеспечивается работа с динамически изменяемыми данными.

Поскольку разрабатываемое веб-приложение сосредоточено на взаимодействии с большим потоком данных, необходимо хранить и изменять данные компонентов и делать это в одном месте. Для этого хорошо подходит библиотека Redux. Основной функционал веб-приложения реализуется, благодаря следующим модулям:

— модуль «Траты»: данный модуль представляет собой основной функционал веб-приложения и позволяет контролировать траты в течение дня, а также редактировать и просматривать историю за все время;

— модуль «Долги»: при внесении информации о долгах и выборе «положительная» или «отрицательная» сумма долга происходит интеграция с другими модулями веб-приложения и реализуется доступ к информации о текущих суммах с учетом новых данных на остальных функциональных модулях;

— модуль «Настройки»: в данном модуле сосредоточен весь функционал редактирования данных профиля и реализована возможность точечной настройки отдельных статей расходов.

Для отслеживания изменений состояния и синхронизации компонентов Redux применяет следующие модули:

— модуль передачи запросов для изменения состояний: после того, как пользователь внесет какие-либо изменения в приложении, например, добавит или удалит запись о долгах, данный модуль получит запрос на обновление состоянии приложения с внесенными изменениями. Также данный модуль может отправлять запрос на сервер, например, получение или удаление новых данных из базы данных;

— модуль изменения состояний: при условии успешной передачи запроса, а также необходимых данных, на изменение состояния, данный модуль выполняет само изменение состояния. После этого происходит загрузка новых данных посредством React-компонент;

— хранилище состояния приложения: в данном объекте находятся все необходимые для работы приложения данные.

Серверная часть приложения построена на базе платформы Node.js в связке с NPM-пакетом фреймворка Express.js. Благодаря данной связке появляется возможность реализовать интерфейс для взаимодействия с клиентом, а также организовать подключение к MongoDB.

2.3.2 Взаимодействие React и Redux

Для того чтобы приступить к разработке внутренней структуры компонента клиентской части приложения, необходимо описать общий принцип работы и взаимодействия React и Redux.

В основе разработки на React лежит техника разделения приложения на отдельные компоненты. В компонентах задаются свойства и состояния — props и state. Оба содержат в себе информацию, которая влияет на конечное отображение приложения, однако, между ними есть фундаментальное отличие. Props служат как параметры функции и передаются в компонент, а state уже находятся внутри компонента. Поведение state можно сравнить с поведением переменных, которые уже объявлены внутри функции.

Благодаря этому можно динамически изменять состояние одного объекта и сразу передавать в свойства других объектов, которые так же изменятся. Вдобавок к этому, компоненты обладают функцией вложенности, имея отношения «родитель-ребенок».

Поскольку в React однонаправленный поток данных, сверху вниз, от родителя к ребенку, но при этом также можно возвращать данные от детей к родителям. Сложность возникает, когда в приложении больше одного уровня, и Становится крайне затруднительно отслеживать и масштабировать разрастающиеся цепочки данных от компонента к компоненту.

Именно для решения данной проблемы применяется Redux. Библиотека позволяет создать единое для всего приложения хранилище, содержащее в себе информацию обо всем приложении и его состояниях сразу. Хранилище представлено в виде большого объекта — JSON-массива. После создания

нового компонента его можно подключить к хранилищу и настроить получение нужных для него данных, при это весь массив данных хранилища передавать не нужно. По аналогии с получением можно настроить передачу данных для изменения хранилища. Благодаря связке React и Redux происходит централизованное управление состоянием в приложениях на React.

Связка позволяет [9, с.64]:

— хранить всё состояние приложения в одном месте — в едином неизменяемом объекте (хранилище, store);

— обеспечить согласованное состояние между компонентами;

— предсказать обновления, т.е. изменения состояния следуют строгому однонаправленному потоку данных;

— упростить отладку, т.е. изменения состояния отслеживаются и становятся более предсказуемыми.

Схема взаимодействия React и Redux представлена на рисунке 2.11.

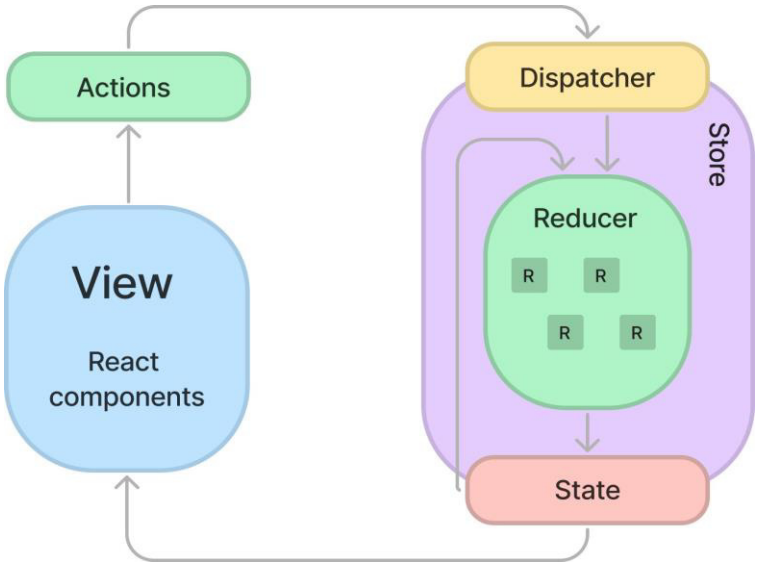

Рисунок 2.11 - Схема взаимодействия React и Redux

После изучения иллюстрированной схемы, связку React.js-Redux использовать в процессе разработки веб-приложения рекомендуется, так как она более детально отражает процесс управления состоянием. Это связано с тем, что React.js и Redux решают разные задачи: React отвечает за отображение

и обновление пользовательского интерфейса, а Redux - за централизованное управление состоянием, что в итоге позволяет более компетентно приходить к результату разработки.

2.4 Проектирование структуры клиентского приложения

Проектирование структуры клиентского приложения (frontend) играет важную роль в разработке веб-приложения, так как именно клиентская сторона отвечает за взаимодействие с пользователем и отображение данных. Рассмотрим структуру клиентской части приложения. На рисунке 2.12 представлена общая структура в связке React и Redux в качестве фреймворка приложения и хранилища состояний приложения соответственно. React-приложение состоит из компонентов, представленных ниже.

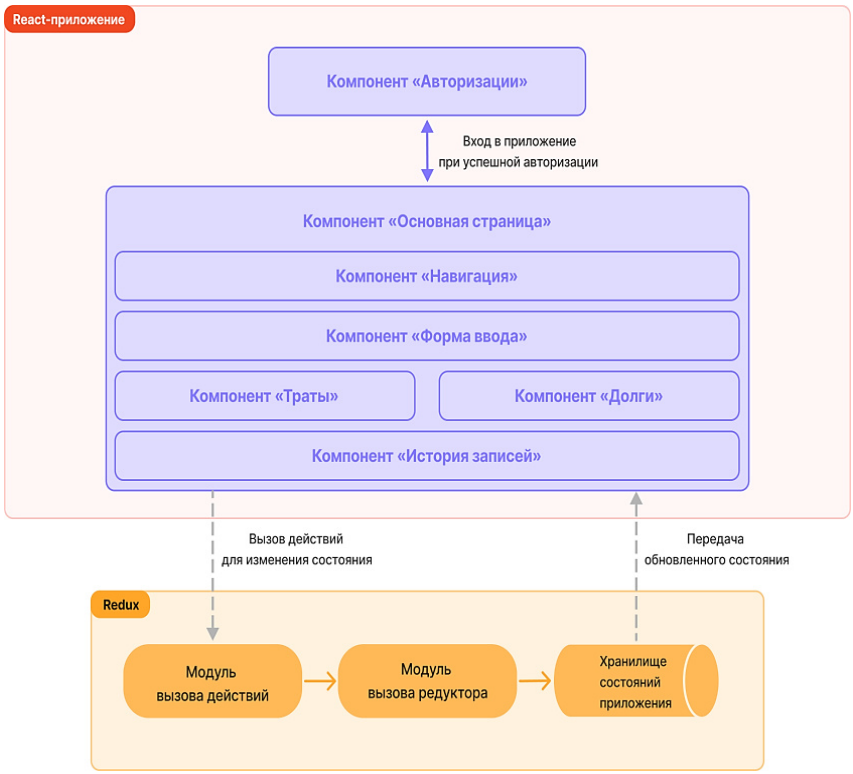

Рисунок 2.12 - Структурная схема взаимодействия React и Redux-хранилища

Компоненты в структуре клиентского приложения React - это независимые и переиспользуемые строительные блоки пользовательского интерфейса.

Каждый компонент отвечает за свой рендеринг и управление состоянием в процессе проектирования веб-приложения [13, с.47].

Рассмотрим каждый компонент более подробно.

Компонент «Основная страница» — компонент, загружаемый в окне браузера по умолчанию после успешной авторизации или при нажатии кнопки «Главная» в компоненте «Навигация».

Данный компонент содержит основной функционал приложения, который также состоит из компонентов. Структура компонента изображена на рисунке 2.13.

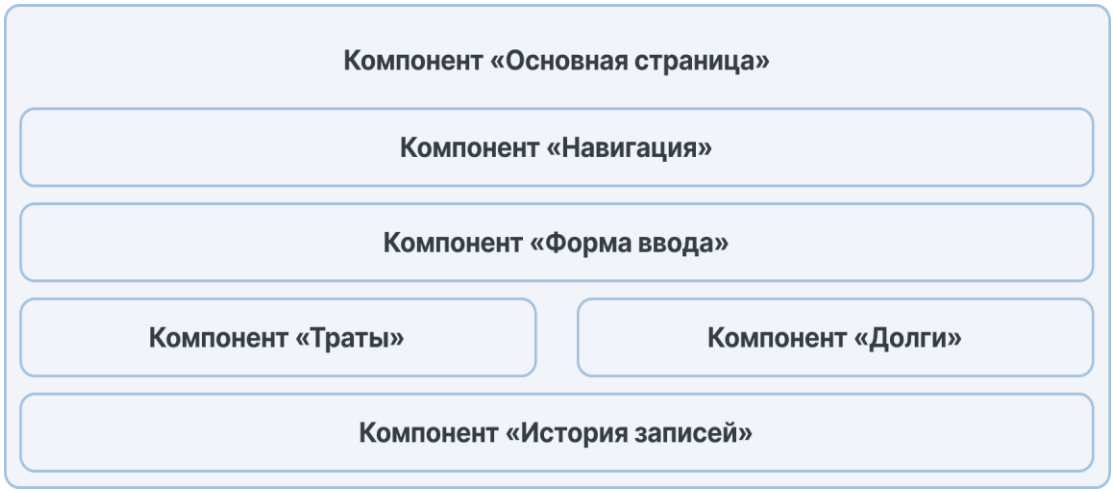

Рисунок 2.13 - Внутренняя структура компонента «Основная страница»

Внутренняя структура компонента «Основная страница» состоит из следующих компонентов:

— компонент «Навигация» — представляет собой «гамбургер», при нажатии на который раскрывается меню с навигацией по приложению; Поскольку данное приложение спроектировано по принципу SPA, иконка «гамбургера» будет присутствовать на всех ключевых компонентах интерфейса клиентской части веб-приложения.

— компонент «Форма ввода» — данный компонент позволяет вводить потраченную сумму и указывать комментарий к ней. При этом, после нажатия на кнопку «Добавить запись», новая запись сразу отображается на главной странице приложения в компоненте «История записей», а остаток от дневной

суммы изменяется в компоненте «Траты»;

— компонент «Траты» — отображает рекомендуемую доступную сумму на день, автоматически изменяется при добавлении новых трат, расходов и долгов. Если дневные расходы превышают рекомендуемую сумму на день, тогда происходит перерасчет рекомендуемой дневной суммы с учетом желаемого процента для накоплений;

— компонент «Долги» — отображает текущее состояние по долгам, при нажатии на данный компонент пользователь перейдет на страницу долгов, где сможет добавить новый долг, просмотреть историю записей или распорядиться положительным балансом долга — суммировать его к общему балансу или погасить им отрицательные долги;

— компонент «История записей» — представляет собой список из внесенных пользователем трат с отображением суммы, комментария и даты добавления записи. При нажатии на кнопку «Далее», откроется страница с историей трат за все время.

