|  |  |  |
| --- | --- | --- |
|  | Изображение выглядит как зарисовка, рисунок, символ, Штриховая графика  Автоматически созданное описание |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**Отчет по итоговому проекту**

по дисциплине

«**Технология разработки программных приложений**»

**Часть 1**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-12-22 | Солобай А.П |
| **Проверил:** | ассистент Петрова А.А. |

МОСКВА 2024 г.

**Оглавление**

[Описание проекта 3](#_Toc159521405)

[Задание 4](#_Toc159521406)

[Часть 1 4](#_Toc159521407)

[Часть 2 4](#_Toc159521408)

[Часть 3 4](#_Toc159521409)

[Часть 4 4](#_Toc159521410)

[Выполнение заданий 5](#_Toc159521411)

[Часть 1 5](#_Toc159521412)

[Часть 2 7](#_Toc159521413)

[Часть 3 12](#_Toc159521414)

[Часть 4 13](#_Toc159521415)

# Описание проекта

**Состав команды:** *Солобай А.П*

**Назначение проекта:** Создание интеллектуального помощника в Telegram для облегчения повседневных задач пользователей.

**Цель:** Целью проекта является создание удобного инструмента, который поможет пользователям в решении повседневных задач, предоставляя им информацию и поддержку в режиме реального времени через мессенджер Telegram.

**Задачи:**

1. Разработка базовой структуры бота на языке Python.

2. Интеграция бота с API Telegram для обработки сообщений и взаимодействия с пользователями.

3. Создание алгоритмов обработки запросов пользователей и предоставления ответов.

4. Интеграция с внешними сервисами и API для получения дополнительной информации (погода).

5. Оптимизация работы бота и обеспечение его стабильной работы в течение длительного времени.

6. Тестирование бота на различных сценариях использования и исправление возможных ошибок и недочетов.

7. Развитие функционала бота на основе обратной связи от пользователей и их потребностей.

**Описание проекта:**

Интеллектуальный помощник в Telegram будет представлять собой бота на языке программирования Python, способного выполнять различные задачи и предоставлять полезную информацию. Он будет использовать различные алгоритмы и API для взаимодействия с пользователями, отвечая на их сообщения.

Задание

## Часть 1

Сформулируйте функциональные требования к своему проекту. Требования должны быть достаточными для описания программного продукта, который вы будете разрабатывать. Рекомендуется оформить требования в виде User Story.

## Часть 2

Оформите User Story в виде карточек. Сделать макет.

## Часть 3

Создайте репозиторий для своего проекта и обеспечьте к нему удаленный доступ для всех членов команды.

## Часть 4

Начинайте разработку проекта. Снабжайте программный код разработанных

классов, методов, функций и т.д. комментариями. Код должен быть легко читаем. В процессе разработки программного продукта необходимо пользоваться выбранной системой контроля версий.

# Выполнение заданий

## Часть 1

1. Как пользователь, я хочу иметь возможность сгенерировать обычный QR-код, чтобы легко передавать информацию через него.

2. Как пользователь, я хочу иметь возможность создать QR-код для подключения к Wi-Fi, чтобы быстро и безопасно передавать данные для подключения к сети.

3. Как пользователь, я хочу иметь возможность сгенерировать надежный пароль определенной длины, чтобы обеспечить безопасность своих аккаунтов и данных.

4. Как пользователь, я хочу иметь возможность получить случайное число из заданного диапазона, чтобы использовать его для различных целей, например, в играх или приложениях для случайных выборов.

5. Как пользователь, я хочу иметь возможность подбросить кубик, чтобы использовать его для принятия случайных решений или в игровых ситуациях.

6. Как пользователь, я хочу получить прогноз погоды для определенного города, чтобы быть в курсе текущих и будущих погодных условий.

7. Как пользователь, я хочу иметь возможность создавать, просматривать и управлять своими заметками, чтобы организовать свои записи и пометки.

8. Как пользователь, я хочу иметь возможность оставить отзыв или задать вопрос об использовании бота, чтобы улучшить работу бота и получить ответ на свой вопрос.

9. Как администратор, я хочу иметь возможность отправлять массовые сообщения всем пользователям бота, чтобы оперативно информировать их о важных событиях или обновлениях.

10. Как администратор, я хочу иметь возможность отправлять персонализированные сообщения конкретному пользователю, чтобы предоставлять индивидуальную поддержку и помощь.

11. Как администратор, я хочу иметь возможность осуществлять поиск пользователей в базе данных по различным параметрам, чтобы эффективно управлять их аккаунтами.

12. Как администратор, я хочу иметь возможность блокировать и разблокировать пользователей, чтобы контролировать доступ к функционалу бота и обеспечивать безопасность сообщества.

13. Как администратор, я хочу иметь возможность просматривать отзывы и вопросы пользователей о работе бота, чтобы эффективно реагировать на запросы и улучшать качество обслуживания.

