Оглавление

[Введение. 3](#_Toc133941772)

[Постановка задачи. 3](#_Toc133941773)

[Цели и задачи. 3](#_Toc133941774)

[Описание архитектуры приложения. 4](#_Toc133941775)

[Разработка базы данных. 5](#_Toc133941776)

[Таблица Users. 5](#_Toc133941777)

[Таблица Files. 5](#_Toc133941778)

[Разработка серверной части. 7](#_Toc133941779)

[Немного о коде сервера. 7](#_Toc133941780)

[Разработка мобильного приложения. 14](#_Toc133941781)

[Окно входа. 14](#_Toc133941782)

[Окно списка дел/задач. 15](#_Toc133941783)

[Окно добавления дел/задач. 15](#_Toc133941784)

[Окно фокусирования. 16](#_Toc133941785)

[Тестирование приложения. 18](#_Toc133941786)

[Заключение 19](#_Toc133941787)

[Список использованных источников 20](#_Toc133941788)

# Введение.

## Постановка задачи.

Целью данной курсовой работы является написать приложение, удобного и простого в использовании инструмента, который позволит пользователям вести список задач и сосредоточиться на выполнении самых важных из них. В частности, данное приложение будет предоставлять следующие возможности:

1. Создание списка задач с возможностью установки приоритетов и категорий для каждой задачи.
2. Отслеживание прогресса выполнения задач.

## Цели и задачи.

Цель данного проекта - создание мобильного приложения, которое поможет пользователям стать более продуктивными, сфокусированными и успешными в выполнении своих задач и целей. Задачи данного проекта следующие:

1. Разработка удобного и интуитивно понятного интерфейса приложения, который будет привлекать пользователей и делать использование приложения легким и приятным.
2. Разработка функционала создания и удаления задач и целей, а также установки приоритетов, дедлайнов и категорий для каждой задачи.
3. Разработка функционала напоминаний и уведомлений, чтобы пользователи не забывали о предстоящих задачах и могли успевать их выполнить в срок.
4. Разработка функционала отслеживания прогресса выполнения задач и целей, а также возможности установки и отслеживания дополнительных целей, которые помогут пользователям добиваться большего успеха.

# Описание архитектуры приложения.

Архитектура мобильного приложения "Неотложные дела" разработана с учетом многих факторов, включая скорость и производительность, масштабируемость, безопасность, удобство использования и простоту разработки.

Приложение разделено на несколько основных компонентов: клиентскую часть, серверную часть и базу данных. Клиентская часть написана на языке Swift, используя современные методы и технологии, такие как MVVM (Model-View-ViewModel) архитектура, SwiftUI — фреймворк. Эти методы и технологии позволяют нам разрабатывать приложение более эффективно, повышая производительность и удобство использования.

Серверная часть написана на языке Python, используя фреймворк Flask. Flask позволяет создавать RESTful API, который обеспечивает взаимодействие между клиентской частью и базой данных, а также обеспечивает безопасность и аутентификацию пользователей.

База данных, используемая в приложении, это MySQL. MySQL является одной из наиболее популярных реляционных баз данных, которая обеспечивает масштабируемость и безопасность нашего приложения.

В целом, архитектура нашего приложения "Неотложенные дела" разработана с учетом многих факторов и использует самые современные методы и технологии, чтобы обеспечить нашим пользователям максимальную производительность, удобство использования и безопасность.

# Разработка базы данных.

Конечная цель нашего приложения "Неотложенные дела" - помочь пользователям лучше организовать свои задачи и цели. Для этого мы разработали базу данных, которая состоит из двух таблиц: Users и files.

## Таблица Users.

Таблица Users содержит информацию о пользователях приложения, включая их уникальный идентификатор (id), имя пользователя (userName) и пароль (password). Уникальный идентификатор (id) генерируется автоматически при создании новой записи.