Вторым основным компонентом является выпадающие меню со списком навигации приложения. Поскольку данное приложение спроектировано по принципу SPA, данный компонент будет присутствовать на всех ключевых компонентах интерфейса клиентской части веб-приложения. Структура компонента «Навигация» изображена на рисунке 2.14.

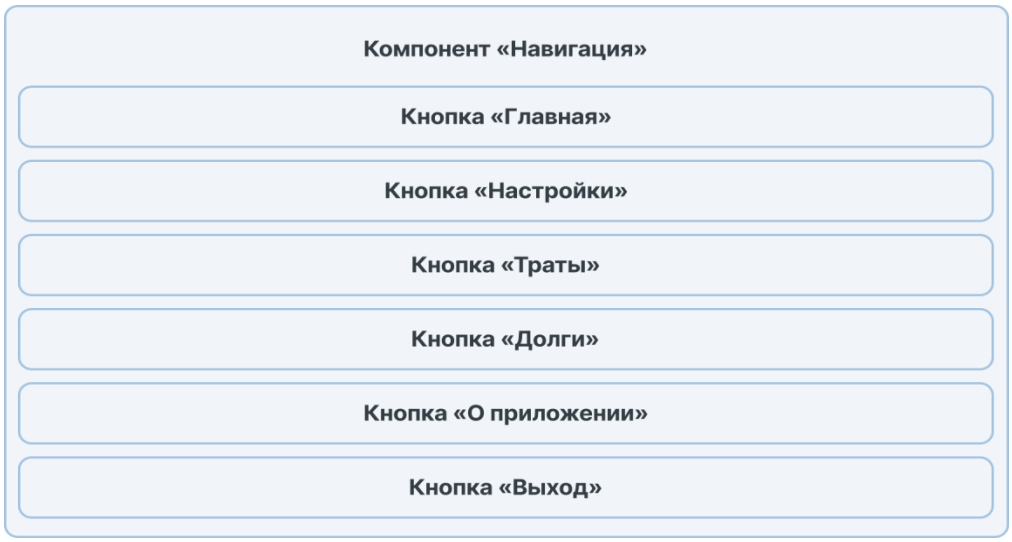

Рисунок 2.14 – Внутренняя структура компонента «Навигация»

Описание компонента «Навигация»:

— кнопка «Главная» перенаправляет пользователя на основную страницу веб-приложения;

— кнопка «Настройки» перенаправляет пользователя в компонент «Настройки», где он может изменить данные профиля или точечно настроить ежемесячные расходы под себя;

— кнопка «Траты» открывает историю всех записей о тратах пользователя в расширенном формате представления данных;

— кнопка «Долги» перенаправляет пользователя в компонент «Долги», где он может вести учет долгов и взаимодействовать с записями о них;

— кнопка «О нас» открывает компонент «О нас», где представлена общая информация о приложении и текущая версия;

— кнопка «Выход» при нажатии на нее выходит из аккаунта пользователя и перенаправляет его на страницу авторизации.

Третий компонент, который будет показан при нажатии на кнопку «Долги» в навигационном меню или на соответствующую иконку с общей суммой долгов на основной странице проиллюстрирован на рисунке 2.15.

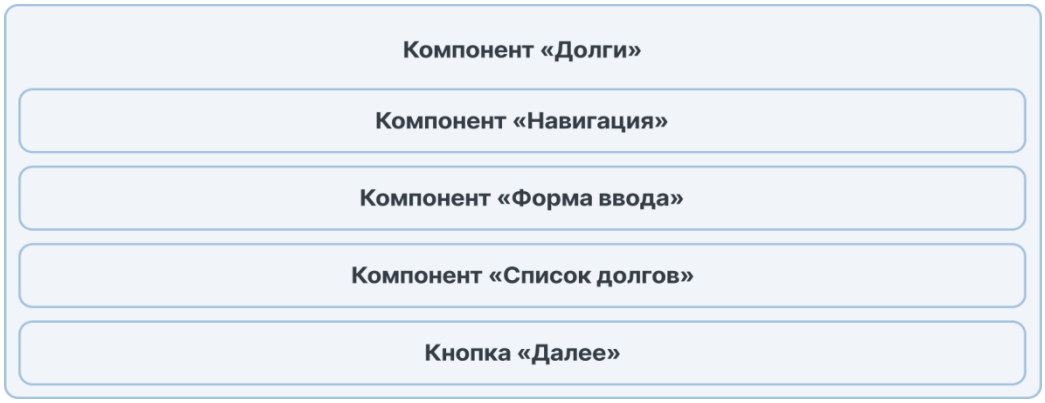

Рисунок 2.15 - Внутренняя структура компонента «Долги»

Основная роль, которую выполняет компонент «Долги», заключается в отображении текущего состояния по долгам. При нажатии на этот компонент пользователь переходит на страницу долгов, где может добавить новый долг, просмотреть историю записей или распорядиться положительным балансом

долга — суммировать его к общему балансу или погасить им отрицательные долги. Описание компонента «Долги»:

— компонент «Форма ввода» — данный компонент позволяет вводить сумму долга и указывать комментарий к ней, а также предоставляет возможность выбрать «отрицательная» или «положительная» сумма долга, при этом, после нажатия на кнопку «Добавить запись», новая запись сразу отображается на странице приложения в компоненте «Список долгов», а на главной странице приложения меняется сумма в компоненте «Сумма долгов»;

— компонент «Список долгов» — содержит в себе записи о долгах с отображением комментариев, даты создания и функциональной кнопкой «Удалить». Также предусмотрена возможность погасить отрицательный долг или перенести сумму положительного долга на текущий месяц в случае возврата долга и появления плюса по долгам;

— кнопка «Далее» загружает предыдущие записи о долгах пользователя в хронологическом порядке.

Заключительный компонент, который помогает структурировать и упрощать разработку, позволяя распределить функционал на отдельные блоки кода и создавать иерархию компонентов для более удобного управления проектом представлен на рисунке 2.16. Компонент «Настройки» отобразится при нажатии на кнопку «Настройки» в навигационном меню, после чего загрузится сам компонент.

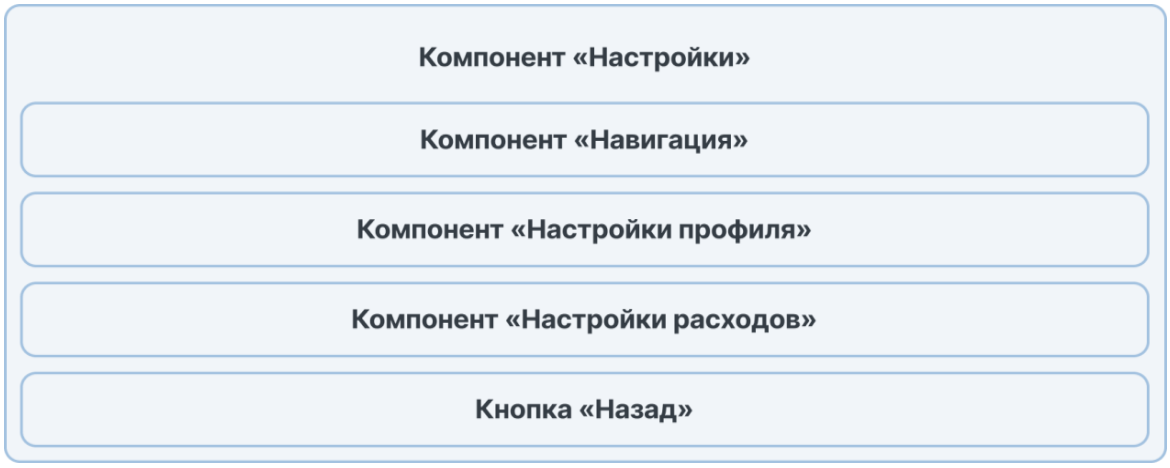

Рисунок 2.16 - Внутренняя структура компонента «Настройки»

Описание компонента «Настройки»:

— компонент «Настройки профиля предоставляет пользователю возможность редактировать настройки данного профиля, а также изменять введенные при первичной настройке приложения данные, а именно, ежемесячный доход и желаемый процент на остаток;

— компонент «Настройки расходов» содержит в себе функционал для более точечной настройки расходов пользователя. Позволяет указать ежемесячные расходы на коммунальные услуги, оплату аренды жилья и т.д;

— кнопка «Назад» возвращает пользователя на основную страницу приложения.

Таким образом, можно смело утверждать, что компонентная архитектура React обеспечивает эффективный и организованный способ построения пользовательских интерфейсов. Она позволяет создавать масштабируемые, поддерживаемые и высокопроизводительные приложения.

2.5 Проектирование структуры базы данных

Проектирование базы данных представляет собой, ничто иное, как процесс создания структурированного плана организации, хранения и управления данными для обеспечения целостности, согласованности и эффективности данных. Структура документов в базе данных представлена в таблице 2.1.

Таблица 2.1 - Структура документов в базе данных

Название документа	Ключи	Данные
Post	number	Number
	user	id пользователя
	amount	Number
	comment	String
	date	Date
	user	id пользователя
	income	Number

Продолжение таблицы 2.1

Debt	number	Number		
	user	id пользователя		
	amount	Number		
	comment	String		
	date	Date		
Profile	expenses	Объект	Ключ	Значение
			housing	Number
			rent	Number
			transportation	Number
			services	Number
			subscriptions	Number
	savings	Number		
	days	Number		
User	name	String		
	email	String		
	password	String		
	date	Date		

Для того, чтобы обеспечить масштабируемость и доступность хранимых в базе данных, хорошим решением было разбить все данные, которые могут поступать на сервер, на четыре коллекции.

На основании табличных данных была построена структура документов в базе данных со связями, которая представлена на рисунке 2.17. Хорошо структурированная база данных помогает сэкономить дисковое пространство за счёт исключения лишних данных, поддерживает точность и целостность данных, обеспечивает удобный доступ к ним. Внутри самой базы данные содержатся в следующих коллекциях, которые в контексте представляют собой группу объектов, каждый из которых может быть любого типа.

