Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись инициалы, фамилия
« 12 » 06 2021 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01- “Информатика и вычислительная техника”
код и наименование специальности

Web-приложение совместный органайзер
тема

Руководитель	11.06.21 подпись, дата	доцент, канд. техн. наук должность, ученая степень	С.Н. Титовский инициалы, фамилия
Выпускник	10.06 подпись, дата		М.Б. Субботкин инициалы, фамилия
Нормоконтролер	11.06.21 подпись, дата	доцент, канд. техн. наук должность, ученая степень	С.Н. Титовский инициалы, фамилия

Красноярск 2021

РЕФЕРАТ

Выпускная квалификационная работа по теме: «Web-приложение совместный органайзер» / ИКИТ СФУ, Красноярск, 2021. Содержит 65 страниц, 19 рисунков, 5 таблиц , 16 источников.

Ключевые слова: WEB-ПРИЛОЖЕНИЕ, JavaScript, TypeScript, Redux, React, Node.js, MongoDB, Express.

Цель: разработка веб-приложения совместного органайзера, функциональными составляющими которого являются, совместный календарь событий с возможностью создания и редактирования событий несколькими пользователями, а так-же возможность создания и редактирования общих и личных списков задач.

Задачи: обзор предметной области, изучение аналогов, разработка клиентской части приложения, разработка серверной части приложения.

В результате выполнения работы будет спроектировано и реализовано web-приложение, с возможностью создания событий в календаре событий, и возможностью создания списка общих и личных задач.

СОДЕРЖАНИЕ

ВЕДЕНИЕ

С каждым новым днем технологии все больше и больше интегрируются в нашу жизнь, захватывая все новые и новые сферы деятельности. На заре развития информационных технологий и технологий связанных с ЭВМ, они применялись в очень узких сферах таких как оборона и научная деятельность. И с каждым новым днем они внедрялись во все более и более бытовые сферы жизни человека. Сегодня мы живем в мире - где даже самые рядовые вещи тесно связанны с технологиями. Большой рывок данной тенденции дало развитие интернета как, доступной и удобной среды для разработки и распространения сервисов с разнообразным функционалом, от сервисов для мониторинга погоды до аналогов классических десктопных приложений.

Но повсеместная автоматизация процессов не дала человеку расслабиться, а только лишь увеличила темп жизни и количество ежедневных задач человека.

Что создало потребность в доступном инструменте, для систематизации персональной и коллективной деятельности.

Для облегчения этой задачи и были созданы органайзеры. Органайзер- это приложение для фиксирования задач, событий и дел, и предоставление информации в удобном виде[1]. Они помогают контролировать выполнения поставленных задач. В большинство современных приложений с подобным функционалом в основе стоят два модуля, это календарь событий, модуль приложения стилизованный под календарь или же в виде списка с подписанными датами, который необходим для создания задачи привязанной к определенной дате, и список дел, список бессрочных задач, так же существуют разные вариации таких списков с разбиением на модули и подзадачи. Так как человек иногда должен выполнять какую-либо деятельность внутри коллектива, то появляется потребность в приложениях с поддержкой совместного использования.

Несмотря на большое количество подобных приложений, каждое приложение отличается как функционалом так и дизайном интерфейса. Поэтому необходимо провести анализ и сравнение с основными функциями, которыми должна обладать приложение.

1 Анализ предметной области

1.1Обзор аналогов

1.1.1 Google Calendar

Приложение[2] имеет удобный интуитивно понятный интерфейс рис.1.1. Функционал приложения включает в себя, создание задач трех типов встреча, задача и напоминание. Все они привязаны к определенной дате с возможностью настройки периода повторения, задача отличается от заметки наличием описания. Встреча имеет дополнительно поля ссылку на “Google Meet” и местоположение. Так же интерфейс позволяет фильтровать выдачу задач по времени: день, неделя, месяц, год, расписание (все задачи идут последовательно), выборка на 4 дня. Данное приложение не предоставляет интерфейс группового взаимодействия по умолчанию, но календарем встреч можно делиться м пользователями.

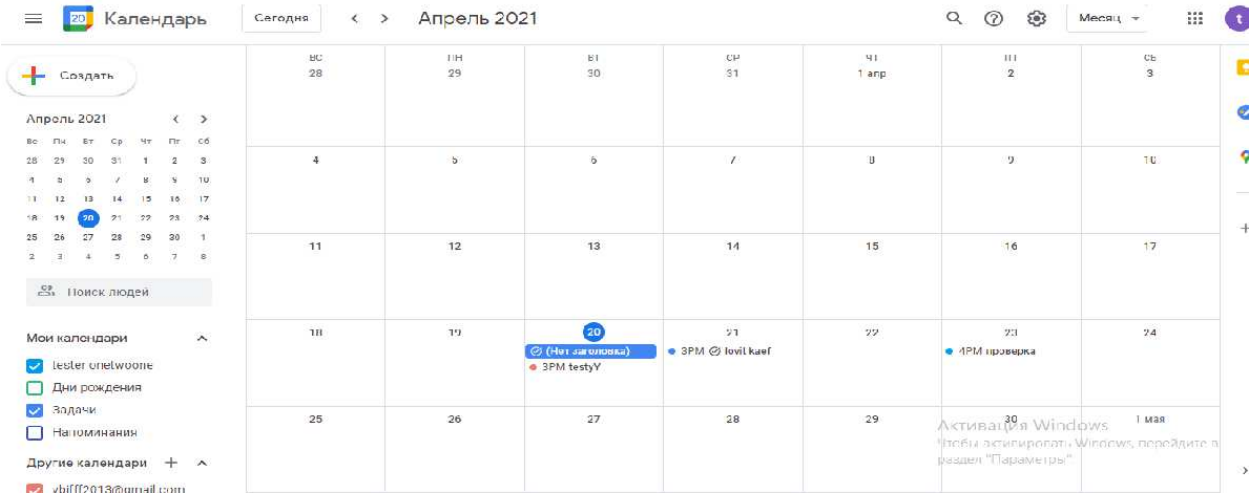

Рисунок 1.1- интерфейс web версии приложения Google Calendar

Минусом данного приложения можно посчитать, регистрация только с помощью аккаунта Google, а так же запутанный интерфейс настройки.

1.1.2Coiz Familiy Oraganaizer

Приложение обладает мобильной и веб версией[3]. При первом входе в приложение необходимо зарегистрироваться. У приложения есть функционал такой как, добавление задач в календарь событий, создание списков задач и создание списка покупок, и возможность загрузки фотографий в альбом, еще можно создавать персональные задачи или скрывать задачи от какого-либо пользователя. События в календаре состоит из поля названия задачи, поля установки времени от и до, с возможностью установки задачи на каждый день, поле установки периода повторения задачи, список пользователей для которых предназначена эта задача, поле для записи адреса, установка оповещения и поле описания задачи. Список

дел имеет стандартный вид, создается список, список имеет одно поле это его название, и добавления задач, состоящих тоже из одного поля это само название задачи .

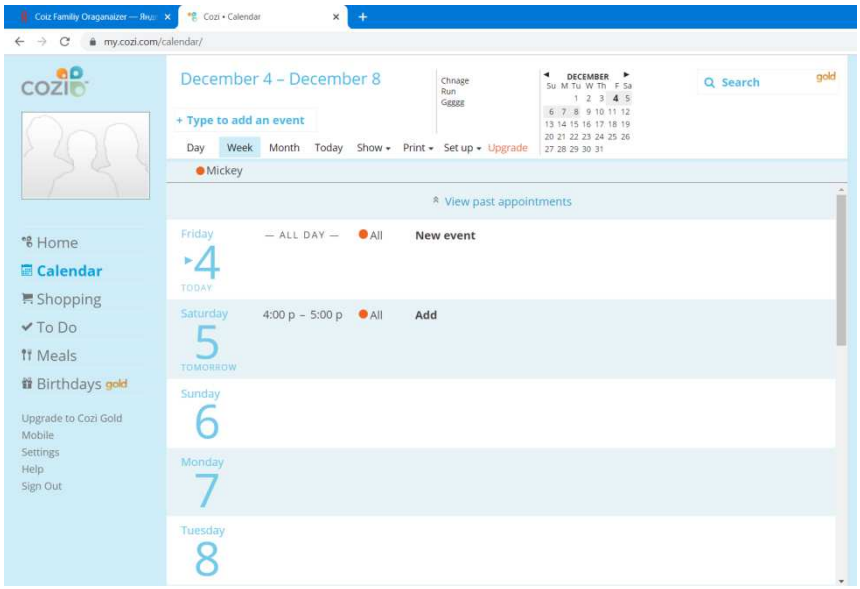

Рисунок 2- интерфейс Coiz Familiy Oraganaizer.

1.1.3 Weeek

Приложение так же имеющие web-версию и версию для смартфонов[4]. Данное приложение является календарем проектов и имеет более сложный функционал чем у других аналогов рассмотренных в этой главе. Пользователь может создавать вроксспэйсы в которые можно приглашать пользователей по email. Вокрспэйс делиться на две рабочие части это задачи и доски.

В окне задач можно создать проект, и в него добавлять задачи привязанные к дате, так же можно заполнить описание задачи и ее название, так же есть функция делегирования задачи участнику воркспэйса. Интерфейс досок прост и интуитивно понятен в списке “доски” можно создать доску и при нажатии на нее откроется интерфейс работы с ней, он состоит из колонок куда можно добавлять задачи и модифицировать их изменяя название описание или делегируя их.

Минусами можно считать сложный интерфейс задач и интерфейс добавления пользователя в команду.

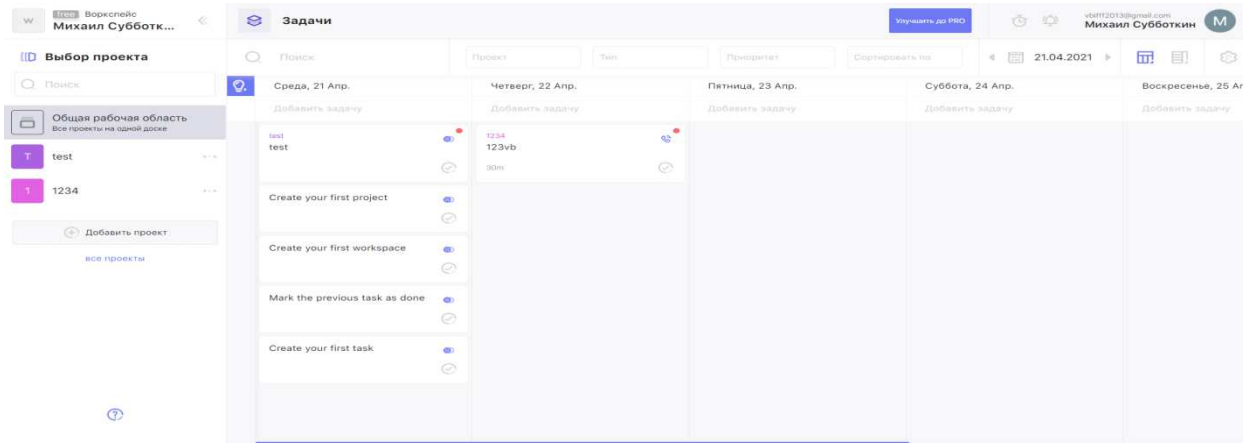

Рисунок 3 – интерфейс Weeek

1.2Анализ рассмотренных ресурсов

В ходе обзора были проанализированы существующие программные продукты и составлена таблица. Основными направлением сравнения аналогов является их функционал, интерфейс, удобство совместного пользования. Результаты анализа аналогов приведены в таблице 1.1.

Таблица 1.1-Сравнительная таблица аналогов

	Google Calendar	Coiz Familiy Oraganaizer	Week
Функционал			
календарь событий	+	+	+
список дел	-	+	+
Удобство совместного использование			
Процедура добавление нового пользователя	-	+	-
Возможность изменения прав доступа к задачам	+	+	-
Интерфейс			
перегруженность интерфейса	-	+	+
Выразителность	+	-	+

1.3Цель создания приложения

Целью разработки является, приложение обладающие календарем событий, списком дел, удобством совместного пользования и интуитивно понятным интерфейсом.

Система должна удовлетворять следующим требованиям

-Интерфейс программы должен быть интуитивно понятен

-Программа должна быть отказа устойчивой

-Данные о задачах, пользователях и группах хранятся в базе данных.

1.4Функциональные возможности

В приложении будут присутствовать три функциональных модуля, календарь событий, в который каждый пользователь может добавлять события. События для календаря должны будут иметь следующие поля: название, описание, дата, период повторения, список пользователей которым будет видна задача. Модуль списков задач, должен будет обладать следующим функционалом, создание списка, то есть необходимо будет записать название списка и указать список участников группы которые будут видеть этот список, возможность добавления задач в список. Третьим модулем является групповой чат. Телеграмм бот будет иметь две команды, первая будет привязывать бота, к аккаунту, а вторая будет отправлять сообщение в приложение.

1.5 Вывод по главе

В данной главе был произведен обзор и анализ аналогов данного приложения, на основании анализа были поставлены цели создания данного приложения, а так же был описан функционал, который необходимо будет представить в данном приложении.

2 Проектирование

2.1 Структура приложения

2.1.1 Архитектура приложения

На стороне клиента находится пользовательский интерфейс, его “состояние” и логика его отрисовки. На сервере расположена бизнес-логика и логика взаимодействия с хранилищем данных[5].

Приложение можно спроектировать как многостраничное или одностраничное.

Многостраничное приложение работает следующим образом, каждый блок приложения является отдельной страницей, и при переходе между ними клиент запрашивает у сервера необходимый, HTML, JavaScript, CSS, это позволяет достичь более комфортной навигации по приложению, но это ухудшает пользовательский опыт, так как при использовании приложения страница браузер полностью перезагружается. Этот подход к проектированию хорошо себя показывает при разработки информационного ресурса с большим количеством данных с которыми ненужно активно взаимодействовать.

Web-приложение спроектированное как одностраничное приложение выгружает HTML, JavaScript и CSS единожды на сторону клиента и динамически подгружает по мере необходимости данные с сервера[6], данный подход улучшает пользовательский опыт, так как отсутствует перезагрузка страницы при изменении состояния. При использовании такого подхода к проектированию приложения, серверная часть может значительно уменьшится в своем объеме и сложности, что приведет к снижение нагрузки на сервер, так как часть логики будет выполняться на стороне клиента.

Данное приложение будет спроектировано как одностраничное, так как подразумевается активное переключение между информационными блоками, и перезагрузка страницы, ухудшит пользовательский опыт.

2.1.2 Взаимодействие клиента и сервера

Коммуникация между клиентом и сервером во всем приложении должна происходить в едином стиле за исключением модуля чата, так как там

необходимо передача данных со стороны сервера по их готовности. Для этого можно применить Long Polling, WebSocket и SSE.

Первый из методов поддержки отправки данных по их готовности это Long Polling. Суть данного метода состоит в том что на сервер отправляется http запрос на получение данных и соединение не разрывается пока у сервера не будет необходимых данных или же выйдет тайм-аут. Данный подход хорошо показывает себя при относительно редких случаях передачи данных, но в чате данные могут передаваться с большой интенсивностью, а так же проблемы с порядком получения данных.

WebSocket это протокол поверх TCP/IP позволяющий установить двухсторонние соединение клиента и сервера. JavaScript имеет удобный встроенный API для работы с сокетами в браузере.

SSE(Server Side Events)- это спецификация, при которой клиент может настроить соединение с сервером на основе http, минусом данного подхода является односторонние соединение, по нему только сервер может отправлять данные.

Таким образом лучшим выбором будет WebSocket так как, он не создает большой нагрузки на сервер, имеет удобный API и взаимодействие клиента и сервера будут иметь одинаковую структуру, что облегчит разработку приложения.

Остальная часть клиента будет взаимодействовать с сервером, по протоколу Http.

2.1.3 Архитектура клиентской части

Интерфейс клиентской части приложения будет разбит на компоненты, для снижения сложности кода, облегчения последующего сопровождения и дополнения приложения. Структура интерфейса представлена на рис.2.1

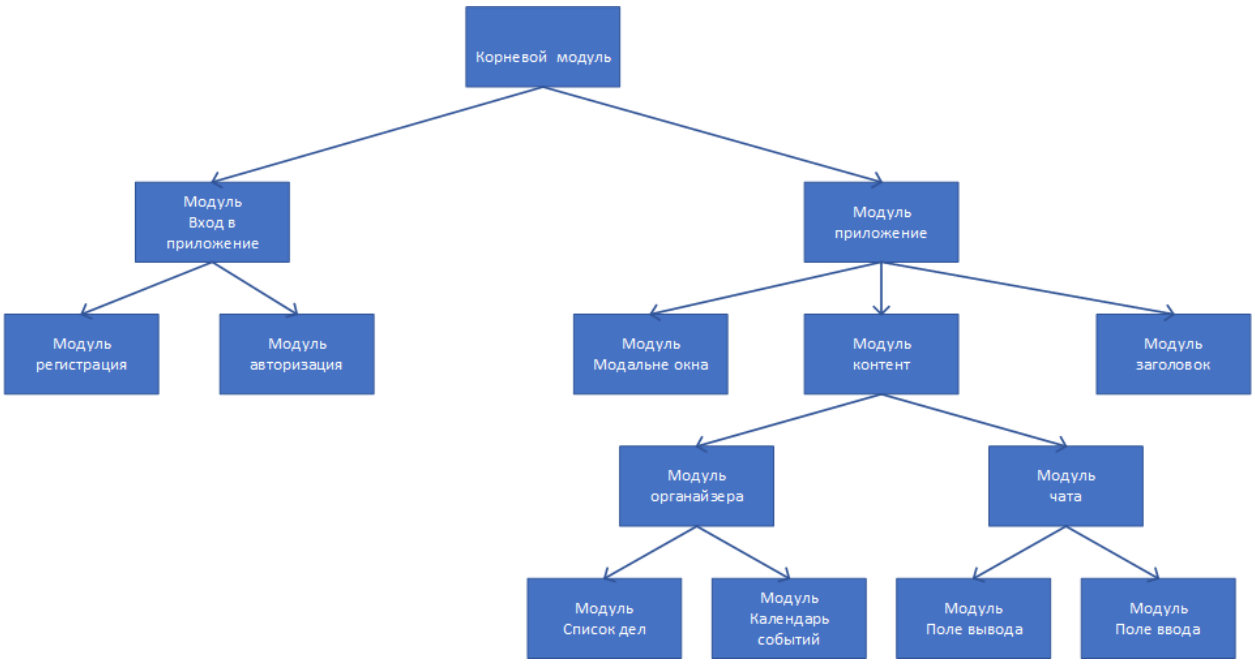

Рисунок 2.1– схема интерфейса клиентской части приложения.