Она была создана следующим образом:

CREATE TABLE `users` (

`id` int NOT NULL AUTO\_INCREMENT,

`username` varchar(255) NOT NULL,

`password` varchar(255) NOT NULL,

PRIMARY KEY (`id`)

)ENGINE=InnoDB AUTO\_INCREMENT=25

DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

## Таблица Files.

Таблица files содержит информацию о задачах и целях пользователей приложения. Каждая запись в таблице files связана с пользователем из таблицы Users посредством поля user\_id. Другие поля, содержащиеся в таблице files, включают в себя имя задачи (taskName), дату (date), приоритет (priority), теги (tags) и заметки (notes). Уникальный идентификатор (id) генерируется автоматически при создании новой записи.

Она была создана следующим образом:

CREATE TABLE `files` (

`id` int NOT NULL AUTO\_INCREMENT,

`user\_id` int NOT NULL,

`taskName` varchar(255) DEFAULT NULL,

`date` datetime DEFAULT NULL,

`priority` varchar(1) DEFAULT NULL,

`tags` varchar(25) DEFAULT NULL,

`notes` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `fk\_files\_users\_idx` (`user\_id`),

CONSTRAINT `fk\_files\_users` FOREIGN KEY (`user\_id`) REFERENCES `files` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

)

ENGINE=InnoDB AUTO\_INCREMENT=21

DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

Использование реляционной базы данных, такой как MySQL, позволяет нам эффективно организовать и хранить данные пользователей. Мы можем легко получать информацию о задачах и целях пользователей, а также о пользовательских учетных записях. Также база данных обеспечивает защиту данных пользователей и контролирует доступ к ним посредством аутентификации и авторизации.

# Разработка серверной части.

Данный сервер написан на языке программирования Python с использованием фреймворка Flask. Он представляет собой RESTful API, который обрабатывает запросы клиентов и возвращает соответствующие им ответы.

Этот сервер состоит из нескольких компонентов, включая файлы с маршрутами, модули базы данных и утилиты для работы с файлами. Каждый маршрут определен в соответствующем файле и связан с определенной функцией обработчика.

В целом, этот сервер представляет собой надежную и функциональную основу для приложения управления задачами. Он может быть легко масштабирован и настроен в соответствии с потребностями другого приложения.

## Немного о коде сервера.

В коде сервера присутствуют такие функции:

create\_user() - функция, которая создаёт нового пользователя в базе данных. Он вызывается при отправке POST запроса на эндпоинт /create\_user.

Пример использования:

@app.route('/create-user', methods=['POST'])

def create\_user():

data = request.json

name = data['username']

passw = data['password']

try:

connection = pymysql.connect(

host=host,

port=3306,

user=user,

password=password,

database=db\_name,

cursorclass=pymysql.cursors.DictCursor

)

with connection.cursor() as cursor:

select\_query = "SELECT \* FROM users WHERE username = '" + name + "';"

cursor.execute(select\_query)

if cursor.fetchone() is not None:

response = {'message': 'User with this username already exists'}

status\_code = 409

else:

insert\_query = "INSERT INTO users (username, password) VALUES ('" + name + "', '" + passw + "');"

cursor.execute(insert\_query)

connection.commit()

response = {'message': 'User created successfully'}

status\_code = 201

except Exception as ex:

print(ex)

response = {'message': 'Error creating user'}

status\_code = 500 # Код HTTP 500 Internal Server Error возвращается, если произошла внутренняя ошибка сервера

return jsonify(response), status\_code

Алгоритм таков:

1. Получение json-файла с именем и паролем пользователя.
2. Попытка присоединиться к базе данных.
3. Если попытка присоединения успешная то он отправляет запрос на проверку существования пользователя с таким именем в базе данныx.
4. Если пользователь найден, то формируется сообщение о том что пользователь есть в базе данных.
5. Если же пользователя с таким именем нет, то он создаёт такого пользователя.
6. Если же на каком то этапе он не что то пошло не так, то он формирует ошибку с текстом и кодом ошибки.
7. После код и текст объединияются в один json - файл и возвращаются по адресу отправки.

