

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Кафедра информационных систем и программной инженерии

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту по дисциплине
"Технологии разработки мобильных приложений"
на тему
Разработка прототипа мобильного приложения
«Многозадачный календарь»

Выполнил:

студент гр. ПРИ-120
Парахин К.В.

Приняла:

преподаватель кафедры
ИСПИ
Петрова А.И.

Владимир, 2023

Аннотация

В данном курсовом проекте производилась разработка прототипа мобильного приложения «Многозадачный календарь»

Проект состоит из 5 основных этапов, включающих в себя основные три этапа, такие как: «Анализ предметной области», «Проектирование системы», «Разработка клиент-серверного приложения», «Тестирование системы», «Развертывание мобильного приложения».

Разработка клиент-серверного приложения подразумевает реализацию в отдельных репозиториях серверного и клиентского приложений, а также проведение их совместного взаимодействия.

Развертывание мобильного приложения подразумевает под собой сборку и запуск готовых приложений в одном образе.

Также курсовой проект включает в себя задачу попытки выкладывания приложения в магазине мобильных приложений.

Реализованная система может применяться для таких целей, как, например, создание и хранение мероприятий, групп, задач и отчетов пользователей, просмотр и редактирование календаря с мероприятиями, получение уведомлений и напоминаний о начале событий.

Реализованная система не использует разделения по ролям пользователей.

Курсовой проект представлен на 87 страницах, основных рисунков - 29, использованных источников – 5, приложений – 3, иллюстрационный материал на 2 листах формата А1.

ABSTRACT

In this course project, a prototype of the mobile application "Multitasking Calendar" was developed

The project consists of 5 main stages, including the main three stages, such as: "Domain analysis", "System Design", "Client-server application development", "System testing", "Mobile application deployment".

The development of a client-server application implies the implementation of server and client applications in separate repositories, as well as their joint interaction.

The deployment of a mobile application implies the assembly and launch of ready-made applications in one image.

Also, the course project includes the task of trying to upload an application in the mobile app store.

The implemented system can be used for such purposes as, for example, creating and storing events, groups, tasks and user reports, viewing and editing a calendar with events, receiving notifications and reminders about the beginning of events.

The implemented system does not use separation by user roles.

The course project is presented on 87 pages, the main figures - 29, the sources used – 5, appendices – 4, illustrative material on 2 sheets of A1 format.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	5
1.1. Описание предметной области.....	5
1.2. Сравнительный обзор аналогов.....	6
1.3. Описание набора функций системы	11
1.5. Словарь предметной области	12
2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ	14
2.1. Функциональная декомпозиция системы	14
2.2. Описание состава данных.....	17
2.3. Требования к разрабатываемой системе.....	20
3. РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ	23
3.1. Выбранный вариант разработки программной системы.....	23
3.2. Описание взаимодействия приложений.....	25
3.3. Реализация серверного приложения	28
3.4. Описание структуры серверного приложения.....	29
3.5. Спецификация API.....	31
3.6. Описание разработки клиентского приложения	35
3.7. Интерфейс мобильного приложения.....	38
4. ТЕСТИРОВАНИЕ СИСТЕМЫ.....	57
4.1. Тестирование запросов к REST API серверного приложения.....	57

					ВлГУ.09.03.04.25 ПЗ					
Изм.	Лист	№ докум.	Подп.	Дата	Разработка прототипа мобильного приложения «Многозадачный календарь» Пояснительная записка			Лит.	Лист	Листов
Разраб.		Парахин К.В.						У	2	87
Пров.		Петрова А.И.						ПРИ-120		
Н. контр.										
Утв.										

4.2. Тестирование приложения с помощью тестовых кейсов	60
5. РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ.....	66
5.1. Развертывание сервера.....	66
5.2. Развертывание мобильного приложения	68
6. РАЗРАБОТКА ЛОГОТИПА ПРИЛОЖЕНИЯ	70
7. ВЕДЕНИЕ РЕПОЗИТОРИЕВ ПРОЕКТА	71
ЗАКЛЮЧЕНИЕ	73
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	74
ПРИЛОЖЕНИЕ А. СПЕЦИФИКАЦИЯ API (ПРОДОЛЖЕНИЕ).....	75
ПРИЛОЖЕНИЕ Б. КОД СЕРВЕРНОГО ПРИЛОЖЕНИЯ	79
ПРИЛОЖЕНИЕ В. КОД КЛИЕНТСКОГО ПРИЛОЖЕНИЯ	84

ВВЕДЕНИЕ

Для начала выполнения разработки прототипа информационной системы «Многозадачный календарь» требуется выполнить следующие шаги по проектированию:

- описать предметную область
- рассмотреть основные аналоги, провести их сравнение
- выделить основные термины предметной области, сформировать словарь предметной области

предметной области

- выделить ее основные функциональные требования, с помощью основных

UML-диаграмм описать составы данных и категории рассматриваемой системы

- провести описание основных прецедентов пользователей, сущности

системы

- провести динамическое моделирование системы

- выбрать средства разработки и основные фреймворки, требующиеся для реализации функционала

- выполнить разработку клиент-серверного мобильного приложения в репозитории (или различных репозиториях), провести тестирование полученного приложения.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		4

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

Предметная область была названа: «Многозадачный календарь» (потому что объединяет функции как личного календаря, так и делового ToDo календаря, который интегрируется с действиями пользователей в группе).

«Почему многозадачный календарь действительно полезен для современного человека?»

Ведение своего личного, а также рабочего календаря является очень актуальной задачей для школьников, студентов, рабочих специалистов и остальных групп людей. Жизнь каждого из этих людей каждый день состоит из множества занятий, встреч, поездок и большого «груза» самостоятельной работы.

Чтобы упростить жизнь современным людям, нужно дать им возможность учета своей активности и занятости, а именно: вести свои занятия, планировать мероприятия и события, а также задачи и самостоятельную работу, которые требуется выполнить в определенный срок. Особенно удобно это будет сделать в виде мобильного приложения, причем вход в свою учетную запись можно совершать не только с одного устройства, а с любого, которое окажется под рукой.

Кроме того, оно сможет подсказать о начале мероприятий, чтобы пользователи не забывали на них присутствовать, в «дружелюбной» форме напоминать о выполнении задач и поставленных целей, а также формировать отчеты о занятости отдельного пользователя и достигнутых им результатов деятельности в различных сферах активностей.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		5

1.2. Сравнительный обзор аналогов

Для того, чтобы погрузиться в эту тему необходимо провести сравнительный анализ нескольких сервисов, предоставляющих функционал личного многозадачного календаря.

Мною были выделены 4 основных календарных сервиса: Google Calendar, Teamup, TimeBlocks, Microsoft Outlook calendar.

Краткое описание каждого из этих сервисов:

1) Google Calendar — сервис для планирования встреч, событий и дел, разработанный компанией Google. Google Календарь стал доступен в бета-версии 13 апреля 2006 года.

Очень часто используется в IT-компаниях (например, в той, которой я сейчас работаю, удобен, прост и многозадачен). Дает возможность создавать звонки, встречи, планировать некоторые задачи и получать напоминания о мероприятиях.

2) Календарь Teamup — это простое цифровое приложение для совместного использования календаря, которое упрощает и оптимизирует общение вашей команды. Это приложение для организации и планирования является простым, безопасным и масштабируемым. Это приложение для организации и планирования позволяет пользователям обмениваться планами, событиями, расписаниями и обновлениями статуса. Вы получаете полную видимость того, кто и что делает в команде, а также автономность, создавая индивидуальный доступ к календарю для отдельных лиц и групп.

3) TimeBlocks — это многоплатформенное приложение-календарь, которое позволяет создавать структурированные повестки дня для обсуждений и встреч в команде и помогает придерживаться планов и расписаний. Эта альтернатива Google - календарю имеет простой и элегантный интерфейс, который предоставляет вам подробное представление о ваших событиях.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		6

4) Календарь Microsoft Outlook — это хорошая альтернатива календарю Google, которая предлагает достаточное количество инструментов для удовлетворения потребностей большинства людей в календаре и планировании. Это бесплатное приложение имеет простой, незагроможденный интерфейс, который намного лучше многих платных альтернатив, доступных в Магазине Windows. Используя календарь Microsoft, вы можете создавать встречи и события, организовывать собрания, просматривать групповые расписания, просматривать календари, отправлять календари кому-либо по электронной почте и т.д.

На рисунке 1 ниже я привел скриншот пользования Google – календарем (как наиболее популярного и знакомого мне сервиса):

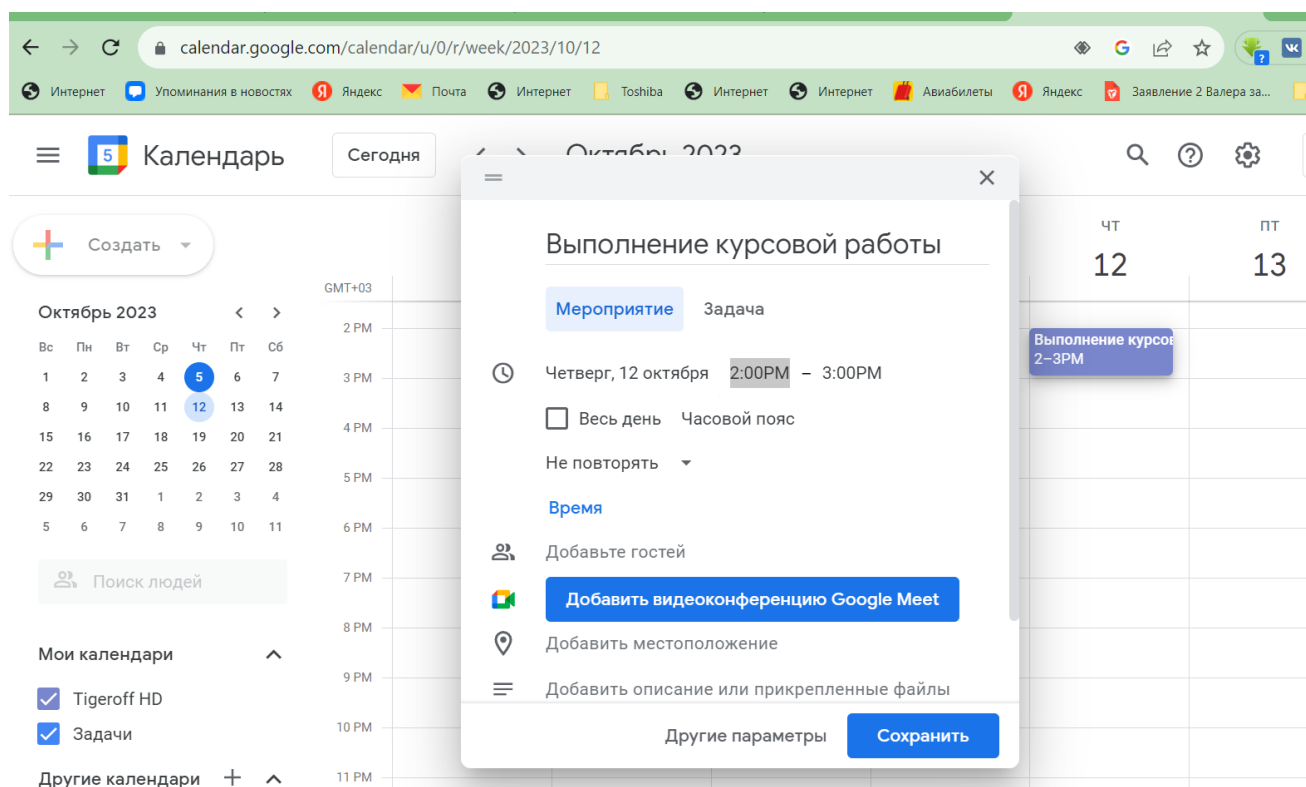


Рисунок 1. Пользование Google – календарем

Затем мною были выделены некоторые критерии, по которым я буду проводить обзор и сравнение этих сервисов:

Критерии	Краткое описание
Дизайн	Оформление контента, совокупность всех графических элементов на веб-странице сервиса
Удобство использования	Понятная и доступная структура сервиса, хорошая видимость и оформление основных элементов сервиса
Функциональность календарного сервиса	<p>Практичность использования календарного сервиса, наличие на нем всех нужных опций и возможностей:</p> <ul style="list-style-type: none"> - создание различных типов мероприятий - добавление задач и целей - возможность рассылки уведомлений и напоминаний
Динамичность работы	Скорость работы сервиса, то есть постоянное обновление информации на нем, адаптивность и скорость работы
Интеграция	Возможность интеграции сервиса планирования мероприятий в другие программные продукты. Автоматическое определение и подхватывание нового контента

Таблица 1. Сравнительные критерии оценки сервисов

Ниже приведена таблица с сравнением четырех сервисов – календарей по выделенным в таблице 1 критериям пользователя-эксперта (то есть меня самого).

Эксперт выставил свои оценки по основным 5 критериям, таким как: дизайн, удобство использования, функциональность календарного сервиса, динамичность работы, интеграция – по пятибальной шкале (где 1 – это очень плохая оценка, когда продукт не соответствует описанию, 2 – плохое качество и реализация, 3 – удовлетворительная реализация и недостаточный функционал, 4 – неплохой показатель при некоторых недостатках, 5 – идеальное значение показателя).

Ниже получились следующие оценки (так как сервисы достаточно похожие, то и оценки оказались тоже достаточно схожи).

Таблица 2. Расчет средней оценки сервисов-календарей по выделенным критериям

Критерии /Сервисы	Google - calendar	Teamup	TimeBlocks	Календарь Microsoft Outlook
URL-ссылка	https://calendar.google.com/calendar	https://www.teamup.com/	https://timeblocks.com/	https://www.microsoft.com/en-us/microsoft-365/outlook
Дизайн	5	5	4	5
Удобство использования	5	4	4	4
Функциональность календарного сервиса	5	5	4	5
Динамичность работы	5	5	5	5
Интеграция	5	4	3	5
Средняя оценка	5	4,6	4	4,8

Таблица 2. Сравнительная таблица с итоговыми средними баллами оценок сервисов

Итоги и выводы по проведенному сравнительному анализу:

Наивысшую среднюю оценку по итогу сравнения набрал сервис Google Calendar (средняя оценка 5), затем с небольшим отставанием идут сервис календаря Outlook от компании Microsoft (4,8) и свободнораспространяемый календарь TimeUp (оба набрали среднюю оценку 6). Среднюю оценку на уровень хуже получил сервис TimeBlocks (средняя оценка 4,7).

При этом практически все сервисы-календари оказались достаточно удобны, красивы (то есть имеют приятный и утонченный дизайн), информативны и динамичны. Функциональность у них тоже достаточно похожа и ориентирована в основном на пользование специалистами внутри компаний (для планирования в основном рабочих мероприятий).

Так как по итогу сравнения наибольшую оценку набрал сервис календаря от компании Google – то как ориентир для разработки собственного многозадачного календаря я возьму именно его (заостряя внимание на внедрение интеграции не только рабочих, но и учебных, а также личных мероприятий – о чем я говорил выше в разделе «Почему многозадачный календарь действительно полезен для современного человека?»)

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		10

1.3. Описание набора функций системы

Основные функции разрабатываемой системы, определяющие границы предметной области:

- зарегистрироваться в системе (пользоваться функционалом только имея учетную запись)
- войти в систему (авторизоваться в приложении с любого другого мобильного устройства)
- создавать группы пользователей, выбирая их тип (рабочая, учебная, личная) – выбирать параметры интеграции пользователей в своей группе
- планировать мероприятия для себя или для группы пользователей, к которой он принадлежит
- рассылать уведомления на устройства приглашенных пользователей
- получать push-напоминания о скором начале ближайших мероприятий
- просматривать свой календарь, удалять из него мероприятия, редактировать их
- добавлять задачи в отдельном разделе приложения, прикреплять их к своим событиям в календаре (либо к другим пользователям)
- получать отчет от календаря за определенный период времени (о том, сколько задач осталось выполнить, сколько времени было проведено в личных/групповых мероприятиях, сколько было отдыха/учебы/работы)

1.4. Описание категорий пользователей системы и прецедентов

Система не включает в себя явное разделение пользователей по ролям, то есть все пользователи после регистрации получают статус обычного пользователя, по умолчанию.

Регистрация не требует подтверждения отдельными ответственными лицами (все делается автоматически).

Модерация в календаре тоже никакая не планируется проводиться, поэтому отдельно роль модератора в рамках системы не выделяется.

Но пользователи могут создавать группы (в рамках рабочего или учебных процессов) – в данных группах может быть назначен свой собственный администратор, который имеет возможности назначать задачи другим пользователям, а также планировать мероприятия, выбирая и удаляя с него участников.

Обычно, администратор всегда присутствует в рамках учебной группы (лектор) или рабочей деловой группы (менеджер проекта).

1.5. Словарь предметной области:

Календарь (calendar) – набор событий, задач и мероприятий пользователя в рамках определенного промежутка времени. Графически показывается в виде таблицы, сгруппированной по датам и часовым промежуткам.

Задача (task) – некоторая абстрактная деятельность (или цель - goal), которую пользователь хочет (или должен) выполнить в определенный срок (или в рамках какого-то мероприятия)

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		12

Goal (цель) – поставленная задача, отражающая развитие пользователя в рамках деятельности в течение какого-то срока.

Мобильное устройство (mobile device) – смартфон с операционной системой Android, имеющий выход в сеть Интернет и обладающий совместимостью к используемому SDK (далее упоминаться будет именно понятие «мобильное устройство»)

Мероприятие (meet) – некоторая совместная деятельность группы пользователей (или одного человека), которое планируется провести в определенное время (оно создается в календаре).

Видеоконференция (или по-простому «созвон» - call) – мероприятие, которое проводится в рамках удаленного аудио или видео - разговора в какой-то программе (например, Google Meet, Zoom, Discord).

One-To-One – созвон двух пользователей в рамках обсуждения локальных задач и достижения целей.

StandUp – созвон множества пользователей из одной деловой (обычно рабочей) группы с целью обсуждения задач на небольшой промежуток времени. Проводится регулярно, может быть циклично запланирован.