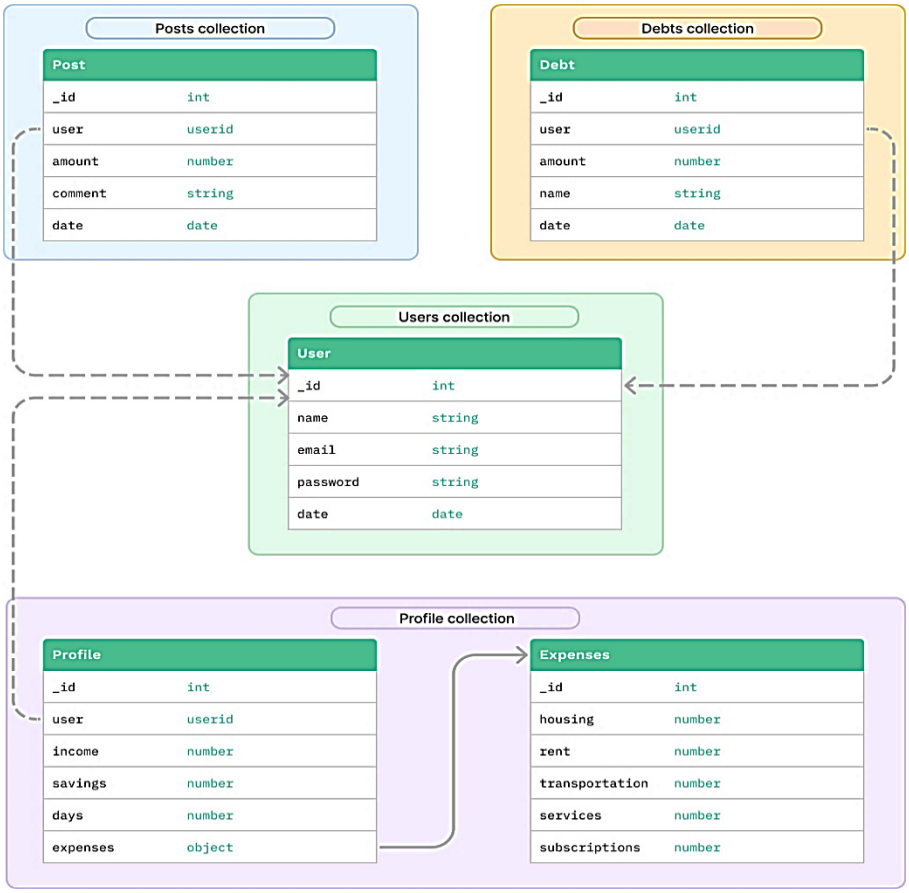

Рисунок 2.17 – Структура базы данных

Коллекции позволяют производить некоторые действия со всеми входящими в них объектами:

1) Коллекция Users. Содержит данные о зарегистрированных пользователях. Документы имеют следующие поля:

1.1._id - уникальный идентификатор пользователя;

1.2.name - логин пользователя;

1.3.email - электронная почта пользователя;

1.4.password - пароль пользователя;

1.5.date - дата регистрации пользователя.

2) Коллекция Posts. Хранит данные с записями пользователя о тратах. Содержит следующие поля:

2.1._id - уникальный идентификатор записи о трате;

2.2.user - идентификатор пользователя;

2.3.amount - сумма траты;

2.4.comment - комментарий о трате;

2.5.date - дата создания записи.

3) Коллекция Debts. Хранит данные с записями пользователя о долгах. Имеет следующие поля:

3.1._id - уникальный идентификатор записи о долге;

3.2.user - идентификатор пользователя;

3.3.amount - сумма долга;

3.4.name - комментарий о долге;

3.5.date - дата возврата долга.

4) Коллекция Profile.Хранит данные с записями пользователя о долгах. Имеет следующие поля:

4.1._id - уникальный идентификатор профиля;

4.2.user - идентификатор пользователя;

4.3.income - сумма доходов пользователя;

4.4.savings - желаемый процент накоплений;

4.5.days - количество дней.

4.6.expenses - объект, напрямую связан с расходами.

Подводя итог проектирования веб – приложения «Дашборд управления личными финансами» отметим, что клиентское приложение будет построено по принципу SPA-это обусловлено тем, что данный подход обеспечивает быстроту и гибкость по сравнению со стандартным многостраничным сайтом. Выполнено проектирование архитектуры веб-приложения. Для разработки веб-приложения будет использоваться язык HTML в качестве описания статической информации, CSS - для стилизации элементов. Фреймворк React в связке с библиотекой Redux в данном веб- приложении будет служить для обеспечения логики и функциональности приложения. Серверная часть приложения будет построена на базе платформы Node.js с использованием NPM-пакетов Express.js и Mongoose. База данных будет развернута с использованием СУБД MongoDB [12, с.33].

3 Обоснование технико-экономической эффективности результатов ВКР

3.1 Реализация клиентской части

Клиентская и серверная стороны представляют собой два основных компонента архитектуры веб-приложений, которые взаимодействуют для обеспечения функциональности и взаимодействия с пользователем. Клиентская часть приложения представляет из себя древовидную структуру и состоит из большого количества компонентов, каждый из которых отвечает за определенную часть пользовательского интерфейса и логику ее отрисовки. Компоненты удобно объединять в модули, в зависимости от функций, которые они выполняют. Клиентская сторона браузера отправляет запросы на сервер для получения данных и отображения их пользователю. Она также обрабатывает пользовательский ввод и осуществляет валидацию данных на стороне клиента. Рассмотрим ключевые модули и корневой файл App.js.

Корневой файл App.js представляет из себя корневой файл и является точкой входа в приложение. В данном файле сосредоточены все существующие компоненты, посредством объединения которых формируются модули приложения. Пример реализации модулей посредством корневого файла App.js изображен на рисунке 3.1.

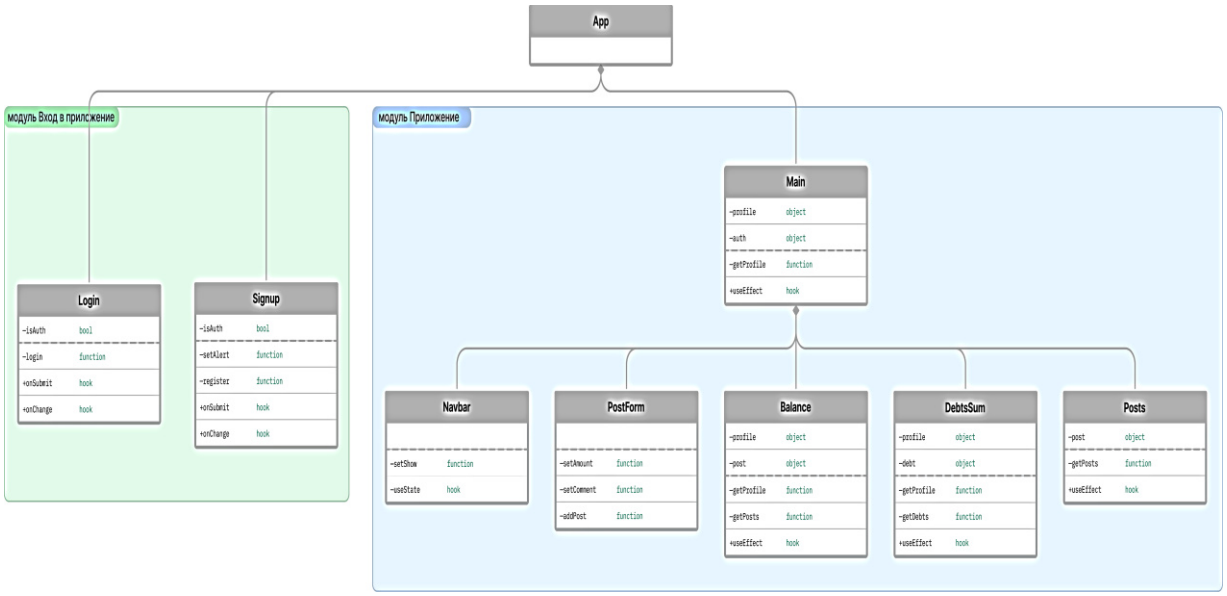

Рисунок 3.1 - Реализация модулей посредством корневого файла App.js

Модуль «Вход в приложение» состоит из двух компонентов:

— компонент Signup отвечает за логику работы регистрации пользователя. Данный компонент состоит из стандартных для регистрации полей ввода данных — «имя пользователя», «электронная почта», «пароль», «повторите пароль»; кнопок — «Зарегистрироваться» и «Назад». Так же предусмотрена возможность перейти к компоненту Login, если пользователь уже был зарегистрирован и хочет войти в приложение. Код компонента Signup представлен в приложении 2.

При нажатии на кнопку «Зарегистрироваться», данные введенные в обязательные поля проверяются на корректность и если данные корректны, передаются в функцию register. В функции register данные преобразуются в формат JSON и отправляются на сервер POST запросом. Если запрос обработан, то в payload передаются данные ответа и запускается функция loadUser, которая отправляет GET запрос и кладет данные в payload для дальнейшей аутентификации пользователя посредством выставления флага isAuthenticated в true. После чего происходит переадресация на страницу первичной настройки приложения;

— компонент Login отвечает за логику работы авторизации пользователя, имеет два поля для ввода данных — «электронная почта» и «пароль», необходимых для аутентификации пользователя в приложении, а также две кнопки — «Войти» и «Назад». Кнопка «Назад» является ссылкой на начальную страницу при первом запуске приложения. Кнопка «Войти» вызывает действие onSubmit которое передает в функцию login два поля email и password. В функции login данные преобразуются в JSON формат и отправляются на сервер POST запросом. Если запрос обработан, то ответ сервера кладется в payload для дальнейшей аутентификации пользователя посредством выставления флага isAuthenticated в true. После чего происходит переадресация на основную страницу приложения.

Каждая функция в этих компонентах обернута в конструкцию try-catch, благодаря которой пользователя оповещают об ошибках ввода и других

проблемах в виде всплывающих окон ошибки. Структура модуля «Вход в приложение» изображена на рисунке 3.2.

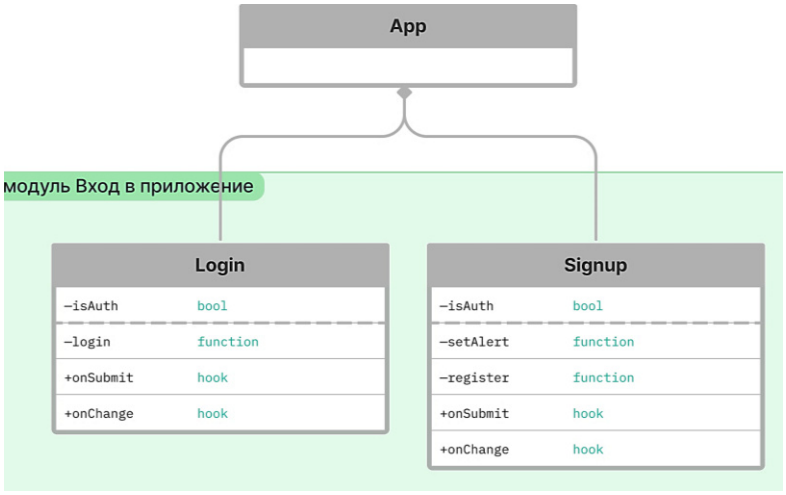

Рисунок 3.2 - Структура модуля «Вход в приложение»

Модуль «Приложение» является самым большим модулем клиентской части приложения и включает в себя модуль «Траты». Клиентский модуль инкапсулирует часть общей функциональности приложения и обычно представляет собой набор взаимосвязанных функциональных частей. Он может включать функции приложения, включая пользовательский интерфейс и бизнес-логику, или части инфраструктуры приложения, такие как службы уровня приложения для аутентификации и авторизации пользователей.

Модуль «Траты», в свою очередь, представляет собой основной функционал веб-приложения и позволяет контролировать траты в течение дня, а также редактировать и просматривать историю за всё время. Точкой входа в модуль выступает компонент Main, который отрисовывает следующие компоненты:

— компонент Navbar имеет два состояния, в закрытом состоянии отображается как скрытое навигационное меню и является кнопкой, при нажатии на которую происходит отрисовка вложенного компонента Menu, включающего в себя ссылки для перехода на все существующие модули и кнопку «Выход». Данный компонент присутствует на всех модулях приложения;

— компонент DebtsSum, который отвечает за загрузку и отрисовку общей суммы долгов. Данный компонент загружает данные из компонента Debts и реализован с применением стандартного хука useEffect. Остальные компоненты являются частью модуля «Траты» и будут рассмотрены далее.

Структура модуля «Приложение» изображена на рисунке 3.3.