Login\_user() - функция, которая проверяет на существования пользователя с отправленным именем и паролем. Она вызывается при отправке POST запроса на эндпоинт /user.

Пример использования:

@app.route('/user', methods=['POST'])

def Login\_user():

data = request.json

name = data['username']

passw = data['password']

try:

connection = pymysql.connect(

host=host,

port=3306,

user=user,

password=password,

database=db\_name,

cursorclass=pymysql.cursors.DictCursor

)

with connection.cursor() as cursor:

select\_query = "SELECT \* FROM users WHERE username = '" + name + "' AND password = '" + passw + "';"

cursor.execute(select\_query)

if cursor.fetchone() is not None:

response = {'message': 'User with this username already exists'}

status\_code = 409 # Код HTTP 409 Conflict возвращается, если ресурс уже существует

else:

response = {'message': 'User with this username not find'}

status\_code = 404 # Код HTTP 404 Запрашиваемый ресурс не найден на сервере

except Exception as ex:

print(ex)

response = {'message': 'Error Login user'}

status\_code = 500 # Код HTTP 500 Internal Server Error возвращается, если произошла внутренняя ошибка сервера

return jsonify(response), status\_code

Алгоритм таков:

1. Получение json-файла с именем и паролем пользователя.
2. Попытка присоединиться к базе данных.
3. Если попытка присоединения успешная то он отправляет запрос на проверку существования пользователя с таким именем в базе данныx.
4. Если пользователь найден, формирует сообщение и статус-код
5. Если же такого пользователя нет в базе данных либо пользователь был найден но пароль не правильный то он формирует ошибку с текстом и кодом ошибки.
6. После код и текст объединияются в один json - файл и возвращаются по адресу отправки.

add\_data() - функция, которая добавляет новую задачу в базу данных. Она вызывается при отправке POST запроса на эндпоинт /add\_data.

Пример использования:

@app.route('/add\_data', methods=['POST'])

def add\_data():

data = request.json

name = data['username']

passw = data['password']

try:

connection = pymysql.connect(

host=host,

port=3306,

user=user,

password=password,

database=db\_name,

cursorclass=pymysql.cursors.DictCursor

)

with connection.cursor() as cursor:

select\_user = "SELECT \* FROM users WHERE username = %s AND password = %s"

cursor.execute(select\_user, (name, passw))

usr = cursor.fetchone()

if not usr:

response = {'message': 'Invalid username or password'}

status\_code = 401

else:

insert\_file = "INSERT INTO files (user\_id, taskName, date, priority, tags, notes) VALUES (%s, %s, %s, %s, %s, %s)"

cursor.execute(insert\_file, (usr['id'], data['taskName'], data['date'], data['priority'], data['tags'], data['notes']))

connection.commit()

response = {'message': 'Data added successfully'}

status\_code = 200

except Exception as ex:

print(ex)

response = {'message': 'Error adding data'}

status\_code = 500

finally:

connection.close()

return jsonify(response), status\_code

Алгоритм таков:

1. Получение json-файла с именем и паролем пользователя.
2. Попытка присоединиться к базе данных.
3. Если попытка присоединения успешная то он отправляет запрос на проверку существования пользователя с таким именем в базе данныx.
4. Если пользователь найден, то он узнаёт его id и отправляет Insert запрос для добавления элемента.
5. Если всё прошло успешно то он создаёт json - файл с кодом успешного добавления и текстом .
6. Если же на каком то этапе он не что то пошло не так, то он формирует ошибку с текстом и кодом ошибки.
7. После код и текст объединияются в один json - файл и возвращаются по адресу отправки.

get\_data() - функция, которая добавляет новую задачу в базу данных. Она вызывается при отправке POST запроса на эндпоинт /\verb|data|.