Дистанционная лекция – мероприятие учебного типа, проводящееся среди лектора и студентов университета. В отличие от некоторых рабочих мероприятий, в рамках этого события может быть назначен единый администратор для группы.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Функциональная декомпозиция системы

Основные прецеденты:

- Управлять своим аккаунтом (пользователь системы)
 - Зарегистрироваться
 - Авторизоваться
- Создать группу пользователей
 - Выбрать тип группы (учебная, рабочая, личная)
 - Добавить участников в группу
- Просматривать визуальный календарь мероприятий
 - Просматривать свой календарь (на главной странице)
 - Просматривать календарь пользователей в группе
- Создавать задачи
 - Распределять задачи в группе по исполнителям
 - Прикреплять задачи к определенным мероприятиям
- Запланировать мероприятие
 - Выбрать тип мероприятия
 - Выбрать время и продолжительность
 - Добавить пользователей из группы на мероприятие
 - Отослать уведомления о приглашении на мероприятие
- Получать уведомления о мероприятиях
 - Получать уведомление о приглашении на мероприятие

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		14

- Получать дополнительно напоминание о наступлении мероприятия
(за 10 мин до начала, например)

- Получать отчеты своем календаре (опционально, по событиям или задачам)

Диаграмма прецедентов в нотации UML изображена ниже на рисунке 2:

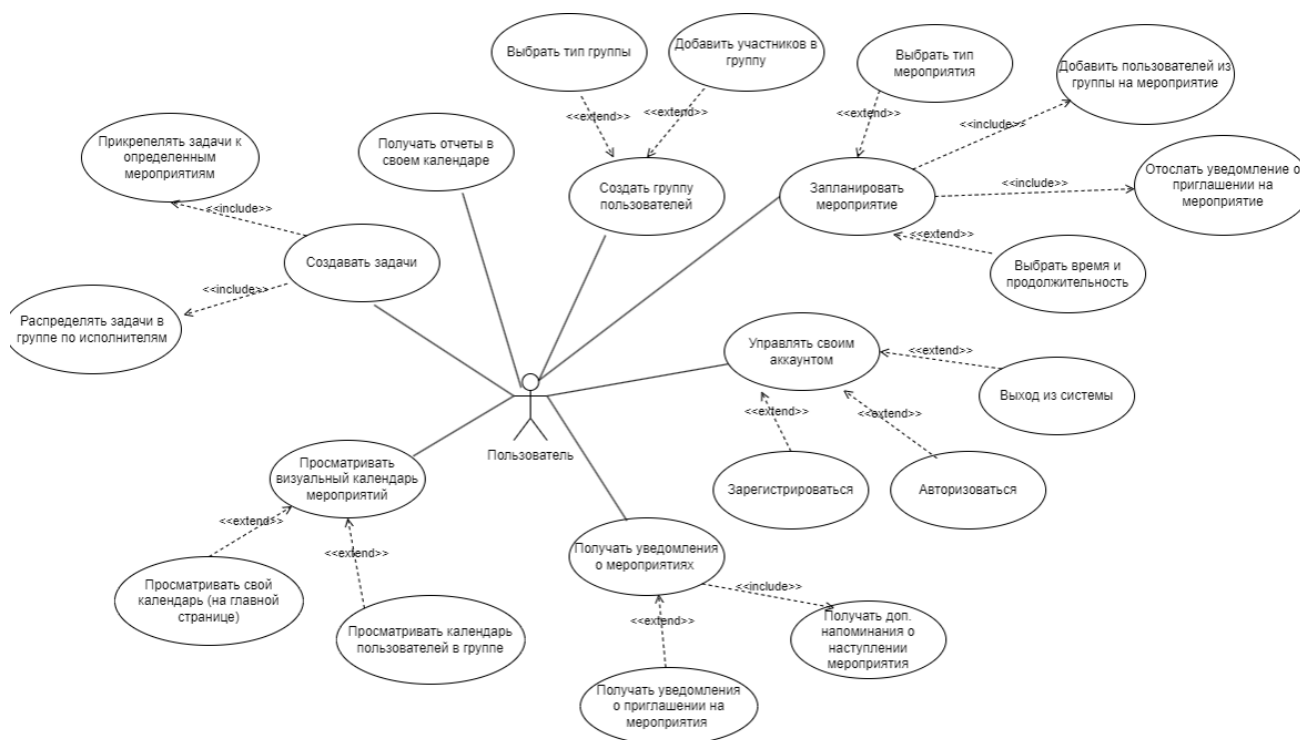


Рисунок 2. Диаграмма прецедентов

Описание некоторых прецедентов

Прецедент «Зарегистрироваться»

Предусловие: Пользователь решил впервые воспользоваться мобильным приложением «Многофункциональный календарь»

Действующее лицо: пользователь

Основной поток: пользователь открывает приложение, нажимает кнопку «Зарегистрироваться», вводит учетные данные для регистрации, затем получает

ссылку на указан адрес электронной почты, по которой он должен перейти – для завершения процесса создания новой учетной записи

Альтернативный поток:

Пользователь не перешел по отправленной ссылке в течение 10 минут. Тогда процесс создания новой учетной записи отменяется и промежуточные данные удаляются.

Пользователь уже зарегистрирован в системе. Тогда при попытке зарегистрироваться с уже сохраненными в системе учетными данными, он будет уведомлен о наличии зарегистрированной учетной записи с введенными данными.

Постусловие: При наличии зарегистрированной учетной записи, пользователь будет перенаправлен на страницу авторизации.

Прецедент «Авторизоваться»

Предусловие: Пользователь решил воспользоваться функционалом системы при уже имеющейся учетной записи, то есть пользователь уже зарегистрирован в системе.

Действующее лицо: пользователь

Основной поток: пользователь открывает мобильное приложение, на главной странице видит раздел для авторизации, затем вводит свои данные и получает доступ к своей личной учетной записи и к личному кабинету.

Альтернативный поток: Пользователь ещё не зарегистрирован в системе. Тогда при попытке авторизоваться с некоторыми данными, он будет уведомлен об отсутствии требуемой учетной записи.

Постусловие: При отсутствии зарегистрированной учетной записи, пользователь будет перенаправлен на страницу регистрации.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		16

2.2. Описание состава данных

Выделим следующие абстракции данных (сущности системы):

- User (пользователь с некоторыми учетными данными)
- Group (часть некоторого сообщества, к которой прикреплен пользователь)
- Event (событие (или мероприятие), предполагающее встречу/созвон – или какую-то другую совместную деятельность пользователей из одной группы)
- Task (задача, поставленная на исполнение администратором группы другому пользователю (или себе))
- Report (отчет о мероприятиях, событиях, задачах в календаре пользователя за определенный промежуток времени).

- Calendar (составная сущность – представляет собой отображение (или правильнее сказать группировку) события и задач пользователя по промежуткам времени). Планируется создать отдельную активность для отображения своего календаря (или календаря любого другого пользователя из той же группы). На диаграмме классов не отображается (так как создается средствами ORM на уровне SQL базы данных – для связи «многие ко многим»).

Дополнительно используются перечисления GroupType (тип группы: рабочая, учебная, личная), TaskType (какой – то абстрактный тип задачи: например, постановка цели (goal), присутствие на встречах, выполнение задания), EventType (тип события: Personal (Solo), OneToOne, StandUp, Meet и так далее).

Диаграмма классов (уровня проектирования) системы в нотации UML представлена ниже на рисунке 3:

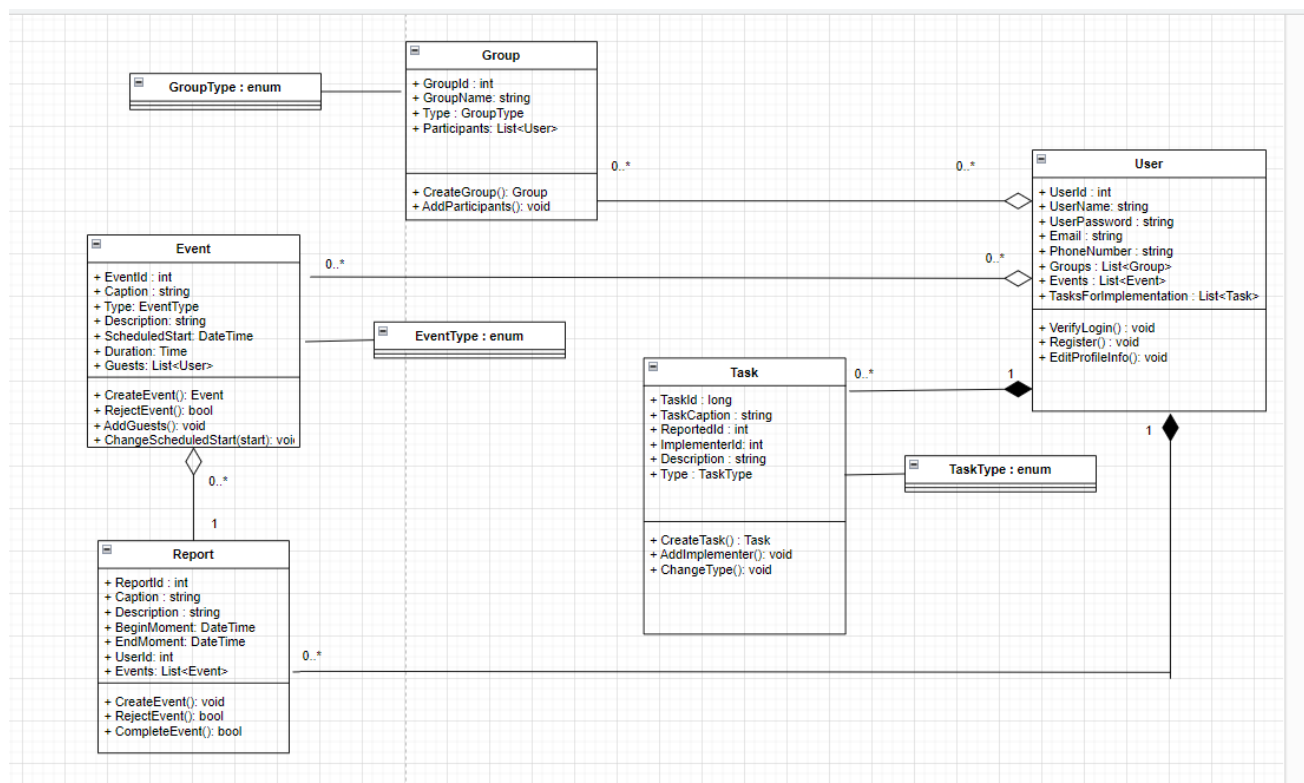


Рисунок 3. Диаграмма классов уровня проектирования

Диаграмма схемы базы данных в нотации ER представлена ниже на рисунке 4:

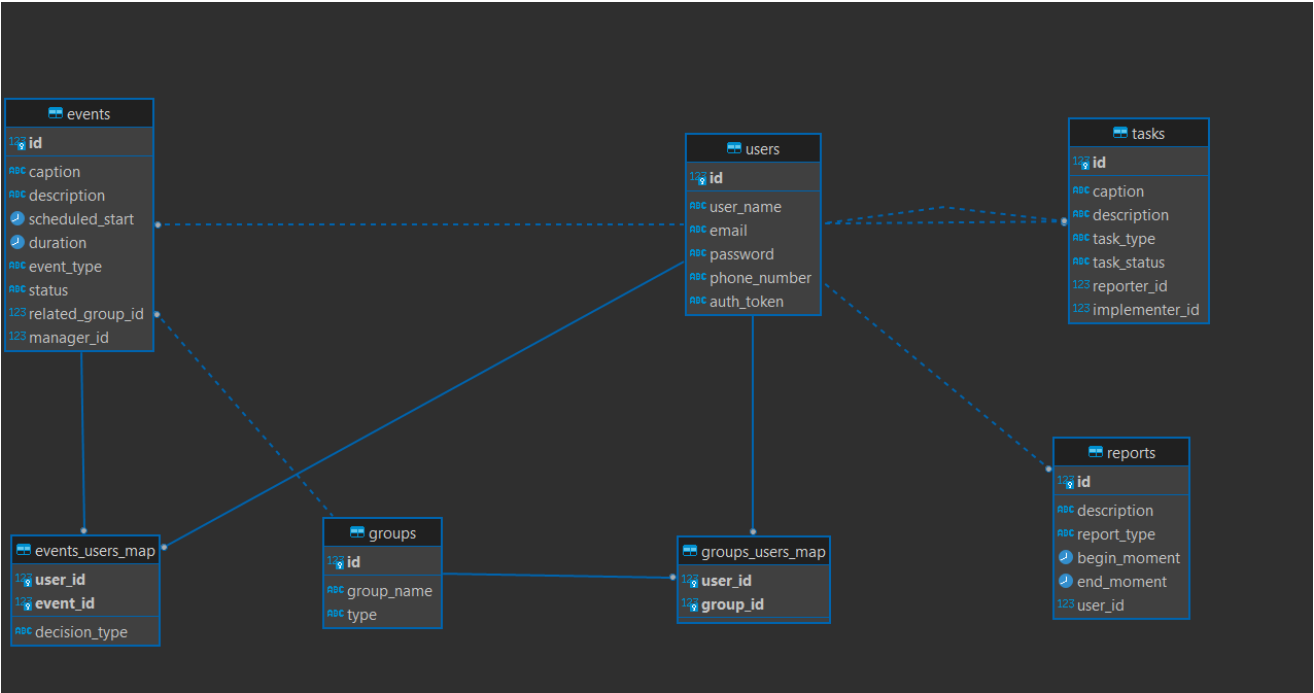


Рисунок 4. Схема базы данных

2.3. Требования к разрабатываемой системе

Выделим основные категории требований, которые необходимо предъявить к анализируемой выше разрабатываемой системе:

1) Функциональные требования

Список функциональных требований, минимально необходимых для прототипа мобильного приложения:

1.1) Функция 1

1.1.1) Название - возможность использовать авторизацию пользователей

1.1.2) Описание - пользователь получает доступ к функционалу на сервере с помощью специального сгенерированного токена, получаемого после авторизации (после входа в систему). Используется серверная аутентификация.

1.1.3) Назначение – предозначить доступ только запросам от авторизованных пользователей.

1.1.4) Зависимости – фреймворк ASP.NET Core 5, Google JWT Bearer Authentication.

1.2) Функция 2.

1.2.1) Название – осуществление регистрации пользователя

1.2.2) Описание – возможность регистрации пользователя в системе

1.2.3) Назначение – возможность сохранять учетную запись пользователя в базе данных и использовать ее повторно для доступа к личным данным.

1.3) Функция 3.

1.3.1) Название – подтверждение создание новой учетной записи

1.3.2) Описание – возможность подтверждения регистрации новой учетной записи пользователя путем подтверждения учетной записи через ссылку, высылаемую на указанный адрес электронной почты

1.3.3) Назначение – верифицировать используемый в настройках учетной записи адрес электронной почты

1.4) Функция 4.

1.4.1) Название – возможность распределения пользователей по группам

1.4.2) Описание – возможность создания пользователем групп различных типов (рабочих, учебных, личных), добавления в него пользователей и разграничения прав и прав на просмотр чужих мероприятий.

1.4.3) Назначение – сохранение групп с пользователями для создания общих мероприятий и назначения задач на исполнение участникам группы

1.5) Функция 5.

1.5.1) Название – возможность планирования пользовательских мероприятий

1.5.2) Описание - возможность создавать мероприятия, приглашать туда других пользователей (мероприятия будут проводиться во внешних системах проведения видеоконференций – например, Google Meet или Discord)

1.5.3) Назначение – сохранение групповых мероприятий с ссылками на проведение их в системах видеоконференции

1.6) Функция 6.

1.6.1) Название – возможность создания пользовательских задач

1.6.2) Описание – возможность прикреплять к пользователям задачи, синхронизировать их с событиями в их календарях

1.6.3) Назначение – сохранение пользовательских задач в базе данных

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		21

1.7) Функция 7

1.7.1) Название – возможность получать отчеты

1.7.2) Описание – возможность создавать отчеты по задачам/мероприятиям (для анализа занятости пользователя и его успехов по выполнению намеченных планов)

1.7.3) Назначение – просмотр и сохранение пользовательских отчетов в базе данных

1.8) Функция 8.

1.8.1) Название – получать уведомления о мероприятиях

1.8.2) Описание – возможность организации системы уведомлений и напоминаний (о приглашении на групповое мероприятие, приближение начала данного мероприятия) для каждого пользователя-участника мероприятия

1.8.3) Назначение – получение mail-сообщений на свой адрес электронной почты

1.8.4) Зависимость – библиотека System.NET.Mail.Kit и средства работы с SMTP - сервером

2) Нефункциональные требования (то есть требования, определяющие свойства, которые система должна демонстрировать):

2.1) Производительность - трафик запросов (не менее 10000 одновременных запросов от разных пользователей в небольшой промежуток времени)

2.2) Надёжность (система работает 24 на 7 и предоставляет возможность для работы с ней)

2.3) Платформенно-ориентированность (возможность запускать приложение на операционных системах семейства Android – на устройствах с API не ниже API 31, то есть ОС Android 11)

2.4) Масштабируемость системы.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		22

3. РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ

3.1. Выбранный вариант разработки программной системы

При разработке данного мобильного клиент-серверного приложения планируется использовать следующий список основных библиотек и фреймворков:

1) Для разработки серверной части приложения использовать библиотеку ASP.NET Core на платформе .NET 7.

Выбор данной платформы связан с ее удобством, наличием всего необходимого функционала для создания Restful WEB-API приложения, а также моими личными предпочтениями и опытом разработки.

ASP.NET Core – новейший фреймворк для разработки веб-приложений на платформе .NET, который предоставляет быстрые и продвинутое решения для создания постоянно обновляемых и улучшаемых систем, устойчивых к рефакторингу.

2) Для разработки клиентской части приложения использовать фреймворк Flutter.

Выбор данного фреймворка связан с его популярностью на рынке, удобством и простотой использования, а также наличием огромного количества плагинов и виджетов – благодаря которым возможно воплотить все основные визуальные решения с минимальными временными затратами. Кроме того, Flutter разрабатывается и поддерживается компанией Google – а так как я выбрал после сравнительного обзора аналогов именно продукт Google Calendar – то это имело решающее значение при выборе целевой платформы.

3) Для работы с БД использовать ORM – фреймворк Entity Framework и СУБД Postgre SQL.

Выбор в сторону Entity Framework был сделан в результате выбора реализации серверной части приложения на платформе .NET и желания использовать доменный подход к работе с моделями. Entity Framework является одним из продуктов компании Microsoft, постоянно обновляется и поддерживается.

Выбор в сторону Postgre SQL был сделан из-за популярности данной СУБД на рынке, а также удобством использования: как с точки зрения взаимодействия со стороны кодовой базы и ее поддержкой провайдером Entity Framework, так и со стороны простоты пользовательского интерфейса приложения pgAdmin – для просмотра состояния таблиц баз данных в ручном режиме.

Также планируется использовать следующие прикладные инструменты разработки и тестирования:

1) Для разработки серверного приложения планируется использовать IDE Microsoft Visual Studio – как наиболее удобное, популярное, а также свободно распространяемое программной средство разработки приложений на .NET.

2) Для разработки мобильного клиентского приложения планируется использовать IDE Android Studio – как приложение, предложенное в качестве основного для обучения работы с мобильными приложениями, а также свободно распространяемое компанией Jet Brains.

3) Для тестирования REST API HTTP-запросов планируется использовать приложение Postman – наиболее популярное и удобное приложение, используемое для задач тестирования веб-приложений.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		24

3.2. Описание взаимодействия приложений

Ниже на рисунке 5 представлена диаграмма развертывания описанной выше программной системы:

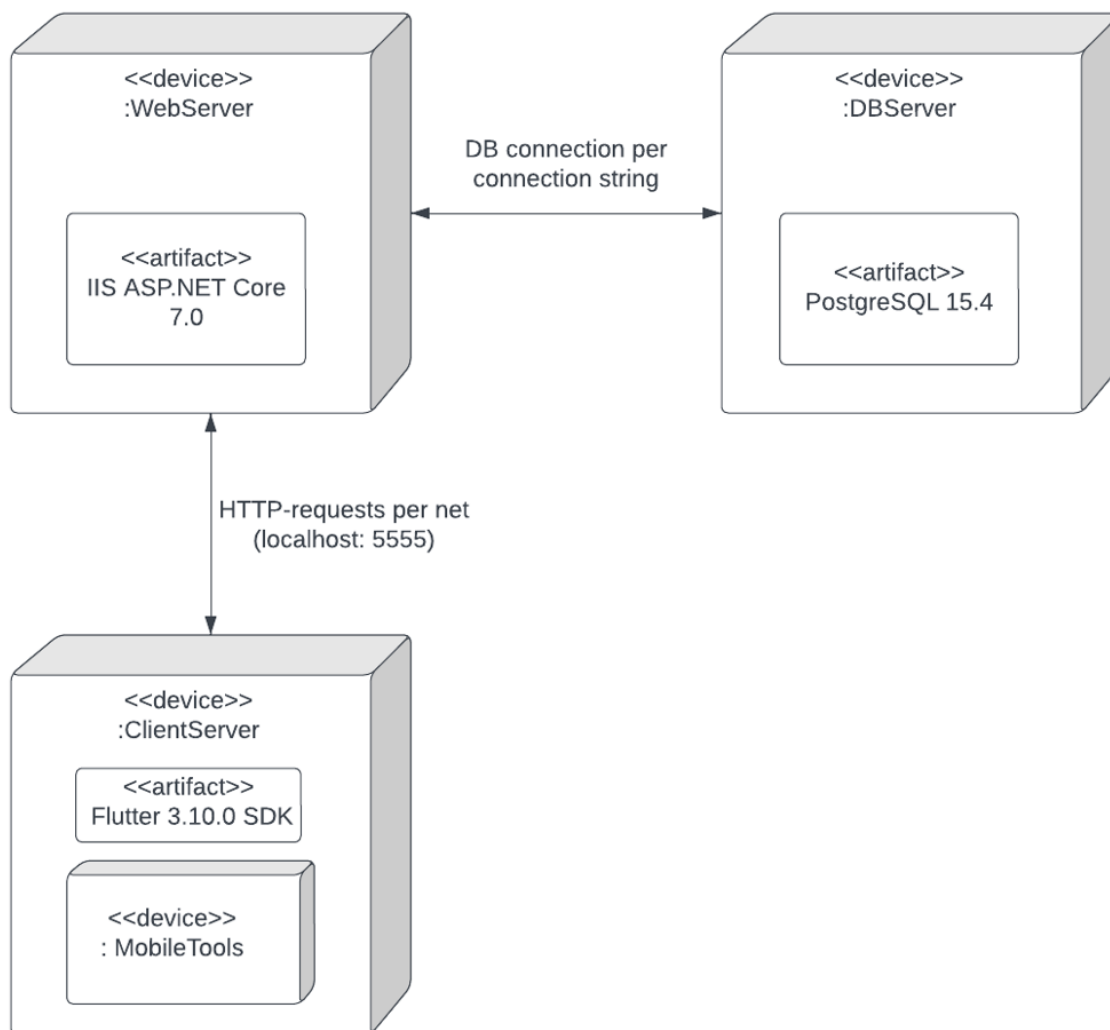


Рисунок 5. Диаграмма развертывания ПС

Построение алгоритма работы приложения

Ниже построим схему одного из основных алгоритмов приложения – схема процесса авторизации пользователя в системе.