Некоторые компоненты должны реагировать на изменения состояния других компонентов или состояния приложения в целом, для этого будет применен шаблон “Наблюдатель”. При использовании данного шаблона проектирования компоненты приложения могут “подписываться” на другие модули, которые в свою очередь при изменении своего состояния рассылают подписчикам информацию. При этом данный паттерн обеспечивает низкую связность между модулями[8].

Для удобной работы с состоянием приложения, будет использован “менеджер” состояния приложения.

2.1.4 Архитектура серверной части

Бизнес-логика приложения не имеет ресурсоёмких задач или каких-либо долгих процедур. Преимущественно вся бизнес-логику состоит из чтения и записи данных в хранилище по каким-либо правилам, за редким исключением при которых, сервер должен еще обработать полученные данные, это задачи по типу валидации данных полученных от клиента или от базы данных. Следовательно сервер должен выполнять быстро такие задачи как чтение и запись.

2.1.5 Архитектура базы данных

Специфика приложения подводит к выводу что данные, с которыми будет оно работать, имеют простую структуру, маленькую связность между собой, данные не будут часто обновляться. Таким образом при выборе между NoSQL и SQL базами данных. Выгоднее будет использовать NoSQL. В таких базах данных используется JSON-подобный формат хранения. Структура документов базы данных описана в таблице 2.1.

Таблица 2.1 - структура документов в базе данных.

<table>
<tr><th>Название документа</th><th>Ключи</th><th colspan="4">Данные</th></tr>
<tr><td rowspan="7">Room</td><td>Users</td><td colspan="4">массив id</td></tr>
<tr><td rowspan="4">Chat</td><td rowspan="4">массив объектов</td><td colspan="2">Ключ</td><td>Значение</td></tr>
<tr><td colspan="2">date</td><td>String</td></tr>
<tr><td colspan="2">loginUser</td><td>Stirng</td></tr>
<tr><td colspan="2">Msg</td><td>String</td></tr>
<tr><td>Tasks</td><td colspan="4">массив id</td></tr>
<tr><td>Todos</td><td colspan="4">массив id</td></tr>
<tr><td rowspan="7">Task</td><td>Title</td><td colspan="4">Stirng</td></tr>
<tr><td>Description</td><td colspan="4">Stirng</td></tr>
<tr><td>Date</td><td colspan="4">Stirng</td></tr>
<tr><td>Time</td><td colspan="4">Stirng</td></tr>
<tr><td rowspan="2">Users</td><td rowspan="2">Object</td><td colspan="2">Ключ</td><td>Значение</td></tr>
<tr><td colspan="2">Логин пользователя</td><td>Bool</td></tr>
<tr><td>Period</td><td colspan="4">Stirng</td></tr>
<tr><td>Room</td><td colspan="4">id комнаты</td></tr>
<tr><td rowspan="5">Todo</td><td>Title</td><td colspan="4">Stirng</td></tr>
<tr><td rowspan="3">Tasks</td><td rowspan="3">массив объектов</td><td>Ключ</td><td>Значение</td></tr>
<tr><td>name</td><td>String</td></tr>
<tr><td>value</td><td>Bool</td></tr>
<tr><td rowspan="2">Users</td><td rowspan="2">Объект</td><td>Ключ</td><td>Значение</td></tr>
<tr><td>Логин пользователя</td><td>Bool</td></tr>
<tr><td>Room</td><td colspan="4">Id комнаты</td></tr>
<tr><td rowspan="4">User</td><td>Email</td><td colspan="4">Stirng</td></tr>
<tr><td>Login</td><td colspan="4">Stirng</td></tr>
<tr><td>Password</td><td colspan="4">Stirng</td></tr>
<tr><td>Room</td><td colspan="4">Id комнаты</td></tr>
</table>

2.2 Выбор технологий разработки

2.2.1 Пользовательский интерфейс

Для разработки пользовательского интерфейса можно отменить три технологии Vue.js, Angular и React.

Все три технологии практически идентичны по своим техническим характеристикам и функциональным возможностям и при создании не большого приложения, нет никакой разницы какую технологию использовать. У этих технологий имеется различия только в сложности и “экосистеме”.

Vue.js- это фреймворк с маленьким порогом входа, позволяющий быстро начать разработку, но его минусом является “экосистема”, он имеет куда меньшее распространение, относительно других технологий, а поэтому имеет меньший объем доступной информации и меньшее количество утилит для работы с ним и более сложную интеграцию с другими технологиями.

Angular фреймворк с куда более мощной поддержкой и большим количеством информации, но он обладает высоким порогом входа и более сложной структурой приложения, что не имеет смысл при разработки данного приложения.

React это библиотека которая обладает маленьким порогом вхождения, большим сообществом сформировавшимся вокруг данной технологии, а так же поддержкой разработчика, активно развивающего данную технологию, и ее “экосистему”.

Для разработки пользовательского интерфейса будет использоваться библиотека React. Эта библиотека была выбрана мной так как у меня был опыт практического применения. React представляет возможность разработки одностраничных приложений на основе компонентов, что позволяет произвести декомпозицию разметки и логики описывающей работу интерфейса[9], так как React позволяет инкапсулировать разметку и логику, так же он позволяет поддерживать уровень абстракции при работе с компонентой, не отвлекаясь на реализацию вложенных компонентов. В свою очередь компоненты описываются с пошью JSX. JSX -это расширение синтаксиса языка JavaScript с помощью которого можно вставлять код в разметку, что повышает удобства разработки.

Так же React представляет хуки это удобный инструмент для работы с состоянием приложения, а так-же построение пользовательских хуков

облегчает поддержу кода и опять же увеличивает уровень абстракции компонента[10].

2.2.2 Управление приложением

Для управления состоянием приложения можно использовать такие технологии как MobX, Redux и React Context API.

MobX это библиотека контроля состояния приложения с маленьким порогом входа, позволяющая менять и получать состояние приложения с помощью геттеров и сеттеров. Она не имеет строгих правил ее использования.

Redux это библиотека основана на Flux-архитектуре и обязывает разработчика следовать строгому шаблону, для работы с ней, что приводит к сильному разрастанию приложения. Но взамен библиотека предлагает развитый интерфейс взаимодействия с react-приложениями и большое количество утилит облегчающих разработку, а так же удобство отладки и тестирование приложения.

React Context AP это набор react-хуков позволяющих управлять состоянием приложения, это не отдельная библиотека, так что интеграция в приложения не вызовет никаких сложностей, но минусами являются сложность работы с приложением, которое имеет большое состояние, и малое количество информации.

Я выбрал Redux так как он предоставляет хорошо описанную документацию и утилиты для удобной отладки приложения. Это предсказуемый и централизованный контейнер состояния[11]. Принцип работы данной библиотеки следующий, все данные находятся в “состоянии” это можно представить как объект, далее если необходимо что-то изменить, оправляется действие, действие- это объект у которого есть обязательное поле “type” так же могут быть другие поля. Изменение состояния происходит в функции “редукторе”, в которую передается текущие состояние и действие.

Так как Redux не поддерживает асинхронность, а она является необходимым свойством клиент-серверного приложения с отзывчивым интерфейсом. Для решение этой проблемы необходимо использование “redux-thunk” этот “middleware” позволяет отправлять действие с задержкой или при выполнении какого-либо условия[12].

2.2.3 Среда исполнения

Серверная часть приложения может быть реализованна с помощью Python, PHP, Node.js.

PHP- это веб-ориентированный язык программирования. Он обладает огромным сообществом, и в связи с этим большим объемом документации и прочей информации, хорошо налажено взаимодействие с SQL базами данных. Написание кода на PHP сопровождается быстротой его написания и быстротой развертывания. Минусом данного языка является его блокирующий ввод/вывод, эти операции для данного приложения являются основными и то что они будут нагружать сервер, является заметным минусом.

.NET – это фреймворк, обладающий большим сообществом, а следовательно большим количеством информации и пакетов помогающих в разработке. Значимым плюсом является высокая производительность, но он является синхронным языком с блокирующими операциями ввода/вывода.

Node.js- это среда выполнения JavaScript[13]. Node.js обладает большим сообществом и большим количеством пакетом способных заметно уменьшить время затраченное на разработку серверной части приложения. Этот язык является асинхронным, что позволяет ему быстро выполнять операции ввода/вывода, а следовательно и в целом ускорить данное приложение. Так же так как клиентская часть будет написана на JavaScript, то это заметно сэкономит время на разработку приложения.

Таким образом оптимальным будет выбор Node.js.

На стороне сервера будет взаимодействия большого числа классов между собой, что при использовании JavaScript может привести к трудно уловимым ошибкам, из-за не строгой типизации. В свою очередь TypeScript частично разбирается с этой проблемой, так же TypeScript позволяет использовать интерфейсы и классы в более очевидной и понятной форме.

Этот язык программирования, компилируется в JavaScript, тем самым позволяя использовать пакеты написанные на JavaScript и совмещать оба этих языка. Так же он предоставляет удобное представление классов и интерфейсов.

Express- это пакет предназначенный для создание серверов веб-приложений, с минималистичным API, из-за своей легковесности он слабо влияет на производительность приложения[14].Так же является одним из самых старых пакетов для разработки и поэтому очень хорошо описан и

задокументирован. Его минусом является отсутствия рекомендованного метода организации кода, что увеличивает по времени процесс проектирования серверной части приложения на основе Express.

Hapi- тоже предназначен для создание веб-сервера, но имеет более сложный API, но присутствует поддержка мощной системой плагинов которая ускоряет разработку и уменьшает затраты на сопровождение и доработку уже готового продукта. Он в отличии от того же Express, он заметно проигрывает в производительности.

Koa- является продуктом разработанным командой разработчиков Express, который является улучшенной версий Express, он является более производительным и с более понятной структурой кода. Но его минусом является его относительная молодость и нестабильность, что усложняет его интеграцию с другими пакетами.

Таким образом для веб-сервера выбран пакет Express так как он сочетает скорость и стабильность.

HTTP запросы будут обрабатываться при помощи “Router” это поможет декомпозировать логику обработки запросов в разные классы в зависимости от url на который пришел запрос. Принцип обработки запросов WebSocket отличается от HTTP запросов тем что экземпляр WebSocket инициализированный на сервере “слушает” единственный url на который присылаются запросы и уже логика обработки запроса зависит от типа запроса.

2.2.4 СУБД

Redis- это база данных хранящая данные в виде ключ-значение, отличается высокой гибкостью, скоростью и маштабироемостью.

MongoDB- это NoSQL база данных в которой данные хранятся JSON-образном формате[15]. Была выбрана данная база данных так как она предоставляет сервер базы данных с удобным web-приложением, с помощью которого легко управлять базой данных, так же MonogBD представляет подробную документацию по своему API, а так же главным удобством является возможность написания запроса на JavaScript. На стороне сервера взаимодействие с базой данных происходит с помощью пакета “mongoose” которые предоставляет удобный интерфейс по созданию документов и запросов к базе данных[16].

MongoDB была выбрана в связи с опытом работы с этой базой данных.

2.3 Функциональность приложения

Необходимые функциональные возможности приложения иллюстрируются диаграммой, приведенной на рис. 2.2

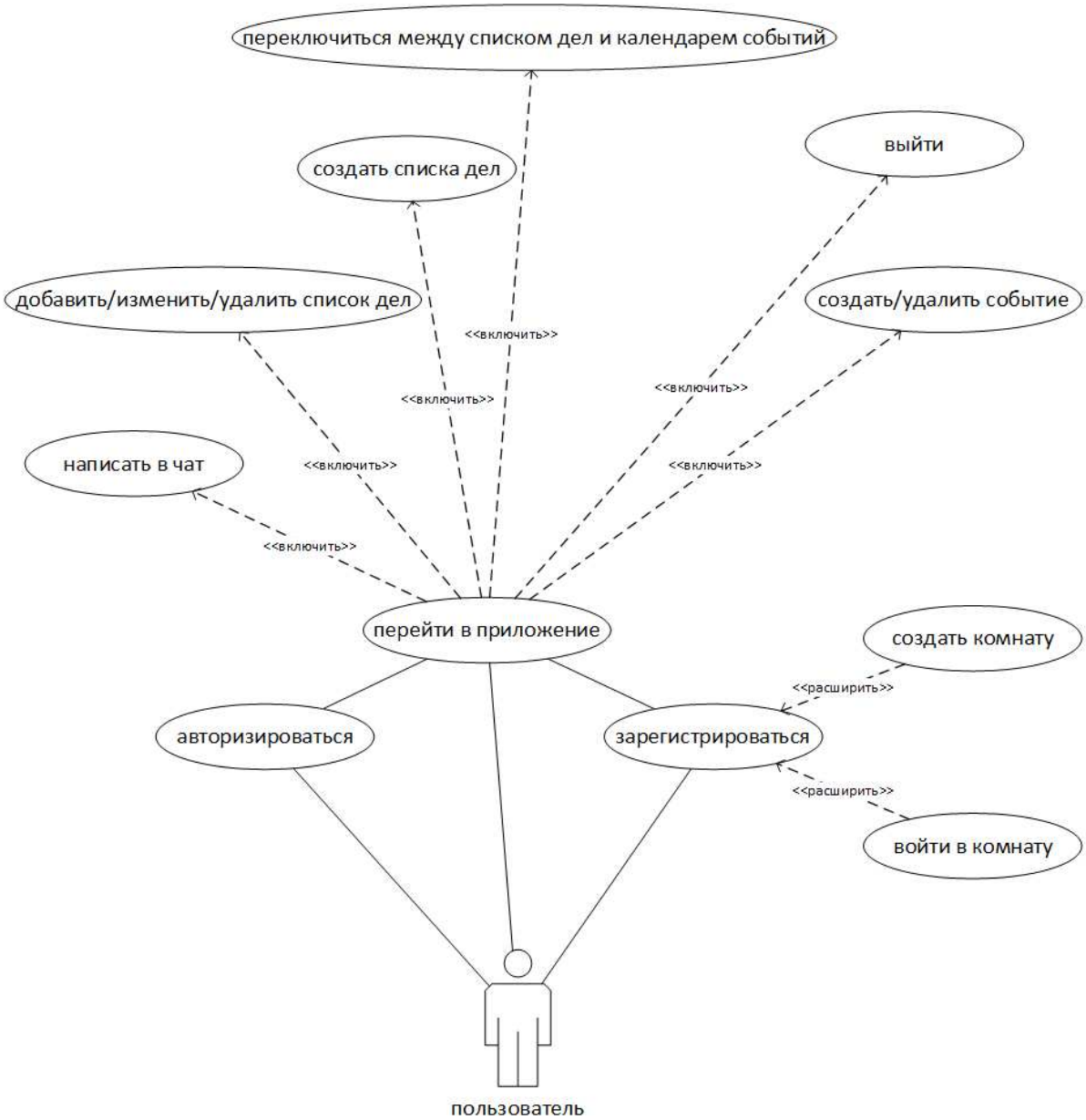

Рисунок 2.2- Диаграмма прецендентов.

Логика переходов следующая при входе в приложение пользователь если уже авторизировался в браузере, то он переходит сразу в приложение, если нет он может либо авторизироваться либо зарегистрироваться. В приложении можно переключиться между списком дел и календарем событий, а так же создать, удалить или добавить что-то.

Функциональные возможности рис2.2 приложения реализуются модулями:

- Модуль входа
- Модуль заголовка
- Модуль “Календарь событий”
- Модуль “списка дел”
- Модуль чат
- Модуль модальные окна

2.3.1 Модуль входа

При первом входе в приложение пользователя перекидывает на окно регистрации в нем можно зарегистрироваться и создать новую комнату либо войти в уже существующею, так же можно перейти в модуль авторизации и ввести данные уже зарегистрированного пользователя. При вводе корректных пользователя перекидывает в окно с самим приложением и записываются данные в локальное хранилище браузера для запоминания пользователя.

2.3.2 Модуль заголовка

В модуле заголовка располагается кнопка выхода из приложения и id комнаты, которое нужно вводить при регистрации что бы попасть в уже созданную комнату. При выходе данные из локального хранилища стираются.

2.3.3 Модуль “календарь событий”

В модули “календарь событий” можно создавать события, при нажатии на кнопку “+” всплывет модальное окно в котором можно будет заполнить поля, при нажатии на кнопку “OK”, данные из полей отправятся на сервер где их проверят и при корректности данных их сохранят в базе данных и вернут клиенту обновленный список событий. Так же при нажатии на кнопку “сменить вид” изменится вид отображения событий либо в формате месяца либо в формате недели. Так же можно менять список отображаемых

событий изменяя дату, с помощью двух кнопок уменьшая дату на месяц или недели или же увеличивая на тот же срок. Если календарь находиться в формате месяца, то при нажатии на ячейку дня, всплывет модальное окно в котором будут отображены события записанные на этот день, рядом с ними будет кнопка удаления события, при нажатии на названия события всплывет модальное окно с описание “события”. У календаря событий в форме недели схожий интерфейс, при одном различии в том что список событий на день отображется сразу же. Так же в из модуля “Календарь событий” можно перейти в список дел по нажатию на кнопку.