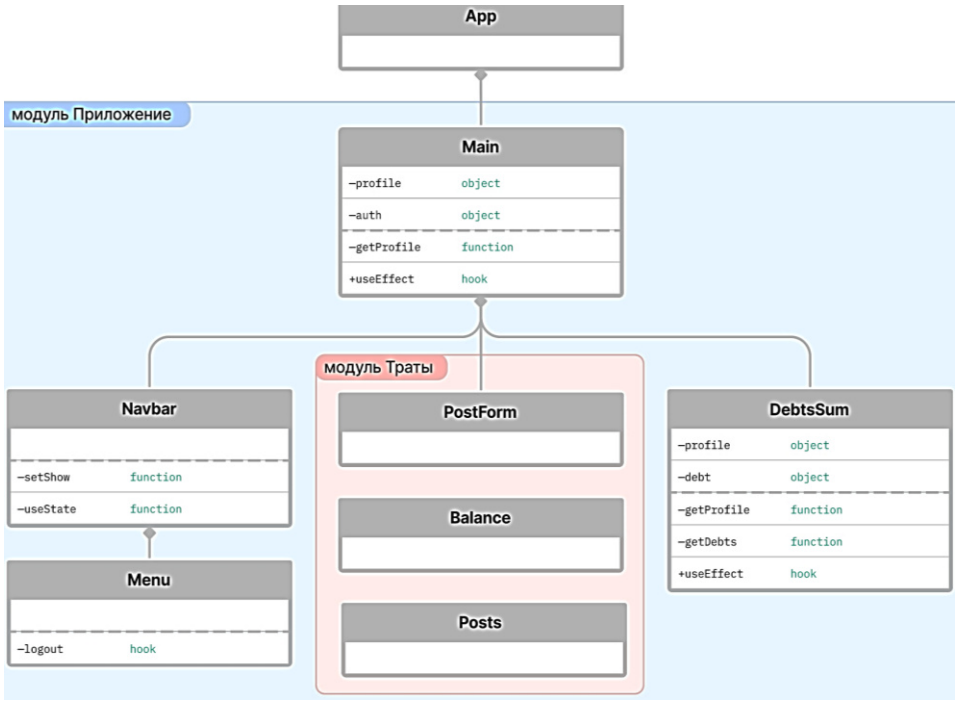

Рисунок 3.3 - Структура модуля «Приложение»

Модуль «Траты» является вложенным в модуль «Приложение» и отвечает за бизнес-логику, выполняющуюся на стороне клиента. В контексте веб-приложения данный модуль представляет собой основной функционал приложения, позволяет контролировать траты в течение дня, а также редактировать и просматривать историю за всё время за счет отрисовки следующих компонентов:

— компонент PostForm, в котором реализованы функции добавления новой записи путем внесения данных в текстовые поля и нажатия кнопки «Добавить». После того как пользователь нажал кнопку, обработчик onSubmit отправляет данные на валидацию на стороне клиента, при успешной валидации поля amount и comment передаются в функцию addPost.

В функции addPost данные преобразуются в JSON формат и

отправляются на сервер POST запросом. Если запрос обработан, то в payload передаются данные ответа и с помощью функции setAlert на стороне клиента появляется всплывающее уведомление о том, что запись создана. Созданная запись сохраняется в базу данных с помощью библиотеки mongoose, посредством обращения к методу save в маршруте для записей;

— компонент Balance, в данном компоненте сосредоточена бизнес-логика расчета рекомендуемой суммы на день с помощью применения хука useEffect и отрисовка данной суммы благодаря возвращемому фрагменту;

— компонент Posts отвечает за подгрузку и отображение добавленных записей, каждая из которых отрисовывается благодаря вложенному компоненту PostItem, возвращаемые от сервера записи выстраиваются в коллекции в порядке убывания даты создания записи с помощью метода sort.

Структура модуля «Траты», с описанием всех компонентов изображена на рисунке 3.4.

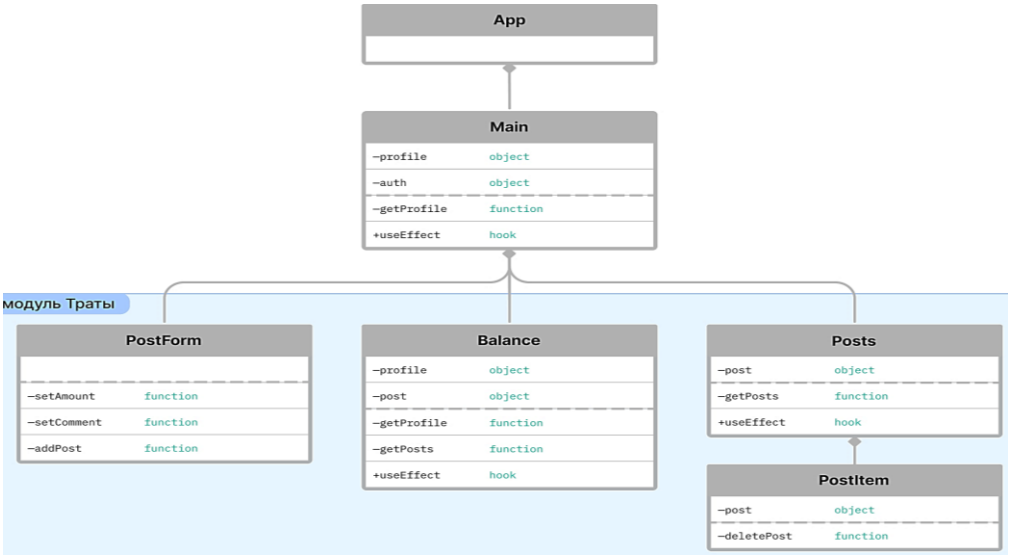

Рисунок 3.4 - Структура модуля «Траты»

Модуль «Настройки» включает в себя все присутствующие возможности редактирования пользовательских данных и более тонкой настройки приложения. Точкой входа является корневой файл App.js.

В зависимости от выбранного раздела настроек, происходит отрисовка следующих компонентов:

— компонент Profile, в котором происходит отрисовка ежемесячного дохода, желаемого процента на остаток и ежемесячных расходов на различные услуги. Данные для отрисовки поступают из базы данных с помощью функции getProfile, которая обращается к API сервера по пути api/profile посредством GET запроса. После чего в payload передаются данные пользователя, отображение которых происходит с помощью обращения к нужному полю объекта посредством оператора доступа;

— компонент EditProfile, в котором пользователь может изменить первоначальные настройки профиля. Сначала данный компонент получает данные пользователя с помощью хука useEffect и вызова в нем функции getProfile, после чего происходит отрисовка данных на стороне клиента. Затем пользователь редактирует данные и нажимает кнопку «Сохранить», в результате чего обработчик onSubmit передает измененные поля в функцию createProfile. В функции createProfile происходит обращение к серверу для изменения данных. При успешной обработки запроса в payload передаются данные ответа и с помощью функции setAlert у пользователя появляется всплывающее уведомление о том что настройки успешно отредактированы. После чего происходит переход на общую страницу настроек с помощью функции navigate и передачи в нее параметра /profile;

— компонент EditExpenses, в котором пользователь может внести свои ежемесячные расходы на различные услуги и подписки. В процессе взаимодействия пользователя с полями ввода, благодаря событию onChange происходит мгновенное изменение объекта на основе введенных пользователем данных. После чего пользователь нажимает кнопку «Сохранить», тем самым вызывая обработчик onSubmit, который передает новые данные в функцию addExpenses. В функции addExpenses переданные в нее данные преобразуются в JSON формат и отправляются на сервер в виде AJAX запроса с помощью передатчика axios. Если ответ от сервера о внесении новых данных в базу данных успешный, происходит переход на общую страницу настроек с помощью функции navigate. На основной странице настроек выполняется

отрисовка новых данных с помощью применения стандартного хука useEffect и вызова в нем функции getProfile.

Структура модуля «Настройки» изображена на рисунке 3.5.

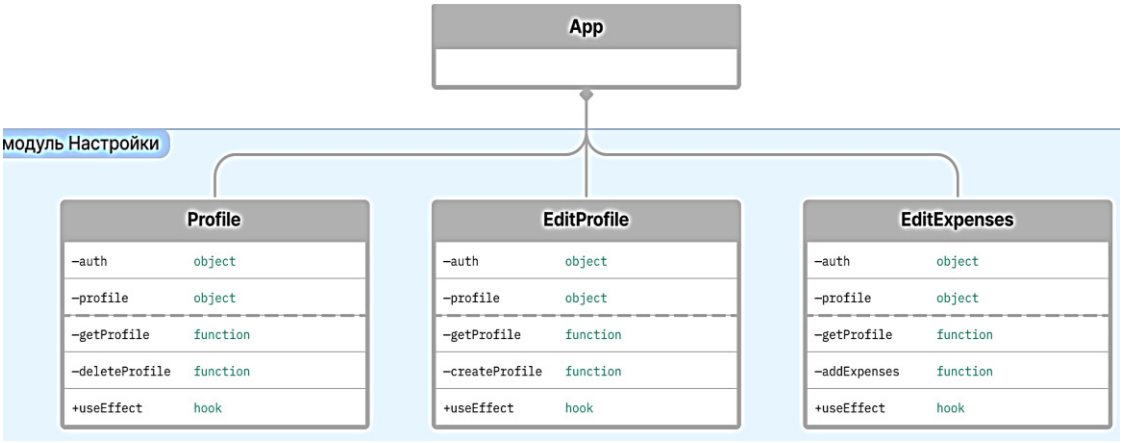

Рисунок 3.5 - Структура модуля «Настройки»

Модуль «Долги» отвечает за логику работы приложения в части добавления, редактирования и удаления записей о долгах. Точкой входа является компонент Debts, через который происходит отрисовка следующих компонентов:

— компонент DebtForm в котором реализованы функции создания новой записи путем ввода данных в текстовые поля, изменение данных в полях ввода происходит мгновенно при помощи хука useState, и нажатия кнопки «Добавить». После того как пользователь нажал кнопку, обработчик onSubmit запускает первичную валидацию данных на стороне клиента, при успешной валидации поля amount, name, date и returnDate передаются в функцию addDebt. В функции addDebt данные преобразуются в JSON формат и отправляются на сервер POST запросом. При успешной обработки запроса в payload передаются данные ответа и с помощью функции setAlert на стороне клиента появляется всплывающее уведомление о том, что запись создана;

— компонент DebtItem, с помощью данного компонента происходит отрисовка записей о долгах, выстраивание происходит в порядке убывания даты создания записи с помощью применения метода sort к коллекции в базе данных.

Структура модуля «Долги» изображена на рисунке 3.6.

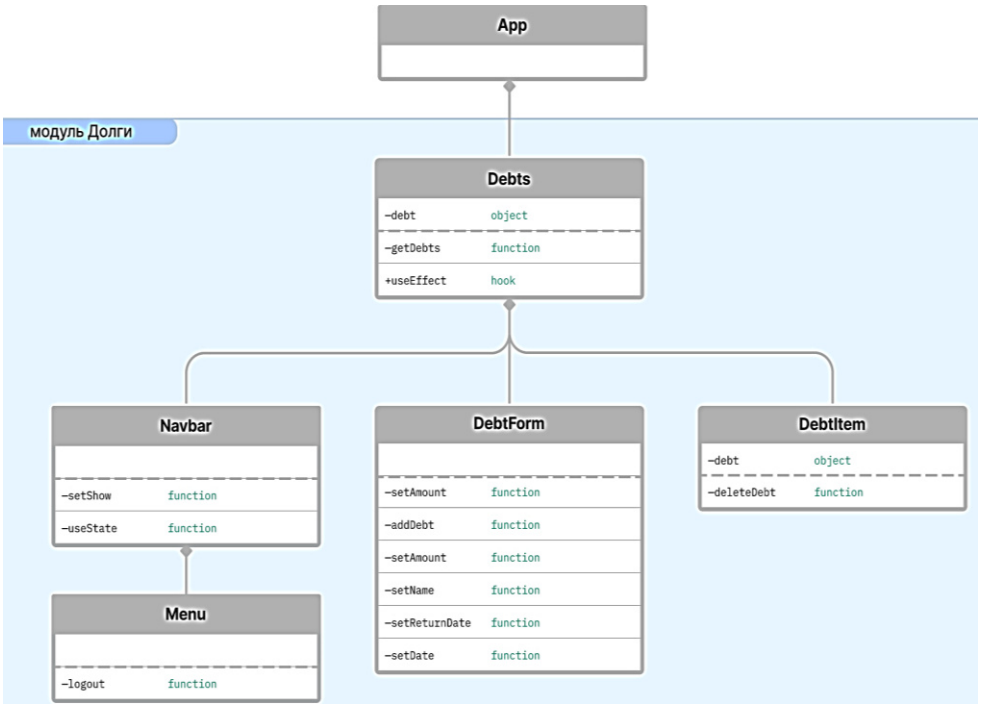

Рисунок 3.6 - Структура модуля «Долги»

Необходимо выделить, что при реализации клиентской части важно учитывать, что для корректной работы пользователя с системой необходимо уделять особое внимание пользовательскому интерфейсу. Например, для мобильных приложений важны такие принципы, как удобство, архитектура информации, дизайн и пользовательские действия. Также при реализации клиентской части важно обеспечивать взаимодействие с сервером, проводить модульное тестирование и учитывать возможности дальнейшего развития. Например, можно расширять функциональность, добавлять новые типы моделей, оптимизировать производительность и обеспечивать безопасность данных [15, с.27].