Построение схемы работы алгоритма (алгоритм регистрации пользователя в системе представлен ниже на рисунках 6.1 – 6.2):

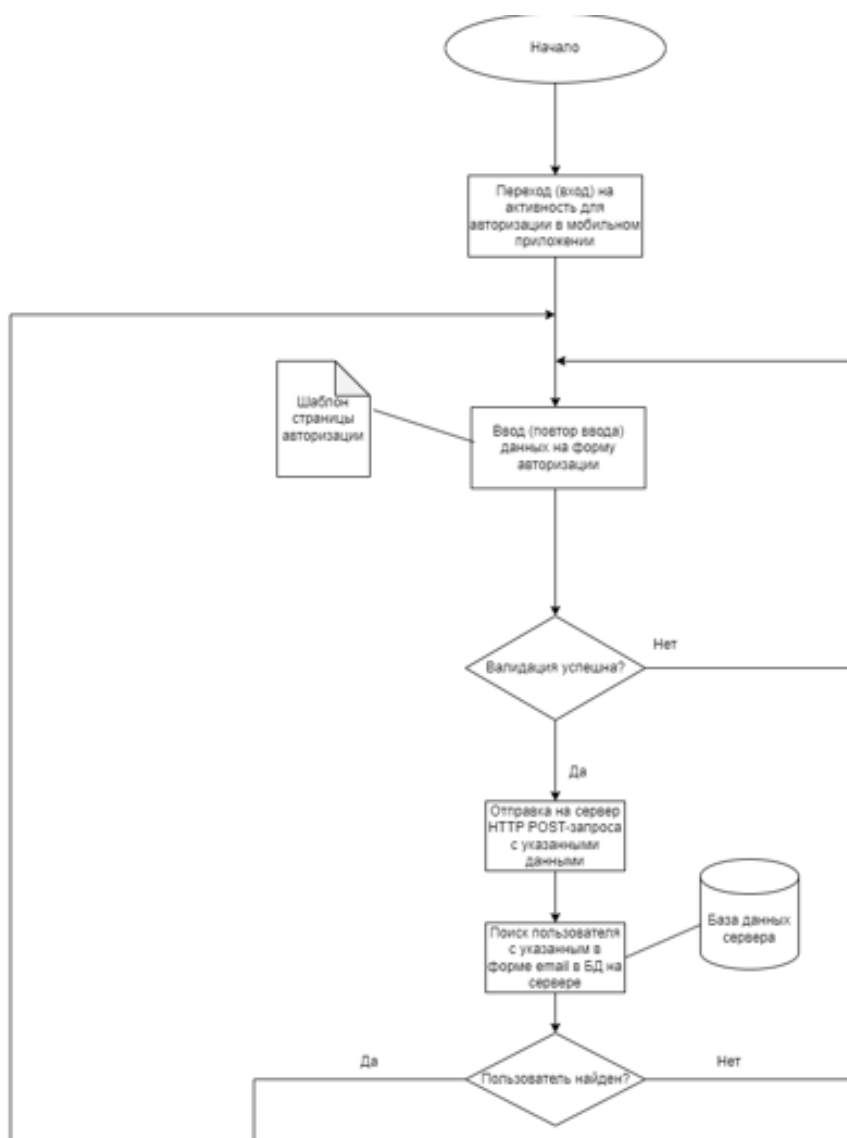


Рисунок 6.1. Схема алгоритма авторизации пользователя (часть 1)

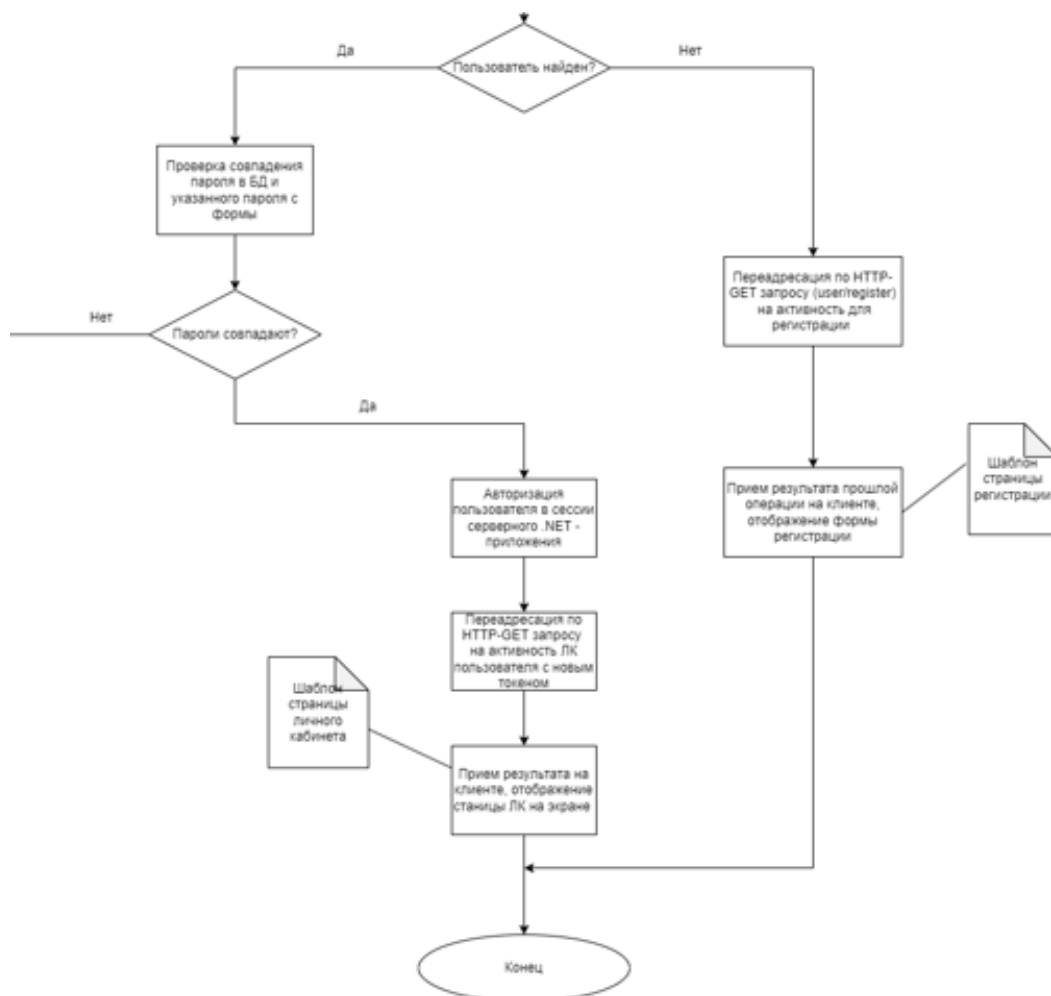


Рисунок 6.2. Схема алгоритма авторизации пользователя (часть 2)

3.3. Реализация серверного приложения

Используемые технологии

В качестве платформы для разработки серверной части приложения был выбран ASP.NET Core благодаря его высокопроизводительности и кроссплатформенности.

В качестве СУБД была выбрана PostgreSQL — свободная объектно-реляционная система управления базами данных. Существует в реализациях для множества UNIX-подобных платформ, включая AIX, различные BSD-системы, HP-UX, IRIX, Linux, macOS, Solaris/OpenSolaris, Tru64, QNX, а также для Microsoft Windows.

Аутентификация была использована на основе Identity от ASP.NET Core – с моими собственными доработками (генерацией токена на основе токеном JWT и проверкой токенов на уровне middleware серверного приложения).

Используемые библиотеки

Newtonsoft.Json – библиотека для сериализации и десериализации объект в JSON.

Serilog – сторонняя библиотека для логгирования приложения (как в консоли, так и в Google Cloud)

Entity Framework Core - современный модуль сопоставления "объект — база данных" для .NET.

xUnit - инструмент тестирования (тестирование покрывает только основные интеграционные сценарии – типа сценария регистрации. И тестируется в принципе работоспособность приложения – тест на hc (health checks)).

3.4. Описание структуры серверного приложения

Разработка серверного приложения производилась на платформе ASP.NET Core. Для создания проекта под веб-приложение использовался шаблон проекта под названием ASP WEB-API. Для остальных проектов использовался шаблон библиотеки классов.

Были созданы следующие проекты:

- Contracts (проект, в котором располагаются основные транспортные модели, необходимые для десериализации моделей запросов к API и сериализации моделей ответов данного API)

- Logic (проект, в котором располагаются основные инфраструктурные классы-обработчики, например, классы, отвечающие за логику создания и подтверждения учетных записей пользователя, за отправку сообщений на электронную почту и т.д.)

- Models (проект, в котором располагаются основные доменные и сторожевые модели, необходимые для ведения процесса взаимодействия с обработанными внешними контрактами и взаимодействием с объектной моделью базы данных)

- PostgreSQL (проект, в котором располагаются основные классы, отвечающие за создания контекста объектно-реляционной модели базы данных, а также репозитории, отвечающие за операции с базой данных PostgreSQL).

- ToDoCalendarServer (основной проект приложения, в котором находится основной сервис, работающий асинхронно все время с времени запуска приложения, а также точка входа в приложение. В отдельной папке лежат

RESTFul – контроллеры, благодаря которым осуществляется взаимодействие по протоколу REST API).

- ToDoCalendarServer.Tests (тестовый проект, проверяющий корректную сборку и работу сервисного веб-приложения, являющегося точкой входа в данное приложение).

Ниже на рисунке 7 показан скриншот обозревателя решений с полной структурой всех проектов

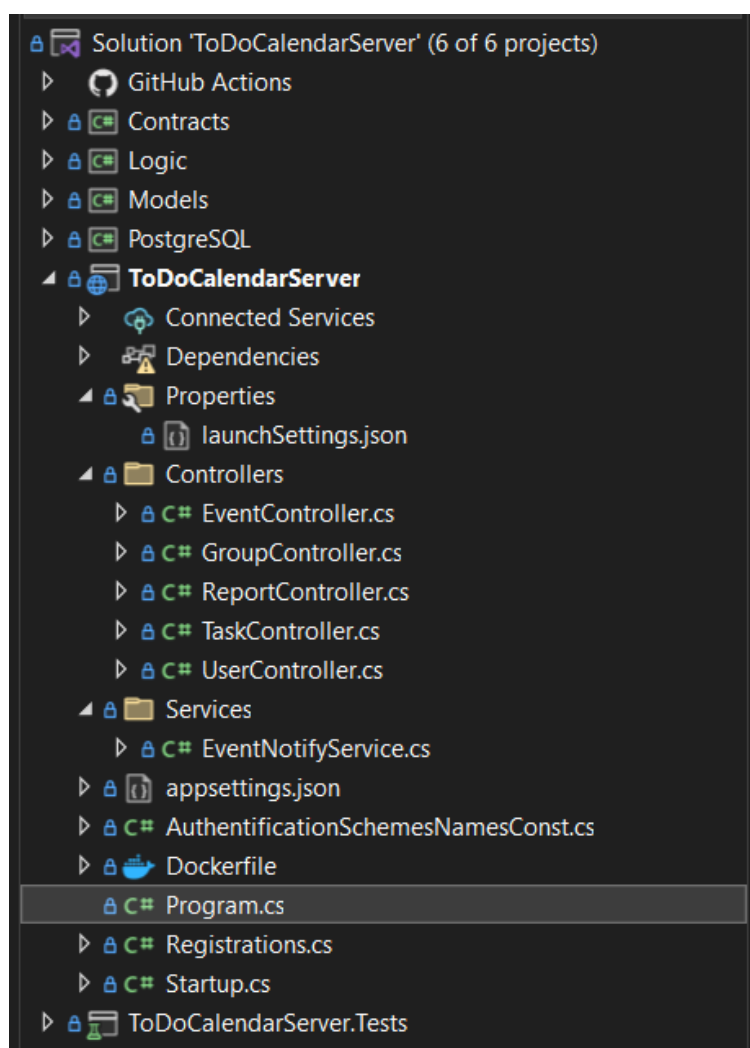


Рисунок 7. Структура проектов серверного приложения

3.5. Спецификация API

Все запросы начинаются с пути к выделенному серверу порту – а именно порту <https://localhost:5201>.

Все запросы, содержащие тело – должны иметь заголовок вида:

Content-type: Application/json

При ошибке авторизации возвращается ошибка 401 и текстом «Unauthorized».

При ошибке доступа возвращается ошибка со статусом 403 и текстом «Forbidden» или ошибка со статусом 400 и текстом «Bad Request»

При ошибке наполнения тела запроса возвращается ответ со статусом 400 и описанием ошибки в объекте errors.

При ошибке исполнения запроса, связанной с данными из тела, возвращается ответ со статусом 500 – «Internal Server Error».

Токен пользователя (необходимый для подтверждения на сервере факта аутентификации пользователя) – должен передаваться в качестве заголовка запроса (либо альтернативный вариант – в самом теле запроса: "token": "0895439408").

Явную авторизацию сервер не использует (используется собственная схема аутентификации) – поэтому в запросе не требуется указывать дополнительно заголовок *Authorization: <User-token>*

Таблица 2. Спецификация запросов в API <https://localhost:5201/>

Регистрация пользователя	
Метод	URL
POST	/users/register
Аутентификация	Не требуется
Тело запроса	
{ "email": "parahinvaleri5@gmail.com", "name": "Tigeroff1", "password": "tigeroff2002",	

"phone_number": "8-903-255-50-27"	
}	
Ответ	
Статус	Тело
200 OK	{ "result": true, "out_info": "Code confirmation was performed for user with email parahinvaleri5@gmail.com with code: 009709.\nRegistrating new user parahinvaleri5@gmail.com with id 3 with creating new auth token -1892586296" }
Авторизация пользователя	
Метод	URL
POST	/users/login
Аутентификация	Не требуется
Тело запроса	
{ "email": "parahinvaleri5@gmail.com", "password": "tigeroff2002" }	
Ответ	
Статус	Тело
200 OK	{ "result": true, "out_info": "Login existed user Tigeroff1 with new auth token 2112168000" }
Получение информации о пользователе	
Метод	URL
GET	/users/get_info
Аутентификация	Требуется – проверяется токен для пользователя с текущим id
Тело запроса	
{ "user_id": 3, "token": "2112168000" }	
Ответ (варианты для нового пользователя с id = 3 и старого пользователя с id = 1)	
Статус	Тело
200 OK	{ "requested_info": "{\"user_name\": \"Tigeroff1\", \"password\": \"tigeroff2002\", \"user_email\": \"parahinvaleri5@gmail.com\", \"phone_number\"

	:\"8-903-225-50-27\", \"user_groups\": [], \"user_tasks\": [], \"user_events\": [], \"user_reports\": []}, \"result\": true, \"out_info\": \"Info about user with with id 3 has been received\" }
200 OK	{ \"requested_info\": \"{ \"user_name\": \"Kirill Parakhin\", \"password\": \"kirill2002\", \"user_email\": \"kirill.parakhin@altenar.com\", \"phone_number\": \"8-904-255-50-27\", \"user_groups\": [{ \"group_name\": \"Add dotnet pls\", \"group_type\": \"Educational\" }, { \"group_name\": \"ADF\", \"group_type\": \"Job\" }], \"user_events\": [{ \"caption\": \"StandUp meeting\", \"description\": \"Every week project standup meeting\", \"scheduled_start\": \"2023-10-29T03:00:00+03:00\", \"duration\": \"00:30:00\", \"event_type\": \"StandUp\", \"event_status\": \"Cancelled\", \"manager\": { \"user_name\": \"Kirill Parakhin\", \"user_email\": \"kirill.parakhin@altenar.com\", \"phone_number\": \"8-904-255-50-27\" } }, { \"group\": null, \"guests\": null }] }, \"result\": true, \"out_info\": \"Info about user with with id 1 has been received\" }

Запланировать новое мероприятие	
Метод	URL
POST	/events/schedule_new
Авторизация	Требуется
Тело запроса	
{ \"user_id\": 1, \"token\": \"0895439408\", \"caption\": \"New december olimpiad discussion\", \"description\": \"Discussion about ICPC decemper tour olimpiad\", \"scheduled_start\": \"2023-11-05T18:00+00:00\", \"duration\": \"00:30:00\", \"event_type\": \"OneToOne\", \"event_status\": \"NotStarted\", \"group_id\": 10, \"guests_ids\": [2]	

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		33

}	
Ответ	
Статус	Тело
200 OK	{ "result": true, "out_info": "New event with id = 9 related to group 10 with name Add dotnet pls has been created" }
Добавить гостя в созданное мероприятие	
Метод	URL
PUT	/events/update_event_guests
Авторизация	Требуется
Тело запроса	
{ "event_id": 8, "user_id": 1, "token": "0895439408", "guests_ids": [3] }	
Ответ	
Статус	Тело
200 OK	{ "result": true, "out_info": "Existed event with id = 8 personal for manager with id 1 has been modified" }
Получить полную информацию о мероприятии (и связанных с ним группой и гостями)	
Метод	URL
GET	/events/get_event_info
Авторизация	Требуется
Тело запроса	
{ "event_id": 8, "user_id": 1, "token": "0895439408" }	
Ответ	
Статус	Тело
200 OK	{ "requested_info": { "caption": "StandUp meeting",

	<pre> "description": "Every week project standup meeting", "scheduled_start": "2023-10- 29T03:00:00+03:00", "duration": "00:30:00", "event_type": "StandUp", "event_status": "Cancelled", "manager": { "user_name": "Kirill Parakhin", "user_email": "kirill.parakhin@altenar.com", "phone_number": "8-904-255-50-27" }, "group": { "group_name": "Add dotnet pls", "group_type": "Educational", "participants": [] }, "guests": [] }, "result": true, "out_info": "Info about event with id = 3 has been received" } </pre>
--	--

Продолжение описания спецификации API представлено в приложении А.

3.6. Описание разработки клиентского приложения

Клиентская часть приложения была разработана с использованием фреймворка Flutter.

Описание структуры клиентского приложения

Для реализации мобильного клиентского приложения использовался фреймворк Flutter. Данная библиотека «из коробки» предоставляет большое количество возможностей – например, реализация как Android, так и IOS мобильного приложения, а также настройку используемых пакетов и внешних библиотек и модификацию скриптов по сборке и отладке.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		35

Для создания визуальной части мобильного приложения во Flutter использовались классы-виджеты, которые были разделены на следующие группы:

- Виджеты без состояния (то есть виджеты, которые не меняют своих данных (и их своевременное отображение) в разное время взаимодействия с ними). К таким виджетам можно отнести следующие виджеты: виджет главной страницы (main.dart), виджет домашней страницы, предоставляющей точку входа в календарь (home.dart), виджет для аутентификации (authorization_page.dart), виджеты для выполнения регистрации (register.dart) и авторизации (login.dart).

- Виджеты с изменением состояния (те виджеты, на которых необходимо производить подгрузку/изменение данных и т.д.). К таким виджетам можно отнести следующие виджеты: виджет главной страницы календаря (user_page.dart), виджет личного кабинета пользователя (user_info_map.dart), виджет с предоставлением дополнительной информации пользователю (additional_page.dart), виджеты для добавления новой группы (GroupPlaceholderWidget.dart), новой задачи (TaskPlaceholderWidget.dart), нового мероприятия (EventPlaceholderWidget.dart) и нового отчета пользователя (ReportPlaceholderWidget.dart), виджеты для отображения «контента» пользователя, то есть его визуального табличного календаря с мероприятиями (events_list_page.dart), его групп (groups_list_page.dart), задач (task_list_page.dart) и отчетов (reports_list_page.dart).

Для передачи данных между двумя HTTP-клиентами (клиентским и серверными REST API приложениями) будут использоваться json-объекты, которые для выполнения передачи между слушателями необходимо корректно обрабатывать (то есть сериализовать/десериализовать в формат .json). Поэтому в клиентском приложении (в папке models) были созданы модели по аналогии с серверными моделями – задача которых состоит в том, чтобы послать запрос на сервер и получить от него ответ.