2.3.4 Модуль “списки дел”

В модуле “списки дел” , при нажатии на кнопку “+” всплывет модальное окно в котором можно будет заполнить поля, при нажатии на кнопку “OK”, данные из полей отправятся на сервер где их проверят и при корректности данных их сохранят в базе данных и вернут клиенту обновленный список дел. При наведении на список дел всплывают две кнопки “удалить” и кнопка “+”, при нажатии на кнопку удалить, список дел удаляется из базы данных, и сервер отправляет обновленный список клиенту, при нажатии на кнопку “+”, всплывает модальное окно, с аналогичным алгоритмом работы, при успешном обработке на сервере, возвращается обновленный список дел с добавленным в него делом. Также при нажатии на дело его можно пометить как выполненное(название дела будет зачеркнуто) или наоборот пометить как не выполненое.

2.3.5 Модуль чат

Модуль чат- это единственный модуль в приложении который изменяется в реальном времени. При написании другим участником комнаты сообщения оно автоматически отправиться всем участникам комнаты, и отобразиться в окне чата, так же пользователь может сам написать в окне набора сообщений.

2.3.6 Модуль модальные окна

Этот модуль используется почти во всех сценариях использования приложения. При рендере этого окна также блокируется интерфейс остальной части программы. Сам интерфейс модального окна зависит, от

контекста его вызова, при нажатии на кнопку “+” в режиме “календарь событий” вызывается редактор с полями:

-название

-описание

-дата

-время

-период повторения

-поле выбора пользователей которым будет видно это событие.

Так же в окне есть кнопка “OK” которая отправляет, данные на сервер, и поле куда выводятся ошибки если пользователь ввел некорректные данные.

При нажатии на эту же кнопку в режиме “список дел ” вызовет редактор с полями:

-название

- поле выбора пользователей которым будет видно это событие.

Остальной интерфейс аналогичен выше описанному. Еще модальное окно с редактором всплывает при нажатии на кнопку “+” на самом списке дел. В этом редакторе есть только одно поле это название.

Еще один тип модального окна- это окно которое отображает список “событий” на конкретный день.

Последним типом окна является окно отображения данных о “событии” в нем находятся поя которые нельзя редактировать это поля:

-название

-описание

-время

-период повторения.

2.3.7 Telegram бот

У бота будет команда “reg” вместе с которой записывается email и пароль, она привязывает чат к определенному пользователю. Команда “send” отсылает сообщение из телеграмм бота на сервер, где оно будет отправляться другим клиентам.

2.4 Вывод по проектированию

В результате выполнения второй главы была разработана архитектура клиентской, серверной частей и структура базы данных, был выбран стек технологий и паттерны проектирования применяемый при разработке web-приложения. А так же описан функционал приложения и его интерфейс, которые будут реализованы в web-приложении.

3 Программная реализация приложения

3.1 Реализация клиентской части

Клиент имеет древовидную структуру и состоит из множества компонентов отвечающих за определенную часть UI и логику отрисовки ее. Компоненты можно объединить в модули, в зависимости от выполняемой ими функции. Структура модулей приведена на рис.3.1.

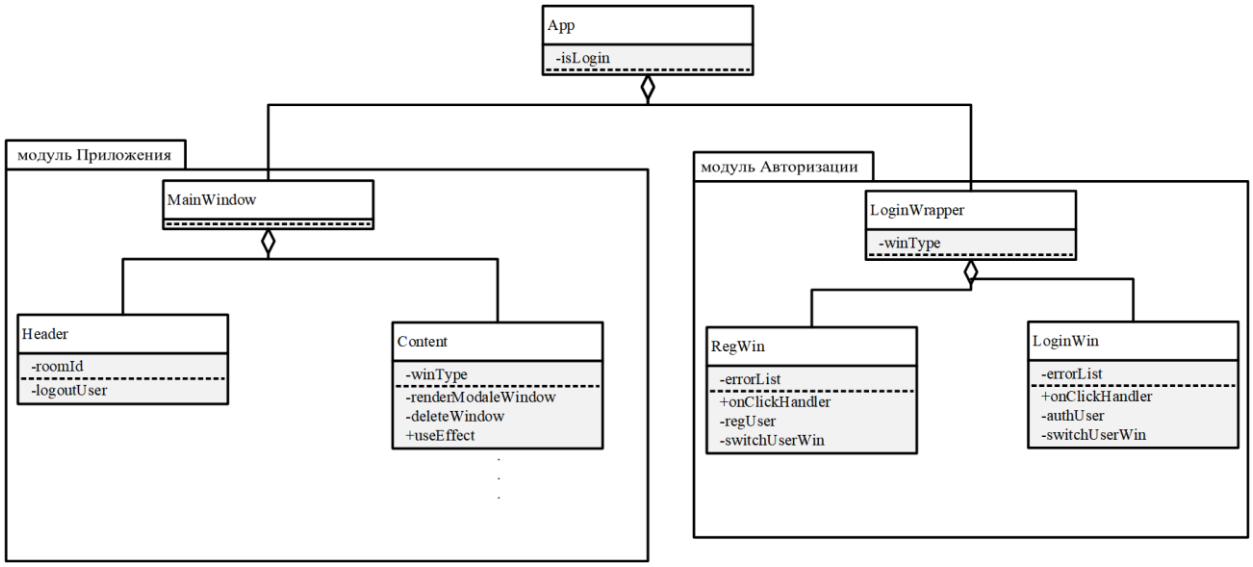

Рисунок 3.1-структура клиентской части приложения

Точкой входа в приложение является корневой модуль, который отвечает за объединение всех остальных модулей в себе.

Модуль входа отвечает за окна регистрации и авторизации пользователя и логику обработки данных.

Модуль приложения отвечает за основной функционал работы приложения. Эти два модуля являются вложенными в корневой модуль.

Модуль чат отвечает за создание интерфейса ввода и вывода сообщений.

На том же уровне вложенности расположен модуль главное окно, его задачей является отрисовка интерфейса органайзера. Модули список дел, календарь событий, кнопки меню, модальные окна являются вложенными в модуль главное окно, и отвечают за интерфейсы и логику работы: списка дел, календаря событий кнопок меню и модальных окон соответственно.

3.1.1 Корневой модуль

Корневой модуль состоит из одного компонента, которым является компонент App. Дочерними элементами являются два компонента MainWindow и LoginWinWrapper. Эти компоненты отрисовываются в зависимости от того авторизирован ли пользователь.

3.1.2 Модуль входа

Модуль входа состоит из трех основных компонентов это компонент LoginWrapper, RegWin, LoginWin. Первый компонент отвечает за логику отрисовки компонентов в зависимости от состояния приложения. По умолчанию переменная состояния winType равна “REG”, что приводит к отрисовке RegWin, если winType равна “AUTH”, то рендериться LoginWin.

Компонент RegWin отвечает за UI и логику работы регистрации. Компонент состоит из двух кнопок “создать новую комнату”, “войти в комнату”, полей “e-mail”, “логин”, “пароль”, “повторить пароль”, “ID комнаты”, и кнопки стилизованной под текст “Авторизироваться”. При нажатии на кнопку “создать новую комнату” данные из полей компонуются (поле ID комнаты игнорируется) в объект “data” и передаются в “действие” regUser. При нажатии “войти в комнату” алгоритм аналогичный только в regUser передается объект с двумя полями “data” и “room” данные прописанные в поле ID комнаты. Кнопка “Авторизироваться” вызывает “действие” switchUserWin с аргументов “AUTH”.

Компонент LoginWin имеет два поля “E-mail” и “пароль”, две кнопки “назад” и “OK”. Кнопка “назад” вызывает “действие” switchUserWin с аргументов “REG”. Кнопка “OK” вызывает “действие” authUser с аргументов в виде объекта с данными из формы.

Так же оба компонента обладают полем вывода ошибки, из переменной состояния errorList.

Диаграмма состояний данного модуля представлена на рис.3.2

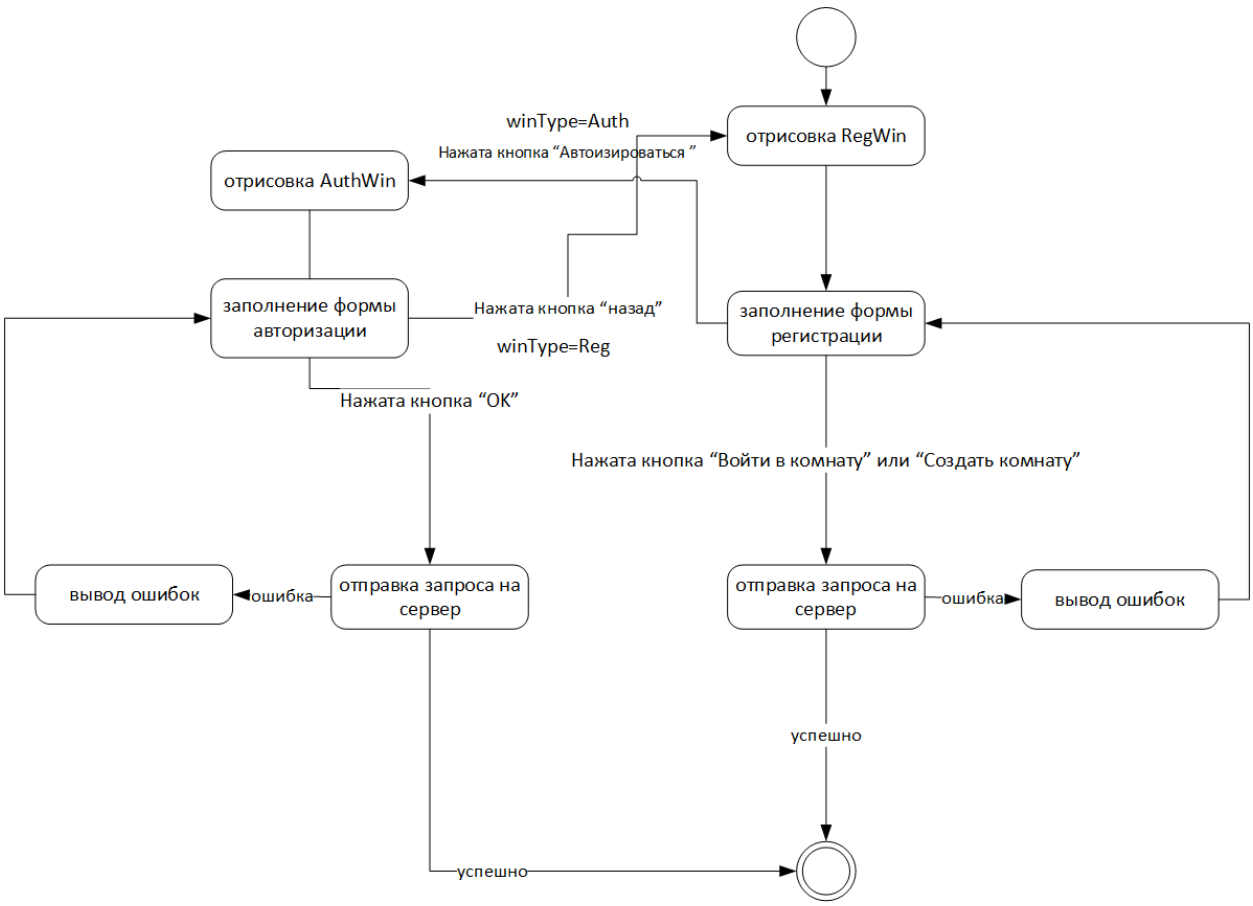

Рисунок 3.2-Диаграмма состояний модуля входа

3.1.3 Модуль приложения

Модуль приложения-это самый большой модуль на стороне клиента он состоит из нескольких подмодулей (рис.3.3). Точкой входа в этот модуль является компонент MainWindow, он отрисовывает два компонента это Header и Content.

Компонент Header имеет небольшой функционал, это кнопка “выйти”, которая вызывает “действие” logOutUser, и отображение переменной состояния roomId.

Компонент Content отвечает за отрисовку двух компонентов MainBlock и ChatBlock, так же он отвечает за отрисовку модального окна, в зависимости от переменной состояния winType, это достигается за счет двух хуков стандартного хука useEffect(отвечает за вызов функций отрисовки и удаления, при изменении переменной winType) и кастомного хука useModaleWin(он содержит функции отрисовки модального окна относительно переменной winType и функцию удаления модального окна).

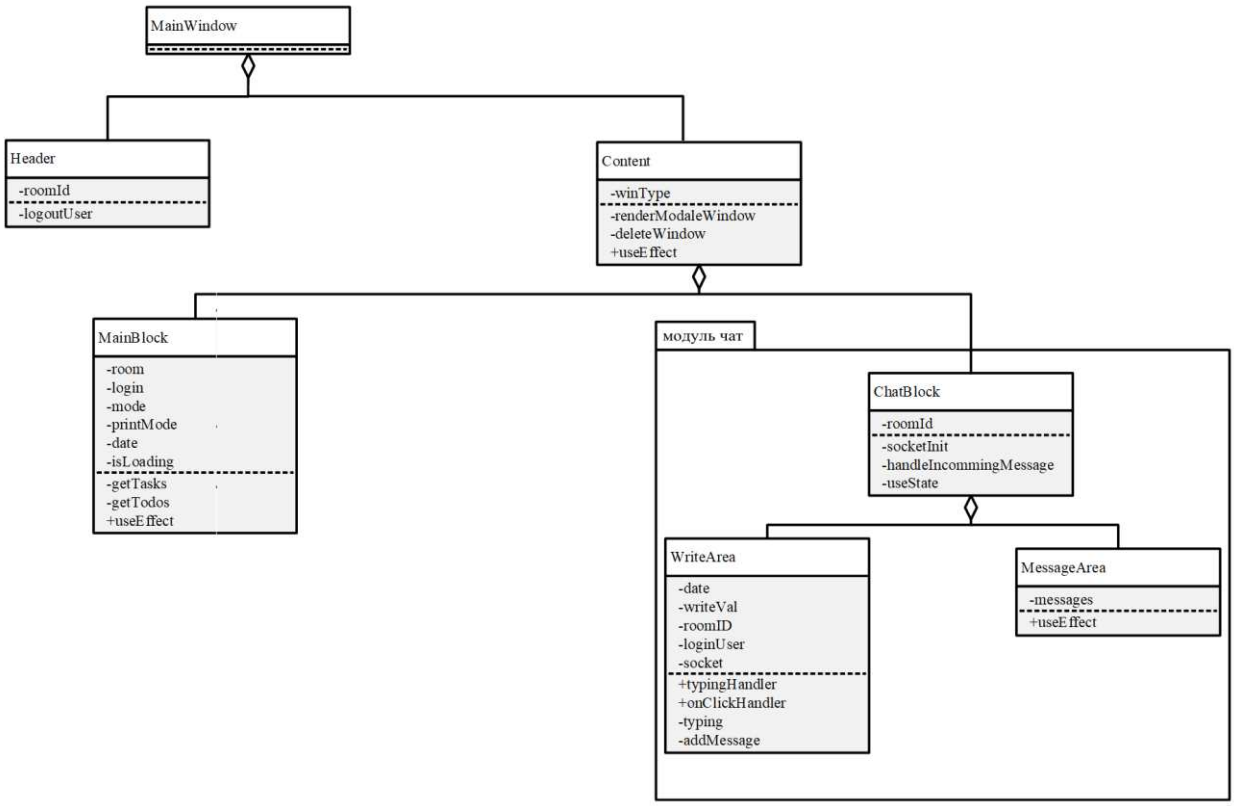

Рисунок 3.3- структура модуля приложения

3.1.4 Модуль чат

Модуль чат является подмодулем приложения. Данный модуль состоит из трех компонентов это компонент “обертка” ChatBlock, WriteArea, MessageArea. ChatBlock является родительским компонентом для двух других, так же при первом рендере компонента вызывается действие socketInit с аргументами переменной состояния roomId и обработчиком события, которое подключиться к созданному сокету, как обработчик пришедшего сообщения, этим обработчиком является вызов другого действия handleIncomingMessage. Для того что бы вызывать socketInit были использованы два хука- это useState(он использовался для создания флага isFirstRender и функции setIsFirstRender, которая заменяла значение флага) и хук useEffect(в нем была прописана логика работы при первом рендере компоненты).

WriteArea этот компонент отвечает за обработку и отправку сообщений, и состоит он из текстового поля ввода и кнопки “отправить”. На текстовом поле весит обработчик onChange, который вызывает отправку “действия”

typing, которое в свою очередь приводит к изменению переменной состояния writeVal, которая вставляется как текст в это поле. Это может понадобиться для облегчения валидации, если она понадобится в дальнейшем. При нажатии на копку “отправить” вызывается “действие” addMessage в аргументы которого передаются переменная состояния socket и объект содержащий в себе переменные состояния wrtieVal, toomId, loginUser, date.

MessageArea компонент, который отвечает за отображение сообщений хранящихся в переменной состояния messages, которая является массивом. При первом рендере компоненты, message с помощью метода “map” каждый элемент массива передается в компоненту Message, которая является конструктором одного сообщения. На рис 3.4 представлена диаграмма состояний модуля чат.