3.2 Описание интерфейса для связи между клиентом и сервером

Описание клиент-серверной архитектуры подразумевает чёткое распределение ролей: клиенты обращаются за информацией, а серверы отвечают на запросы и выполняют необходимую обработку данных.

Поскольку данное веб-приложение было разработано при помощи Redux-хранилища, в каждом из реализованных компонентов операции с данными реализованы с помощью функций-действий для подготовки данных к отправке на сервер. Для реализации передачи данных между клиентской и серверной частью приложения был разработан специальный API, с помощью которого клиентская часть может подключиться к серверу, используя функции-действия, благодаря чему полученные данные загружаются в Redux-хранилище.

API HTTP - запросов с указанием маршрутов — это коллекция низкоуровневых точек входа в приложение. Они позволяют клиенту работать непосредственно с ресурсами приложения, оставляя решение о том, как представить информацию пользователю, клиенту. Маршруты в API — это URL, к которому можно обратиться разными HTTP-методами. Маршрут может иметь несколько конечных точек (эндпоинтов). Эндпоинт выполняет конкретную задачу, принимает параметры и возвращает данные клиенту. Определим методы HTTP, которые используются для разных операций в процессе разработки веб-приложения:

— GET — для получения (чтения) ресурсов;

— POST — для создания ресурсов;

— POST/PUT/PATCH — для обновления ресурсов;

— DELETE — для удаления ресурсов;

— OPTIONS — для получения полного описания маршрута.

Передача данных реализована при помощи четырех методов: POST, GET, DELETE, PUT. Для осуществления передачи данных была использована JavaScript-библиотека Axios, которая позволяет преобразовывать данные, которые поступили со стороны клиента, в необходимый для осуществления AJAX-запроса JSON-формат.

Для корректного функционирования запросов и ответов сервера были написаны специальные маршруты, благодаря которым сервер понимает куда именно нужно отправить данные. API HTTP - запросов с указанием маршрутов и комментариями представлены в таблице 3.1.

Таблица 3.1 - API HTTP-запросов

Данные	Тип запроса	Адрес	Комментарий
email, password, userid	POST	api/auth	Запрос на авторизацию пользователя
userid	GET	api/auth	Запрос на аутентификацию пользователя
name, email, password	POST	api/users	Запрос на регистрацию пользователя
userid	GET	api/profile	Отправка запроса на получение профиля пользователя
income, savings, days, userid	POST	api/profile	Запрос на изменение информации о пользователе
userid	DELETE	api/profile	Отправка запроса на удаление пользователя по id
housing, rent, transportation, services, subscriptions, userid	PUT	api/profile/ expenses	Запрос на добавление пользовательских данных
amount, comment, date, userid	POST	api/posts	Отправка запроса на создание записи о тратах
userid	GET	api/posts	Запрос на получение записей о тратах
postid	DELETE	api/posts	Отправка запроса на удаление записи по id
amount, comment, date, userid	POST	api/debts	Отправка запроса на создание записи о долгах
userid	GET	api/debts	Запрос на получение записей о долгах
debtid	DELETE	api/debts	Отправка запроса на удаление записи по id

Таким образом, именно методы HTTP помогают при проектировании веб-приложения обеспечивать безопасность, эффективность и надёжность веб-сервисов. Они определяют, какие операции выполняются на сервере: получение информации, отправка форм, обновление или удаление данных.

3.3 Реализация серверной части

Реализация серверной части при проектировании веб-приложения подразумевает динамическую генерацию контента в ответ на запрос пользователя.

Сервер интерпретирует запрос, читает необходимую информацию из базы данных, комбинирует извлечённые данные с шаблонами HTML и возвращает ответ, содержащий сгенерированный HTML. В данном веб-приложении сервер реализован на базе Node.js и фреймворка Express. Подключение к базе данных осуществляется посредством библиотеки Mongoose.

Данная библиотека предназначена для работы с MongoDB и позволяет сопоставлять документы коллекций базы данных с объектами классов. При запуске сервера происходит подключение к базе данных по адресу, указанному в файле config, благодаря чему обеспечивается возможность загрузки и сохранения информации.

Для обеспечения приема запросов от клиента были реализованы следующие модули:

— модуль аутентификации (модуль auth), выполняющий процесс аутентификации, валидации и наличия пользователя;

— модуль пользователь (модуль users), выполняющий процесс добавления нового пользователя с применением написаной схемы данных, шифрования пароля при помощи библиотеки bcrypt, а также присваивания пользователю уникального веб-токена для обращения к базе данных и предоставления доступа только к тем коллекциям, за которыми закреплен токен

пользователя;

— модуль профиль (модуль profile), реализующий функционал предоставления доступа к базе данных для создания, редактирования и удаления настроек пользователя посредством аутентификации токена и обращения к нужному типу запросов, таких как POST, GET, DELETE и вызова в них методов save или remove;

— модуль записи (модуль posts), реализующий доступ к коллекциям в базе данных для создания, редактирования и удаления записей о тратах при помощи написанных для этого маршрутов и типов запросов, таких как POST, GET, DELETE и вызова в них методов save или remove. Код модуля posts представлен в приложении 3;

— модуль долги (модуль debts), реализующий отправку запросов и получение ответов по адресу api/debts, для получения доступа к коллекциям в базе данных для создания, редактирования и удаления записей о долгах.

Реализованные модули подключаются, при помощи внутреннего для фреймворка Express, инструмента Router, который получает соответсвующий адрес запроса и направляет его к нужному модулю.

Модуль auth принимает запросы по адресу /api/auth, модуль users использует /api/users, модуль profile обрабатывает запрос по адресу /api/profile, модуль posts использует /api/posts, а модуль debts принимает запросы по адресу /api/debts.

Исключением является объект, поле которого равно «/», в нем описан путь до корневого файла клиентской части веб-приложения, по пути которого происходит отрисовка компонента Home, предназначенного для первичного запуска приложения.

Значение серверной части в веб-приложении заключается в том, что она:

— поддерживает внешние компоненты. Предоставляет им необходимые данные, обеспечивая эффективное выполнение их функций;

— снабжает веб-приложения необходимыми ресурсами. Помогает поддерживать их общую эффективность и производительность;

— позволяет формировать контент веб-сайта под конкретного пользователя. Динамические сайты могут выделять контент, который более актуален в зависимости от предпочтений и привычек пользователя;

— даёт возможность взаимодействовать с пользователем сайта. Например, посылать уведомления и обновления по электронной почте или по другим каналам.

Диаграмма взаимодействия разрабатываемых функций и подключенных сторонних модулей серверной части разрабатываемого веб-приложения представлена на рисунке 3.7.

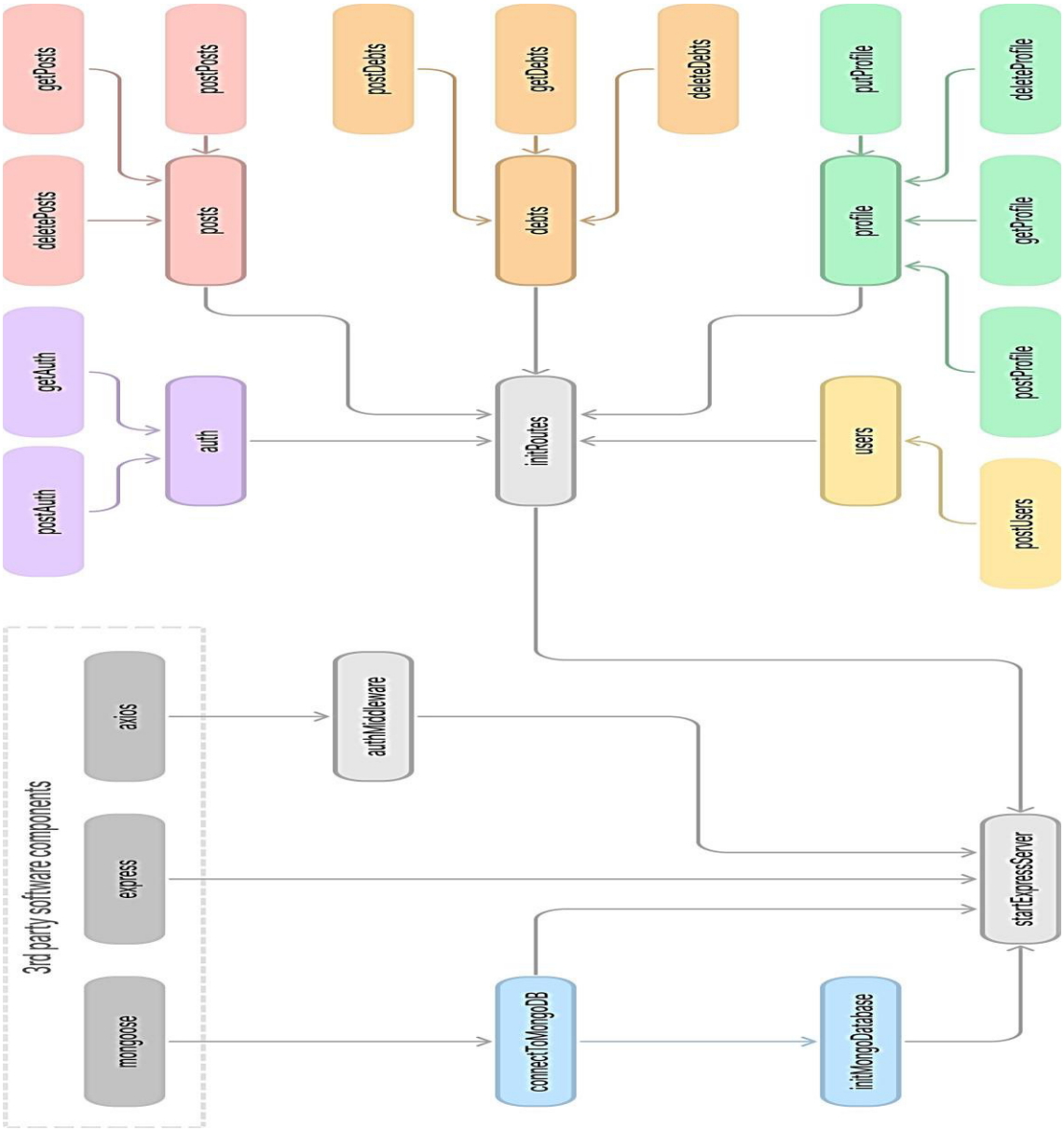

Рисунок 3.7 - Функции серверной части разрабатываемого веб-приложения

Подводя итог реализации серверной части, акцентируем еще раз ее роль в проектировании веб-приложения. Именно она выступает в качестве основы, на которой работают внешние компоненты, снабжая веб-приложения необходимыми ресурсами и помогая поддерживать их общую эффективность и производительность. Несомненно, хорошо спроектированный бэкенд, должен быть масштабируемым, надёжным и безопасным. Это гарантирует, что веб-приложение сможет адаптироваться к возросшим нагрузкам и приспособиться к растущей базе пользователей.