Ниже на рисунке 8.2 показана структура папок и виджетов данного клиентского мобильного приложения:

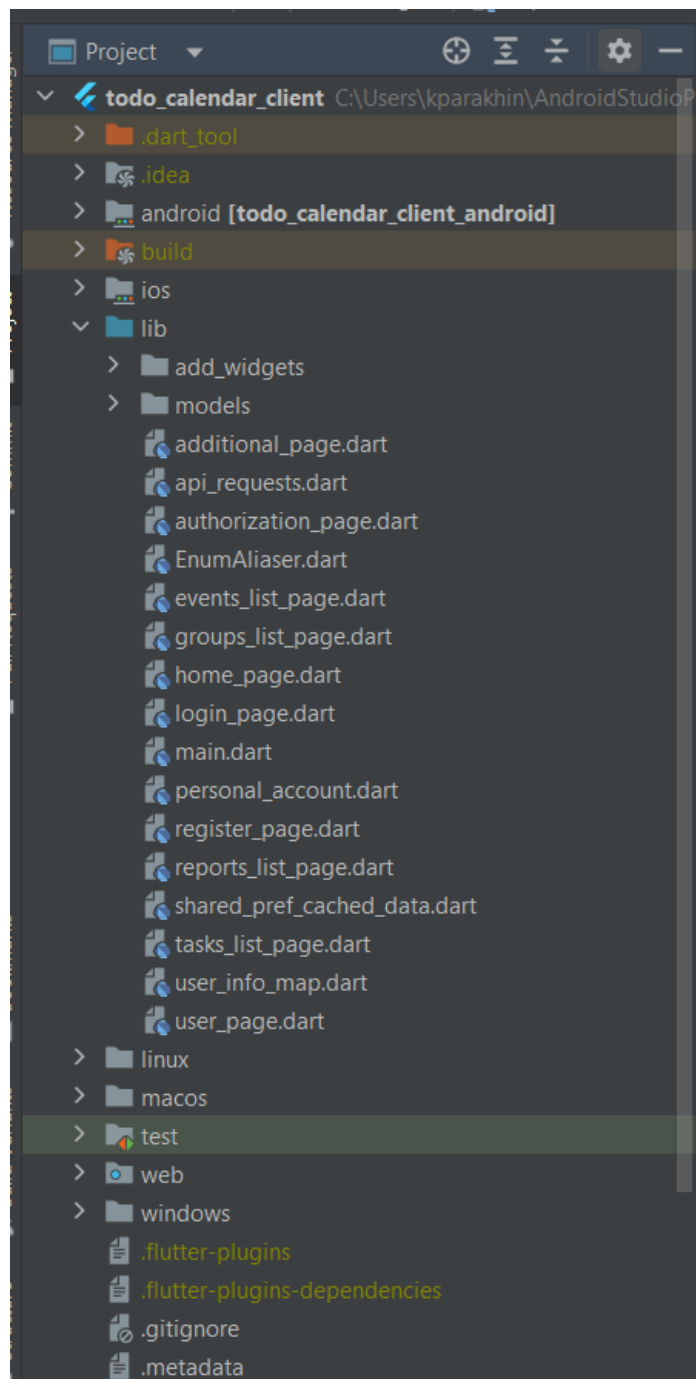


Рисунок 8.2. Структура папок клиентского Flutter – приложения

3.7. Интерфейс мобильного приложения

Создадим основные страницы (виджеты) – для регистрации, авторизации, для личного кабинета пользователя (с перечислением мероприятий, групп, задач и отчетов пользователя), а также дополнительные виджеты для добавления/редактирования мероприятия в календарь, новой группы, новой задачи на реализацию, нового отчета (по мероприятиям или отчетам).

По аналогии, есть виджеты для работы с группами, задачами и отчетами текущего пользователя – получаемые в качестве результата на GET-запрос по текущему идентификатору пользователя.

Также присутствует инструмент личного кабинета пользователя – для просмотра информации о текущих мероприятиях в календаре пользователя, списка его групп, задач и отчетов.

Ниже приведена стартовая страница, открывающаяся при запуске данного мобильного приложения:

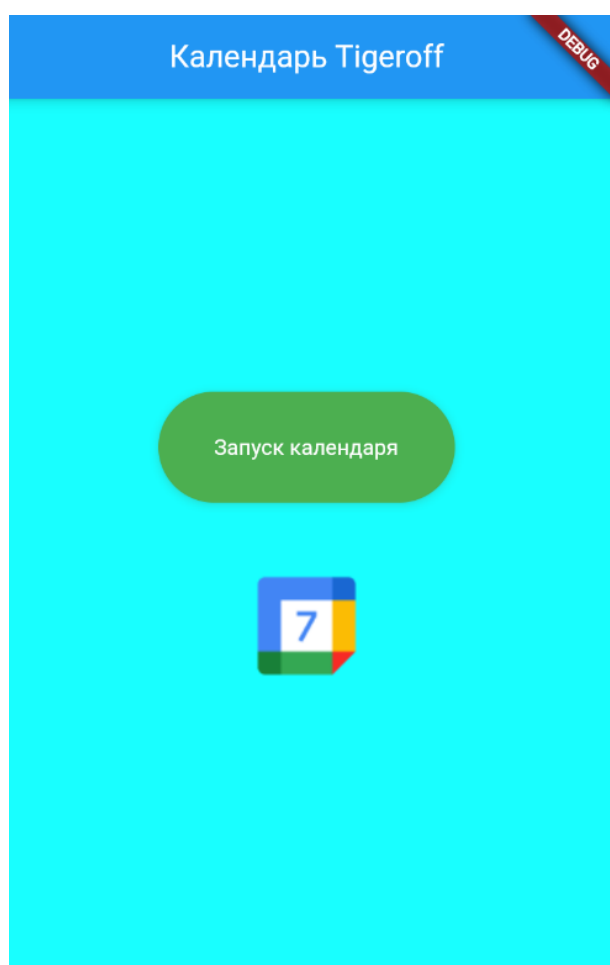


Рисунок 9.1. Стартовая страница приложения

Далее представлена домашняя страница для нового (то есть неаутентифицированного пользователя). По умолчанию, ему предлагается либо авторизоваться в системе, либо создать новый аккаунт (зарегистрироваться).

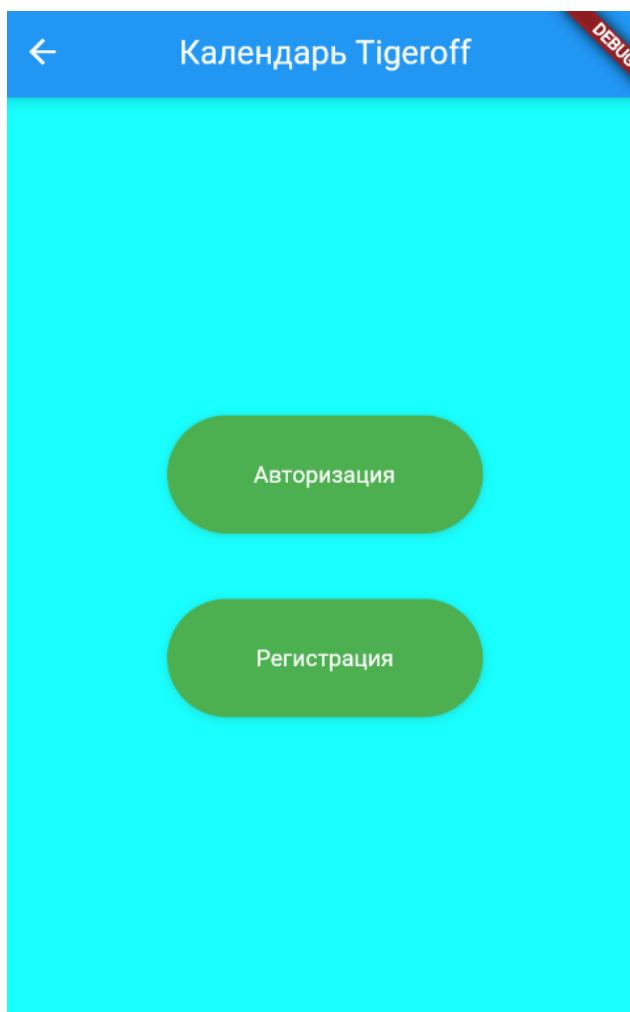


Рисунок 9.2. Домашний виджет для нового пользователя

В случае успешного ответа – загружается страница личного кабинета (`user_page.dart`), при этом параметры идентификатора пользователя и его токен аутентификации сохраняются в локальном кэше с использованием пакета `shared_preferences`.

Благодаря этому при аутентификации (входе в аккаунт) соответствующие пользователю данные в кэше будут добавляться (или обновляться), а при выходе из аккаунта – локальный кэш будет очищаться.

```
import 'package:shared_preferences/shared_preferences.dart';
```

Краткий листинг реализации кэширования (которое производится в локальном файле текущего мобильного устройства):

```
class MySharedPreferences {
    static const String _keyData = 'myData';
    static const String _keyExpiration = 'expirationTime';

    // Function to save data with an expiration date to SharedPreferences
    Future<bool> saveDataWithExpiration(String data, Duration
expirationDuration) async {
        try {
            SharedPreferences prefs = await SharedPreferences.getInstance();
            DateTime expirationTime = DateTime.now().add(expirationDuration);
            await prefs.setString(_keyData, data);
            await prefs.setString(_keyExpiration,
expirationTime.toIso8601String());
            print('Data saved to SharedPreferences.');
```

```
            return true;
        } catch (e) {
            print('Error saving data to SharedPreferences: $e');
            return false;
        }
    }

    // Function to get data from SharedPreferences if it's not expired
    Future<String?> getDataIfNotExpired() async {
        try {
            SharedPreferences prefs = await SharedPreferences.getInstance();
            String? data = prefs.getString(_keyData);
            String? expirationTimeStr = prefs.getString(_keyExpiration);
            if (data == null || expirationTimeStr == null) {
                print('No data or expiration time found in SharedPreferences.');
```

```
                return null; // No data or expiration time found.
            }
            DateTime expirationTime = DateTime.parse(expirationTimeStr);
            if (expirationTime.isAfter(DateTime.now())) {
                print('Data has not expired.');
```

```
                // The data has not expired.
                return data;
            } else {
                // Data has expired. Remove it from SharedPreferences.
                await prefs.remove(_keyData);
                await prefs.remove(_keyExpiration);
                print('Data has expired. Removed from SharedPreferences.');
```

```
                return null;
            }
        } catch (e) {
            print('Error retrieving data from SharedPreferences: $e');
```

```
            return null;
        }
    }

    // Function to clear data from SharedPreferences
    Future<void> clearData() async {
        try {
            SharedPreferences prefs = await SharedPreferences.getInstance();
            await prefs.remove(_keyData);
            await prefs.remove(_keyExpiration);
            print('Data cleared from SharedPreferences.');
```

```
        } catch (e) {
            print('Error clearing data from SharedPreferences: $e');
```

```
        }
    }
}
```

Изм.	Лист	№ докум.	Подп.	Дата

Страницы авторизации и регистрации представлены на рисунках ниже. На данных страницах, как и на всех остальных страницах, на которых происходит ввода каких-то данных – происходит валидация текста:

Рисунок 9.3. Виджет для авторизации пользователя

Рисунок 9.4. Виджет для регистрации пользователя

Главная страница пользователя, представляющая собой пример виджета с изменением состояния (в зависимости от изменения локального кэша меняется ее содержимое – в случае отсутствия данных о авторизованном пользователе – осуществляется перенаправление пользователя на страницу для авторизации).

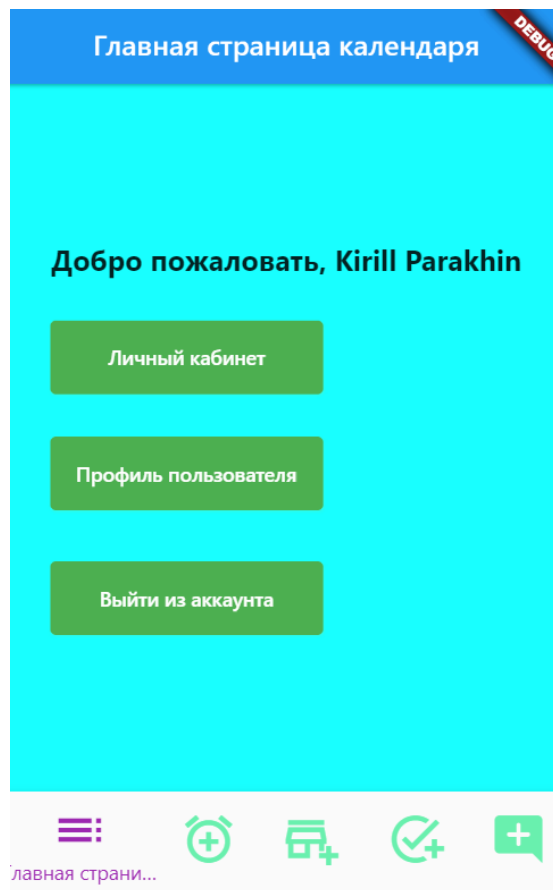


Рисунок 10.1. Главная страница пользователя

На главной странице пользователя (рис.10.1) и странице личного кабинета пользователя присутствует нижнее навигационное меню с различными режимами текущего окна. По умолчанию, открывается главная страница. При нажатии на другую панель – откроется другое состояние виджета.

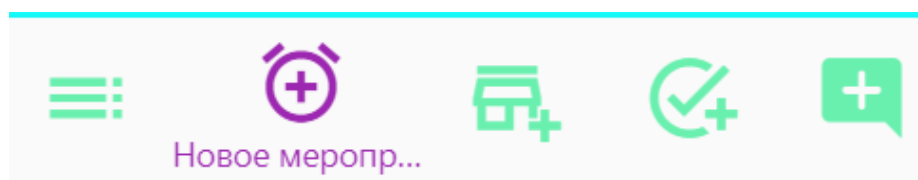


Рисунок 10.2. Переключение плиток нижнего меню

Ниже представлены страница с профильной информацией об аккаунте пользователя:



Рисунок 10.2. Информация об аккаунте текущего пользователя

Страница с календарными мероприятиями пользователя

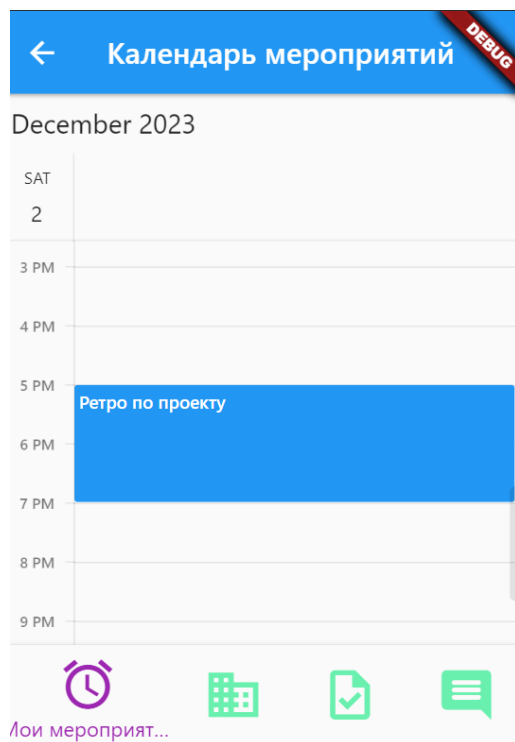


Рисунок 11.1. Виджет календаря пользователя с мероприятиями

Мероприятия настраиваются и показываются в режиме дневного просмотра. При этом время в календаре используется локальное время мобильного устройства пользователя (+3 UTC, по умолчанию).

При этом сам виджет календаря и выделенные в нем мероприятия интерактивны – при клике на них можно посмотреть подробную информацию о них – а также увидеть список участников

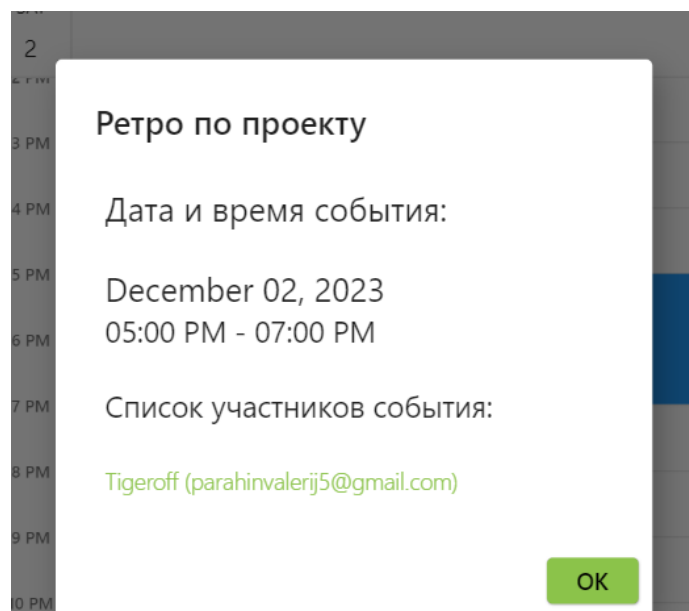


Рисунок 11.2. Показ информации о мероприятии

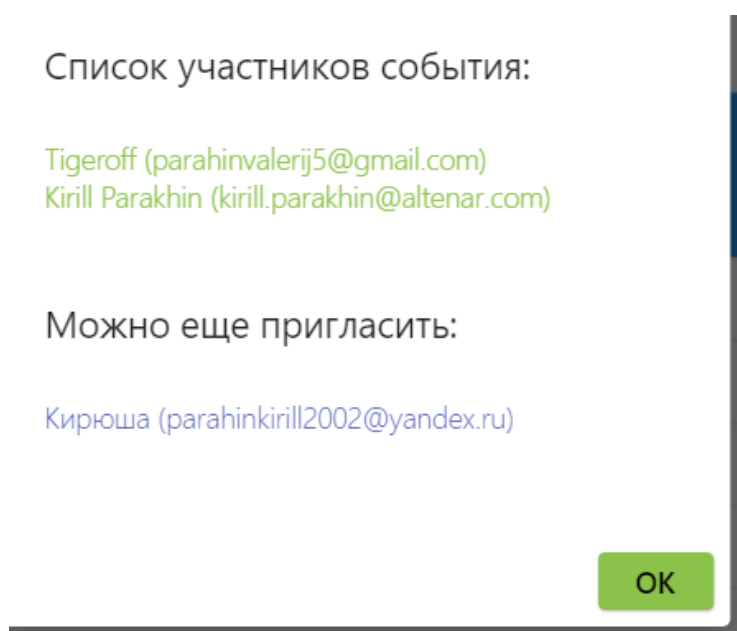


Рисунок 11.3. Список участников события

Фрагмент листинга получения информации о мероприятиях пользователях с API сервера:

```
var json = jsonDecode(cachedData.toString());
var cacheContent = ResponseWithToken.fromJson(json);

var userId = cacheContent.userId;
var token = cacheContent.token.toString();

var model = new UserInfoRequestModel(userId: userId, token: token);
var requestMap = model.toJson();

var url = Uri.parse(uri);
final body = jsonEncode(requestMap);

try {
  final response = await http.post(url, headers: headers, body: body);

  var jsonData = jsonDecode(response.body);
  var responseContent = GetResponse.fromJson(jsonData);

  if (responseContent.result) {
    var userRequestedInfo = responseContent.requestedInfo.toString();

    var data = jsonDecode(userRequestedInfo);
    var userEvents = data['user_events'];

    var fetchedEvents =
      List<EventInfoResponse>
        .from(userEvents.map(
          (data) => EventInfoResponse.fromJson(data)));

    setState(() {
      eventsList = fetchedEvents;
    });
  }
}
```

Фрагмент формирования коллекции доменных моделей мероприятий для использования их при отображении в виджете:

```
List<EventAppointment> meetings =
  List.from(
    fetchedEvents.map((data) =>
      new EventAppointment(
        data.start,
        data.duration,
        data.caption)));
```

Фрагмент использования виджета календаря из пакета syncfusion_flutter_calendar:

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Календарь мероприятий'),
        leading: IconButton(
```