Пример использования:

@app.route('/data', methods=['POST'])

def get\_data():

data = request.json

name = data['username']

passw = data['password']

try:

connection = pymysql.connect(

host=host,

port=3306,

user=user,

password=password,

database=db\_name,

cursorclass=pymysql.cursors.DictCursor

)

with connection.cursor() as cursor:

# find the user with the given username and password

select\_user = "SELECT \* FROM users WHERE username = %s AND password = %s"

cursor.execute(select\_user, (name, passw))

usr = cursor.fetchone()

if not usr:

# if user not found, return an error response

response = {'message': 'Invalid username or password'}

status\_code = 401

else:

# find the files with the same user\_id as the found user

select\_files = "SELECT \* FROM files WHERE user\_id = %s"

cursor.execute(select\_files, (usr['id'],))

files = cursor.fetchall()

response = {'files': files}

status\_code = 200

except Exception as ex:

print(ex)

response = {'message': 'Error getting files'}

status\_code = 500

finally:

connection.close()

return jsonify(response), status\_code

Алгоритм таков:

1. Получение json-файла с именем и паролем пользователя.
2. Попытка присоединиться к базе данных.
3. Если попытка присоединения успешная то он отправляет запрос на проверку существования пользователя с таким именем в базе данныx.
4. Если пользователь найден, то он узнаёт его id и отправляет Select запрос на получение данных данного пользователя
5. Если всё прошло успешно то он создаёт json - файл с ответом от базы данных (с таблицей данных).
6. Если же на каком то этапе он не что то пошло не так, то он формирует ошибку с текстом и кодом ошибки.
7. После код и текст объединияются в один json - файл и возвращаются по адресу отправки.

# Разработка мобильного приложения.

"Неотложные дела" - это мобильное приложение для управления списком задач на iOS. Приложение имеет несколько основных окон, каждое из которых выполняет свою функцию.

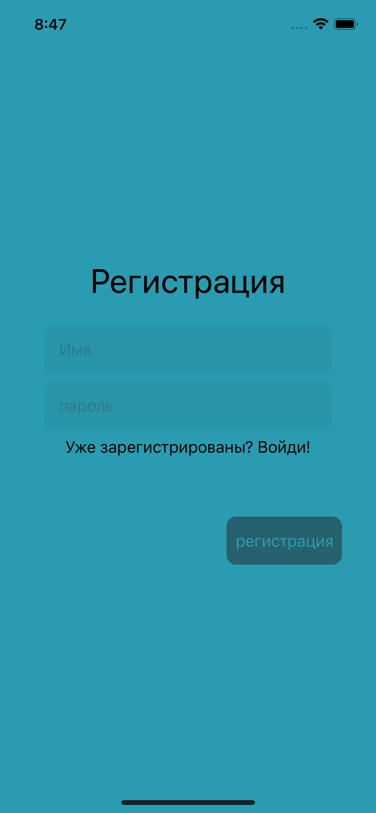
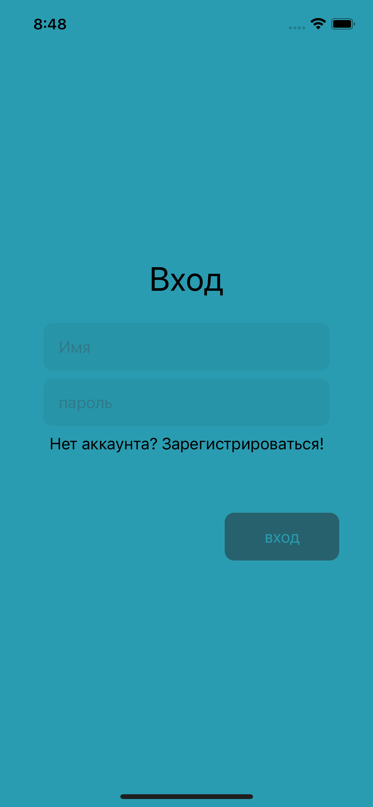
В данном приложении присутствуют 4 окна:

1. Окно входа/регистрации.
2. Окно списка дел/задач.
3. Окно добавления дел/задач.
4. Окно фокусирования.