3.4 Тестирование веб-приложения

Тестирование веб-приложения представляет собой процесс проверки работы программы, к которой пользователь получает доступ через веб-браузер по сети. Основная цель заключена в том, чтобы убедиться окончательно, что веб-приложение работает корректно, стабильно и безопасно. Основными задачами тестирования можно считать выявление и устранение ошибок, проверку соответствия требованиям, обеспечение совместимости с различными браузерами и устройствами, проверка производительности под нагрузкой, выявление уязвимостей [19].

При расчёте экономической эффективности тестирования веб-приложения важно учитывать такие факторы, как сложность и объём тестов, расходы на один тест и частота проведения тестирования.

Техническая сторона проекта предполагает анализ разработки транзакций, т.е. процесса, который включает проверку функциональности, производительности и безопасности работы веб-приложения при выполнении транзакций (например, регистрации, входа в систему, оплаты) основными элементами которой являются (рисунок 3.8):

— таблица с описанием операций;
— форма добавления новых транзакций;
— интерактивная диаграмма доходов/расходов.

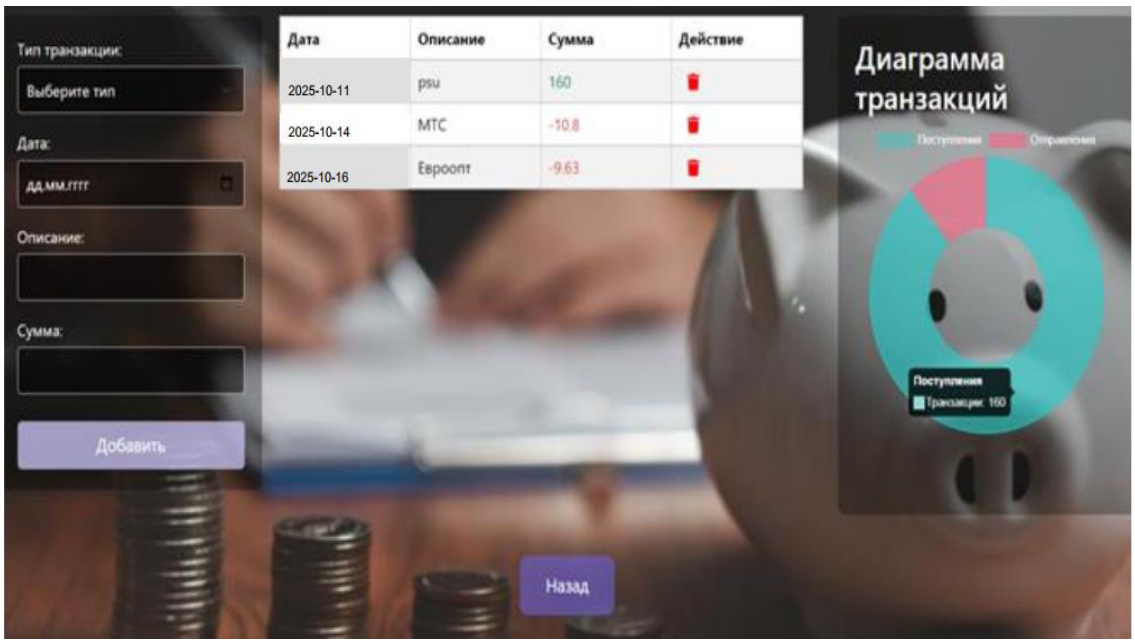

Рисунок 3.8 – Страница транзакции веб-приложения

Следующим элементом стал процесс тестирования управления затратами. Как видно из рисунка 3.9 процесс составляют:

— круговая диаграмма распределения расходов;

— форма добавления финансовых целей;

— возможность удаления записей.

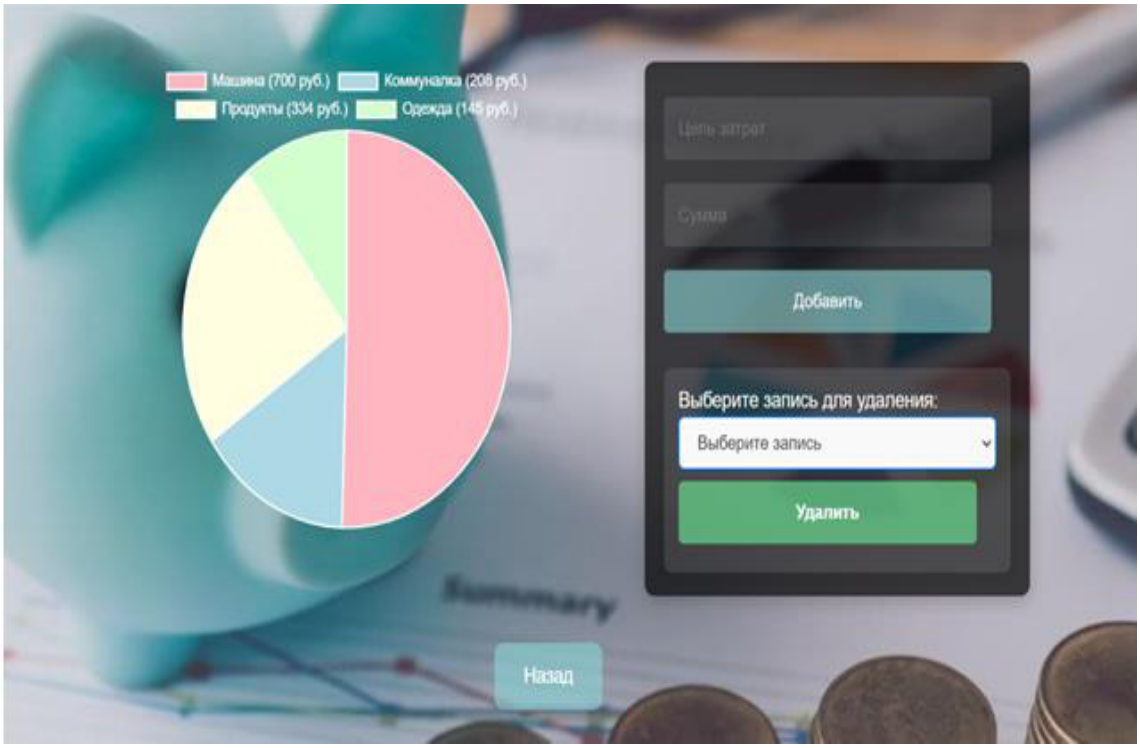

Рисунок 3.9 – Страница управления затратами

Далее было протестировано страницы управления накоплениями. При проектировании веб-приложения, в том числе страницы управления накоплениями, тестирование включает проверку функциональности, удобства использования (юзабилити), безопасности и производительности. Цель — убедиться, что все функции выполняются так, как ожидалось, и интерфейс интуитивно понятен для пользователя. Страница включает (рисунок 3.10):

— список существующих целей накопления;

— форма создания новых накоплений с выбором категории;

— функция добавления средств к существующим обвинениям;

— предварительный просмотр прогресса накопления.

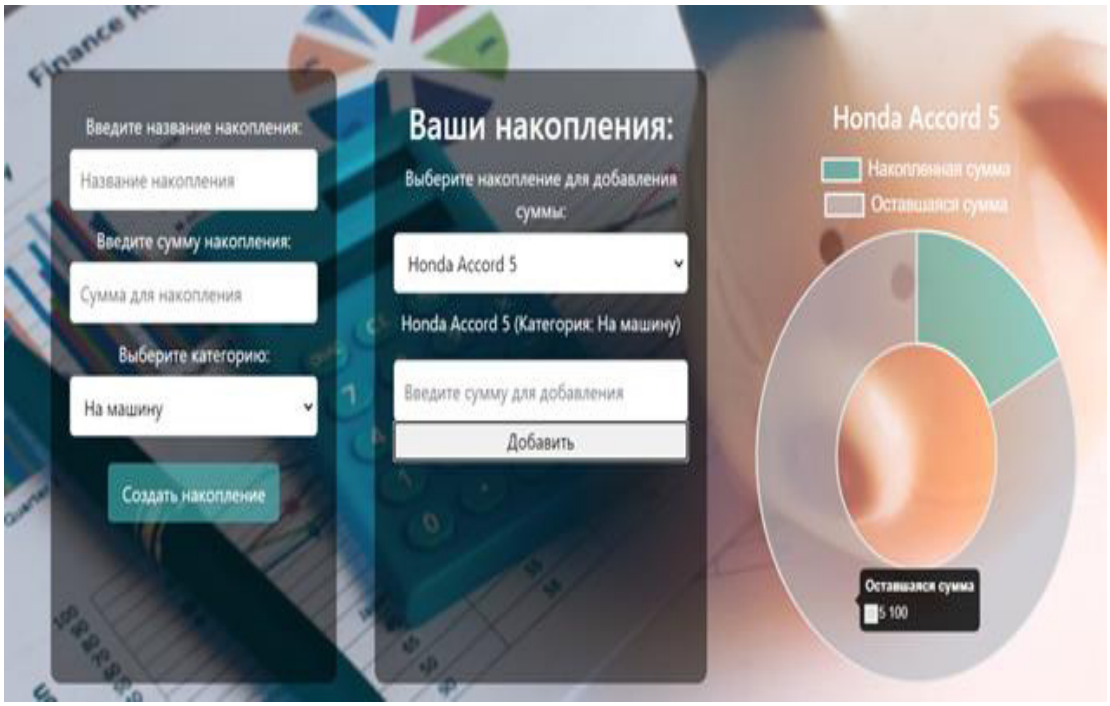

Рисунок 3.10 – Страница управления накоплениями

Что касается особенностей реализации веб-приложения, то основными можно считать следующие позиции:

Компонентный подход: все элементы программы реализованы как основные компоненты React.

Хуки состояния: использование useState для управления локальным состоянием.

Навигация: React Router для клиентской маршрутизации.

Формы проверки: проверка обязательных полей перед отправкой.

Адаптивный дизайн: поддержка различных размеров экрана.

Для проведения тестирования был развернут локальный сервер Node.js с адресом http://localhost/.

Клиентская часть была протестирована посредством запуска в браузерах Google Chromeи Mozilla Firefox. Проверялась возможность подключения к серверу, корректная работа приложения, сохранение, редактирование и удаление данных с помощью панели разработчика в браузере.

Также был протестирован мобильный интерфейс веб-приложения при помощи запуска мобильной симуляции смартфона в Google Chrome. Корректность работы маршрутов и валидации данных была протестирована с помощью Postman.

Тестирование показало успешную работу всех модулей веб-приложения, корректную обработку запросов сервером и доступ к базе данных.

3.5 Расчет показателей экономической эффективности

Экономический эффект от реализации любого информационного продукта (ИТ-проекта) нужен для оценки целесообразности его внедрения и дальнейшего использования. Оценка экономического эффекта помогает:

— оценить технико-экономическое обоснование проекта;

— снизить риски;

— контролировать результат.

Что касается непосредственно дашбордов, то они позволяют сэкономить время на сбор и анализ данных, предоставляя всю критически важную информацию на одном экране. Некоторые факторы, которые способствуют экономии времени при использовании дашбордов [4, с.28]:

— автоматизация процессов. Дашборд автоматизирует сбор и анализ данных, что значительно сокращает время, затрачиваемое на рутинные задачи;

— быстрый доступ к информации. Все данные находятся в одном месте, что упрощает их анализ и управление;

— концентрация ключевых показателей. Дашборд предоставляет обзор самой важной информации, необходимой для принятия обоснованных решений по расходам или вложениям личных средств;

— визуализация данных. В отличие от традиционных отчётов, дашборд не утопает в массивах цифр, а представляет информацию в визуально понятном формате;

— централизация информации. Все необходимые данные собраны в едином месте, что устраняет необходимость переключения между различными файлами или документами.