```
        icon: Icon(Icons.arrow_back),
        onPressed: () {
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(
              builder: (context) => UserPage(),
            ),
          ),
        ),
      ),
    body: SfCalendar(
      view: CalendarView.week,
      firstDayOfWeek: 1,
      initialDisplayDate: DateTime.now(),
      initialSelectedDate: DateTime.now(),
      dataSource: MeetingDataSource(getAppointments(eventsList)),
    ),
  ),
),
```

Страницы со списком групп, задач и отчетов пользователя

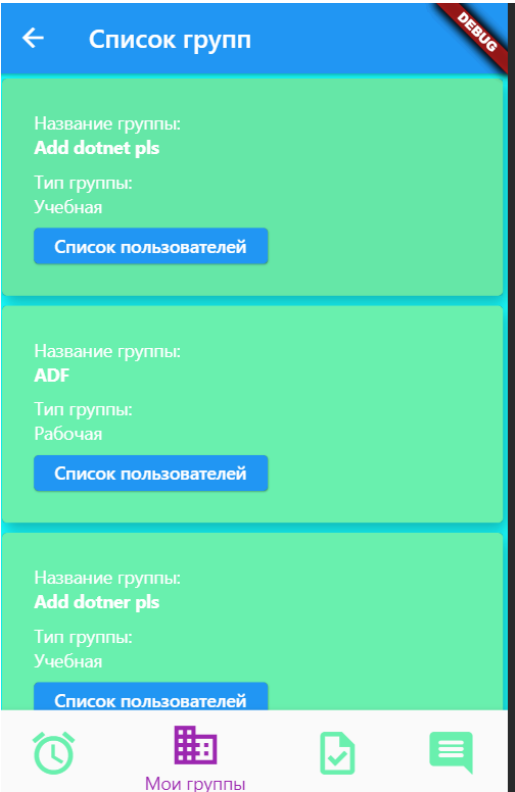


Рисунок 12.1. Виджет со списком групп пользователя

У каждой группы можно посмотреть список ее пользователей (красным выделяется текущий пользователь) – и посмотреть снимок его календаря:

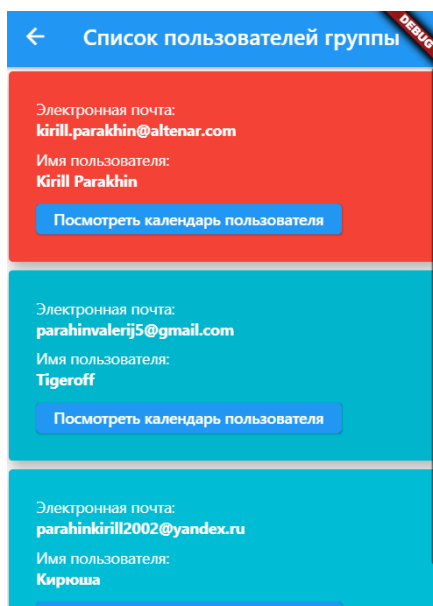


Рисунок 12.2. Список пользователей группы

Ниже представлен виджет со списком задач текущего пользователя. Как и мероприятия – задачи также можно редактировать – для этого в приложении присутствует отдельная страница:

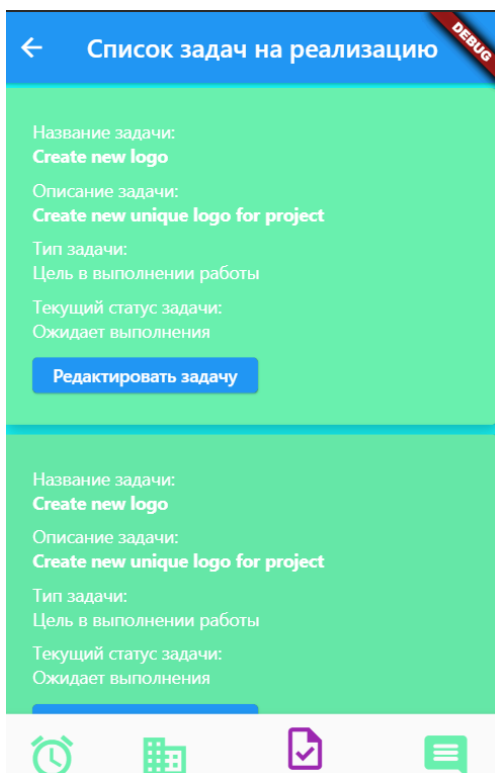


Рисунок 13.1. Виджет со списком задач пользователя

Изм.	Лист	№ докум.	Подп.	Дата

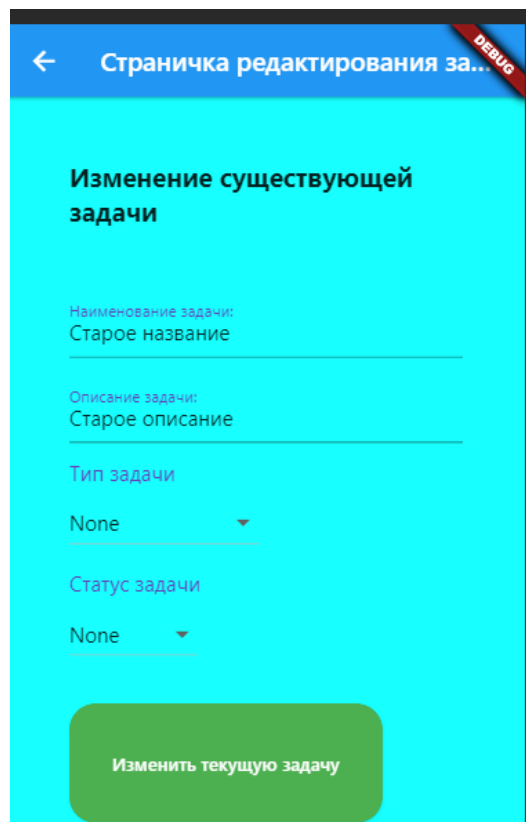


Рисунок 13.2. Виджет с редактированием существующей задачи

В случае, если задачи (или группы отсутствуют) – приложение предложит на этих виджетах создать новые сущности (рис.13.3):

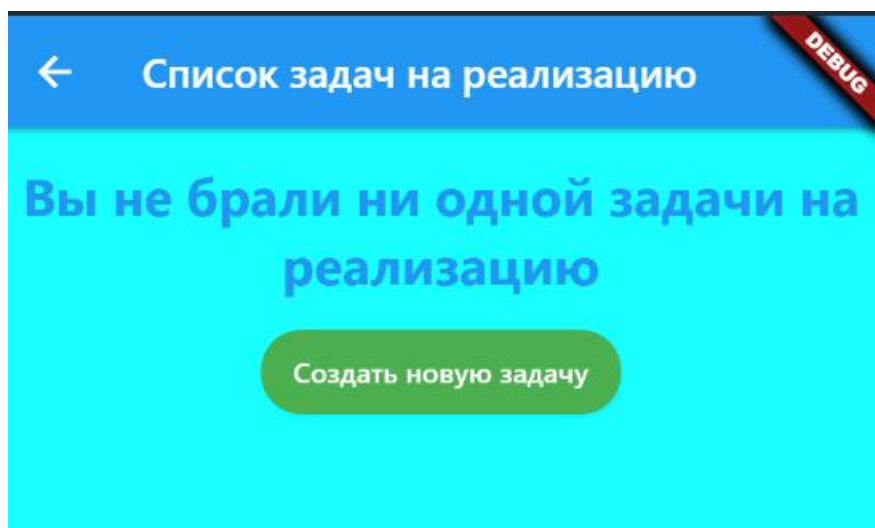


Рисунок 13.3. Пустой список задач (состояние виджета)

Также пользователь может просматривать созданные отчеты (задачи создаются по прогрессу выполнения задач или посещению мероприятий за определенный задаваемый отрезок времени):



Рисунок 14. Виджет со списком отчетов пользователя

В отчете по задачам указывается:

- Кол-во задач для реализации за указанный период
- Процент не начатых задач
- Процент задач в процессе выполнения
- Процент выполненных задач

В отчете по мероприятиям указывается:

- Кол-во запланированных мероприятий за указанный период
- Общая реальная продолжительность мероприятий за данный период
- Процент посещения запланированных мероприятий

Виджеты для создания новых данных

Все виджеты, использующие для создания/редактирования данных имеют встроенную валидацию исходной строки на стороне клиента (например, проверка пустоты строк, сравнение времени окончания и начала планируемого мероприятия (или отчета) и т.д.).

Для значений, соответствующим типам и статусам создаваемых сущностей – будут использоваться выпадающие списки для перечислений.

Для выбора даты и времени начала и окончания создаваемого мероприятия использовался специальное визуальное диалоговое окно Flutter.

Фрагмент листинг считывания и формирования данных для запроса по созданию нового мероприятия:

```
String caption = eventCaptionController.text;
String description = eventDescriptionController.text;
String scheduledStart = selectedBeginDateTime.toString();

int durationMs =
    selectedEndDateTime.millisecondsSinceEpoch
        > selectedBeginDateTime.millisecondsSinceEpoch
    ? selectedEndDateTime.difference(selectedBeginDateTime).inMilliseconds
    : DateTime(0, 0, 0, 0, 30, 0).millisecondsSinceEpoch;

var durationSeconds = (durationMs / 1000).round();

var hours = (durationSeconds / 3600).round();

var remainingSeconds = durationSeconds - hours * 3600;

var minutes = (remainingSeconds / 60).round();

var seconds = remainingSeconds - minutes * 60;

var duration = hours.toString().padLeft(2, '0')
    + ':' + minutes.toString().padLeft(2, '0')
    + ':' + seconds.toString().padLeft(2, '0');
```

Функции для выбора даты и времени начала/окончания мероприятия (совместно друг с другом):

```
Future<DateTime?> pickDate() => showDatePicker(
    context: context,
    initialDate: selectedBeginDateTime,
    firstDate: DateTime(2023),
    lastDate: DateTime(2025)
);
```

```
Future<TimeOfDay?> pickTime() => showTimePicker(
    context: context,
    initialTime: TimeOfDay(
        hour: selectedBeginDateTime.hour + 1,
        minute: 0));
```

```
Future pickBeginDateTime() async {

    DateTime? date = await pickDate();
    if (date == null) return;

    final time = await pickTime();

    if (time == null) return;

    final newDateTime = DateTime(
        date.year,
        date.month,
        date.day,
        time.hour,
        time.minute
    );
    setState(() {
        selectedBeginDateTime = newDateTime;
    });
}
```

```
Future pickEndDateTime() async {

    DateTime? date = await pickDate();
    if (date == null) return;

    final time = await pickTime();

    if (time == null) return;

    final newDateTime = DateTime(
        date.year,
        date.month,
        date.day,
        time.hour,
        time.minute
    );