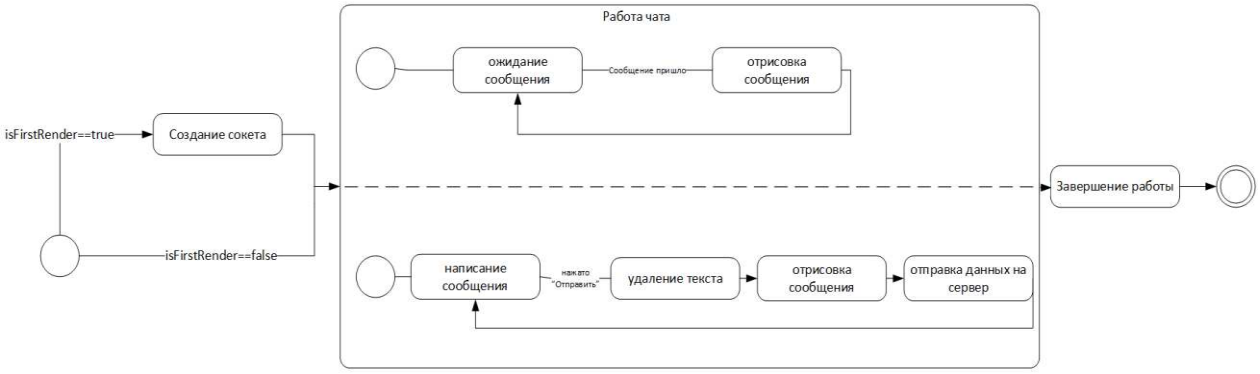

Рисунок 3.4-Диаграмма состояний модуля чат

3.1.5 Модуль главное окно

В этом модуле находится логика работы календаря событий и списка дел (рис 3.5). Если переменная состояния isLoading равна “true”, то отрисовывается компонент Loader, который отвечает за отрисовку и анимацию индикатора загрузки. Отрисовка следующих компонентов зависит от переменной состояния mode, если она равна “CALENDAR”, то отрисовывается компоненты: Calendar, DateBlock, Buttons, если mode равен “WEEKLIST” то отрисовываются все тоже кроме Calendar, вместо него WeekList. И наконец если mode равен “TD_LIST” от вместо двух выше описанных компонентов отрисовывается ToDoList, а так же не отрисовывается компонента DateBlock.

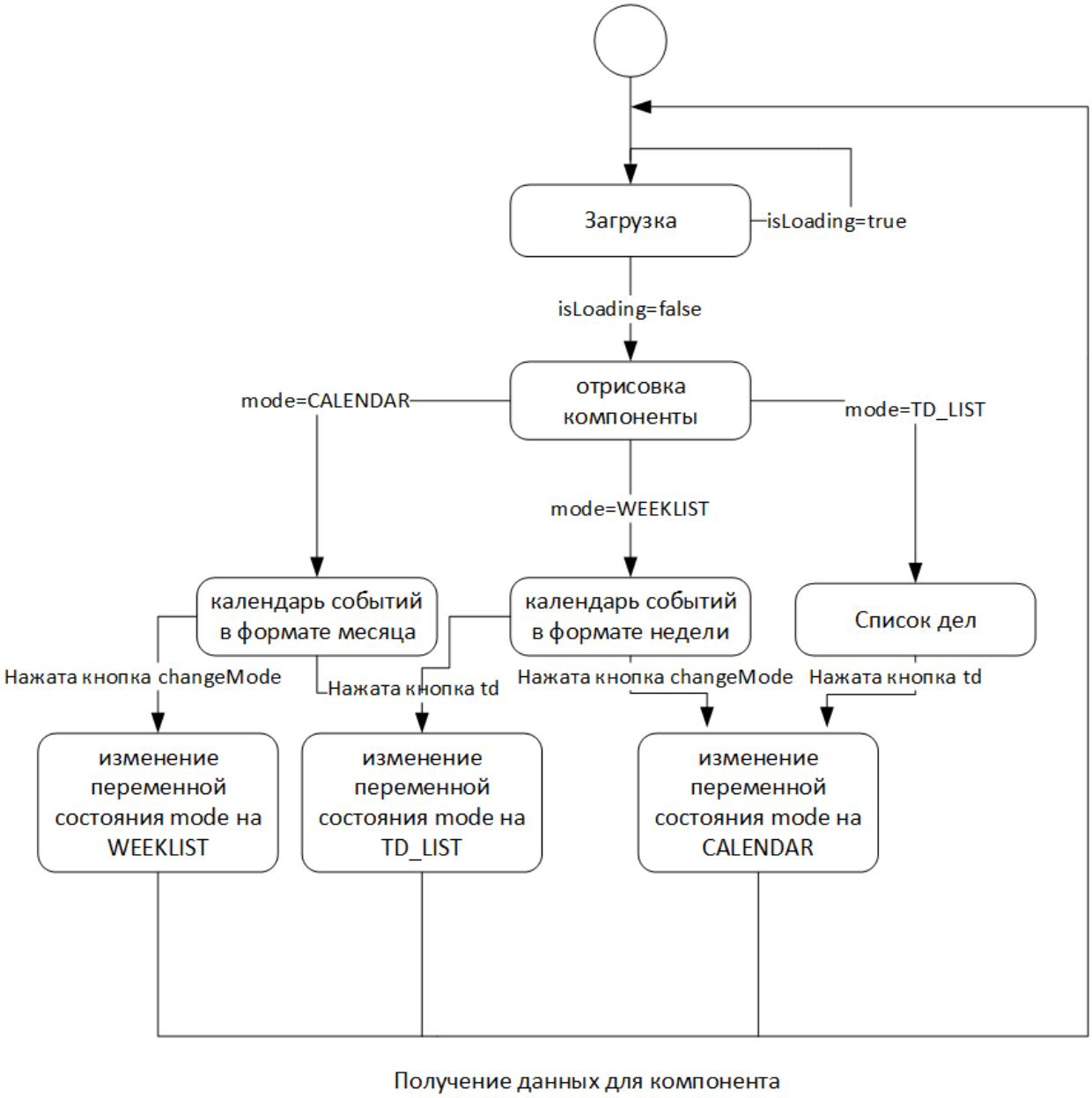

Рисунок 3.5-Диаграмма состояний модуля главное окно.

3.1.6 Модуль список дел

Модуль состоит из трех компонентов ToDoList, TdCard, TdCardItem (рис3.6).

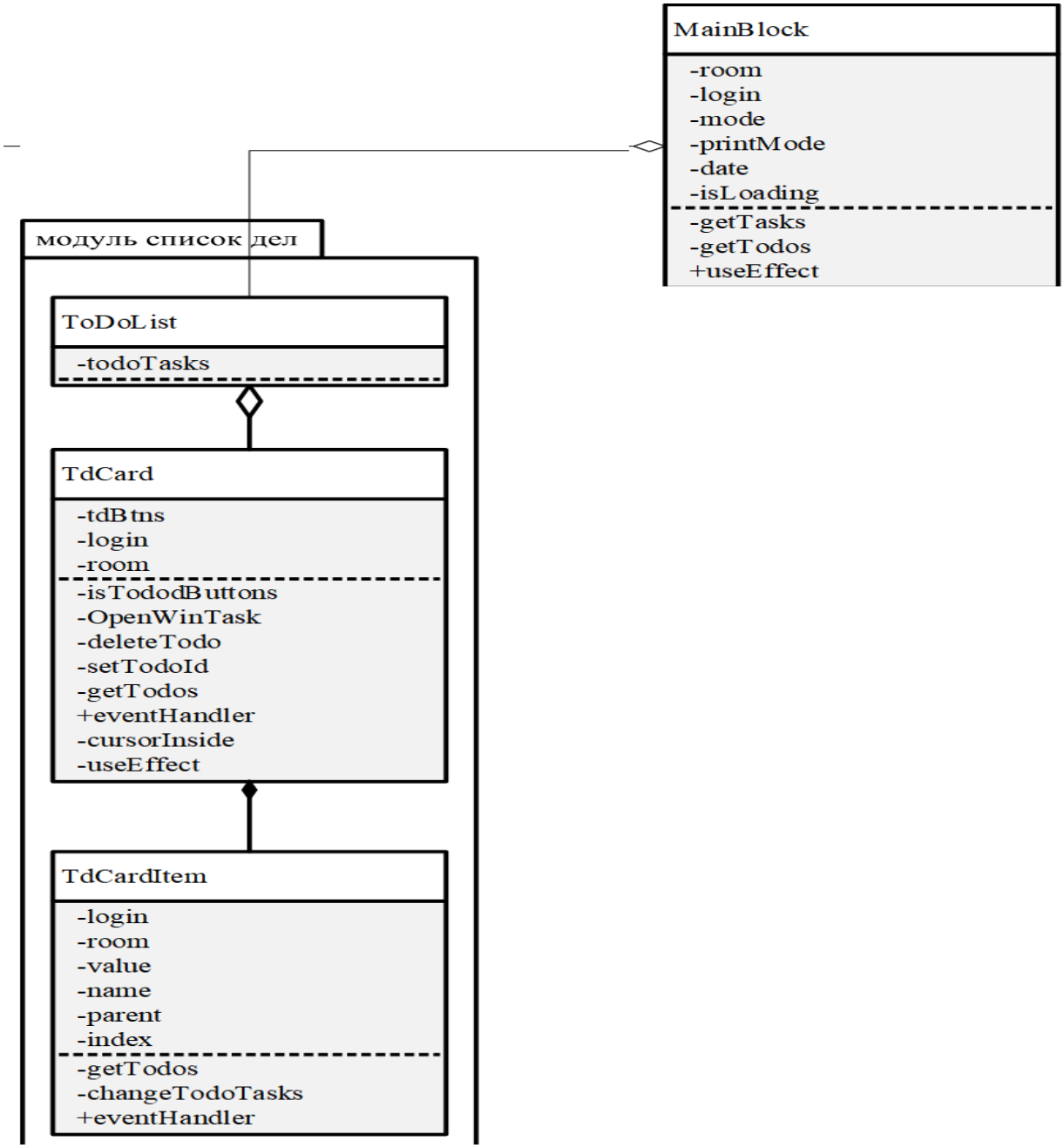

Рисунок 3.6-структура модуля список дел

В первом компоненте на основе данных из объекта todoTasks, который является переменой состояния, отрисовываются компоненты TdCard.

В TdCard описана логика рендера списка дел. Так как при наведении на список дел, на нем отрисовываются две кнопки, то в этой компоненте расположена логика отрисовки. Для описания логики используются два хука это useState, который обрабатывает переменную flag, хук useEffect, который вызывается при изменении переменной flag. useEffect добавляет или удаляет

класс “visibilityHidden” со списка дел, в зависимости от значения переменной flag. Так же используется две функции cursorInside в ней определяется находиться ли курсор внутри списка дел или он его покинул. Функция eventHandler, где находятся два условие, если тип события “mouseover” и переменная состояния todoBtns равна “true”, то вызывается функция инициализированная в хуке useState, которая меняет переменную flag на “true” и вызывает “действие” isTodoButtons. Следующие условие если курсор не внутри списка дел и переменная окружения todoBtns равна true, то меняется flag на “false” и вызывается “действие” isTodoButtons. На самом списке дел расположены два обработчика событий это onMouseOver и onMouseOut курсор наведен на объект и курсор покинул объект соответственно. Эти два обработчика вызывают функцию eventHandler.

В самом компоненте расположены компоненты DeleteBtn AddBtn, TdCardItem. React для облегчения обмена данными между родительским и дочерними компонентами предоставляет инструмент “props”, это можно сравнить с передачей аргументов в функцию, только родитель передает это компоненте. Таким образом с помощью этого инструмента из TodoList в TdCard передаются данные хранящиеся в переменой окружения todoTasks. Так переданная переменная id используется в этом компоненте для маркировки списка чтобы его можно было однозначно определить для определения его координат. Этим же способом переменные передаются в TdCardItem, DeleteBtn и AddBtn. Последние два компонента являются шаблонными кнопками, использующиеся помимо этого компонента еще в ряде других, но они имеют только одинаковый внешний вид, а функции которые повешены на их обработчики событий отличаются, таким образом из компонента TdCard в них передаются в DelteBtn функции вызывающие действие deleteTodo с аргументом id, getTodos с login и room. В TdCarditem передаются переменные для их отрисовки.

Компонент TdCardItem состоит из HTML тега input со свойством type равным “checkbox”, значенее которой по умолчанию зависит от переменной value переданной из TdCard. Так же на этом элементе весит обработчик событий onChange, который вызывает функцию eventHandler. Еще одним элементов является тег “div”, в котором надпись равная переменой name из TdCard является перечеркнутой если переменная value равна true. На ней так же вести обработчик событий onClick который вызывает функцию eventHandelr. Эта функцию вызывает два “действия” changeTodoTask и getTodos. На рис.3.7 находиться диаграмма поведения данного модуля.

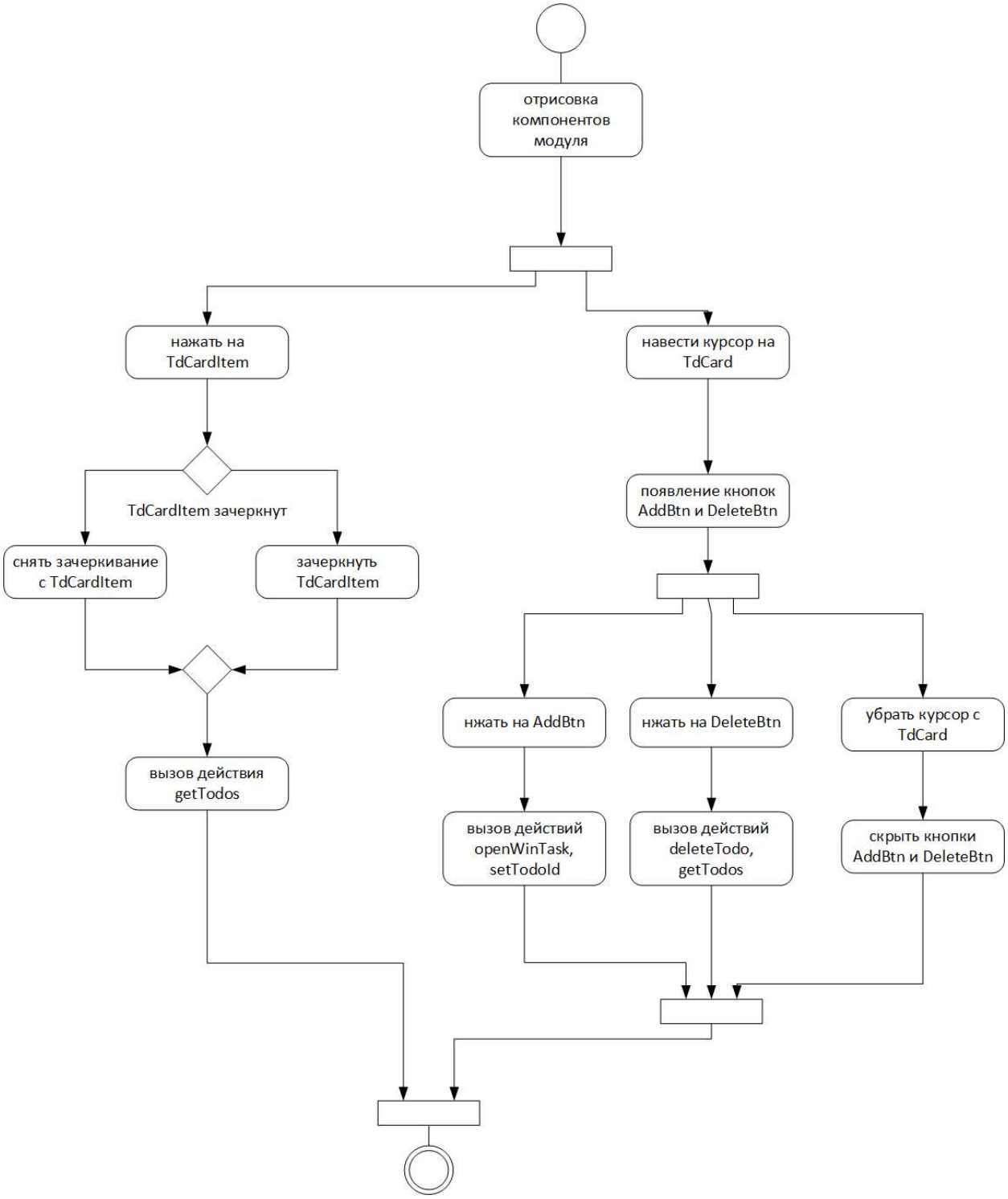

Рисунок 3.7-Диаграмма поведений модуля список дел

3.1.7 Модуль календарь событий

Этот модуль состоит из трех подмодулей рис 3.8 отображения данных в формате календаря, модуль отображения данных в формате недели, и отображение даты.