## Окно входа.

Окно входа - это первое окно, которое увидит пользователь при запуске приложения. Здесь пользователю будет предложено войти в свой аккаунт, если у него уже есть учетная запись, или зарегистрироваться, если ее еще нет.

На этом окне находится два поля: для ввода логина и пароля. Также есть кнопка "Войти" и кнопка "Зарегистрироваться". В случае, если пользователь нажмет на кнопку "Зарегистрироваться", он будет перенаправлен на страницу регистрации, где ему потребуется ввести свои данные для создания нового аккаунта.

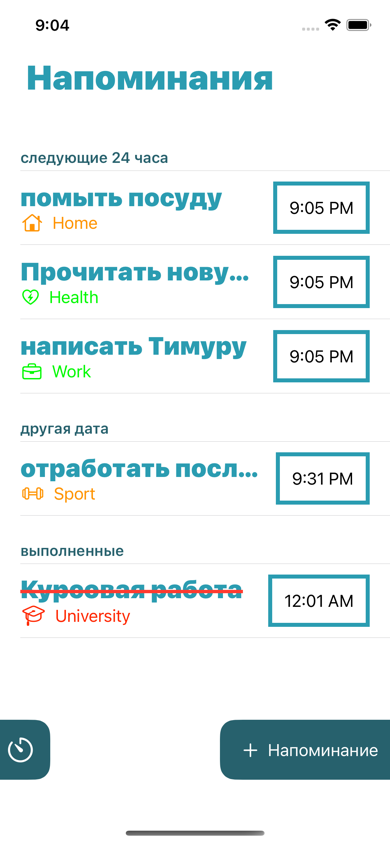


## Окно списка дел/задач.

После успешной аутентификации пользователя в приложении, он будет перенаправлен на главный экран приложения. Главный экран приложения отображает список всех задач, которые пользователь создал. Список дел разбит на три группы: задачи, которые должны быть выполнены в течение 24 часов, задачи, срок выполнения которых истек после 24 часов и список выполненных задач.

Каждая задача в списке будет содержать информацию, такую как: название задачи, дата выполнения, приоритет и категория. Кроме того, у каждую задачу можно нажать и высветится подробная информация о задачи.

Внизу главного экрана будет расположена кнопка "Добавить", которая перенаправляет пользователя на экран добавления новой задачи. Также на главном экране есть кнопка "Фокусирование ", которая открывает экран с функцией таймера обратного отсчета.



## Окно добавления дел/задач.

Окно добавления дел/задач позволяет пользователю добавить новую задачу в свой список дел. При открытии окна пользователю отображается форма для заполнения информации о задаче, включая:

Название - это обязательное поле, в котором пользователь указывает краткое название задачи.

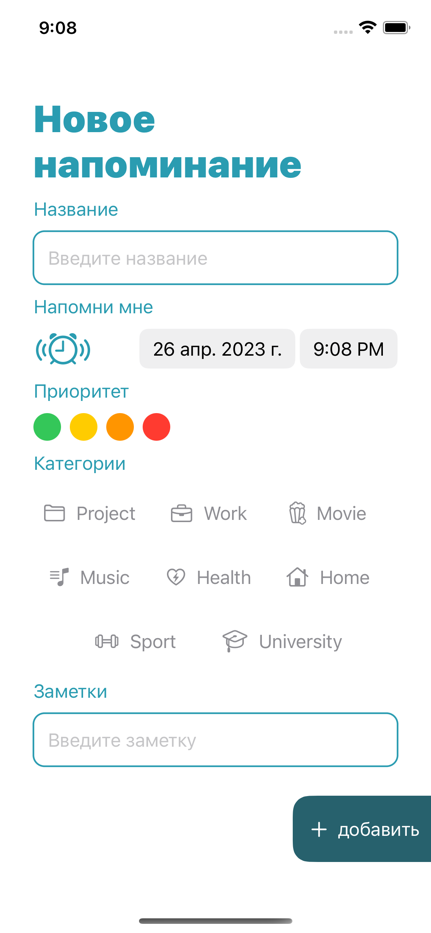
Дата выполнения - поле для указания даты, когда задачу нужно выполнить. Если задача не имеет конкретного срока выполнения, можно оставить это поле пустым.