    setState(() {
        selectedEndDateTime = newDateTime;
    });
}
```

Ниже приведены виджеты для создания нового мероприятия (и последовательного заполнения информации о нем):

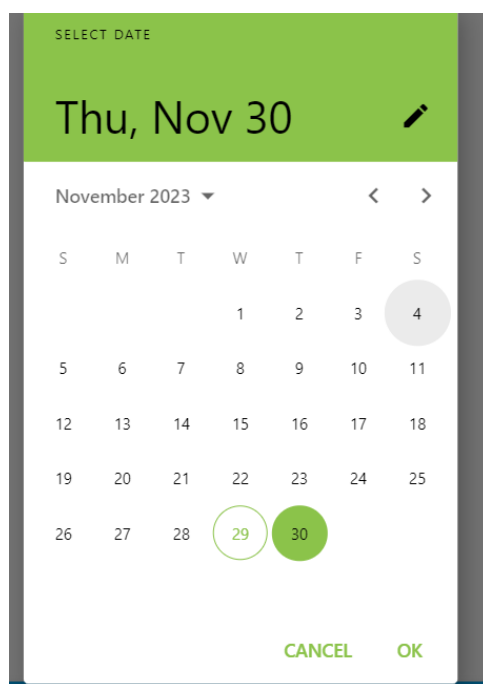


Рисунок 15.1.

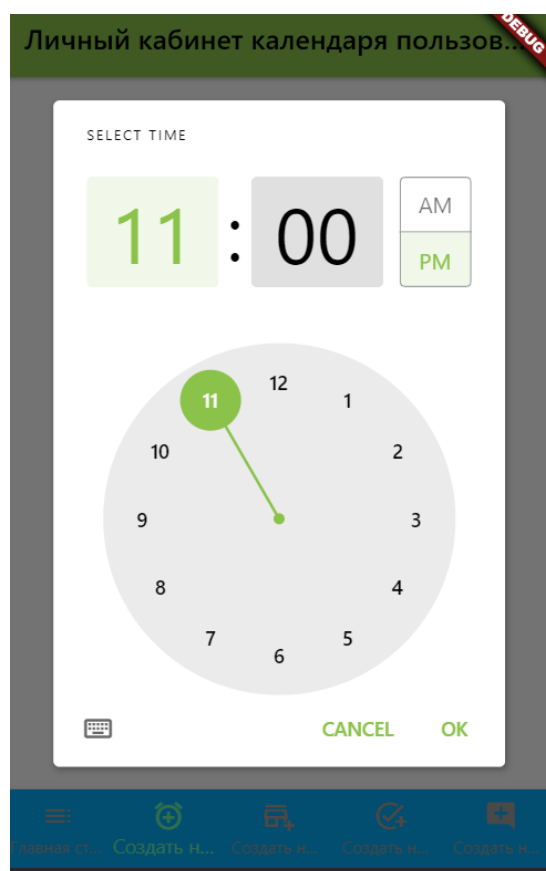


Рисунок 15.2.

Страница создания мероприятия

DEBUG

Наименование мероприятия:

Название мероприятия не может быть пустым

Описание мероприятия:

Описание мероприятия не может быть пустым

Время начала мероприятия

2023/12/7 23:34

Время окончания мероприятия

Время окончания 2023/12/7 23:00 должно быть больше времени начала

Тип мероприятия

Personal

Статус мероприятия

NotStarted

☰

🕒

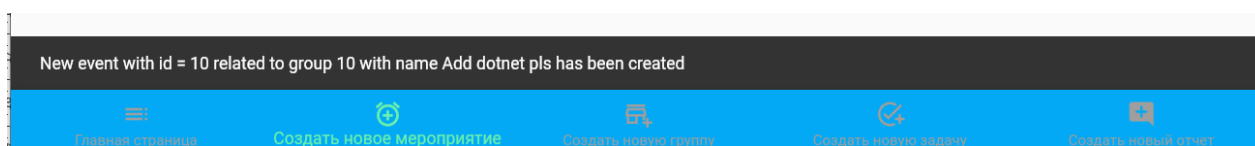
🏠+

✅+

💬+

Новое мероприятие

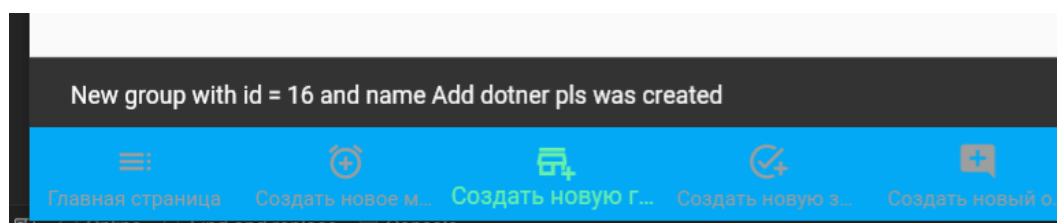
Рисунок 15.3.



Рисунки 15.1 – 15.4. Виджеты для создания нового мероприятия и вывод текста результата отправленного запроса

Ниже приведен виджет для создания новой группы (по умолчанию, добавляется в качестве участника создатель группы – для добавления новых участников присутствует дополнительный виджет в личном кабинете создателя группы):

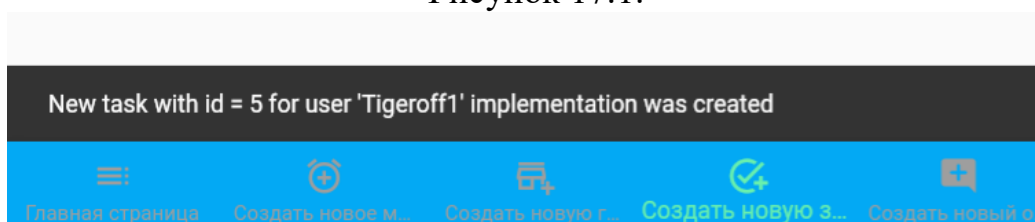
Рисунок 16.1.



Рисунки 16.1 – 16.2. Виджеты для создания новой группы и вывод текста результата отправленного запроса

Ниже приведен виджет для создания новой задачи (по умолчанию, задача назначается на реализацию ее менеджером (или создателем). На дополнительном виджете редактирования в личном кабинете можно переназначить ее на другого пользователя.

Рисунок 17.1.



Рисунки 17.1 – 17.2. Виджет для создания новой задачи и вывод текста результата отправленного запроса

Ниже приведен виджет для создания нового отчета (с валидацией даты начала и окончания периода – одно из дополнительных ограничений по созданию нового отчета – отчетный период не должен превышать одного календарного месяца):

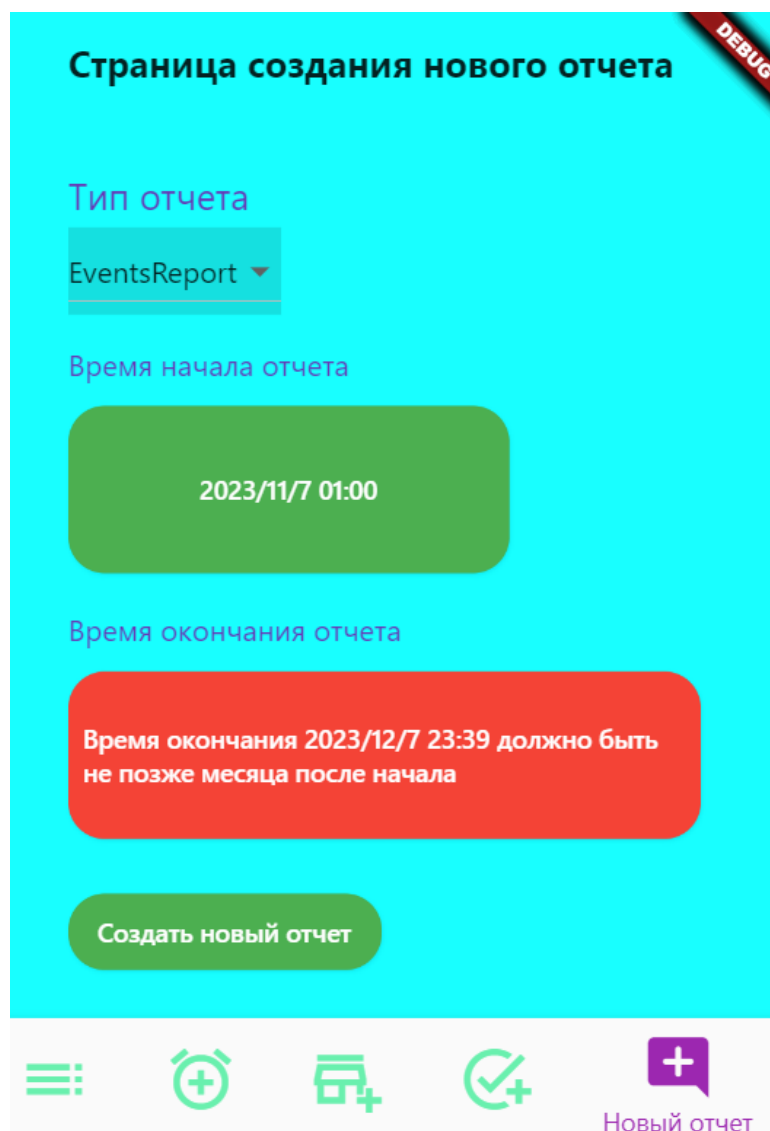
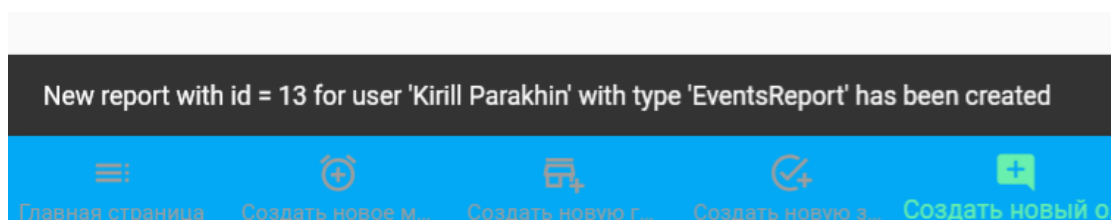


Рисунок 18.1.



Рисунки 18.1 – 18.2. Виджет для создания нового отчета по пройденным мероприятиям пользователя и вывод текста результата отправленного запроса

4. ТЕСТИРОВАНИЕ СИСТЕМЫ

4.1. Тестирование запросов к REST API серверного приложения

Для тестирования HTTP-запросов, поступаемых «внутри и наружу» серверного приложения я использовал программу Postman.

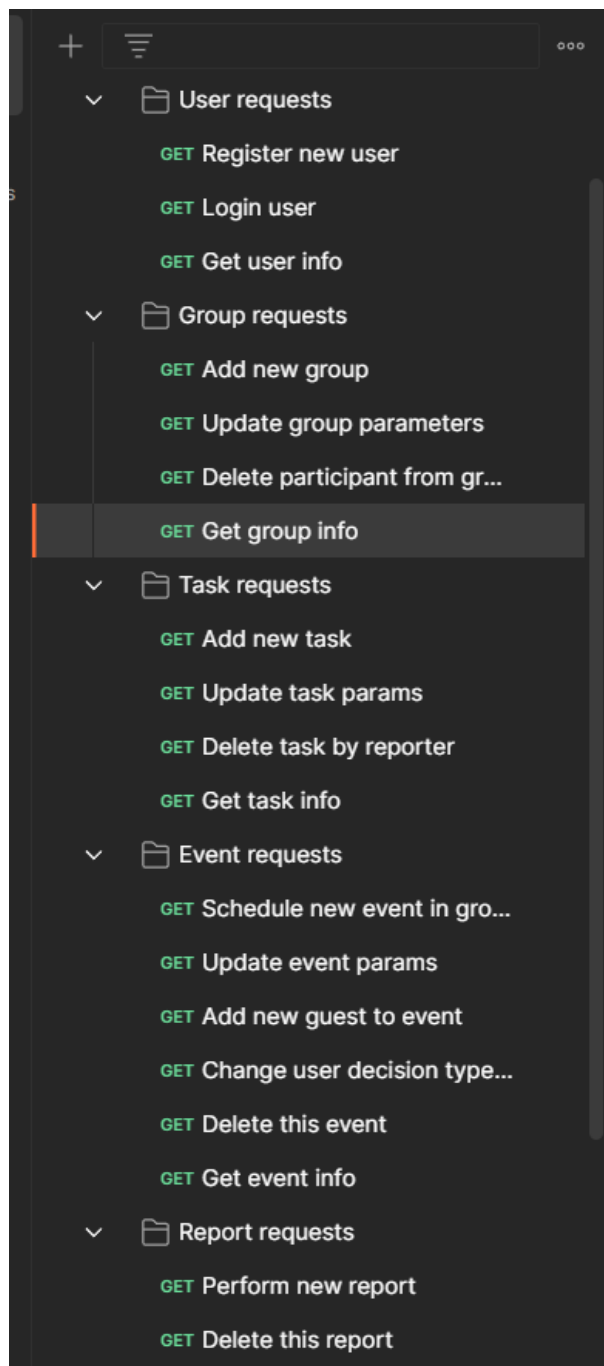


Рисунок 19. Список коллекций запросов к API

Ниже на рисунках 20 – 21 – показаны примеры запросов, связанных с созданием и получением информации об основных сущностях системы (то есть пользователях и их мероприятиях):

Запросы к endpointy /users:

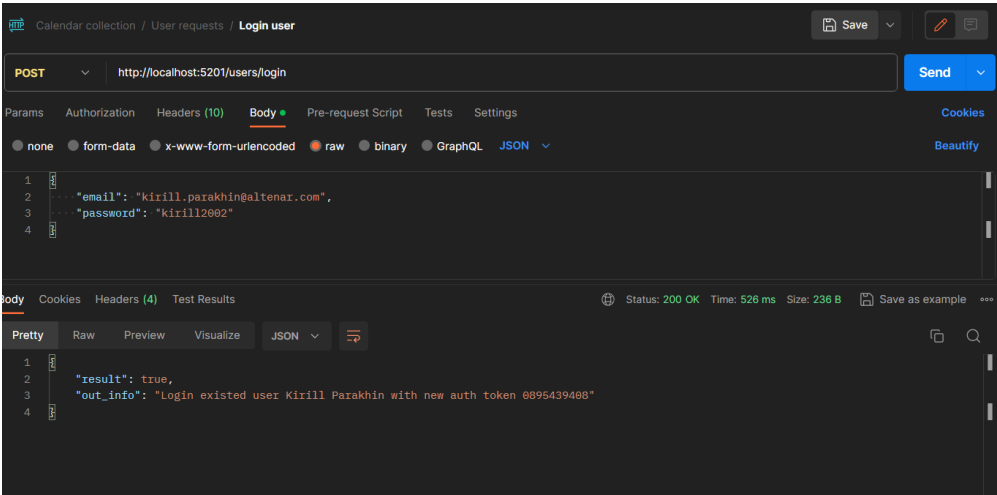


Рисунок 20.1. Тестирование запроса на авторизацию пользователя

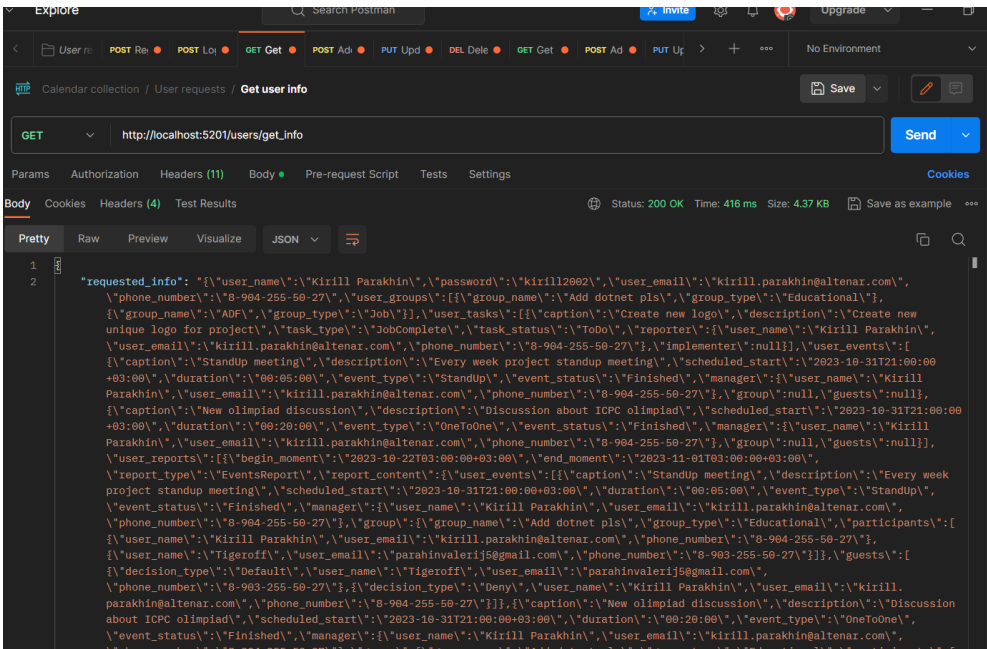


Рисунок 20.2. Тестирование запроса на получение полной информации о пользователе

Запросы к endpointy /events:

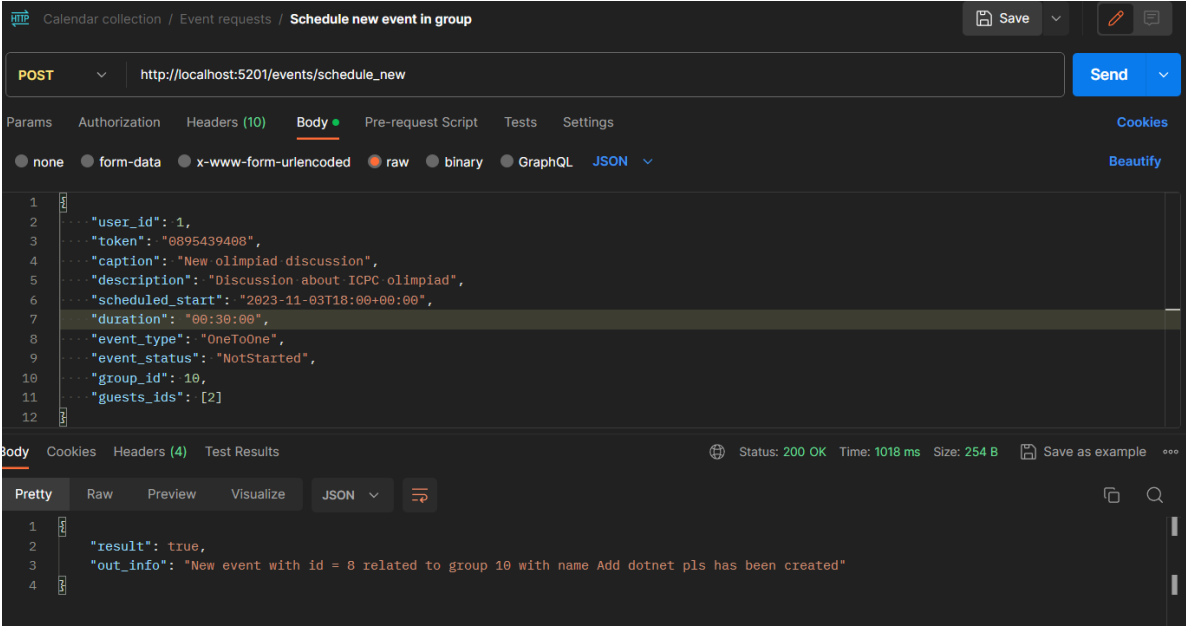


Рисунок 21.1. Тестирование запроса на создание нового мероприятия

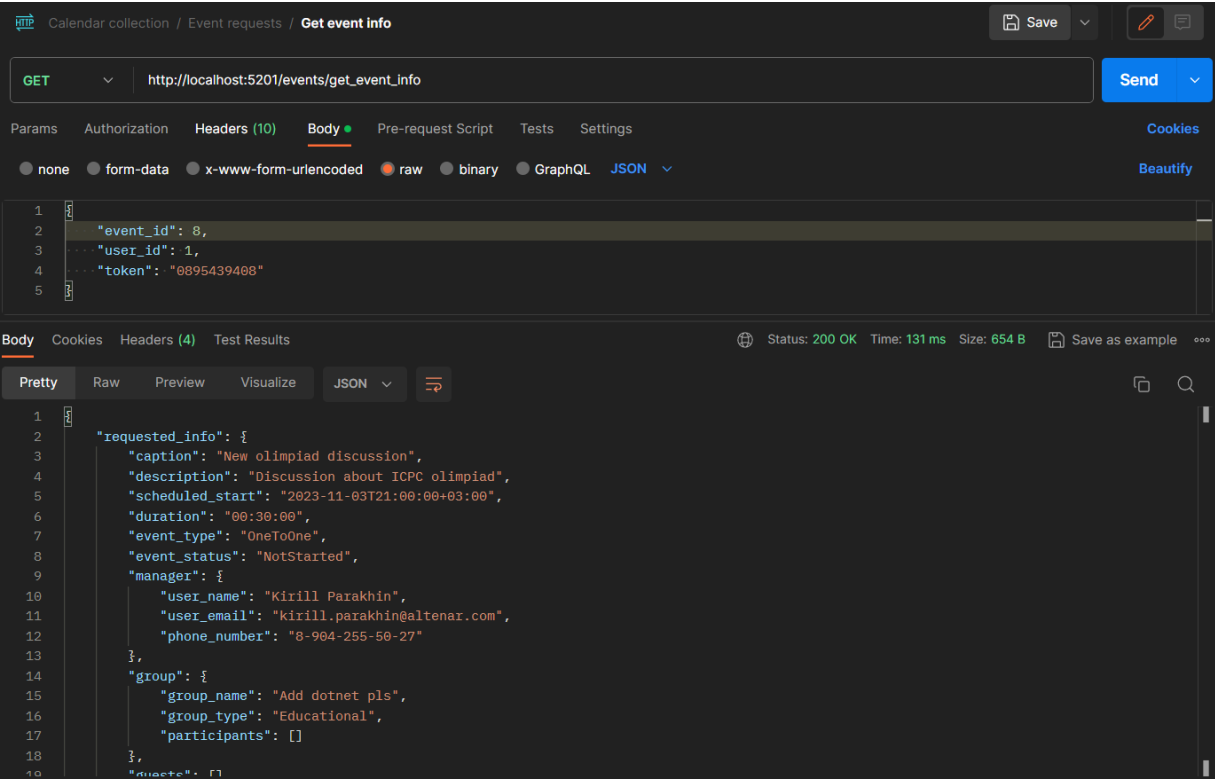


Рисунок 21.2. Тестирование запроса на получение полной информации о выбранном мероприятии

4.2. Тестирование мобильного приложения с помощью тестовых кейсов

В данном разделе напишем несколько сценариев для тестирования виджетов в нашем приложении.

1. Сценарий регистрации нового пользователя в системе

- 1) Пользователь заходит на страницу регистрации;
- 2) Пользователь вводит требуемые данные для регистрации;
- 3) Пользователь получает ответ;
 - a. Пользователь может получить успешный результат регистрации;
 - b. Пользователь может получить сообщение о провальной регистрации.
 - c. Пользователь может получить сообщение о проблеме с соединением к серверу.

Пользователь заполняет данные, требуемые для регистрации, и нажимает кнопку «Зарегистрироваться»:

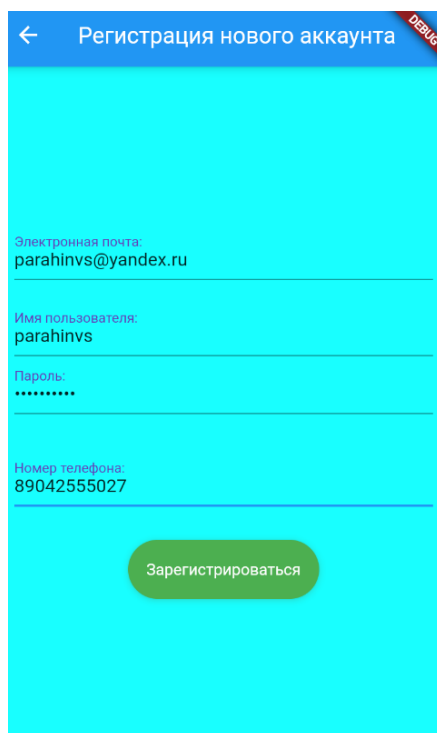


Рисунок 22.1. Скриншот ввода существующего аккаунта для регистрации

В том случае, если пользователь введёт электронную почту, с которой уже существует какой-то аккаунт в системе – то ему будет выведено сообщение о провальной регистрации. Для показа диалоговых сообщений мною будет использоваться виджет showDialog (или AlertDialog).

На стороне сервера идёт проверка на существование пользователя, в нашем случае пользователь существует, возвращается ответ с кодом 400 (BadRequest), обрабатываем ответ и выводим окно с сообщением для пользователя:

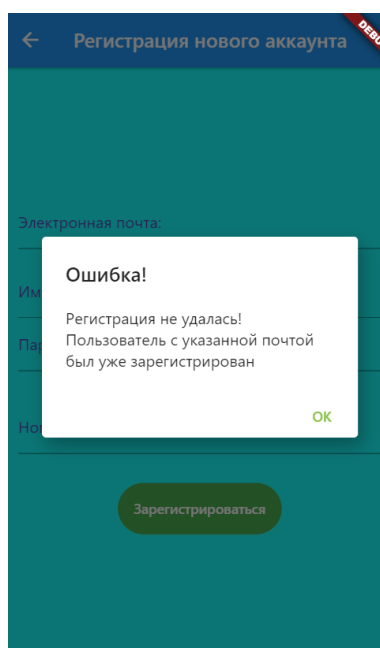


Рисунок 22.2 Скриншот вывода ошибки при помощи showDialog

После того, как пользователь уведомлён о невозможности использования текущей почты, он может ввести другую почту (например, parahinvs1@yandex.ru и имя «Кирюша»):

Далее пользователю будет выведено диалоговое окно о необходимости подтверждения его аккаунта в течение 5 минут. Для этого ему необходимо перейти на указанный при регистрации адрес электронной почты – и перейти по ссылке, отправленной ему в письме:

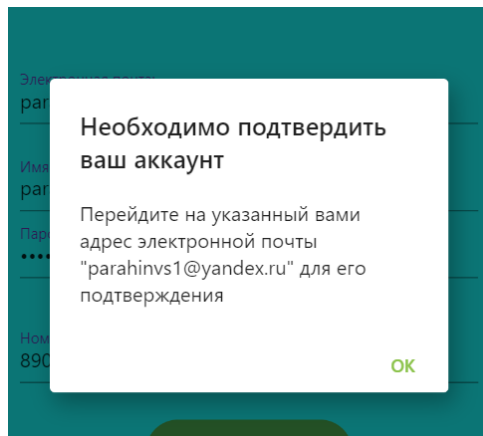


Рисунок 22.3. Уведомление о подтверждении аккаунта

Сообщение о подтверждении регистрации по почте:

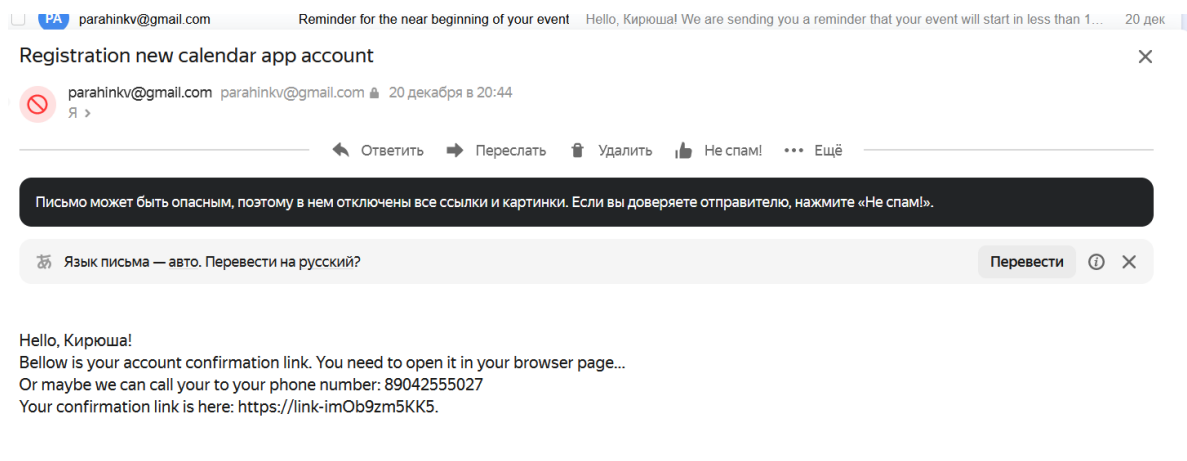