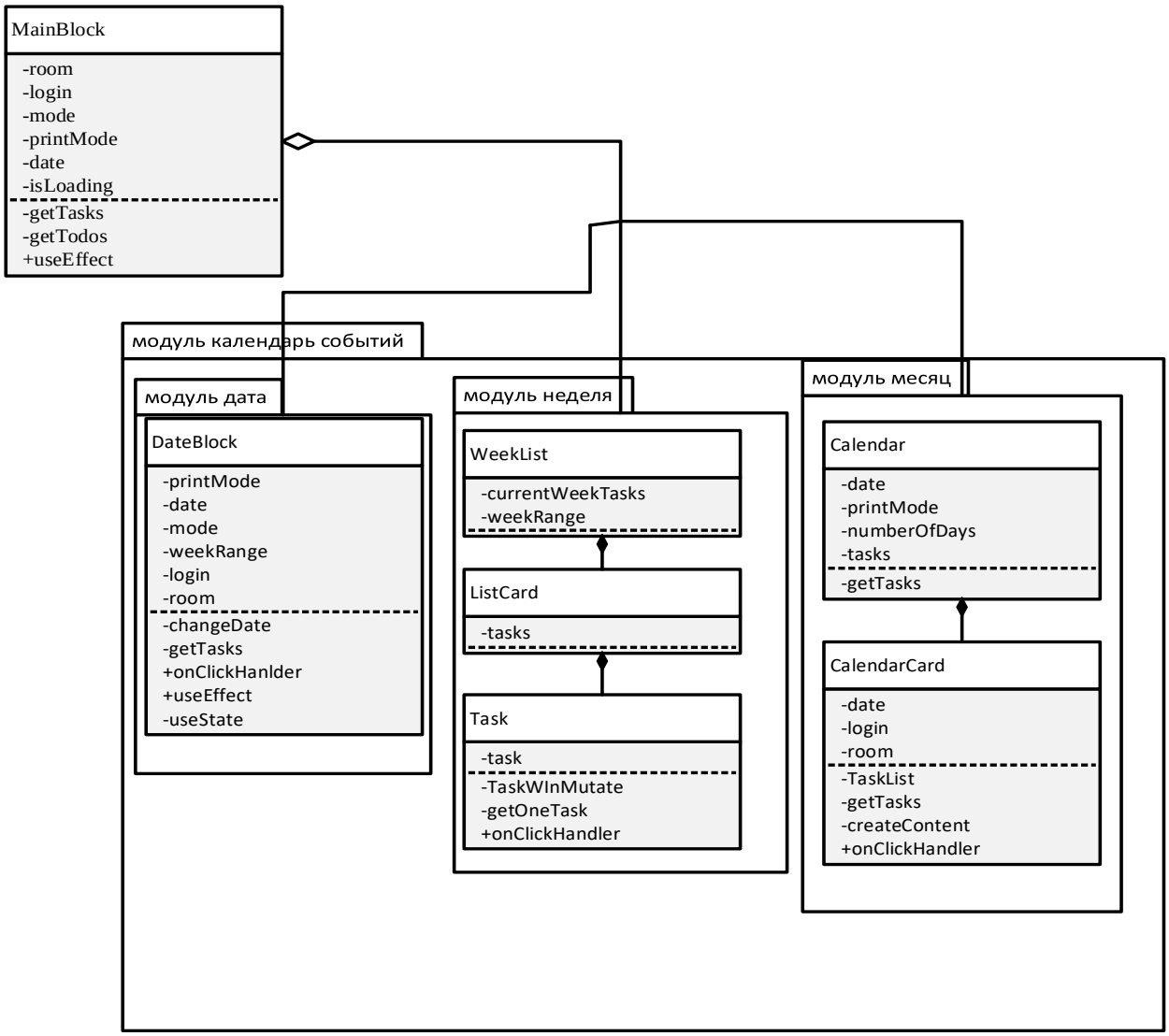

Рисунок 3.8- структура модуля календарь событий

Модуль отображения даты состоит из одного компонента dateBlock. Модуль отрисовывает две кнопки и поле для вывода даты между ними. На обе кнопки привязаны обработчики событий onClick и функции которые они вызывают onClickHandler. Эта функция вызывает “действия” changeDate, getTasks и вызывает функцию setDateFlag с аргументом “true” инициализированную в хуке useState. Так же в компоненте присутствуют два хука useEffect, первый вызывает функцию setDateFalg с аргументом “true” при изменении переменной состояния mode, второй хук содержит в себе следующую логику, если переменная dateFlag равна “true”, то в HTML объект с классом “dateBlock__date” вставляется дата в формате месяц.год или день.месяц-день.месяц, это зависит от переменной состояния mode в первом случае она равна “CALENDAR” во втором “WEEKLIST”. Дата храниться в переменных состояния date и weekRange, который является

объектом с полями first и last это первый и последний день недели в формате ПН-ВС.

Модуль отображения данных в формате календаря состоит из двух компонентов Calendar и CalendarCard. Calendar отвечает за отрисовку компонентов CalendarCard количеством равному переменной состояния numberOfDays и передачи данных в каждую компоненту в зависимости от ее номера из переменой окружения tasks. CalendarCard возвращает элемент с тэгом div с слушателем события onClick и функцией на нем onCkickHandler. Эта функция вызывает “действия” openTaskList, getTasks. Данные расположенные в этом элементе зависят от функции createContent. В этой фукции пишется количество задач переданных в этот компонент, а так же число тоже переданное из родительского компонента. Схема работы модуля показана на рис.3.9.

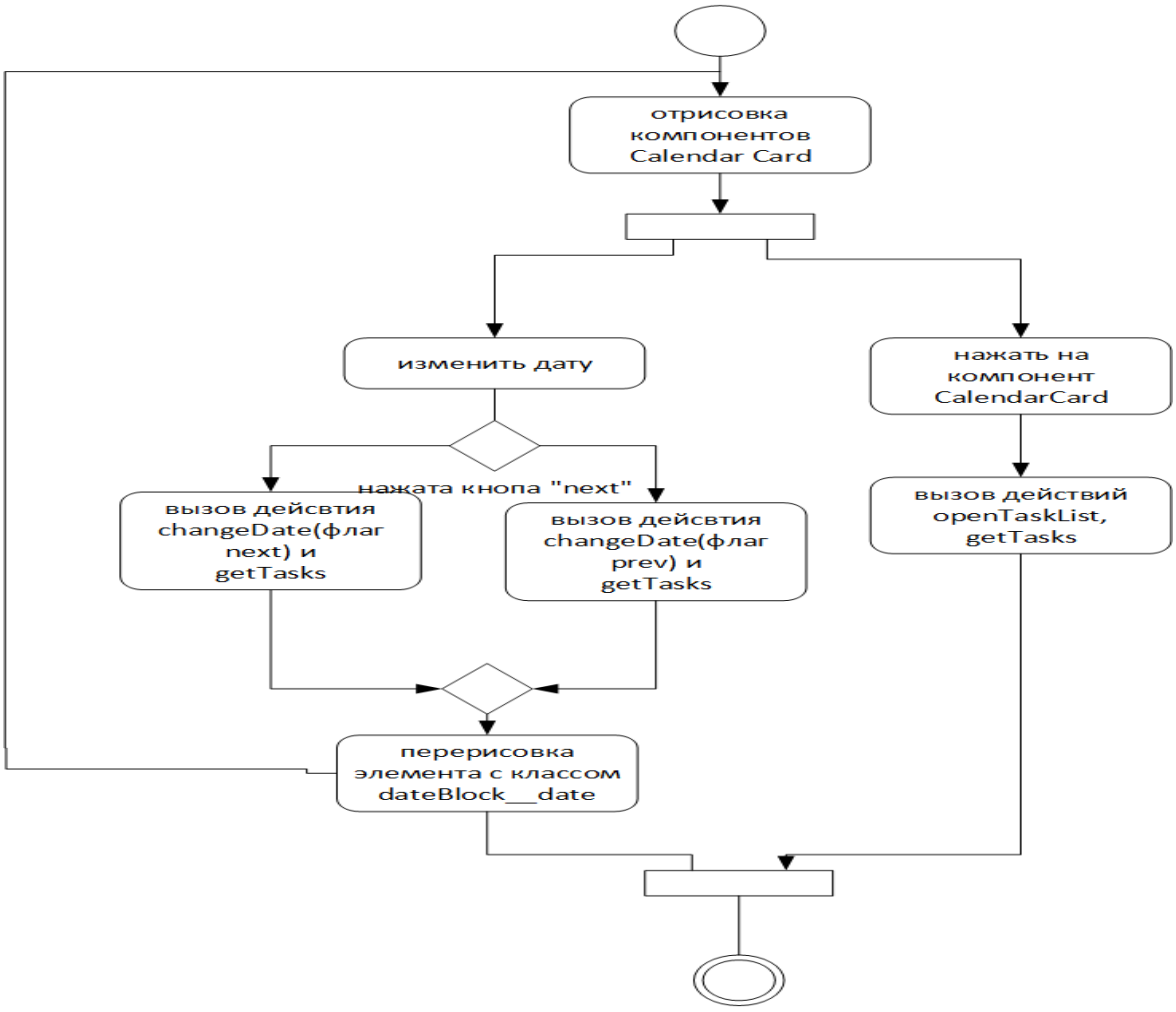

Рисунок 3.9-Диаграмма деятельности модуля календарь событий.

Модуль отображения данных в формате недели состоит из WeekList, ListCard, Task. WeekList аналогично компоненту Calendar он отрисовывает

компоненты ListCard и передает в них данные, эти данные берутся из переменных состояния weekRange, currentWeekTasks. Компонент ListCard состоит из блока в который записана дата и список задач состоящий из компонентаTask, эти данные получены из родительского компонента. Компонент Task этот компонент возвращает div с привязанной к нему функцией onClickHandler, которая при срабатывании вызывает “действие” getOneTask, TaskWinMutate. Так же в этом элементе отрисовывается переменная time и title.

3.1.8 Модуль кнопки меню

Это модуль состоит из одного компонента Buttons. В этой компоненте можно выделить три сценария отрисовки компоненты все они зависят от переменной состояния mainBlockItem, которая может быть равна “CALENDAR”, “WEEKLIST” или “TD_LIST”, сценарии представлены в таблице 3.1.

Таблица 3.1-Сценарии действия кнопок

переменная/ кнопка	CALENDAR		WEEKLIST		TD_LIST	
	Рис	Функция	Рис	функция	Рис	функция
mode	week.svg	changeMainBlock(Weeklist)	calendar.svg	changeMainBlock(Calendar)	-	-
td	todo1.svg	changeMainBlock(TD_LIST)	todo1.svg	changeMainBlock(TD_LIST)	time.svg	changeMainBlock(Calendar)
new task	nTask.svg	openWinTask(inputeTask)	nTask.svg	openWinTask(inputeTask)	nTask.svg	openWinTask(todo)

3.1.9 Модуль модальных окон

Этот модуль состоит из компонентов TaskWindow и TaskListWin (рис3.10). Первый модуль состоит из шаблонной кнопки CloseBtn в которую передаются функции вызова “действия” closeWinTask, closeErrorList, компонент InputsConteiner в который передается переменная type в зависимости от ее значения она настраивается на ввод или вывод данных, соответственно она равна “taskInput” или “taskOutput”, так же есть компонент отрисовки ошибок и шаблонная кнопка OkBtn, ее функционал так же

зависит от значения переменной type, при значении “taskOutput” передаются функции closeWinTask и clearErrorList, иначе передается функция sendDataToAction, эта функция вызывает функцию getDataFromForm, которая забирает из полей формы данные и компонует их в объект, далее в зависимости от значения переменной type происходит вызов “действий”, если type равен “taskInput”, то вызывается createTask с аргументами объект с данными и переменная состояния room, так же еще вызывается getTasks. При значении “todo” addTask и getTodos, и наконец при “todoTask” setTodoTask, getTodos. Так же этот компонент содержит “обертку”, которая блокирует экран при нажатии на нее вызываются “действия” closeWinTask, clearErrorList. В этой компоненте используется хук useEffect, который при изменении переменных состояния closeWin или closeWinTodo вызывает “действие” closeWinTask.

Компонент TaskListWin состоит из такой же “обертки” с таким же функционалом, шаблонной кнопки CloseBtn с аргументом closeTaskList, и списком задач который состоит из компонентов ListItem в каждый компонент передается переменная состояния tasks.time и tasks.title. Компонент ListItem отрисовывает выше описанные переменные и шаблонную кнопку DelteBtn в которую передается вызовы “действий” deletOneTask, getTasks, так же на “обертку” повешен обработчик событий onClick с функцией onClickHandler. Это функция вызывает “действие” getOneTask и TaskWinMutate.

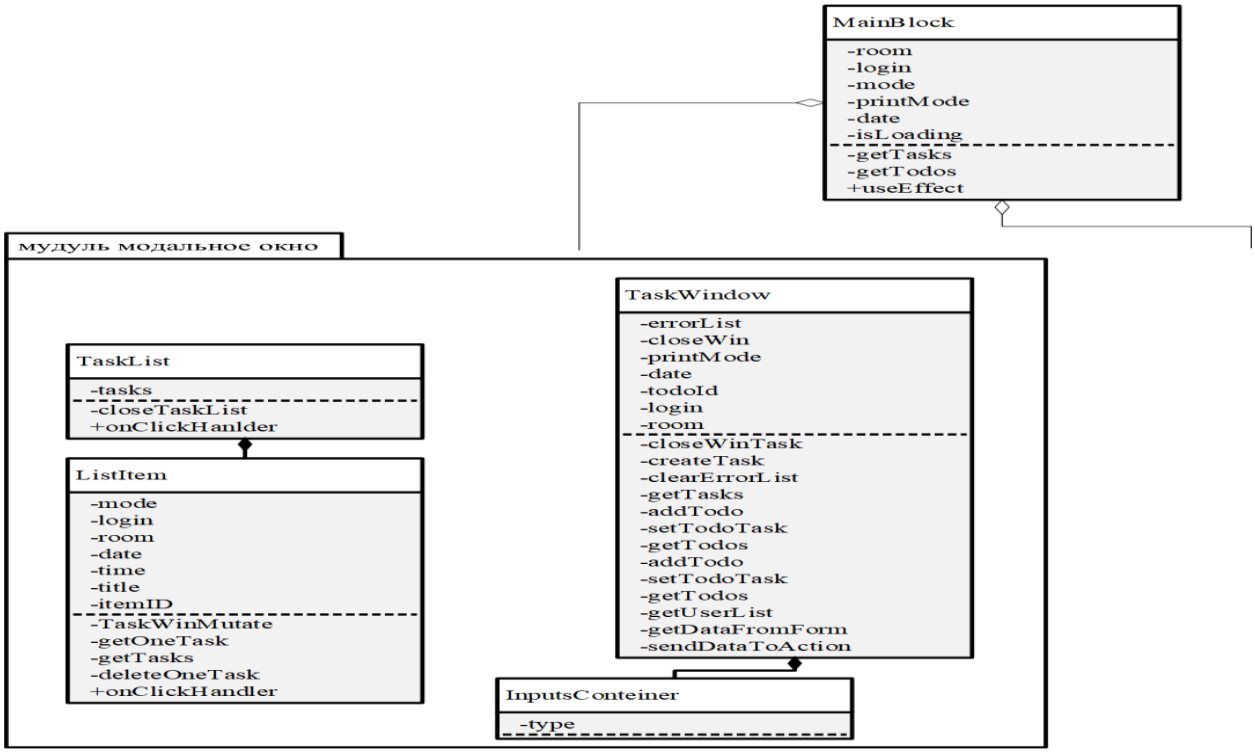

Рисунок 3.10-структура модуля модальное окно

3.2 Действия и их эффекты.

В таблице представленной в Приложении Б описаны функции действий и их эффекты.

3.3 Описание взаимодействия между клиентом и сервером

В таблице 3.2 представлены http-запросы, uri конечной точки и комментарий к запросу. В таблице 3.3 представлено описание типов сообщений передаваемых с помощью сокетов и логика их обработки

Таблица 3.2 API http-запросов

Данные	Тип запроса	Адрес	Комментарий
Объект с данными task, roomId	POST	/api/task/one	Отправка запроса на создание события
taskId	GET	/api/task/one	Отправка запроса на получения события
taskId	DELETE	/api/task/one	Отправка запроса на удаления события по Id
Room, Login, Mode, date	GET	/api/task/list	Отправка запроса на получения событий
email, login, password	POST	/api/user/perosn	Запрос на регистрацию пользователя
Email, password	GET	/api/user/perosn	Запрос на авторизацию пользователя
roomID	GET	/api/user/list	Запрос на получение пользователей из комнаты

Окончание таблицы 3.2

Данные	Тип запроса	Адрес	Комментарий
Room,user	GET	/api/todo/lists	Запрос на получение списка дел
Room,todo	POST	/api/todo/lists	Запрос на создание списка дел
Id	DELETE	/api/todo/lists	Запрос на удаления списка дел
todoId,task	POST	/api/todo/task	Запрос на создание задачи
todoId,task,value	PATCH	/api/todo/task	Запрос на обновление задачи

Таблица 3.3 типы сообщений

Тип	Подтип	Запрос	Действие
connect	-	-	На заданный сокет навешиваются события “message” и “close”
message	Reg	roomID	Регистрирует сокет и отправляет собщение {type:”chat_msg”,data:msg}
	Msg	Msg, roomID, date, loginUser	Записывает сообщение в кэш
close	-	-	Удаляет сокет из кэша

3.4 Серверная часть приложения

Серверная часть приложения реализована на платформе Node.js на языке TypeScript и фреймворке “Express”, который используется для создания web-сервера и обработки http-запросов, пакет “ws” используется для обработки web-сокетов. Для работы с базой данных используется пакет Mongoose. WebSoket-ы используется для обработки сообщений из чата, сообщения. Все

остальные запросы обрабатываются с помощью сервера созданного с помощью Express.

Эту часть приложения можно разделить на несколько слоев. Первый слой это- App, который является точкой входа в приложение, он инкапсулирует три основных блока приложения Server, Storage и Tbot(рис 3.11).