Чтобы посчитать экономию времени при использовании дашбордов, можно использовать следующие показатели [5, с.69]:

— сокращение времени на сбор и анализ данных. Как правило, до 80% тратится меньше времени на эти процессы, получая всю критически важную информацию на одном экране;

— ускорение процесса принятия решений. Изучение сырой документации может занимать часы, в то время как просмотр ключевых данных на дашборде занимает не больше 10 минут;

— освобождение от изучения многочисленных источников данных. Дашборд освобождает пользователя от таблиц, отчётов, списков, реестров.

Также можно учитывать, что именноавтоматизация отчётов и использование дашбордов позволяет сократить время на их подготовку. По некоторым исследованиям, до 70% времени, ранее затрачиваемого на ручное составление, теперь освобождается для стратегических задач. Экономический эффект от применения дашборда для снижения вероятности допущения ошибок в расчётах и планировании может включать:

— определять точки для оптимизации – именно дашборды позволяют агрегировать данные из различных учётных систем, а также автоматически рассчитывать итоговые показатели;

— визуализировать личные расходы и доходы по разным направлениям;

— осуществлять поиск проблемных зон – отражение ниши, способствующей повышению расходов с целью своевременного исправления возникшей ситуации;

— снижать ошибки в расчётах - автоматизация планирования и бюджетирования помогает избежать ошибок, которые возникают из-за большого объёма ручного труда и многочисленных перепроверок данных.

Автоматизация расчётов обычно требует меньше времени, чем выполнение задач в ручном режиме. Например, при автоматизации расчёта списка трат на месяц экономия времени может достигать до 28 % по сравнению с ручным расчётом.

Существует несколько методик для обоснования экономической эффективности, одна из наиболее популярных - это сопоставление показателей деятельности до внедрения разрабатываемого продукта и тех показателей, которые будут достигнуты в результате использования разработанного программного обеспечения. Именно она и была применена для расчета эффективности описываемого проекта [11, с.98].

При оценке экономической эффективности веб-приложения по управлению личными финансами целесообразно оценить трудовые и стоимостные показатели. Данные показатели дают возможность измерить экономию от внедрения разработанного веб-приложения для обработки бюджетных параметров относительно прежнего варианта, при котором использовались таблицы Exel. В проектном варианте, при внесении данных в разработанное веб- приложение, будет затрачено 2,8 минут, если появляется новая статья доходов или расходов и необходимо добавить новые параметры в базу. Если изменения не вносятся, то 2 минуты. Учитывая, что примерно 52% параметров внесены в базу веб-приложения, среднее время, необходимое на внесение данных будем считать равным 2,4 минуты. В день вносится в среднем информация о 30 параметрах, как расходов, так и поступлений, что в базовом варианте составляет 30 * 4,8 минут = 144 минуты или 2,4 часа.

В проектном варианте это 30 * 2,4 = 72 минуты или 1,2 часа. Рассчитаем абсолютное снижение трудозатрат на одну операцию по формуле [6, с.41]:

$$\Delta T = T_0 - T_1 = 2{,}4 - 1{,}2 = 1{,}2 \quad (3.1)$$

Затем рассчитаем коэффициент относительного снижения трудовых затрат ($К_т$) по формуле [6,с.57]:

$$K_T = (\Delta T/T_0) * 100\ \% = 1{,}2/2{,}4 = 50\% \quad (3.2)$$

Далее рассчитаем индекс снижения трудовых затрат или повышение производительности труда (Y_T), учитывая, что времени на подсчеты вручную заметно снижаются [6,с.83]:

$$Y_T = T_0/T_1 = 2{,}4/1{,}2 = 2 \quad (3.3)$$

Сравнительная оценка величины затрат времени и вероятности абсолютной погрешности при финансовых вычислениях с учетом использования дашборда отражена в таблице 3.2.

Таблица 3.2 – Сравнительная оценка показателей деятельности до внедрения разрабатываемого продукта и после

До применения дашборда	После применения дашборда	Отклонения +/-
Т = 144 мин.	Т= 72 мин.	Т ≈ - 72 мин.
$\Delta = 15$	$\Delta = 3$	$\Delta = - 12$
$З_{канц} = 75$ руб.	$З_{канц} = 0$	$З_{канц} \approx -75$ руб.
$\Sigma = 0{,}18\%$	$\Sigma = 0{,}1\%$	$\Sigma = - 0{,}17\%$

Таким образом, можно увидеть, что применения дашборда по управлению личными финансами способствует уменьшению затрат на канцтовары, минимизации вероятности допустимой ошибки в расчетах и

снижения затрат времени на расчеты личного бюджета. Среднеквадратическое (стандартное) отклонение при расчете наступления риска финансовых операций так же заметно снижается, что еще раз доказывает степень эффективности применения веб-приложения. Рассмотрим вариант реализации веб-приложения «Дашборд управления личными финансами» третьим лицам. На разработку программного продукта потребуется в среднем 3 месяца. Общий объем времени, потраченный на разработку программы, составит 360 часов, если учесть стоимость часа в размере 41,67 руб. в час, то получим общую сумму затрат в размере: $З_{осн}$ = 41,67 * (3 * 120) = 15 001,2 руб.

В таблице 3.3 отразим калькуляцию затрат на разработку программного продукта.

Таблица 3.3 - Калькуляция затрат на разработку программного продукта

Статья затрат	Сумма затрат, руб.
Общая сумма затрат	15 001,2
Накладные расходы, в т.ч.амортизации	293,86
Итого	15 295,06

Рассчитаем плановый уровень прибыли с условием, что процент рентабельности составит 25%.

Прибыль рассчитывается по формуле [25,с.105]:

$$П = \frac{Сполн * Р}{100} \quad (3.4)$$

где, $С_{полн}$ – себестоимость;

Р - процент рентабельности.

Подставив значения получаем: П = (15 295,06* 25)/ 100 = 3 823,77 руб.

Цена программного продукта равна сумме полной себестоимости и прибыли и рассчитывается по формуле [25,с.109]:

$$Ц = Сполн + П \quad (3.5)$$

Используя формулу расчетов, получаем итоговую сумму веб-приложения в размере: Ц = 15 295,06+ 3 823,77= 19 118,83 руб.

Далее рассчитаем размер НДС: 19 118,83* 18 =3 441,39 руб.

Чтобы определить окончательную стоимость программного продукта, применим формулу расчетов по методике Шеремета А.Д.[25,с.211]:

$$\text{Цена} = \text{НДС} + \text{Ц} \quad (3.6)$$

Цена = 19 118,83 + 3 441,39 = 22 560,22 руб.

Таким образом, предполагаемая стоимость программного продукта «Дашборд управления личными финансами» составляет 22 560,22 рублей. На момент завершения проектирования веб-приложения было определено 4 потенциальных покупателя, что в конечном итоге позволит получить дополнительную прибыль от проектирования продукта в размере 90 240,88рублей, что еще раз подчеркивает экономическую эффективность разрабатываемого веб-приложения.

Заключение

В ходе написания выпускной квалификационной работы был изучен теоретический аспект, нацеленный на управление персональными финансами, проанализированы существующие инструменты для этой задачи, а также описано создание собственного инструмента на основе полученных знаний. Среди всех преимуществ проектирования веб-приложения можно выделить его доступность независимо от платформы, отображение расходов в виде диаграммы для простоты восприятия и анализа информации, а также современный и минималистичный дизайн интерфейса. Кроме того, одним из плюсов является абсолютная бесплатность для пользователя, чем не могут похвастать конкуренты.

Для реализации веб-сервера был выбран NPM-пакет Express, поскольку он сочетает в себе скорость и стабильность текущей версии. HTTP-запросы будут обрабатываться посредством Router, что позволяет декомпозировать логику обработки запросов и направлять их в нужный класс в зависимости от того, на какой URL пришел запрос в конкретном случае. Выбран принцип разработки веб-приложения.

Само клиентское приложение было построено по принципу SPA, что позволяет обеспечивать быстроту и гибкость по сравнению со стандартным многостраничным сайтом. В ходе проектирования создана архитектура веб-приложения, а так же смоделирована клиент-серверная архитектура, т.е. модель для построения сетевых приложений, в которой функции разделены между клиентом (пользовательским интерфейсом) и сервером (обработкой данных и логикой).

В ходе разработки программного продукта был выбран язык программирования HTML в качестве описания статической информации, CSS — для стилизации элементов. Фреймворк React в связке с библиотекой Redux в данном веб- приложении служит для обеспечения логики и функциональности приложения. Что касается серверная части приложения, то

она была построена на базе платформы Node.js с использованием NPM-пакетов Express.js и Mongoose. База данных развернута с использованием СУБД MongoDB.

После окончания проектирования веб-приложения был развернут локальный сервер Node.js с адресом http://localhost/ для контрольного тестирования программного продукта. Клиентская часть была протестирована посредством запуска в браузерах Google Chrome и Mozilla Firefox. Проверялась возможность подключения к серверу, корректная работа приложения, сохранение, редактирование и удаление данных с помощью панели разработчика в браузере. Также был протестирован мобильный интерфейс веб-приложения при помощи запуска мобильной симуляции смартфона в Google Chrome. Тестирование показало успешную работу всех модулей веб-приложения, корректную обработку запросов сервером и доступ к базе данных.

В заключении был рассчитан экономический эффект от проектирования веб-приложения «Дашборд управления личными финансами», который показал по ряду вычислений положительную динамику в снижении затрат времени на расчеты и определение статей расходов, уменьшение затрат на канцтовары, исключение вероятности допустимой ошибки в вычислениях.

По итогам завершения проекта веб-приложения «Дашборд управления личными финансами» была произведена оценка его стоимости с учетом НДС. Учитывая возможную реализацию данного продукта третьим лицам позволит получить доход в размере 90 240,88 рублей.

Таким образом, в результате выполнения работы было спроектировано и разработано веб-приложение «Дашборд управление личными финансами», которое позволяет вести ежедневный учёт трат и контролировать текущее состояние долговых операций.

Список литературы

1. Адигеев, М.Г. Жизненный цикл программного обеспечения / М.Г. Адигеев. – Ростов-на-Дону: Изд-во ЮФУ, 2023. – 401 с.

2. Аквино, К., Ганди, Т. Клиентская разработка для профессионалов: Node.js, ES6, REST.- СПб.: Питер, 2021. - 512 с.

3. Бабанов, А.М. Технология разработки программного обеспечения: структурный подход. – Томск: ТГУ, 2025. – 157 с.

4. Балдин, К.В. Информационные системы в экономике. – М.: Дашков и К, 2025. – 395 с.

5. Блюмин, А.М. Экономические расчеты в информатике: учеб.пособие. – М.: Издательско-торговая корпорация «Дашков и Ко», 2025. – 384 с.

6. Варзунов, А.В. Анализ и управление бизнес-процессами: учеб. пособие. – Санкт-Петербург: Университет ИТМО, 2022. – 114 с.

7. Гагарина, Л. Г. Инструменты для создания и реализации приложений: учеб. пособие. – Москва: ИНФРА - М, 2023. - 400 с.

8. Гвоздева, В.А., Ворфелева, Е.Н., Неорович, И.С. Информатика, автоматизированные информационные технологии и системы: учеб. - М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2025. – 544 с.

9. Долженко, А.И. Разработка React. – Ростов-на-Дону: Изд-во РГУ, 2023. – 191 с.

10. Изучение React. [Электронный ресурс] – URL: https://habrahabr.ru. (дата обращения: 09.09.2025).

11. Котляров, В.П. Методы расчета экономической эффективности программных продуктов. - Москва: ИНТУИТ, 2024. – 335с.

12. Кумагина, Е.А. Модели жизненного цикла и технологии проектирования программного обеспечения. – Нижний Новгород: изд-во ННГУ, 2022. – 157 с.

13. Коцюба, И.Ю. Структура клиентского приложения: учеб. пособие.– СПб: Университет ИТМО, 2025. – 206 с.

14. Луб, Е.Н. Разработка веб-приложений в ReactJS. - М.: ДМК Пресс, 2023. - 254 с.