Рисунок 22.4. Сообщение на почте

После успешной регистрации пользователя перенаправляет на начальную страницу для попытки авторизации, также снизу мы можем увидеть сообщение об успешной регистрации в системе. Данное сообщение обрабатывалось при помощи `showSnackBar`:

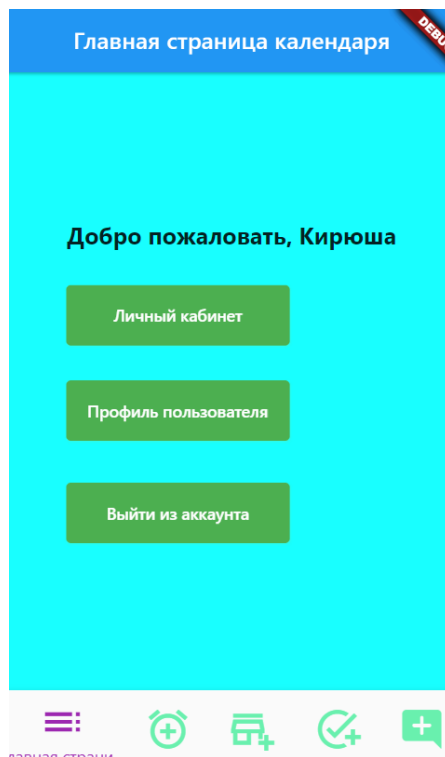


Рисунок 22.5. Скриншот приложения после успешного прохождения регистрации

2. Сценарий авторизации существующего пользователя в системе

Напишем следующий сценарий:

- 1) Пользователь заходит на страницу авторизации;
- 2) Пользователь вводит логин и пароль;
- 3) Пользователь получает ответ;
 - а. Пользователь может получить сообщение о неверном введённом логине или пароле.
 - б. Пользователь может получить сообщение о проблеме соединения с сервером
 - с. Пользователь может успешно пройти авторизацию и попасть на главную страницу

Протестируем авторизацию пользователя с электронной почтой «parahinvs@yandex.ru» и паролем «kirill»:

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		63

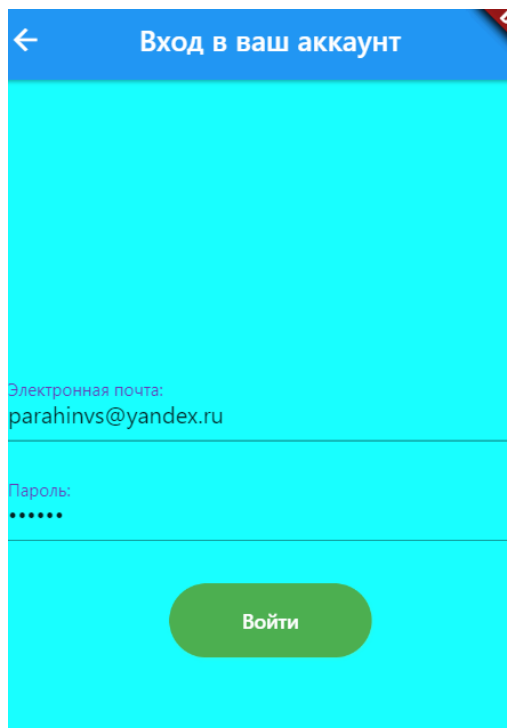


Рисунок 23.1. Скриншот ввода неверного пароля

В таком случае у пользователя либо нет аккаунта, либо введенные данные неверны. Уведомляем пользователя об этом:

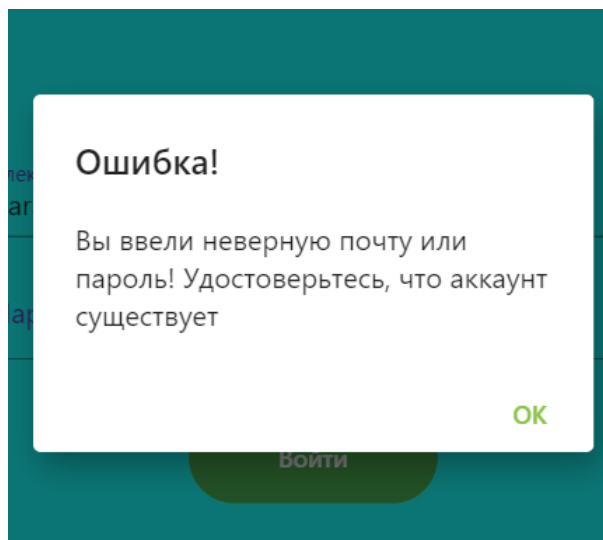


Рисунок 23.2. Скриншот уведомления пользователя

Также существует вероятность, что сервер приложения будет недоступен в момент выполнения запроса авторизации (или ранее запроса регистрации). Тогда пользователь получит соответствующее уведомление в диалоговом окне.

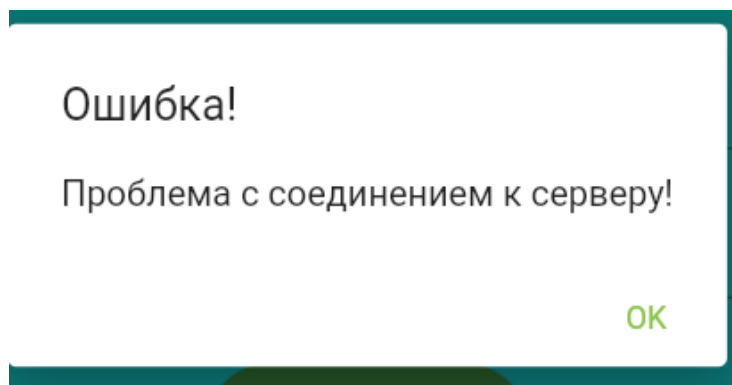


Рисунок 23.3. Уведомление о проблеме соединения с сервером

В случае, если данные пользователя введены верно, пользователь будет перенаправлен на главную страницу приложения (как на рис.22.5)

5. РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ

5.1. Развертывание сервера

Для использования серверного приложения – его удобно развернуть в отдельном контейнере, например, Docker – контейнере.

Чтобы произвести развертывание приложения, необходимо ввести в терминале проекта команду: `docker compose –up build`.

При этом во время развертывания с помощью Dockerfile будет создан образ серверного приложения (под названием `todocalendar-app`), а также образ базы данных `postgres_container` (при этом при создании контейнера – к базе данных будут применены основные миграции по созданию таблиц и наполнению их некоторыми исходными данными).

По итогу, получается контейнер приложения, к которому можно получить доступ в локальной сети по адресу `url: 4040:4040` (рис. 24)

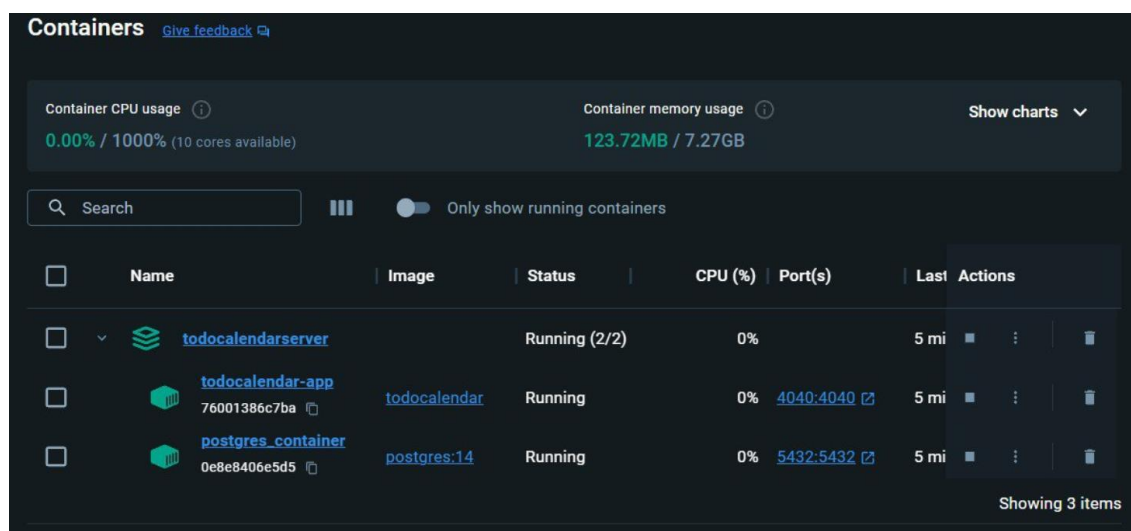


Рисунок 24. Развертывание сервера на Docker

Серверное приложение постоянно проверяет все мероприятия из базы данных – и меняет их статусы при наступлении их приближения, начала и окончания. Также в некоторых случаях отсылает уведомления о их начале по электронной почте (например, за 10 минут до начала всем участникам).

todocalendar-app

76001386c7ba

4040:4040

STATUS

Running (5 minutes ago)

Logs

Inspect

Bind mounts

Exec

Files

Stats

2023-12-06 19:50:53

SELECT e.id, e.caption, e.description, e.duration, e.event_type, e.manager_id, e.related_group_id, e.s

cheduled_start, e.status

2023-12-06 19:50:53

FROM events AS e

2023-12-06 19:50:53

info: PostgreSQL.EventsRepository[0]

2023-12-06 19:50:53

The changes of events were sent to DB

2023-12-06 19:51:53

info: Microsoft.EntityFrameworkCore.Database.Command[20101]

2023-12-06 19:51:53

Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']

2023-12-06 19:51:53

SELECT e.id, e.caption, e.description, e.duration, e.event_type, e.manager_id, e.related_group_id, e.s

cheduled_start, e.status

2023-12-06 19:51:53

FROM events AS e

2023-12-06 19:51:53

info: PostgreSQL.EventsRepository[0]

2023-12-06 19:51:53

The changes of events were sent to DB

2023-12-06 19:52:53

info: Microsoft.EntityFrameworkCore.Database.Command[20101]

2023-12-06 19:52:53

Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']

2023-12-06 19:52:53

SELECT e.id, e.caption, e.description, e.duration, e.event_type, e.manager_id, e.related_group_id, e.s

cheduled_start, e.status

2023-12-06 19:52:53

FROM events AS e

2023-12-06 19:52:53

info: PostgreSQL.EventsRepository[0]

2023-12-06 19:52:53

The changes of events were sent to DB

2023-12-06 19:53:53

info: Microsoft.EntityFrameworkCore.Database.Command[20101]

2023-12-06 19:53:53

Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']

2023-12-06 19:53:53

SELECT e.id, e.caption, e.description, e.duration, e.event_type, e.manager_id, e.related_group_id, e.s

Рисунок 25.1. Логи работы сервера в контейнере

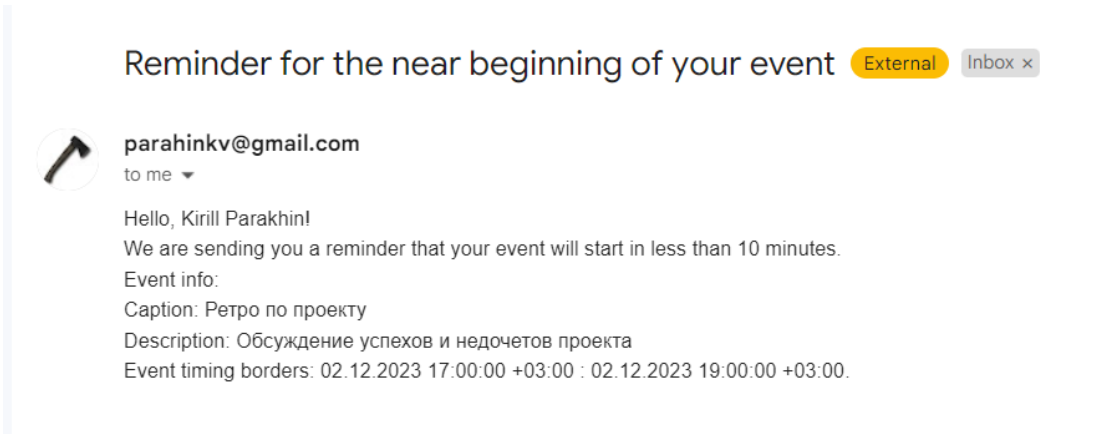


Рисунок 25.2. Оповещение о начале мероприятия по почте

5.2. Развертывание мобильного приложения

После проведения разработки и тестирования логики работы мобильного клиент-серверного приложения – необходимо собрать его в виде исполняемого файла для определенной мобильной операционной системы – в данном случае будем использовать ОС Android.

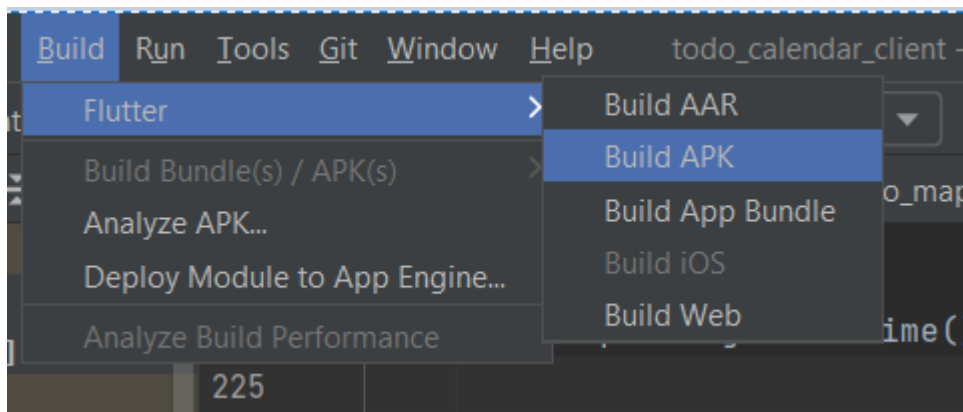


Рисунок 26.1. Выбор сборки APK файла