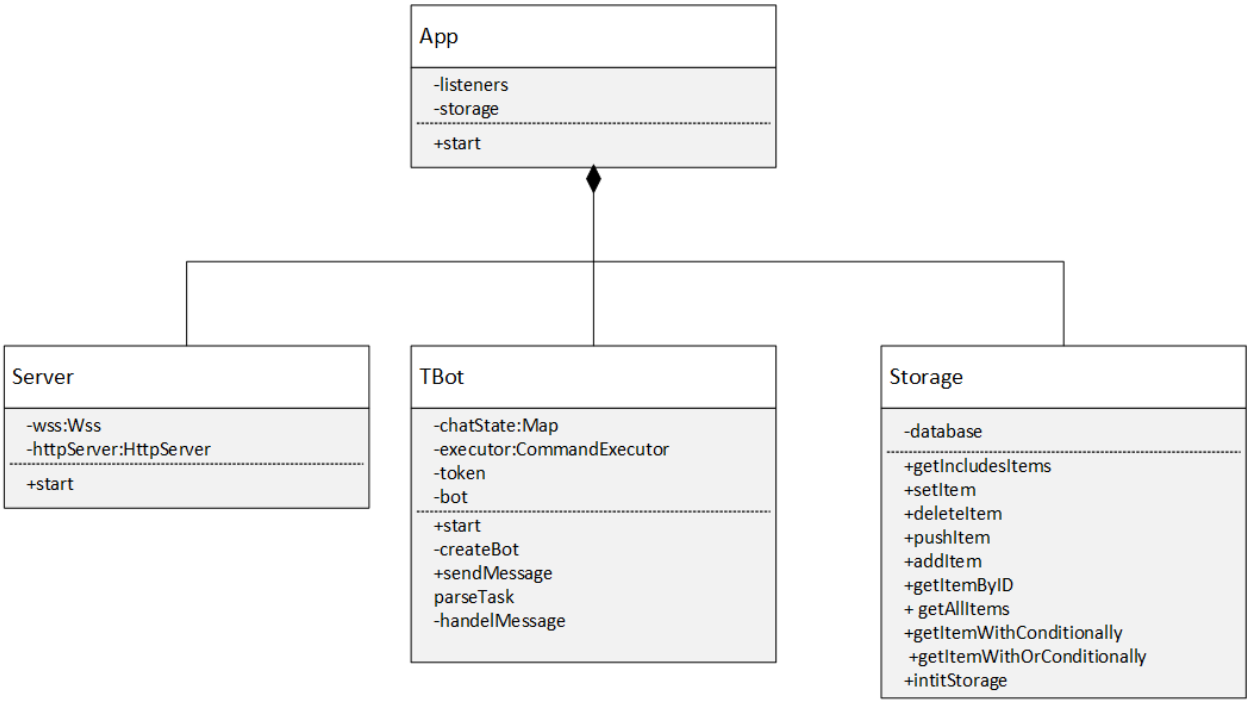

Рисунок 3.11-схема серверной части.

3.4.1 Слой App

Он состоит из одного класса App приложение A1 , свойствами которого являются массив объектов с интерфейсом IListener и объект класса StorageDB. Класс имеет один метод start, который вызывает метод initStorage() у StorageDB и вызывает у каждого элемента массива IListener метод start.

3.4.2 Слой Server

Он состоит из одного метода start и двух свойств, это объекты классов Wss и HttpServer. Метод start запускает прослушивание http-запросов и websocket’ов. Код представлен в приложение A2.

3.4.3 Слой WsServer

Этот слой состоит из одного класса за обработку websocket’ов . Он имеет интерфейс IListener и Observer. Методами класса являются initWss он вызывается в конструкторе класса для создание слушателя порта, connectHandler, messageHanlder, closureHandler эти методы отвечают за обработку сообщений, первый метод отвечает за обработку сообщений при подключение клиента, в нем на действия сокета привязываются методы их обработки на “message” и “close” messageHanlder, closureHandler соответственно. Второй и третий методы вызывают метод handelIncommingMessage объекта msgDistributor. Еще одним методом является метод sendMessage он принимает в аргументах сокет и данные которые нужно отправить и данные для определения работы метода. С помощью структуры case в зависимости от аргумента mode данные высылаются либо группе пользователей либо, определенному получателю. По контракту с интерфейсом Observer класс должен вызвать метод sendMessage с аргументами которые пришли в метод Observer. На рисунке 3.12 представлена структура программы обрабатывающая сообщения, и логика его работы рис 3.13, код представлен в приложении А3.

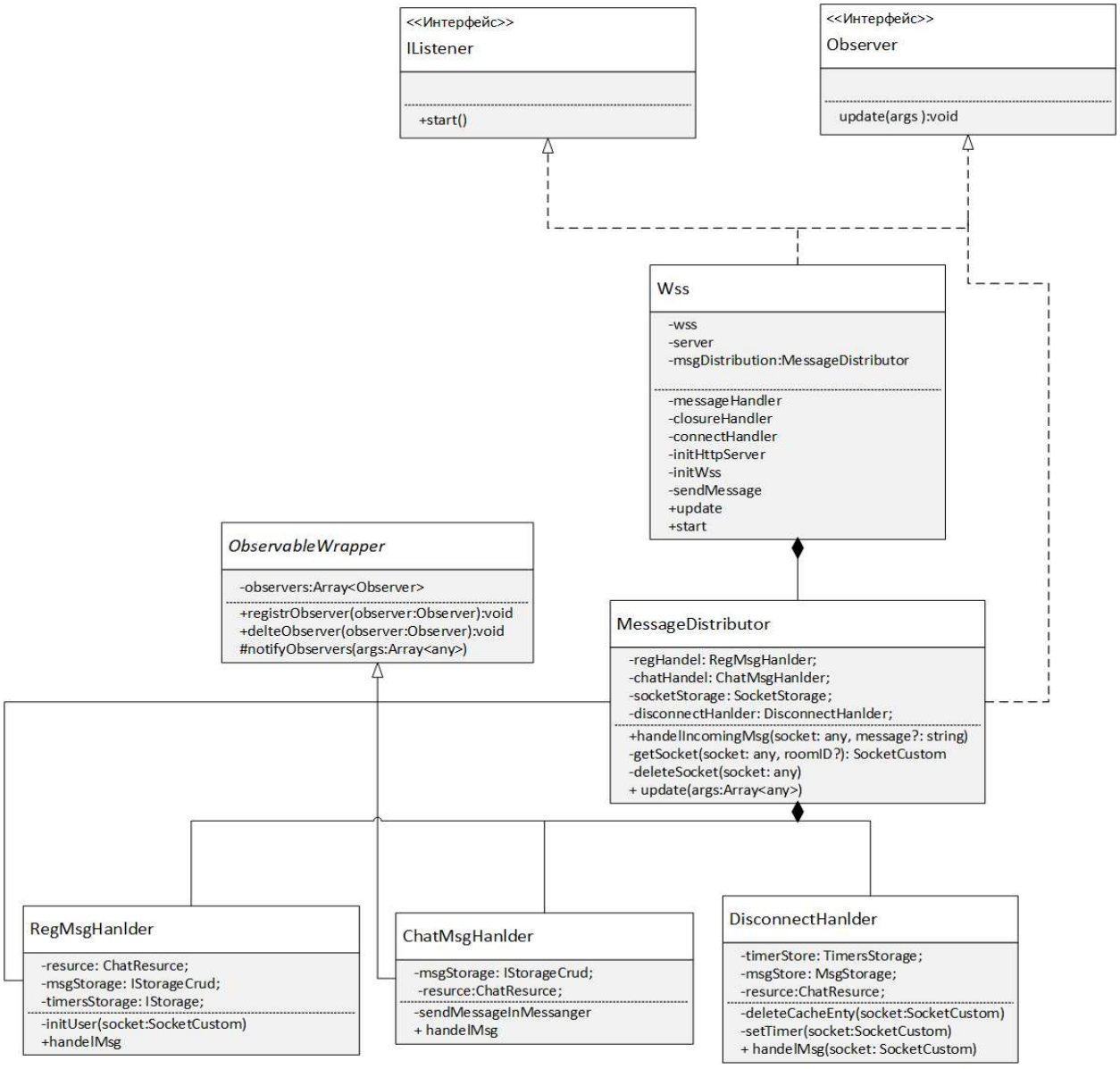

Рисунок 3.12-структура программы обрабатывающая сообщение.

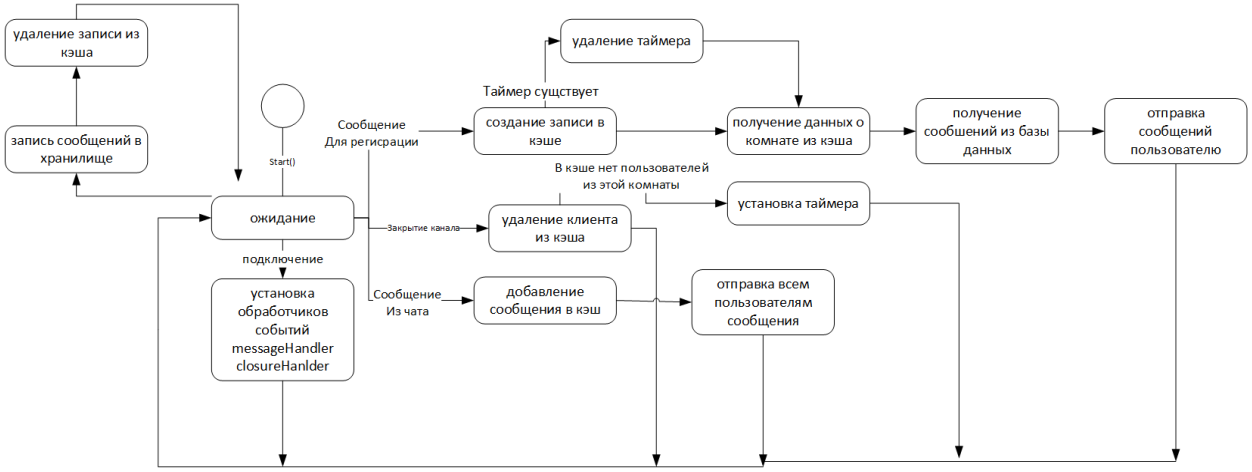

Рисунок 3.13- схема WebSocket сервера.

3.4.4 Слой MessageHandler

Этот слой отвечает за обработку сообщений трех типов сообщения о регистрации сокета, сообщения с данными из чатов и сообщения о закрытии соединения. Этот слой состоит из четырех классов MessageDistributor, RegMsgHandler, DisconnectHandler, ChatMsgHandler.

Первый класс отвечает за распределение сообщение между остальными классами и за регистрацию сокетов или получению если они уже зарегистрированы, это необходимо для сопоставления сокета с кастомным типом “SokcetCustom”-этот тип нужен для добавления к сокету id комнаты и его собственного id. В методе handelIncomingMessage в структуре case происходит вызов функции в зависимости от переменной type полученной из сообщения, типы reg и msg вызывают методы regHandler.handelMsg и chatHandler.HandelMsg соотвественно. В “default” происходит вызов deleteSocket и disconnectHanlder.handelMsg для разрыва соединения. Метод update полученный от интерфейса Observer в нем вызываются методы notifyObservers унаследованный от класса ObservableWrapper.

RegMsgHandler состоит из трех аргументов это обьекты классов MsgStorage, TimerStoarge и ChatResurce. Этот класс имеет публичный метод handelMsg в нем вызывается метод initUser, в который передается аргумент функции socket, в этом методе вызывается метод timersStorage.deleteItemByName и msgStorage.updateItemByName. Далее в handelMsg с помощью ChatResurce.getChat получает сообщение из хранилища и с помощью msgStorage.getItemByName получает сообщение из кэша, дальше оба массива обьектов объединаются и предаются в метод notifyObservers. DisconnectHandler имеет публичный метод handelMsg в нем из хранилища MsgStorage получает массив сокетов, в нем находится сокет переданный в аргумент функции и удаляется, если массив становится пустой, то вызывается метод setTimer, в нем в хранилище TimerStore добавляется запись таймера с функцией deleteCacheEnty, при вызове этого метода из хранилища MsgStorage выгружается записи чата и записываются в бауз данных, далее из хранилищи сообщений и таймеров удаляются записи этой комнаты. Последний класс имеет поля MsgStorage и TBot,ChatResurce, первое поле это объект хранилища сообщений второй объект отвечающий за логику работы телеграмм бота и третье это объект взаимодействия с базой данных. В методе msgHandler полученное сообщение отправляет в метод senMessageInMessenger, где их базы данных выгружаются все комнаты месенджера куда необходимо отправить сообщение и вызывается метод

tbot.sendMessge. Далее из хранилища msgStorage выгружается данные по сокетам и записываются данные в кэш, после этого если сокеты есть вызываются метод notifyObsrver куда передаются сокеты и данные, если же сокетов нет то данные записываются в базу данных.

3.4.5 Слой Local Storages

Этот слой состоит из трех классов MsgStoarge, SokcetStorage, TimerStorage. Первый класс с интерфейсом IStorageCRUD, остальные два с интерфейсом IStorage. У всех трех классов есть методы getItemByName, senItemByName и deleteItemByName и переменная тип Map. Так же у MsgStorage есть еще один метод это метод updateItemByName.Эти методы являются интерфейсом для взаимодействия данных типа Map.

3.4.6 Слой StorageBD

Отвечает за подключение к базе данных которая передается в конструктор и инициализируется в методе initStorage. Так же этот класс хранит статические методы для обращение к базе данных, код представлен в приложении А6.

Таблица 3.4 методы обращение к базе данных

Название	Аргументы	Комментарий
getItemWithOrConditionally	Model, filds, fildsVal, fildsOr, fildsValOr, returnVal	Возвращает объекты соответствующие двум условиям
getItemWithConditionally	Model, filds, fildsVal, returnVal	Возвращает объекты соответствующие условиям
getAllItems	Model, reurnVal	Возвращает все объекты описанные этой моделью
getItemByID	Model,id ,reurnVal*	Возвращает элемент с данным id
addItem	Model,item ,reurnObj**	Создать новый объект
pushItem	Model,query,param,val	Дополнить объект
deleteItem	Model,query,param,val	Удалить объект

Окончание таблицы 3.4

Название	Аргументы	Комментарий
setItem	Model,query,param,val	Изменить объект
getIncludesItems	Model,valName,val,returnVal	Возвращает поля объекта
* это массив строк которые равны тем полям объекта которые необходимо вернуть, по умолчанию это пустой массив и возвращаются все поля. ** это флаг при значении true, метод возвращает объект созданный в базе данных.		

3.4.7 Слой HttpServer

В конструктор он принимает экземпляр класса router и Exress. В методе initServer происходят настройки сервера и подключения “роутов”. Метод getServer возвращает объект server, это http-сервер с привязанными маршрутами.Структура http-сервера описана на рис3.14. Код представлен в приложении А4

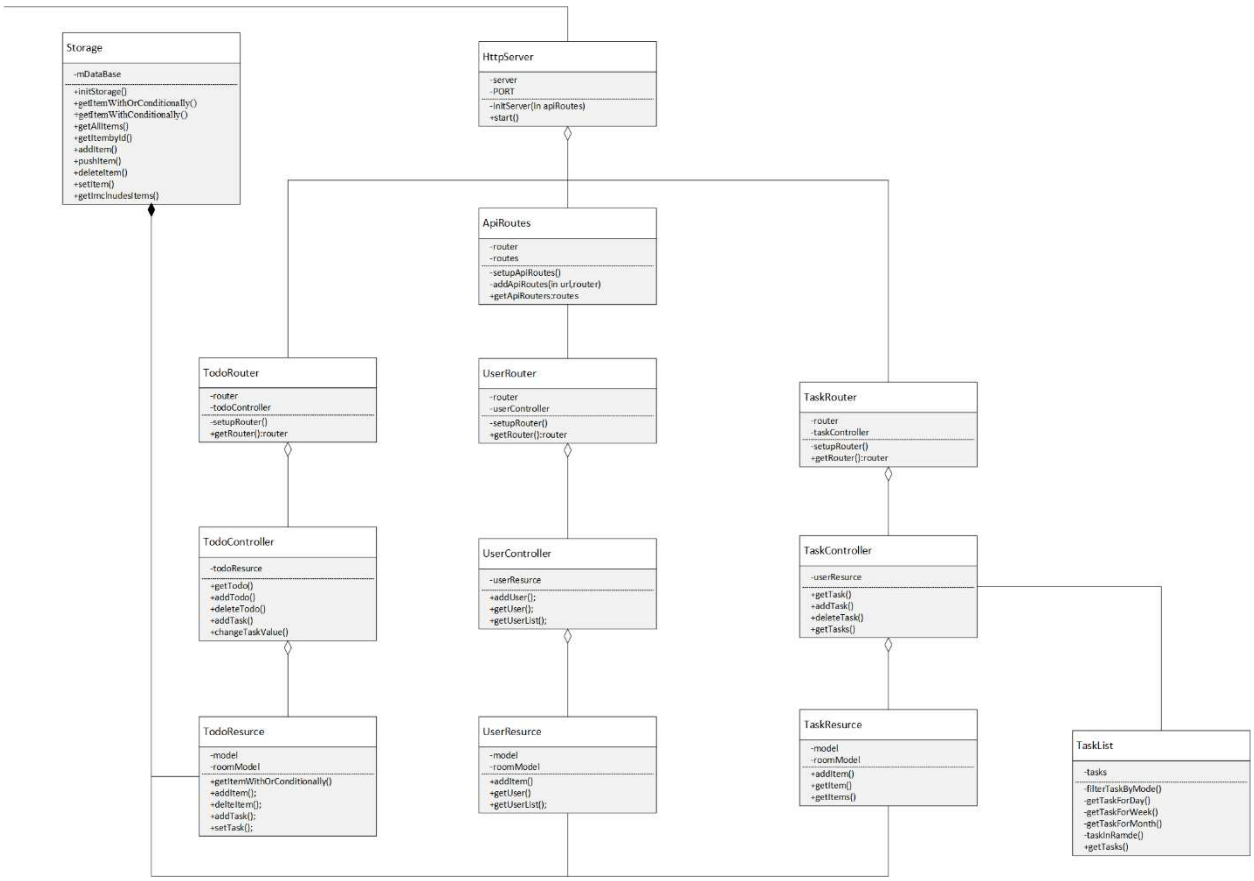

Рисунок 3.14-схема http сервера.

3.4.8 Слой ApiRoute

В конструктор класса передается объект Router из фреймворка Express, который позволяет создавать модульные маршруты, и объект routes который содержит объекты с полями url и router. В поле url находиться строка с маршрутом “/api/task”, “/api/todo”, “/api/user”, “/”. Поле router это такой же объект Express. Router на котором описаны продолжения выше обозначенных маршрутов, а так же описаны конечные точки, это taskRouter, todoRouter, userRouter, Исключением является объект с полем url равным ‘/’, в нем в поле router описан путь до корневой папки с клиентской части, которая отправляется клиенту при первом запросе.