Приоритет - поле для выбора приоритета задачи. Возможные варианты могут включать "Зелёный", "Жёлтый", "Оранжевый" и "Красный".

Категория - поле для выбора категории, к которой относится задача.

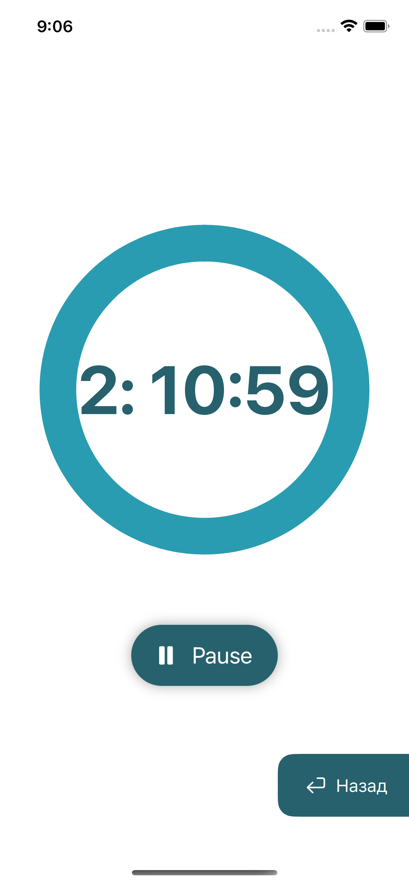
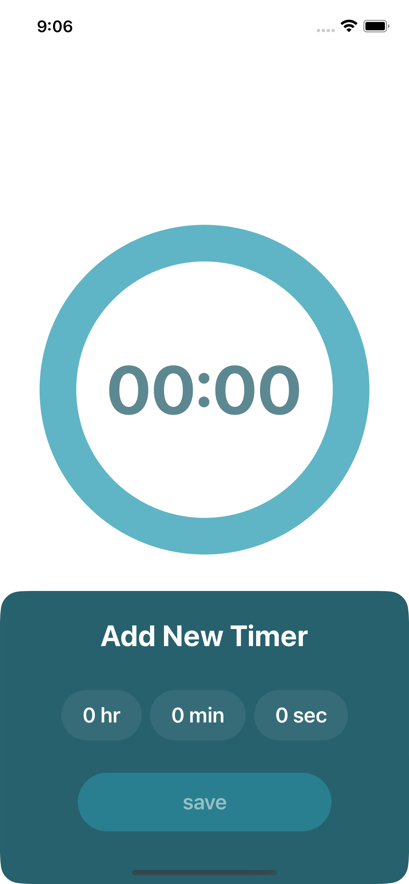
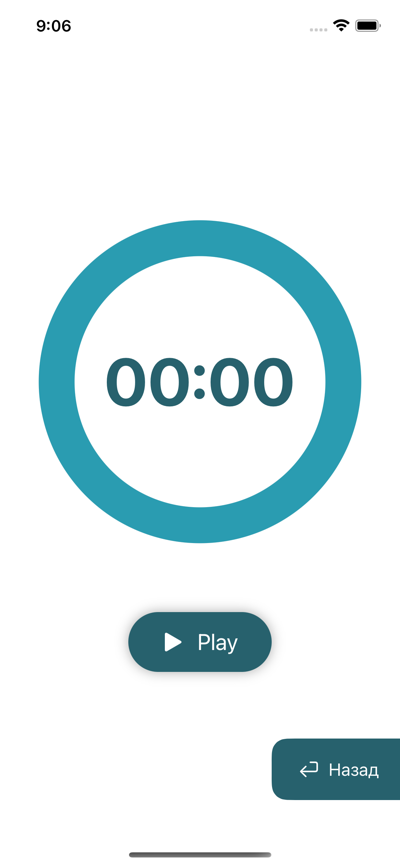
Заметки - необязательное поле, в котором можно указать более подробную информацию о задаче.