```
C:\src\flutter\bin\flutter.bat --no-color build apk

Running Gradle task 'assembleRelease'...
Font asset "MaterialIcons-Regular.otf" was tree-shaken, reducing it from 164
flag when building your app.
Running Gradle task 'assembleRelease'... 80,2s
v Built build\app\outputs\flutter-apk\app-release.apk (21.6MB).
Process finished with exit code 0
```

Рисунок 26.2. Получение готового APK файла

Полученный исполняемый файл (с расширением .ark) - его возможно запустить как в эмуляторе, так и на реальном мобильном устройстве (пример на рисунке 27).

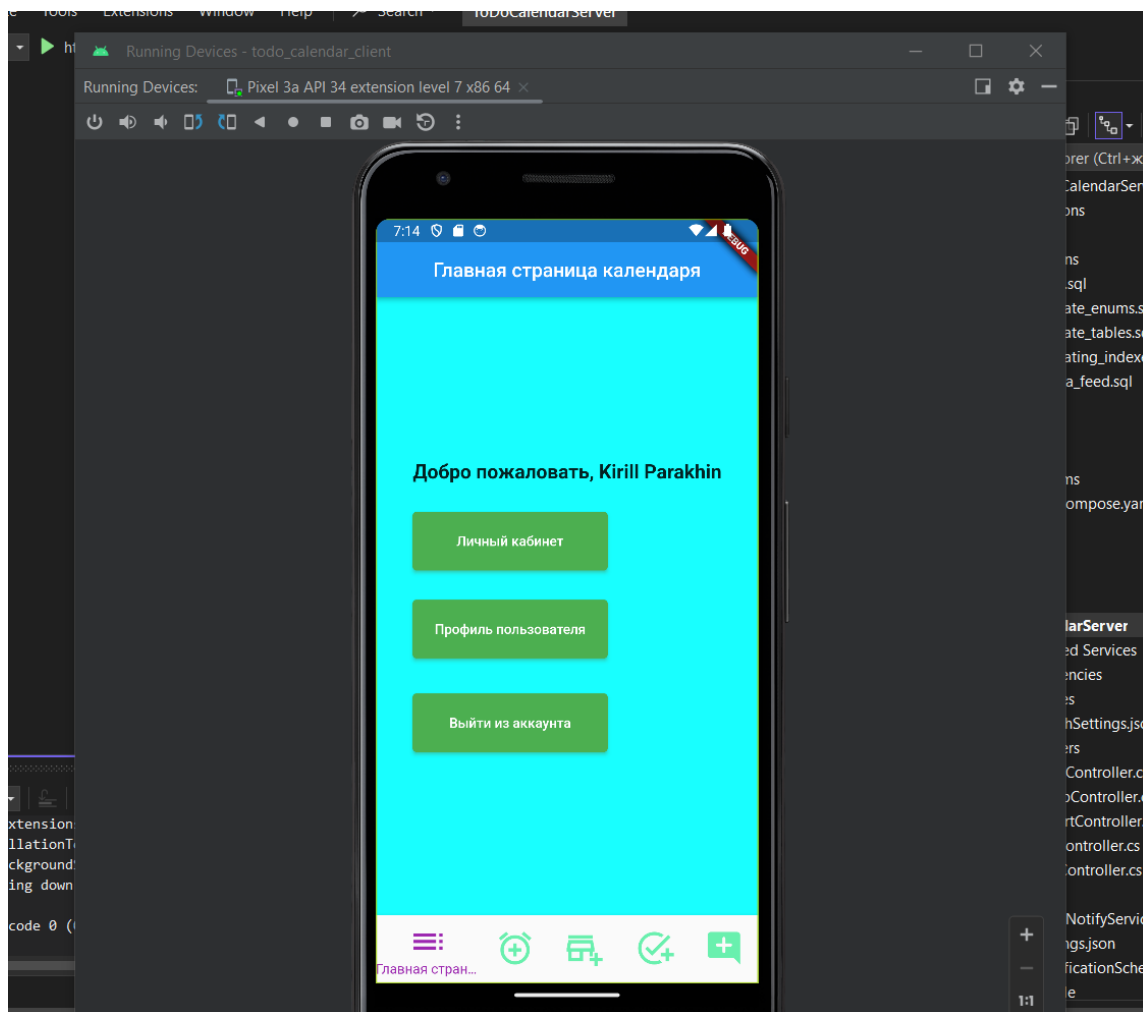


Рисунок 27. Запуск личного кабинета в эмуляторе

6. РАЗРАБОТКА ЛОГОТИПА ПРИЛОЖЕНИЯ

Для прототипа мобильного приложения «Многозадачный календарь» при помощи сервиса FreeLogoDesign был разработан собственный логотип, который включает индивидуальный ник разработчика (Tigeroff) – рисунок 28.



Рисунок 28. Разработка логотипа для мобильного приложения Tigeroff Calendar

7. ВЕДЕНИЕ РЕПОЗИТОРИЕВ ПРОЕКТА

Вся разработка велась с использованием системы контроля версий git и сервиса для хранения удаленных репозиторий Github, на котором размещен код моей программной системы по ссылке: <https://github.com/Tigeroff2002/ToDoCalendarServer> (рис. 29.1 – 29.2 – представлены скриншоты репозиторий с серверной и клиентской частью разрабатываемого курсового проекта, расположенные на сайте GitHub).

Для разработки использовался базовый функционал, предлагаемый системой git. Но, несмотря на то, что разработка велась не в команде (то есть она проводилась только мной) – все равно реализация функционала велась в отдельных ветках, на каждую из задач или изменение создавались коммиты.

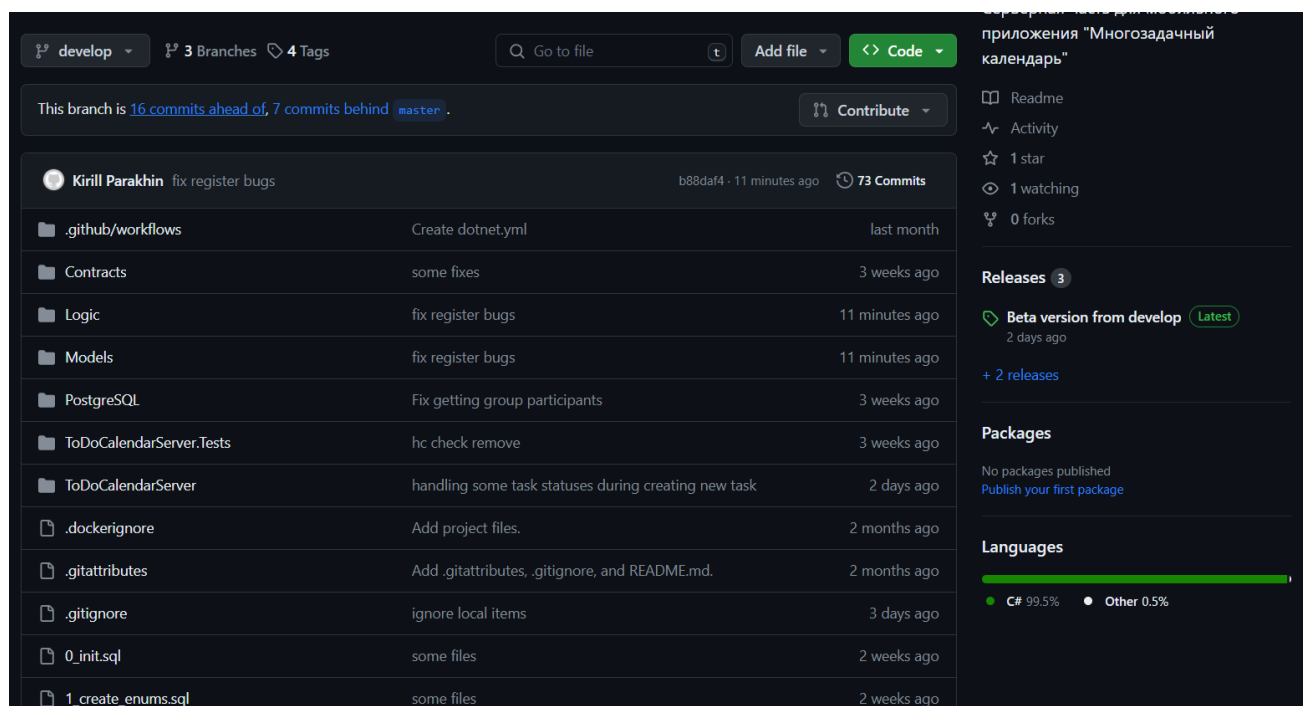


Рисунок 29.1. Репозиторий серверного ASP.NET - приложения

Репозиторий клиентского приложения доступен на Github по ссылке:
<https://github.com/Tigeroff2002/ToDoCalendarClient>

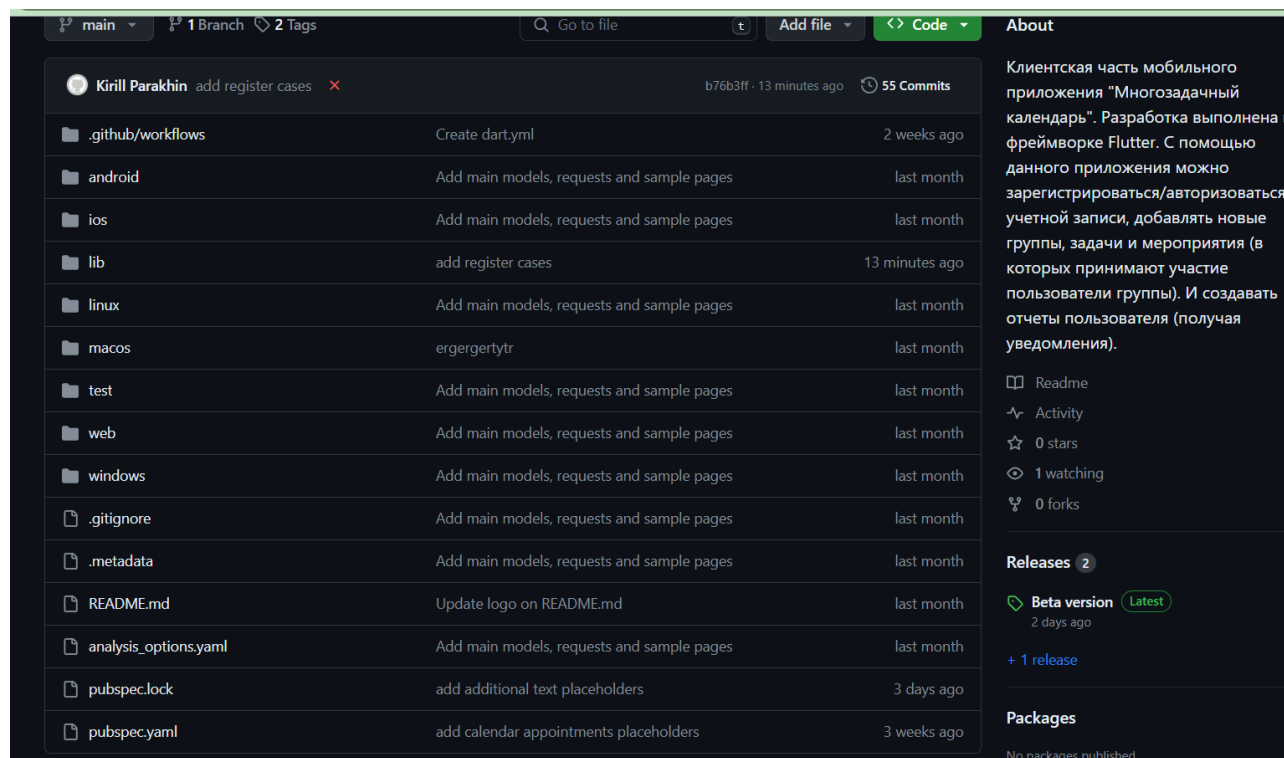


Рисунок 29.2. Репозиторий клиентского Flutter – приложения

По мере внесения изменений – проводилось автоматическая удаленная сборка серверного приложения, а также его развертывание в Docker – контейнере (смотреть рисунки 22 и 23.1)

По окончанию разработки были созданы релизы – то есть готовые собранные приложения, готовые к скачиванию и прикладному использованию.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработано мобильное клиент-серверное приложение, предоставляющее функционал удобного многозадачного календаря пользователя.

Несмотря на то, что разработка велась в одиночку – все основные функциональные требования, а также требования к интерфейсу были выполнены, цели по разработке и тестированию в полной мере достигнуты.

					ВлГУ.09.03.04.23.04.00 ПЗ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		73

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация Android Studio. [Электронный ресурс]
URL: <https://developer.android.com/guide>
2. Официальная документация React Native. [Электронный ресурс]
URL: <https://reactnative.dev/docs/getting-started>
3. Гриффитс, Д. Head First. Программирование для Android /Д. Гриффитс. – СПб.: Питер, 2016. – 704 с.
4. Нативная разработка мобильных приложений [Электронный ресурс]
/ Льюис Ш., Данн М., пер. с англ. А.Н. Киселева. - М. : ДМК Пресс, 2020.
5. Разработка мобильных приложений в среде Android Studio : учебное пособие [Электронный ресурс] / Л. В. Пирская. – Ростов н/Д : ЮФУ, 2019

ПРИЛОЖЕНИЕ А. Спецификация API (Продолжение)

Создание новой группы	
Метод	URL
POST	/groups/create
Аутентификация	Требуется (для существующего пользователя)
Тело запроса	
<pre>{ "user_id": 1, "token": "0895439408", "group_name": "Add dot net please", "group_type": "Job" }</pre>	
Ответ	
Статус	Тело
200 OK	<pre>{ "result": true, "out_info": "New group with id = 15 and name Add dot net please was created" }</pre>
Изменение параметров созданной группы	
Метод	URL
PUT	/groups/update_group_params
Авторизация	Требуется (в случае если пользователь не относится к изменяемой группе – действие отклоняется)
Тело запроса	
<pre>{ "user_id": 1, "token": "0895439408", "group_id": 15, "group_type": "Educational" }</pre>	
Ответ	
Статус	Тело
200 OK	<pre>{ "result": true, "out_info": "Group with id 15 has been modified" }</pre>
403 Forbidden	<pre>{ "result": false, "out_info": "Group has not been modified cause user not relate to that" }</pre>
Получить информацию о группе	
Метод	URL
GET	/groups/get_group_info
Авторизация	Требуется

Тело запроса	
<pre>{ "user_id": 1, "token": "0895439408", "group_id": 10 }</pre>	
Ответ	
Статус	Тело
200 OK	<pre>{ "requested_info": { "group_name": "Add dotnet pls", "group_type": "Educational", "participants": [] }, "result": true, "out_info": "Info about group with id 10 was found" }</pre>
Создать новую задачу	
Метод	URL
POST	/tasks/create
Аутентификация	Требуется
Тело запроса	
<pre>{ "user_id": 1, "token": "0895439408", "caption": "Create new api specification", "description": "Create new api asp .net core specification", "task_type": "JobComplete", "task_status": "ToDo", "implementer_id": 3 }</pre>	
Ответ	
Статус	Тело
200 OK	<pre>{ "result": true, "out_info": "New task with id = 4 for user 'Tigeroff1' implementation was created" }</pre>

Получить информацию о задаче	
Метод	URL
GET	/tasks/get_task_info
Авторизация	Требуется
Тело запроса	
<pre>{ "user_id": 1,</pre>	

<pre>"token": "0895439408", "task_id": 4 }</pre>	
Ответ	
Статус	Тело
200 OK	<pre>{ "requested_info": { "caption": "Create new api specfication", "description": "Create new api asp .net core specfication", "task_type": "JobComplete", "task_status": "InProgress", "reporter": { "user_name": "Kirill Parakhin", "user_email": "kirill.parakhin@altenar.com", "phone_number": "8-904-255-50- 27" }, "implementer": { "user_name": "Tigeroff", "user_email": "parahinvalerij5@gmail.com", "phone_number": "8-904-255-50- 27" } }, "result": true, "out_info": "Info about task with id 4 was received" }</pre>
Создать новый отчет о событиях пользователя	
Метод	URL
POST	/reports/perform_new
Авторизация	Требуется
Тело запроса	
<pre>{ "user_id": 2, "token": "0696142000", "report_type": "EventsReport", "begin_moment": "2023-10-22T00:00:00+00:00", "end_moment": "2023-11-06T00:00:00+00:00" }</pre>	

}	
Ответ	
Статус	Тело
200 OK	{ "result": true, "out_info": "New report with id = 11 for user 'Tigeroff' with type 'EventsReport' has been created" }
Получить информацию по созданному отчету	
Метод	URL
GET	/ reports/get_report_info
Авторизация	Требуется
Тело запроса	
{ "user_id": 2, "token": "0696142000", "report_id": 11 }	
Ответ	
Статус	Тело
200 OK	{ "requested_info": { "begin_moment": "2023-10-22T00:00:00+00:00", "end_moment": "2023-11-06T00:00:00+00:00", "report_type": "EventsReport", "report_content": { "creation_time": "2023-11-05T13:14:21.3063733+00:00" } }, "result": true, "out_info": "Info about report with id 11 for user with id 2 was received" }

Аналогично можно создать отчет пользователя по задачам (указав в теле запроса /reports/perform_new значения поля: "report_type": "TasksReport").

ПРИЛОЖЕНИЕ Б. КОД СЕРВЕРНОГО ПРИЛОЖЕНИЯ

Листинг основных функций класса `UserDataHandler`:

```
public async Task<Response> TryRegisterUser(
    UserRegistrationData registrationData,
    CancellationToken token)
{
    ArgumentNullException.ThrowIfNull(registrationData);

    token.ThrowIfCancellationRequested();

    var email = registrationData.Email;

    var existedUser =
        await _usersRepository
            .GetUserByEmailAsync(email, token);

    if (existedUser != null)
    {
        _logger.LogInformation(
            "User with email {Email} was already in DB",
            email);

        var response1 = new Response();
        response1.Result = false;
        response1.OutInfo = $"User with email {email} was already in DB";

        return await Task.FromResult(response1);
    }

    var shortUserInfo = new ShortUserInfo
    {
        UserEmail = email,
        UserName = registrationData.UserName,
        UserPhone = registrationData.PhoneNumber
    };

    var confirmResponse =
        await _usersCodeConfirmer.ConfirmAsync(shortUserInfo, token);

    var builder = new StringBuilder();
    builder.Append(confirmResponse.OutInfo);
    builder.Append("\n");

    var confirmResult = confirmResponse.Result;

    if (!confirmResult)
    {
        _logger.LogInformation(
            "Wrong user code confirmation received");

        var response2 = new Response();
```

```

        response2.Result = false;

        builder.Append($"Wrong user code confirmation received from user with
email {email}.");
        response2.OutInfo = builder.ToString();

        return await Task.FromResult(response2);
    }

    var user = new User();
    user.Email = email;
    user.UserName = registrationData.UserName;
    user.Password = registrationData.Password;
    user.PhoneNumber = registrationData.PhoneNumber;
    user.GroupingMaps = new List<GroupingUsersMap>();
    user.TasksForImplementation = new List<UserTask>();
    user.EventMaps = new List<EventsUsersMap>();
    user.Reports = new List<Report>();

    string authToken = GenerateNewAuthToken();

    user.AuthToken = authToken;

    _logger.LogInformation("Registrating new user {Email}", user.Email);

    await _usersRepository.AddAsync(user, token);

    _usersRepository.SaveChanges();

    var response = new ResponseWithToken();
    response.Result = true;

    builder.Append(
        $"Registrating new user {user.Email} with id {user.Id}" +
        $" with creating new auth token {authToken}");

    response.OutInfo = builder.ToString();
    response.UserId = user.Id;
    response.Token = authToken;

    return await Task.FromResult(response);
}

public async Task<Response> TryLoginUser(
    UserLoginData loginData,
    CancellationToken token)
{
    ArgumentNullException.ThrowIfNull(loginData);

    token.ThrowIfCancellationRequested();

    var email = loginData.Email;

    var existedUser =
        await _usersRepository
            .GetUserByEmailAsync(email, token);

```

```

if (existedUser == null)
{
    _logger.LogInformation(
        "User with email {Email} was not already in DB",
        email);

    var response1 = new Response();
    response1.Result = false;
    response1.OutInfo = $"User with email {email} was not already in DB";

    return await Task.FromResult(response1);
}

if (existedUser.Password != loginData.Password)
{
    _logger.LogInformation("Password not equals");

    var response2 = new Response();
    response2.Result = false;
    response2.OutInfo = "Password not equals";

    return await Task.FromResult(response2);
}

var authToken = GenerateNewAuthToken();

existedUser.AuthToken = authToken;

await _usersRepository.UpdateAsync(existedUser, token);

_usersRepository.SaveChanges();

var userName = existedUser.UserName;

var response = new ResponseWithToken();
response.Result = true;
response.OutInfo = $"Login existed user {userName} with new auth token {authToken}";
response.UserId = existedUser.Id;
response.Token = authToken;

return await Task.FromResult(response);
}

```

Листинг функций класса EventNotificationsHandler:

```
public async Task HandleEventsAsync(CancellationToken token)
{
    _logger.LogInformation("Starting handling events in target handler");

    while (!token.IsCancellationRequested)
    {
        using var scope =
            _serviceProvider.GetRequiredService<IServiceScopeFactory>().CreateScope();
        var eventRepository =
            scope.ServiceProvider.GetRequiredService<IEventsRepository>();

        var dbEvents = await eventRepository.GetAllEventsAsync(token);

        await HandleEventsNotificationsAsync(dbEvents, token);

        HandleEventStatuses(dbEvents, token);

        eventRepository.SaveChanges();

        Task.Delay(_notifyConfiguration.IterationDelay,
            token).GetAwaiter().GetResult();
    }
}

private void HandleEventStatuses(
    List<Event>? dbEvents, CancellationToken token)
{
    token.ThrowIfCancellationRequested();
    var currentTiming = DateTimeOffset.Now;

    if (dbEvents == null)
    {
        return;
    }

    foreach (var dbEvent in dbEvents)
    {
        if (dbEvent.ScheduledStart <= currentTiming)
        {
            if (dbEvent.Status != EventStatus.Cancelled
                && dbEvent.Status != EventStatus.Finished)
            {
                var oldStatus = dbEvent.Status;
                dbEvent.Status = EventStatus.Live;
                var endEvent = dbEvent.ScheduledStart.Add(dbEvent.Duration);

                if (currentTiming >= endEvent)
                {
                    dbEvent.Status = EventStatus.Finished;
                }
            }
        }
    }
}
```

```

        _logger.LogInformation(
            "Status of event {EventId} was changed" +
            " from {OldStatus} to {NewStatus}",
            dbEvent.Id,
            oldStatus,
            dbEvent.Status);
    }
}
}
}

```

```

private async Task SendMessageFromSmtpAsync(
    string eventName,
    string userName,
    string email,
    CancellationToken token)
{
    token.ThrowIfCancellationRequested();

    var smtpClient = new MailKit.Net.Smtp.SmtpClient();

    await smtpClient.ConnectAsync(
        _smtpConfiguration.Host,
        _smtpConfiguration.Port,
        useSsl: true,
        token);

    await smtpClient.AuthenticateAsync(
        _smtpConfiguration.FromAddress,
        _smtpConfiguration.FromPassword);

    _logger.LogInformation("Connected to smtp server");

    var subject = "Reminder for the near beginning of your event";

    var body = new StringBuilder();
    var numberMinutesOfOffset = _notifyConfiguration.ReminderOffset.TotalMinutes;

    body.Append($"Hello, {userName}!\n");
    body.Append(
        $"We are sending you a reminder that your event" +
        $" will start in less than {numberMinutesOfOffset} minutes.\n");
    body.Append(eventName);

    var mailMessage = new MailMessage(
        new MailAddress(_smtpConfiguration.FromAddress),
        new MailAddress(email))
    {
        Subject = subject,
        Body = body.ToString()
    };

    var response = await smtpClient.SendAsync(
        MimeMessage.CreateFromMailMessage(mailMessage), token);
}

```

ПРИЛОЖЕНИЕ В. КОД КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

Листинг класса модели ResponseWithToken:

```
class ResponseWithToken extends Response{

    final int userId;
    final String? token;

    ResponseWithToken({
        required bool result,
        String? outInfo,
        required this.userId,
        this.token
    }) :super(result: result, outInfo: outInfo);

    factory ResponseWithToken.fromJson(Map <String, dynamic> json) {
        return ResponseWithToken(
            result: json['result'],
            outInfo: json['out_info'],
            userId: json['user_id'],
            token: json['token']
        );
    }
}
```

Листинг класса модели UserRegistrationModel (запрос):

```
class UserRegisterModel {

    final String email;
    final String name;
    final String password;
    final String phoneNumber;

    UserRegisterModel({
        required this.email,
        required this.name,
        required this.password,
        required this.phoneNumber
    });

    Map<String, dynamic> toJson() {
        return {
            'email': email,
            'name': name,
            'password': password,
            'phone_number': phoneNumber
        };
    }

    String serialize() {
        return jsonEncode(toJson());
    }
}
```

Листинг виджета главной страницы пользователя:

```
class UserPage extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primarySwatch: Colors.lightGreen,
      ),
      home: Home(),
    );
  }
}

class Home extends StatefulWidget {

  @override
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  int _currentIndex = 0;

  @override
  void initState() {
    super.initState();
  }
  final List<Widget> _children = [
    PersonalAccount(
      color: Colors.red,
      text: 'Главная страница пользователя',
      index: 0
    ),
    EventPlaceholderWidget(
      color: Colors.green,
      text: 'Страница создания мероприятия',
      index: 1),
    GroupPlaceholderWidget(
      color: Colors.blueAccent,
      text: 'Страница создания новой группы',
      index: 2),
    TaskPlaceholderWidget(
      color: Colors.lime,
      text: 'Страница создания новой задачи',
      index: 3),
    ReportPlaceholderWidget(
      color: Colors.deepOrangeAccent,
      text: 'Страница создания нового отчета',
      index: 4)
  ];

  void onTabTapped(int index) {
    setState(() {
      _currentIndex = index;
    });
  }
}
```

Листинг виджета с календарными мероприятиями пользователя:

```
class EventsListPageWidget extends StatefulWidget {

  @override
  EventsListPageState createState() => EventsListPageState();
}

class EventsListPageState extends State<EventsListPageWidget> {

  @override
  void initState() {
    super.initState();
    getUserInfo();
  }

  final uri = 'http://10.0.2.2:5201/users/get_info';
  final headers = {'Content-Type': 'application/json'};
  bool isColor = false;
  final EnumAliaser aliaser = new EnumAliaser();

  List<EventInfoResponse> eventsList = [];

  Future<void> getUserInfo() async {
    MySharedPreferences mySharedPreferences = new MySharedPreferences();
    var cachedData = await mySharedPreferences.getDataIfNotExpired();

    if (cachedData != null){
      var json = jsonDecode(cachedData.toString());
      var cacheContent = ResponseWithToken.fromJson(json);
      var userId = cacheContent.userId;
      var token = cacheContent.token.toString();
      var model = new UserInfoRequestModel(userId: userId, token: token);
      var requestMap = model.toJson();

      var url = Uri.parse(uri);
      final body = jsonEncode(requestMap);

      try {
        final response = await http.post(url, headers: headers, body: body);

        var jsonData = jsonDecode(response.body);
        var responseContent = GetResponse.fromJson(jsonData);

        if (responseContent.result) {
          var userRequestedInfo = responseContent.requestedInfo.toString();

          var data = jsonDecode(userRequestedInfo);
          var userEvents = data['user_events'];
          var fetchedEvents =
            List<EventInfoResponse>
              .from(userEvents.map(
                (data) => EventInfoResponse.fromJson(data)));
          setState(() {
            eventsList = fetchedEvents;
          });
        }
      } catch (e) {
        if (e is SocketException) {
          //treat SocketException
          print("Socket exception: ${e.toString()}");
        }
      }
    }
  }
}
```



```

    }
    else if (e is TimeoutException) {
        //treat TimeoutException
        print("Timeout exception: ${e.toString()}");
    }
    else
        print("Unhandled exception: ${e.toString()}");
}
}
else {
    setState(() {
        showDialog(
            context: context,
            builder: (context) => AlertDialog(
                title: Text('Ошибка!'),
                content:
                    Text(
                        'Произошла ошибка при получении'
                        ' полной информации о пользователе!'),
                actions: [
                    TextButton(
                        onPressed: () {
                            Navigator.pop(context);
                        },
                        child: Text('OK'),
                    ),
                ],
            ),
        );
    });
}
}

@override
Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(
                title: Text('Календарь мероприятий'),
                leading: IconButton(
                    icon: Icon(Icons.arrow_back),
                    onPressed: () {
                        Navigator.pushReplacement(
                            context,
                            MaterialPageRoute(
                                builder: (context) => UserPage()),
                        );
                    },
                ),
            body: SfCalendar(
                view: CalendarView.week,
                firstDayOfWeek: 1,
                initialDisplayDate: DateTime.now(),
                initialSelectedDate: DateTime.now(),
                dataSource: MeetingDataSource(getAppointments(eventsList)),
            ),
        ),
    );
}
}

```