Это все достигается с помощью методов setupApiRouters и addApiRouter, В первом методе для каждого объекта хранящегося в объекте routes, вызывается метод addApiRouter, который привязывает к Router url и второй Router.

3.4.9 Слой Router

Этот слой состоит из трех классов TaskRouter, TodoRouter, UserRouter. Эти классы отвечают за создание Router, который обрабатывает запросы отправленные на ‘api/task’, ‘api/todo’, ‘api/user’ соответственно. В конструктор каждого класса передаются Router, Task/Todo/UserController. Контроллеры содержат методы обработки запросов.

TaskRouter обрабатывает запросы POST, GET, DELTE на ‘/one’, и GET на ‘/list’. POST запрос вызывает метод addTask и в случае ошибки отправляет ответ со статусом 400. GET запрос вызывает метод getTask и так же в случае ошибки отправляет такой же ответ, DELETE вызывает deleteTask. GET запрос на url ‘list’ аналогичен по контрукции и вызывает метод getTaskList.

TodoRouter обрабатывает запросы POST, GET, DELTE на ‘/todo’, и POST, PATCH на ‘/task’. Этот класс аналогичен предыдущему и вызывает методы на ‘/todo’ addTodo, getTodo, deleteTodo и на ‘/task’ addTask, setTask.

UserRouter обрабатывает запросы GET, POST ‘/user’ и GET на ‘/user/list’, соответственно вызываются две функции getUser,createUser, getUserList.

3.4.10 Слой Controller

Этот слой состоит из трех классов Todo/Task/UserController и отвечает за обработку запроса и формирование данных для ответа если это необходимо. Во всех трех классах в конструктор приходит экземпляр класса Todo/Task/UserResurce.

TaskController помимо экземпляр а класса TaskResurce еще получает в конструкторе класс TasksList. Класс имеет методы getTask, deleteTask, addTask, которые работают аналогично он вызывают методы экземпляр а класса TaskResurce с такими же названиями или же в случае ошибки возвращают ошибку. В свою очередь метод getTaksList создает экземпляр класса TasksList в конструктор которого передает вызов метода getTaskList и данные пришедшие из запроса, и у этого экземпляра вызывается “геттер” task и полученные данные возвращаются из метода.

TodoController обладет методами getTodos, addTodos, deleteTodo, addTask, setTask, которые вызывают одноименные методы у экземпляра класса TodoResurce и в случае ошибки возвращают ошибку.

UserController обладает методами addUser, getUser, getUserList функционал которых аналогичен предыдущим методам.

3.4.11 Слой Resurce

Этот слой отвечает за отправку и прием данных из методов Storge’а для работы с базой данных. Состоит так же как и предыдущие слои из трех классов. В конструктор передается модель User, Task, Todo -это описание структуры объекты, который будет храниться в базе данных и отдельно модель Room. В данных классах используются статические методы класса Storage.

TaskResurce обладает методами getTask, getTaskList, deleteTask, addTask. Первый вызывает метод getItemById и если она ничего не вернула возвращает ошибку иначе возвращает данные. Второй формирует поля запроса и вызывает метод getItemWithOrConditionally и возвращает данные

из нее. deleteTask вызывает метод deleteItem. И addTask вызывает метод addItem.

TodoResurce обладает методами getTodos, addTodo, delteTodo, addTask, setTask. Первая формирует поля запроса и вызывает метод getItemWithOrConditionally и возвращает данные из нее. Вторая вызывает метод addItem с дополнительным аргументом true, который необходимо указать что бы метод вернул id записи сделанной в базе данных и вызывает метод pushItem, для добавления этого id в объект комнаты в базе данных. deleteTodo вызывает метод deleteItem. addTask вызывает метод pushItem, setTask вызывает setItem.

UserResurce обладает методами pushUser,getUser и getUserList. В первом методе вызывается метод getAllItems с параметром email, если он возвращает данные, то вызывается ошибка так как этот email уже занят. Далее если в аргумент метода room равен “undefined”, то вызывается метод addItem с моделью Room, и pushItem с моделью User, иначе вызывается два метода pushItem для Room и для User.

3.4.12 Слой ChatBots

Этот слой отвечает за обработку входных сигналов телеграмм бота, он состоит из классов Tbot, CommandExecutor, CommandFactory, RegCommand, SendCommand рис3.15, код представленв А5. В TBot находятся свойства bot и executor это объект класса CommandExecutor получает от интерфейса IListener метод start, в котором происходит привязка обработчика сообщений к боту. При получении сообщения текст сообщения передается в метод praseText, где текст сообщения делиться на команду и аргументы и возвращаются в виде объекта с полями type и args. Далее в метод handelCommandResponse передается вызов метода executor.getCommandResult. Так же этот класс имеет два публичных метода sendMessage и sendKeyboard, которые отправляют в чат сообщения или клавиатуру.

Класс CommadnExecutor имеет в аргументах CommandFactory и публичный метод getCommandResult в нем вызывается метод createCommand в нем вызывается метод CommandFactory.createCommand из него возвращеается объект с интерфейсом Command, из метода возвращается

вызов метода commandExecute с аргументами объектом Command и chatId, в этом методы у объекта Command вызывается метод execute.

Класс CommandFactory является фабрикой для объектов и снтерфейсом Command в зависимости от аргумента type функции createCommand возвращает объект RegCommand или SendCommand.

Классы RegCommand и SendCommand реализуют интерфейс Command с методом execute. В первом классе происходит вызов функции переданной в конструктор в классе CommandFactory с аргументами в виде email, password, chatId. В следующем классе в методе execute вызывается метод UserResurce.getUserByChatId, создается переменная fakeSocket(из-за ошибки при проектировании класса ChatMsgHandler) и вызывается метод executor с аргументами fakeSocket и данными из класса. Так же на рис 3.16 представлена схема работы данного модуля.

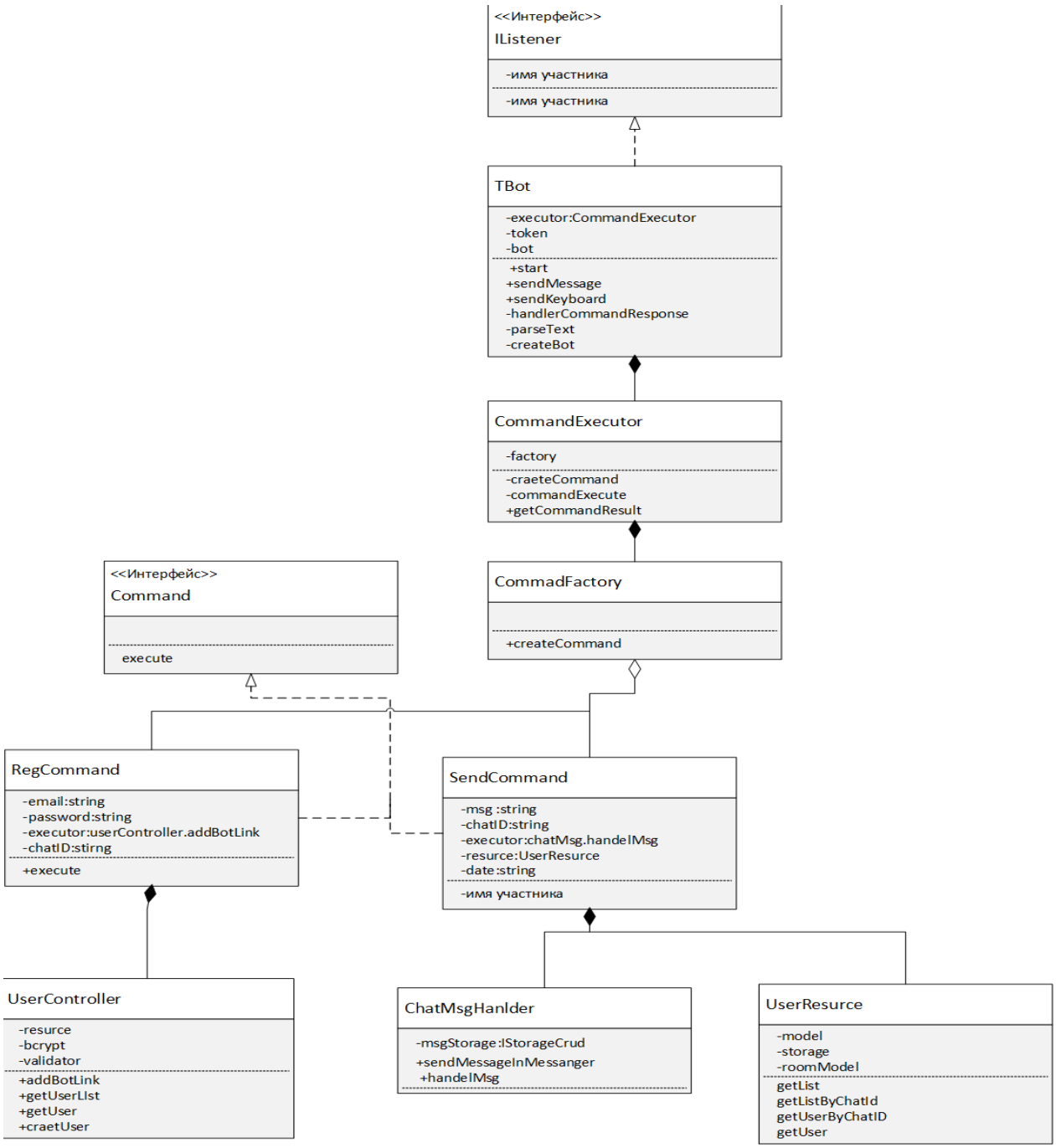

Рисунок 3.15- структура слоя ChatBots

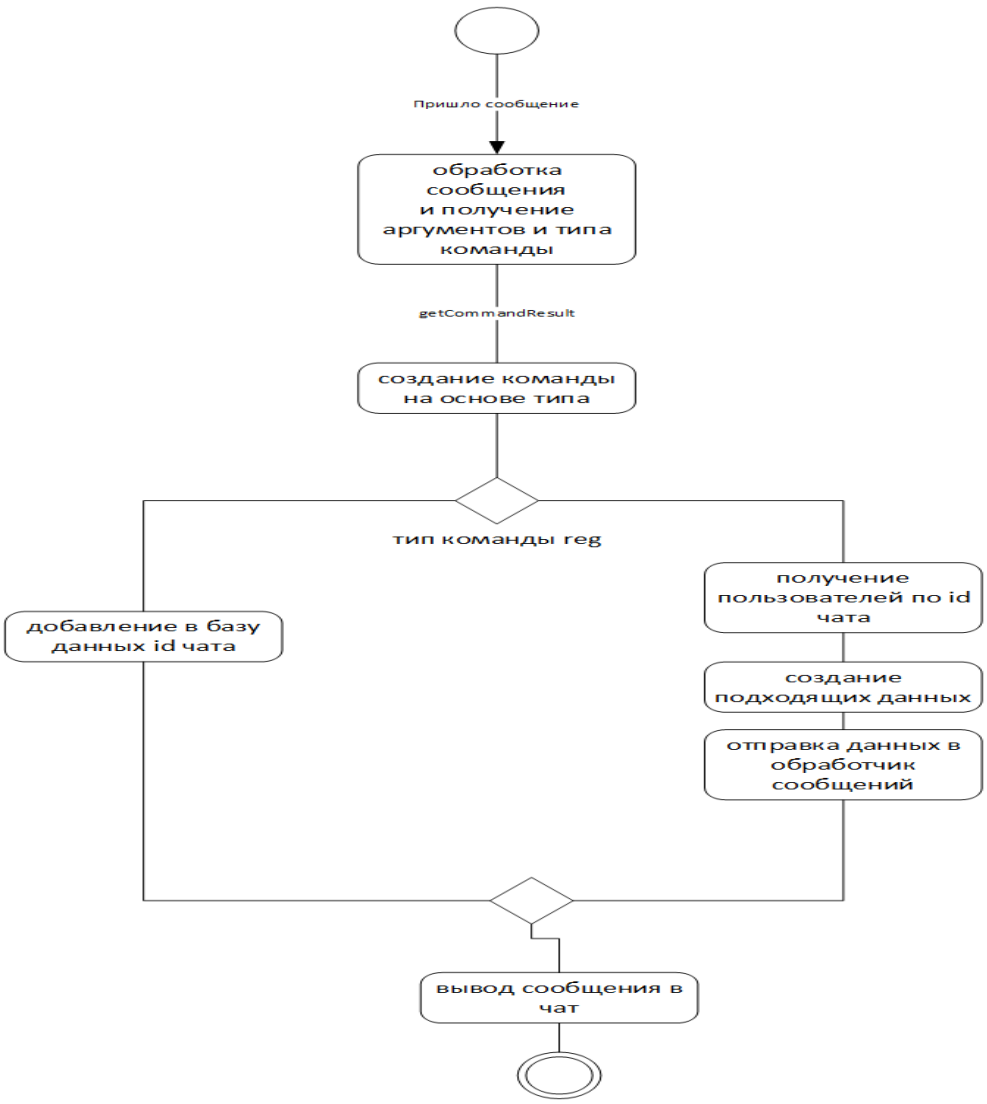

Рисунок 3.16- диаграмма деятельности слоя ChatBots

3.4.13 Вспомогательные Классы и Интерфейсы

Класс ReqValidator имеет одно поле это поле rules в нем хранятся функции валидации полей и один публичный метод это метод validate на вход которого приходит объект с необязательными полями login, password, rPassword, email, далее инициализируется массив result в который поступает либо number либо boolean, и происходит обход по полям этого объекта и вызывается функция из переменной rules с ключом равным имени поля вызывается функция куда передаются в качестве аргументов значения полей объекта.

Класс ObservableWrapper необходим для паттерна наблюдатель, он состоит из трех методов registrObserver для регистрации подписчика, delteObserver для удаления и notifyObservers для их оповещения(вызова у каждого подписчика метода update и передача туда аргументов).

Интерфейсы Observer описывает метод update, для паттерна наблюдатель. IStorage и IStorageCRUD, второй наследуется от первого и добавляет метод updateItemByName к уже существующим методам getItemByName setItemByName deleteItemByName. IListener с методом start, для классов которые должны начинать слушать порты. Command с методом execute.

3.5 Отладка приложения

Для отладка web-приложение было развернуто на “http://loaclhost/”. Клиентская часть запускалась в браузерах Chrome и Opera, а так же с помощью консоли разработчика в браузере Chrome, симулировался запуск приложение в смартфоне. Весь заявленный функционал работал корректно.

Тестирование показало успешную работу всех модулей приложения. Интерфейс приложения в момент тестирования представлен на рис.3.10,3.11,3.12.

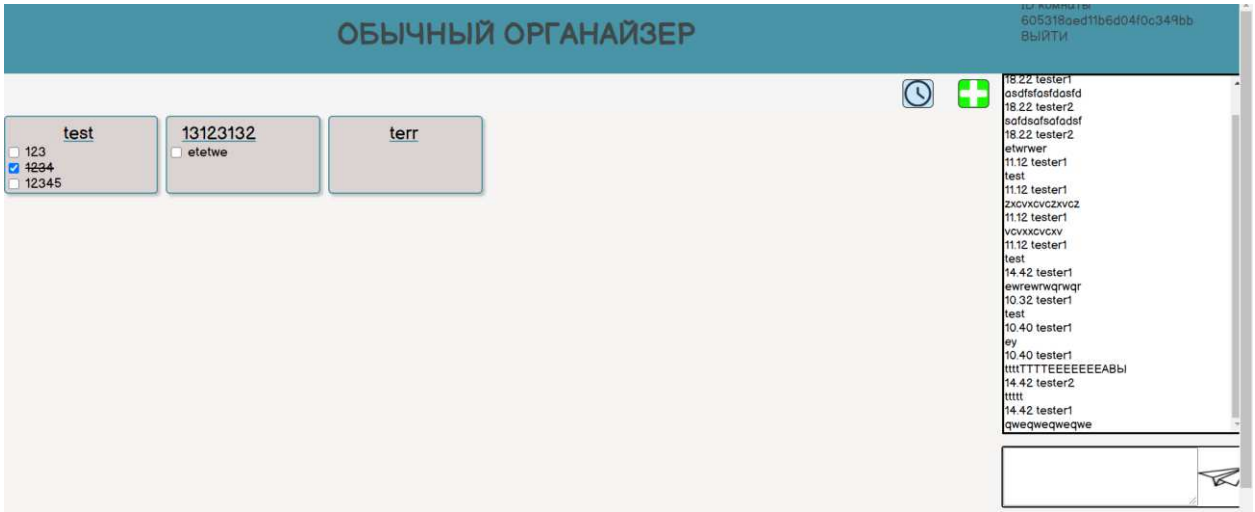

Рисунок 3.17- пользовательский интерфейс список дел в браузере Chrome.