Когда пользователь заполняет информацию о задаче, он может сохранить ее, нажав на кнопку "Добавить". Если пользователь попытается сохранить задачу без указания названия, то напоминание не добавится.



## Окно фокусирования.

В данном окне пользователь может настроить таймер, чтобы отслеживать время, потраченное на задачу. Когда пользователь запускает таймер, он начинает обратный отсчет от указанного времени, которое можно настроить на свое усмотрение. Например, пользователь может установить таймер на 25 минут (такой метод работы называется "техника помидора"), после чего он получит уведомление о том, что время работы истекло.



# Тестирование приложения.

Тестирование это один из важнейших этапов разработки приложения, ведь во время тестирования можно узнать ошибки сделанные программистами и оперативно устранить их, в противном случае пользователи получат не готовый, не доделанное приложение.

Интерфейс данного приложения тестировался в большинства эмуляторах приложения xCode, таких устройств как IPhone 13, Iphone 14 Pro, Iphone SE (3rd generation) а так же Iphone 12 mini. Серверная часть тестировалась на моём личном устройстве, IPhone 13. Тестирование приложения проходило 65 дней. В течении этих дней оперативно были исправлены куча багов и недочётов, так же пару раз было переписано всё приложение.

# Заключение

Курсовая работа была посвящена разработке серверного приложения для управления задачами. Для создания сервера был использован язык программирования Python с использованием фреймворка Flask и базы данных SQL.

В результате работы был разработан сервер, который предоставляет API для создания, удаления задач пользователей. Для работы с сервером были созданы функции добавления и получения данных, а также функции аутентификации и создания пользователей.

Были реализованы механизмы аутентификации и авторизации пользователей с использованием токенов доступа.

В целом, разработанный сервер является эффективным и безопасным решением для управления задачами и может быть использован в различных проектах, где необходимо управление задачами и контроль доступа пользователей.

# Список использованных источников

1. 100 Days of SwiftUI // hackingwithswift URL: https://www.hackingwithswift.com/100/swiftui (дата обращения: 13.10.2022)
2. Как подключиться к MySQL на Python // PythonToday URL: https://www.youtube.com/watch?v=LS42t1VMwuM&t=363s (дата обращения: 21.01.2023).
3. Андрианова А.A., Мухтарова Т.М., Рубцова Р.Г. ТЕХНОЛОГИИ БАЗ ДАННЫХ. - 1-е изд. - Казань: Казанский университет, 2016. - 98 с.
4. Complete SwiftUI Firebase Tutorial: Auth, Sign Up Page, Cloud Firestore, Read & Write Data // Logan Koshenka URL: https://www.youtube.com/watch?v=6b2WAePdiqA&t (дата обращения: 23.01.2023).
5. Как установить и использовать MySQL Workbench // Selectel URL: https://selectel.ru/blog/tutorials/mysql-workbench-installation/ (дата обращения: 15.02.2023).
6. How to Connect to a Database with MySQL Workbench // immotionhosting URL: https://www.inmotionhosting.com/support/website/connect-database-remotely-mysql-workbench/ (дата обращения: 18.02.2023).
7. Flask user’s guide // flask URL: https://flask.palletsprojects.com/en/2.2.x/ (дата обращения: 19.02.2023).