15. Моделирование информационных систем: учеб. пособие / по ред. Лисяк, В.В.. – Ростов-наДону: Изд- во Южного федерального университета, 2022. – 88 с.

16. Радченко, М.Г., Невершестов, Р.О. Интерактивные дашборды и приложения с Plotly и Dash. - М.: 1С-Паблишинг, 2022. - 194 с.

17. Рудинский, И.Д. Технология проектирования автоматизированных систем обработки информации и управления: учеб. пособие / И. Д. Рудинский. – М.: Горячая линия - Телеком, 2024. - 304 с.

18. Рязанцева, Н.Е. CASE-технологии. Современные методы и средства проектирования информационных систем. - М.:БХВ-Петербург, 2022. - 694 с.

19. Тестирование информационных систем. [Электронный ресурс] – http://services/testing/testirovanie-sistem. (дата обращения: 10.11.2025).

20. Томас, М.Т. React в действии. – Санкт-Петербург: Издательский Дом Питер. – 368 с.

21. Тюгашев, А.А. Основы программирования веб-приложений. – СПб.: Университет ИТМО, 2025. – 160 с.

22. Финансовые приложения для учета личного бюджета. [Электронный ресурс]– http:// blog_897/prilozheniya-dlya-uchota-finansov. (дата обращения 29.08.2025).

23. Чистякова, В.И. Проектирование информационных систем: учеб. пособие для бакалавров. – М.: Академия, 2025. – 301 с.

24. Чистов, Д. В. Проектирование информационных систем. – М.: Юрайт, 2023. – 260 с.

25. Шеремет, А.Д. Методика финансового анализа: учеб. пособие. - 2- е изд., перераб. и доп. – М.: Наука, 2023. - 518 с.

26. Функциональное моделирование информационных процессов. [Электронный ресурс] – URL: http://www./designing/methodology _for_bp.html. (дата обращения 18.08.2025).

27. Филатова, В.И. Базы данных. - М.:Сфера, 2022. - 256 с.

28. Харитонов, С.А., Пронь, И.И. Архитектура информационных систем: учеб.пособие для вузов. - М.:1С-Паблишинг,2024. - 682 с.

29. Шарина, А.А. Языки программирования. – М.: Технологии будущего, 2023. - 292 с.

30. Языки программирования высокого уровня. [Электронный ресурс] URL: https://ru.wikipedia.org (дата обращения 20.09.2025).

Приложение 1

Описание бизнес-прецедента

Название прецедента: войти в приложение.

Цель сценария: войти в приложение.

Предусловие: открыта страница входа в приложение.

Основной сценарий:

Ввести адрес электронной почты;

Ввести логин;

Ввести пароль;

Повторить пароль;

Нажать кнопку «Войти».

Постусловие: после проверки введенных данных, если такой учетной записи еще не существует, в базу данных будет добавлен новый аккаунт пользователя и произойдет регистрация. Откроется «основная страница приложения». Если же такие данные уже присутствуют в базе данных, происходит авторизация пользователя.

Название прецедента: первичная настройка.

Цель сценария: первичная настройка приложения.

Предусловие: зарегистрирован новый пользователь, открыта страница первичной настройки.

Основной сценарий:

Ввести ежемесячный доход;

Ввести желаемый процент на остаток в конце месяца;

Нажать кнопку «Продолжить».

Постусловие: введенные пользователем данные сохраняются в базе данных, рассчитывается рекомендуемая сумма на день. Затем откроется «основная страница приложения». Если же введенные данные содержат ошибку или заполнены не все обязательные поля, пользователя оповещают об этом посредством окна ошибки.

Название прецедента: добавить запись о тратах.

Цель сценария: добавить запись о тратах.

Предусловие: открыта основная страница приложения. Основной сценарий:

Ввести сумму траты;

Ввести комментарий на что была потрачена данная сумма;

Нажать кнопку «Добавить».

Постусловие: запись о трате добавлена в базу данных, на основной странице приложения появилась добавленная запись. Если же введенные данные содержат ошибку или заполнены не все обязательные поля, пользователя оповещают об этом посредством окна ошибки, запись не добавляется в базу данных.

Название прецедента: редактировать запись о тратах.

Цель сценария: редактировать запись о тратах.

Предусловие: открыта основная страница приложения, список записей не пуст.

Основной сценарий:

Нажать на существующую запись, откроется окно редактирования записи.

Редактировать необходимые поля записи.

Нажать кнопку «Сохранить».

Постусловие: при правильном редактировании поля запись о трате будет изменена в базе данных, на основной странице приложения появилась отредактированная запись. Если же измененные данные содержат ошибку, пользователя оповещают об этом посредством окна ошибки, отредактированная запись не обновляется в базе данных.

Название прецедента: удалить запись о тратах.

Цель сценария: удалить запись о тратах.

Предусловие: открыта основная страница приложения, список записей не пуст.

Основной сценарий:

Выбрать нужную запись.

Нажать на кнопку «Удалить», которая присутствует для каждой записи в виде значка «Корзина».

Постусловие: выбранная запись будет удалена из базы данных и исчезнет с основной страницы приложения.

Название прецедента: просмотр истории записей.

Цель сценария: просмотр истории записей.

Предусловие: открыта основная страница приложения, список записей не пуст.

Основной сценарий:

Нажать на кнопку «Далее», откроется окно истории записей.

Постусловие: открыта страница просмотра истории записей, на которой отображаются все записи данного пользователя.

Название прецедента: добавить запись о долгах.

Цель сценария: добавить запись о долгах.

Предусловие: открыта страница записей о долгах.

Основной сценарий:

Ввести сумму долга.

Ввести комментарий кому или от кого данная сумма.

Выбрать временной промежуток для долга.

Нажать кнопку «Добавить».

Постусловие: запись о долге добавлена в базу данных, на странице записей о долгах появилась добавленная запись. На основной странице приложения обновится общая сумма долга. Если же введенные данные содержат ошибку или заполнены не все обязательные поля, пользователя оповещают об этом посредством окна ошибки, запись не добавляется в базу данных.

Название прецедента: редактировать запись о долгах.

Цель сценария: редактировать запись о долгах.

Предусловие: открыта страница записей о долгах, список записей не пуст.

Основной сценарий:

Нажать на существующую запись, откроется окно редактирования записи.

Редактировать необходимые поля записи.

Нажать кнопку «Сохранить».

Постусловие: при правильном редактировании поля запись о долге будет изменена в базе данных, на странице записей о долгах появилась отредактированная запись. Если в процессе редактирования была изменена сумма долга, на основной странице приложения обновится общая сумма долга. Если же измененные данные содержат ошибку, пользователя оповещают об этом посредством окна ошибки, отредактированная запись не обновляется в базе данных.

Название прецедента: удалить запись о долгах.

Цель сценария: удалить запись о долгах.

Предусловие: открыта страница записей о долгах, список записей не пуст.

Основной сценарий:

Выбрать нужную запись.

Нажать на кнопку «Удалить», которая присутствует для каждой записи в виде значка «Крестик».

Постусловие: выбранная запись будет удалена из базы данных, исчезнет со страницы записей о долгах. На основной странице приложения обновится общая сумма долга.

Название прецедента: распорядиться остатком.

Цель сценария: распорядиться остатком.

Предусловие: открыта основная страница приложения, список записей не пуст.

Основной сценарий:

Нажать на сумму остатка, откроется окно управления остатком.

Выбрать из двух опций: перенести остаток на следующий период или потратить завтра.

Постусловие: в зависимости от выбора пользователя, остаток либо перенесен на следующий период, либо перенесен на завтрашний день. Если перенесли на завтрашний день, тогда на основной странице приложения обновится дневная сумма. Если перенесли на следующий период, тогда на основной странице приложения обновится общая сумма.

Приложение 2

Компонент Signup

```
import React, { Fragment, useState } from 'react'; import { connect } from 'react-redux';
import { Link, Navigate } from 'react-router-dom'; import { setAlert } from '../../actions/alert';
import { register } from '../../actions/auth';
import PropTypes from 'prop-types';
const Signup = ({ setAlert, register, isAuthenticated }) => { const [formData, setFormData] = useState({
name: '',
email: '',
password: '',
password2: '',
});
const { name, email, password, password2 } = formData;
const onChange = (e) =>
setFormData({ ...formData, [e.target.name]: e.target.value });
const onSubmit = async (e) => { e.preventDefault();
if (password !== password2) { setAlert('пароли не совпадают', 'danger');
} else {
register({ name, email, password });
}
// переадресация в случае успешной регистрации
if (isAuthenticated) {
```

```
return <Navigate to='/main' />;
}
return (
<Fragment>
<div  className='backwards____wrapper
backwards__login'>
<Link to='/' className='btn__backwards '>
<i  class='fa-solid  fa-angle-left  backwards
icon'></i>
<p  className='backwards_____text  text
link'>назад</p>
</Link>
</div>
<main className='signup'>
<form className='signup__form' onSubmit={(e) => onSubmit(e)}>
<div className='signup__wrap'>
<input
type='email'
placeholder='email'
className='login__input  text
input' name='email'
value={email}
onChange={(e) => onChange(e)}
pattern='[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,4}$'
/>
<input type='text'
placeholder='имя'
className='login_input text_input' name='name'
value={name}
onChange={(e) => onChange(e)} minLength='3'
```

```
maxLength='20'
/>
<input type='password' placeholder='пароль'
className='login_input text_input' name='password'
value={password} onChange={(e) => onChange(e)} minLength='6'
/>
<input
type='password' placeholder='повторите пароль' className='login__input text input' name='password2' value={password2}
onChange={(e) => onChange(e)} minLength='6'
/>
<button className='btn text__button signup__submit' type='submit'>
зарегистрироваться
</button>
</div>
<div className='signup__help'>
<p className='text__body'>
уже зарегистрированы?
<br />
<Link className='text__link' to='/login'>
войдите
</Link>
</p>
</div>
</form>
</main>
</Fragment>
);
};
Signup.propTypes = {
```

```
    setAlert: PropTypes.func.isRequired, register: PropTypes.func.isRequired,
isAuthenticated: PropTypes.bool,
    };
    const mapStateToProps = (state) => ({ isAuthenticated:
state.auth.isAuthenticated,
    });
export default connect (mapStateToProps, { setAlert, register })(Signup);
```

Приложение 3

Модуль posts

```
const express = require('express'); const router = express.Router();
const { check, validationResult } = require('express-validator'); const auth =
require('../../middleware/auth');
const Post = require('../../models/Post'); const User = require('../../models/User');
const checkObjectId = require('../../middleware/checkObjectId');
router.post( '/',
[auth, check('amount', 'введите сумму').not().isEmpty()], async (req, res) => {
const errors = validationResult(req); if (!errors.isEmpty()) {
return res.status(400).json({ errors: errors.array() });
}
try {
const user = await User.findById(req.user.id).select('-password');
const newPost = new Post({ amount: req.body.amount, comment:
req.body.comment, date: req.body.date,
user: req.user.id,
});
const post = await newPost.save();
res.json(post);
} catch (error) { console.error(error.message); res.status(500).send ('ошибка
сервера');
}
}
);
module.exports = router;
router.get('/', auth, async (req, res) => { try {
req.user.id;
```

```
const posts = await Post.find({ user: req.user.id }).sort({ date: -1 });
res.json(posts);
} catch (err) {  console.error(err.message); res.status(500).send ('ошибка сервера');
}
});
router.delete('/:id', [auth, checkObjectId('id')], async (req, res) => { try {
const post = await Post.findById (req.params.id);
if (!post) {
return res.status(404).json({ msg: 'Post not found' });
}
// check user
if (post.user.toString() !== req.user.id) {
return res.status(401).json({ msg: 'User not authorized' });
}
await post.remove();
res.json({ msg: 'Post removed' });
} catch (err) { console.error(err.message);
res.status(500).send('Server Error');
}
});
```