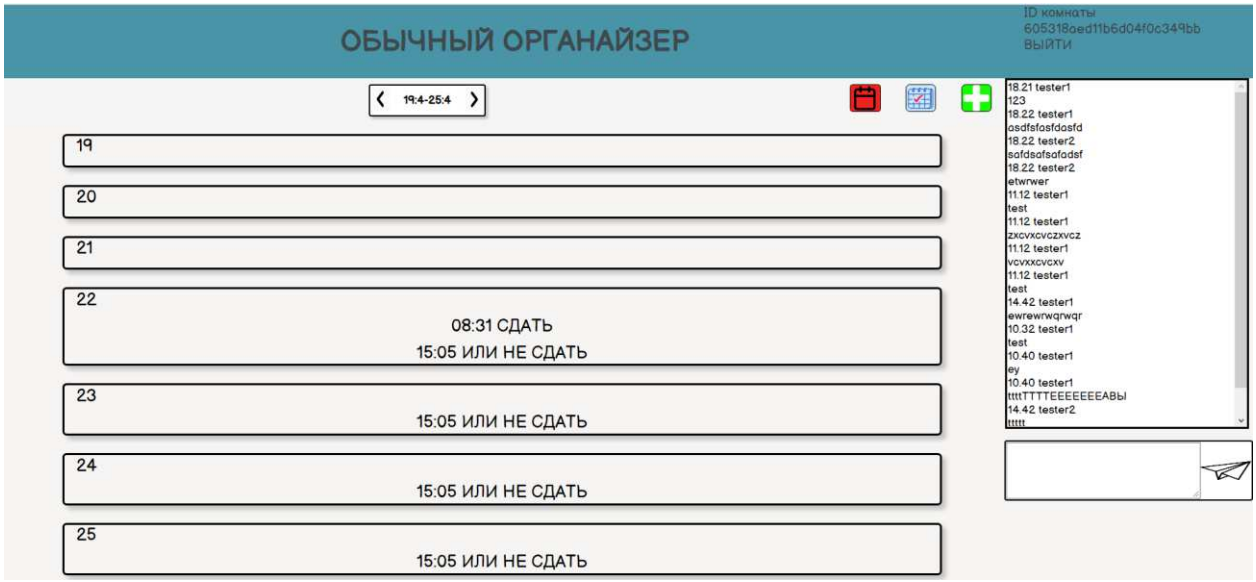

Рисунок 3.18- пользовательский интерфейс календарь событий в браузере Opera.

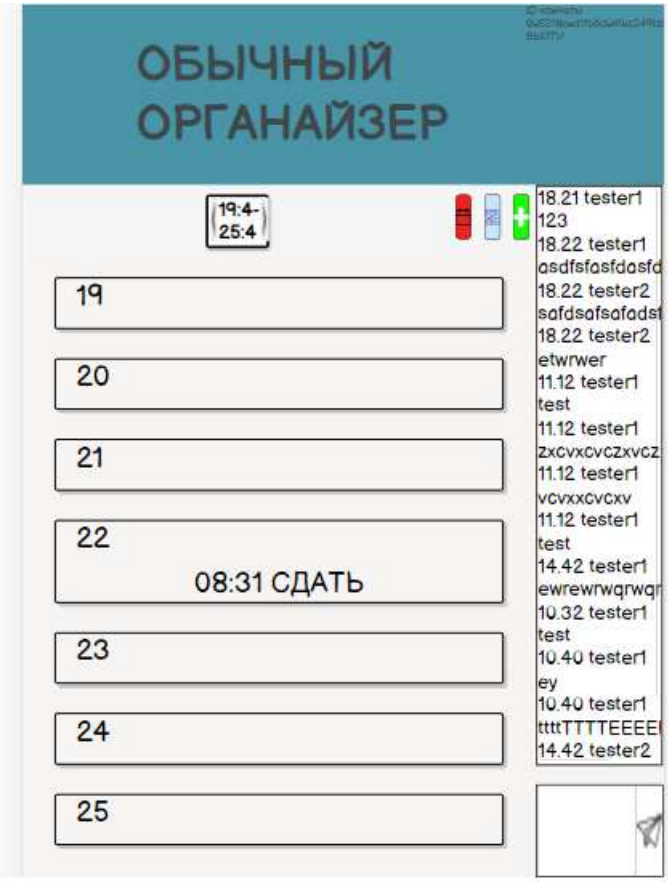

Рисунок 3.19- пользовательский интерфейс календарь событий на iPhone4.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были решены следующие задачи:

1.Выполнен анализ предметной области.

2.Проведен сравнительный анализ аналогов.

3.Составленно техническое задание.

4.Произведенно проектирование архитектуры и выбор средств разработки.

5.Выполенна программная реализация приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Органайзер[Электронный ресурс]: - Режим доступа: https://ru.wikipedia.org/wiki/Персональный_органайзер
2. Google Calendar [Электронный ресурс]: - Режим доступа: https://calendar.google.com/calendar
3. Cozi Family Organizer [Электронный ресурс]: - Режим доступа: https://www.cozi.com/
4. Flyak [Электронный ресурс]: - Режим доступа: https://app.weeek.net/
5. Клиент-сервер [Электронный ресурс]: - Режим доступа: https://ru.wikipedia.org/wiki/Клиент_—_сервер
6. Одностраничное приложение [Электронный ресурс]: - Режим доступа: https://en.wikipedia.org/wiki/%20Single-page_application
7. Web Socket [Электронный ресурс]: - Режим доступа: https://en.wikipedia.org/wiki/WebSocket
8. Наблюдатель [Электронный ресурс]: - Режим доступа: https://en.wikipedia.org/wiki/Observer_pattern
9. React [Электронный ресурс]: - Режим доступа: https://reactjs.org/
10. React хук [Электронный ресурс]: - Режим доступа: https://reactjs.org/docs/hooks-intro.html
11. Redux [Электронный ресурс]: - Режим доступа: https://redux.js.org/tutorials/essentials/part-1-overview-concepts
12. Redux-thunk [Электронный ресурс]: - Режим доступа: https://github.com/reduxjs/redux-thunk
13. Node.js [Электронный ресурс]: - Режим доступа: https://nodejs.org/en/about/
14. Express [Электронный ресурс]: - Режим доступа: https://expressjs.com/
15. MongoDB [Электронный ресурс]: - Режим доступа: https://docs.mongodb.com/manual/introduction/
16. Mongoose [Электронный ресурс]: - Режим доступа: https://www.npmjs.com/package/mongoose

Приложение А

A1

```
import {server} from './Server'
import {tbot} from './../chatbots/t_bot'
import {IListener} from './IServer'
import {storage} from './Storage';

export class App {
   private listeners:Array<IListener>=[];
   private storage;
   constructor(listeners:Array<IListener>, storage) {
      this.listeners=listeners;
      this.storage = storage;
   }
   async start() {
      if (!await this.storage.intitStorage()) {
         console.log("error at init storage!");
         process.exit(1);
      }
      this.listeners.forEach(e=>e.start());

   }

}

export const app=new App( [server,tbot],storage);
```

A2

```
import * as  http from 'http';
import {IListener} from './IServer'
import {HttpServer,httpServer} from './HttpServer'
import {Wss,wss} from './Wss'

export class Server implements IListener{
  PORT:number=4000;
   wss:Wss;
   httpServer:HttpServer;
   constructor(http:HttpServer,ws:Wss){
      this.wss=ws;
      this.httpServer=http;
   }

   start() {
   const server=wss.getWss().listen(this.PORT, () => { console.log('Hi, I\'m WebSokcet server!!!') });
     server.on("request",this.httpServer.getServer());
   }
}

 export const server=new Server(httpServer,wss);
```

A3

```
import * as  http from 'http';
import * as  WebSocket from 'ws';
import {response} from './../webChat/types'
import { Observer } from './../wrappers/ObserverWrapper'
import { MessageDistributor, messageDistributor } from './../webChat/MessageHandler/MessageDistributor'

export class Wss implements Observer {

   private wss;
```

```
    private server;
    private msgDistribution: MessageDistributor;

    constructor(http, ws, msgDistribution: MessageDistributor) {
        this.msgDistribution = msgDistribution;
        this.server=http.createServer();
        this.initWss(ws);
        this.msgDistribution.registrObserver(this);
    }
    private messageHandler = async (socket: any, message: string): Promise<void> => {
        await this.msgDistribution.handelIncomingMsg(socket, message);
    }

    private closureHandler = async (socket: any): Promise<void> => {
        await this.msgDistribution.handelIncomingMsg(socket, JSON.stringify({ msg:"", roomID: "disconnect", type: "",
date: "", loginUser: "" }));
    }

    private connectHandle = (socket, req): void => {
        socket.on('message', this.messageHandler.bind(null, socket));
        socket.on('close', this.closureHandler.bind(null, socket));
    };

    private initWss(ws) {
        this.wss = new ws.Server({ server: this.server ,clientTracking: true });
        this.wss.on('connection', this.connectHandle)
    }
    private sendMessage(socket, obj:response) {
        switch (obj.mode) {
            case "sender":

                socket.send(JSON.stringify(obj.data));
                break;
            case "all":

                obj.data.clients.forEach(e => {
                    if(e.send!==undefined)
                    e.send(JSON.stringify({ type: obj.data.type, message: obj.data.message }))
                });
                break;
            default:
                console.log("error")
                break;
        }
    }

    update(args: Array<any>) {

        this.sendMessage(args[0], args[1]);

    }

    getWss(){
        return this.server;
    }

}

export const wss = new Wss(http, WebSocket, messageDistributor);
```

A4

```
import * as express from 'express';

import {apiRouter} from './../routes/api.router'
```

```
export class HttpServer {
   private server;
   constructor(app,apiRouter){
      this.server=app;
      this.initServer(apiRouter);
   }
   getServer(){
      return this.server;
   }
   private initServer(apiRouter):void{
      this.server.use(express.json({}));
      this.server.use(apiRouter);
   }

}

export const  httpServer=new HttpServer(express(),apiRouter);
```

A5

```
import * as  TelegramBot from 'node-telegram-bot-api'
import { Bot } from './bot'
import { IListener } from "./../classes/IServer"
import { CommandFactory } from "./Commands/CommandFactory"
import { Command } from "./Commands/Command"
import { CommandExecutor } from './CommandExecutor'
enum CHAT_STATUS {
   MESSAGE = 0,//reg //send //create_todo //create_task
   MESSAGE_CHAIN,
   KEYBOARD_CHAIN,

}
const PERIOD_ARRAY:Array<string[]>=[["никогда"],["каждый день"],["каждую неделю"],["каждый
месяц"],["каждый год"]];

export type CommandResponse = {
   fn: Function,
   chatId: string,
   commandType:string
}
type CHAT_STATE={
   wrapper: Function | null;
   state:Array<string>;
   commandType:string

}

const TOKEN: string = "1751041214:AAGNTL9PX2k0TWCUoj2pdTiS5Xr_5oeLoik";

export class TBot implements IListener, Bot {
   private chatState: Map<string, CHAT_STATE> = new Map();
   private executor: CommandExecutor;
   private token: string;
   private bot: any;

   constructor(token: string, tBot: any, executor: CommandExecutor) {
      this.token = token;
      this.createBot(tBot);
      this.executor = executor;

   }
   private createBot(bot: any): void {
      this.bot = new bot(this.token, {
         polling: {
            interval: 300,
```

```
            autoStart: true,
            params: {
                timeout: 10
            }
        }
    })
}
sendKeyboard(chatId: string, msg: string, keyboard: Array<Array<string>>) {
    this.bot.sendMessage(chatId, msg, {
        reply_markup: {
            keyboard: keyboard
        }
    })
}
closeKeyboard(chatId: string, msg: string): void {
    this.bot.sendMessage(chatId, msg, {
        reply_markup: {
          remove_keyboard: true
        }
    })
}
sendMessage(chatId: string, msg: string): void {
    this.bot.sendMessage(chatId, msg);
}
private parseText(text: string): { type: string, args: Array<string> } {
    const commandAndArgs: Array<string> = text.split(" ");
    const type: string = commandAndArgs[0].substring(0, 1) === "/" ? commandAndArgs.shift() : "notACommand";
    const args: Array<string> = commandAndArgs.filter(e => e !== " ");
    return { type, args };
}

handelMessage(response: boolean | CommandResponse) {
    if ((response as CommandResponse).commandType === undefined)
        return true;
    const res = response as CommandResponse;
   this.chatState.set(res.chatId,{wrapper: res.fn,state:[],commandType:res.commandType});
}

private async handelChainMsg(text:string,chatId:string){
    let chatState=this.chatState.get(chatId)
    switch(chatState.commandType){
        case "users":
            const index=chatState.state.indexOf(text);
            if(index===-1)
            {
                chatState.state.push(text);
                this.sendMessage(chatId,`пользователь ${text} добавлен \u2705`)
            }
                else
                 {
                    chatState.state.splice(index,1);
                    this.sendMessage(chatId,`пользователь ${text} удален \u274C`)
                }
                this.chatState.set(chatId,chatState);
            return;// не доходит до конца функции а завершается здесь
        case"disciprion":
        chatState.commandType="date";
        this.sendMessage(chatId, "ВВЕДИТЕ ДАТУ В ФОРМАТЕ гггг-мм-дд");
        break;
        case "date":
        chatState.commandType="time";
        this.sendMessage(chatId, "ВВЕДИТЕ ВРЕМЯ В ФОРМАТЕ чч:мм");
            break;
         case "time":
```

```
            chatState.commandType="period";
            this.sendKeyboard(chatId, "ВЫБЕРЕТЕ ПРИРОД ПОВТАРЕНИЯ",PERIOD_ARRAY);
             break;
          case "period":
            chatState.commandType="users";
            await this.executor.getCommandResult( "/send_users", [], chatId, "date")
             break;
         default: break;
      }
      chatState.state.push(text);
      chatState.wrapper.bind(null,text);
      this.chatState.set(chatId,chatState);
   }

   private handelChain(text: string, chatId: string) {
      switch(text){
         case "отмена":
            this.closeKeyboard(chatId, "ОТМЕНИТЬ");
            this.chatState.delete(chatId)
            break;
            case "далее":
               console.log(this.chatState.get(chatId));
               this.closeKeyboard(chatId, "THE END");
               const obj=this.chatState.get(chatId);
               this.chatState.delete(chatId);
               obj.wrapper(obj.state).execute();
               break;
               default:
                  this.handelChainMsg(text,chatId);
               break;
      }

   }

   start(): void {
      this.bot.on('message', async msg => {
         try {
            const { text, chat, date } = msg;
            if(!this.chatState.has(chat.id)){
            const { type, args } = this.parseText(text as string);
            this.handelMessage(await this.executor.getCommandResult(type, args, chat.id, date));
            }
            else
                  this.handelChain(msg.text, chat.id);

         }
         catch (e) {
            this.sendMessage(msg.chat.id, "ПАЦАНЫ, ЦЕНИТЕ МАТЬ!");
            console.log((e as Error).stack);
            console.log(e.message);
         }
      })

   };
}

export const tbot = new TBot(TOKEN, TelegramBot, new CommandExecutor(new CommandFactory()));
```

А6

```
import * as Mongoose from 'mongoose';

export class Storage {
   private database;
```

```
   private MONGO_URI: string =
"mongodb+srv://us:123@organizer.x9hju.mongodb.net/<dbname>?retryWrites=true&w=majority";
   constructor(mDb) {
      this.database = mDb;
   }
   async intitStorage() {
      try {
         await this.database.connect(this.MONGO_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true
         })
         return true;
      }
      catch (e) {
         console.log(e);
         return false;
      }
   }
   static async getItemWithOrConditionally(model, filds, fildsVal, fildsOr, fildsValOr, returnVals:string[] = []){
      try {
         const conditions = filds.map((e, i) => { return { [e]: fildsVal[i] } });
         const conditionsOr = fildsOr.map((e, i) => { return { [e]: fildsValOr[i] } });
         const test = await model.find({}, returnVals.reduce((a, val) => a + ' ' + val, " ")).or([{ $and: conditions }, { $and:
conditionsOr }])
         return test;
      }
      catch (e) {
         console.log(e);
         return false;
      }

   }

static async getItemWithConditionally (model, filds, fildsVal, returnVals = []){
   try {
      const conditions = filds.map((e, i) => { return { [e]: fildsVal[i] } });
      return await model.find({}, returnVals.reduce((a, val) => a + ' ' + val, " ")).and(conditions);

   }
   catch (e) {
      console.log(e);
      return false;
   }
}
static async getAllItems(model, returnVals = []){
   try {

      return await model.find({}, returnVals.reduce((a, val) => a + ' ' + val, " "));
   }
   catch (e) {
      console.log(e);
      return false;
   }
}
static async  getItemByID(model, id, returnVals= null) {
   try {
      return returnVals === null ?
         await model.findById(id) :
         await model.findById(id, returnVals.reduce((a, val) => a + ' ' + val, " ")).exec();
   }
   catch (e) {
      console.log(e);
      return false;
```

```
    }
}
static async  addItem (model, item, returnObj = false){
    try {
        const obj = new model({ ...item });
        await obj.save();
        return returnObj ? obj : true;
    }
    catch (e) {
        console.log(e);
        return false;
    }
}
static async pushItem(model, query, param, val){
    try {
        const updateDocument = { $push: { [param]: val } };
        await model.updateOne(query, updateDocument)
        return true;
    }
    catch (e) {
        console.log(e);
        return false;
    }
}
static async deleteItem  (model, query, value)  {
    try {
        await model.deleteOne({ [query]: value });
        return true;
    }
    catch (e) {
        console.log(e)
        return false;
    }
}
static async setItem(model, query, param, val){
    try {
        const updateDocument = { $set: { [param]: val } }
        await model.updateOne(query, updateDocument);
        return true;
    }
    catch (e) {
        console.log(e);
        return false;
    }
}
static async getIncludesItems(model, valName, val, returnVals:string[] =[]){
    try {

        return returnVals.length === 0 ?
            await model.find({ [valName]: { $in: val } }) :
            await model.find({ [valName]: { $in: val } }, returnVals.reduce((a, val) => a + ' ' + val, " "));
    }
    catch (e) {
        console.log(e);
        return false;
    }
}
}

export const storage = new Storage(Mongoose);
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись инициалы, фамилия
« 12 » 06 2021 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01- “Информатика и вычислительная техника”
код и наименование специальности

Web-приложение совместный органайзер
тема

Руководитель	11.06.21 подпись, дата	доцент, канд. техн. наук должность, ученая степень	С.Н. Титовский инициалы, фамилия
Выпускник	10.06 подпись, дата		М.Б. Субботкин инициалы, фамилия
Нормоконтролер	11.06.21 подпись, дата	доцент, канд. техн. наук должность, ученая степень	С.Н. Титовский инициалы, фамилия

Красноярск 2021