## Часть 2

Таблица 1 – карточка для обычного QR-кода

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о генерации QR-кода |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Сгенерировать QR-код для передачи обычных данных (текст, URL и т. д.) |
| Цель | Чтобы | Легко поделиться информацией с другими пользователями через сканирование QR-кода. |
| Функциональное требование |  | Функция генерации QR-кода с выбором цвета |

Таблица 2 – карточка для Wi-Fi QR-кода

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о генерации Wi-Fi QR-кода |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Сгенерировать QR-код для подключения к Wi-Fi сети. |
| Цель | Чтобы | Упростить процесс передачи параметров сети для подключения других устройств. |
| Функциональное требование |  | Функция генерации QR-кода для подключения к Wi-Fi сети. |

Таблица 3 – карточка для генератора паролей

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о генерации паролей |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Сгенерировать надежный пароль заданной длины. |
| Цель | Чтобы | Обеспечить безопасность и случайность пароля при создании учетных записей или защищенных данных. |
| Функциональное требование |  | Функция генерации надежного пароля заданной длины. |

Таблица 4 – карточка для бросания кубика

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о бросании кубика |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Бросить кубик и получить случайный результат. |
| Цель | Чтобы | Использовать бота для развлечения или принятия случайных решений. |
| Функциональное требование |  | Функция бросания кубика и получения случайного результата. |

Таблица 5 – карточка для получения случайного числа из заданного диапазона

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о бросании кубика |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Иметь возможность получить случайное число из заданного диапазона. |
| Цель | Чтобы | Использовать его для различных целей, например, в играх или приложениях. |
| Функциональное требование |  | Функция генерации случайного числа из заданного диапазона. |

Таблица 6 – карточка для прогноза погоды

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о прогнозе погоды |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Получить прогноз погоды для заданного города. |
| Цель | Чтобы | Быть проинформированным о погодных условиях в выбранном месте. |
| Функциональное требование |  | Функция получения прогноза погоды для заданного города. |

Таблица 7 – карточка для менеджера заметок

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о менеджере заметок |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Создавать, просматривать, редактировать и удалять заметки. |
| Цель | Чтобы | Организовать свои мысли, планы или списки дел с помощью бота. |
| Функциональное требование |  | Функции создания, просмотра, редактирования и удаления заметок. |

Таблица 8 – карточка для массовой рассылки

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о массовой рассылке |
| Заказчик (actor) | Как | Администратор |
| Примечание | Я хочу | Отправлять сообщения всем пользователям бота. |
| Цель | Чтобы | Распространять важную информацию или объявления среди пользователей. |
| Функциональное требование |  | Функция отправки сообщения всем пользователям бота. |

Таблица 9 – карточка для рассылки конкретным пользователям

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о персонализированной рассылке |
| Заказчик (actor) | Как | Администратор |
| Примечание | Я хочу | Отправлять персонализированные сообщения определенному пользователю. |
| Цель | Чтобы | Общаться с пользователями индивидуально и предоставлять персонализированную поддержку или информацию. |
| Функциональное требование |  | Функция отправки персонализированных сообщений определенному пользователю. |

Таблица 10 – карточка для поиска пользователей

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о поиске пользователей |
| Заказчик (actor) | Как | Администратор |
| Примечание | Я хочу | Искать пользователя в базе данных по идентификатору, имени или другим параметрам. |
| Цель | Чтобы | Найти нужного пользователя для проведения административных действий или предоставления поддержки. |
| Функциональное требование |  | Функция поиска пользователя в базе данных по идентификатору, имени или другим параметрам. |

Таблица 11 – карточка для бана/разбана пользователей

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о бане/разбане пользователей |
| Заказчик (actor) | Как | Администратор |
| Примечание | Я хочу | Блокировать или разблокировать доступ конкретного пользователя к боту. |
| Цель | Чтобы | Управлять безопасностью и поведением пользователей в системе. |
| Функциональное требование |  | Функция блокировки или разблокировки доступа конкретного пользователя к боту. |

Таблица 12 – карточка для отправки фидбека

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о фидбеке |
| Заказчик (actor) | Как | Пользователь |
| Примечание | Я хочу | Иметь возможность оставить отзыв или задать вопрос об использовании бота. |
| Цель | Чтобы | Улучшить работу бота. |
| Функциональное требование |  | Функция для написания отзывов и вопросов об использовании бота |

Таблица 13 – карточка для получения фидбека

|  |  |  |
| --- | --- | --- |
| Заголовок | | Информация о фидбеке |
| Заказчик (actor) | Как | Администратор |
| Примечание | Я хочу | Иметь возможность просматривать отзывы и вопросы пользователей о работе бота. |
| Цель | Чтобы | Эффективно реагировать на запросы и улучшать качество обслуживания. |
| Функциональное требование |  | Функция просмотра отзывов и вопросов пользователей о работе бота. |

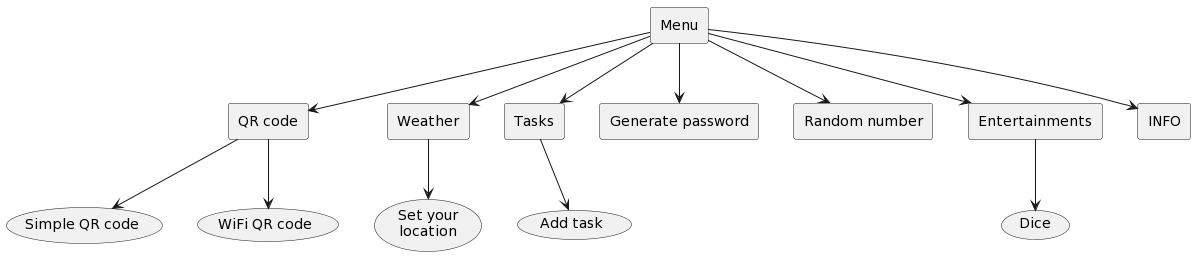


Рисунок 1 – макет

## Часть 3

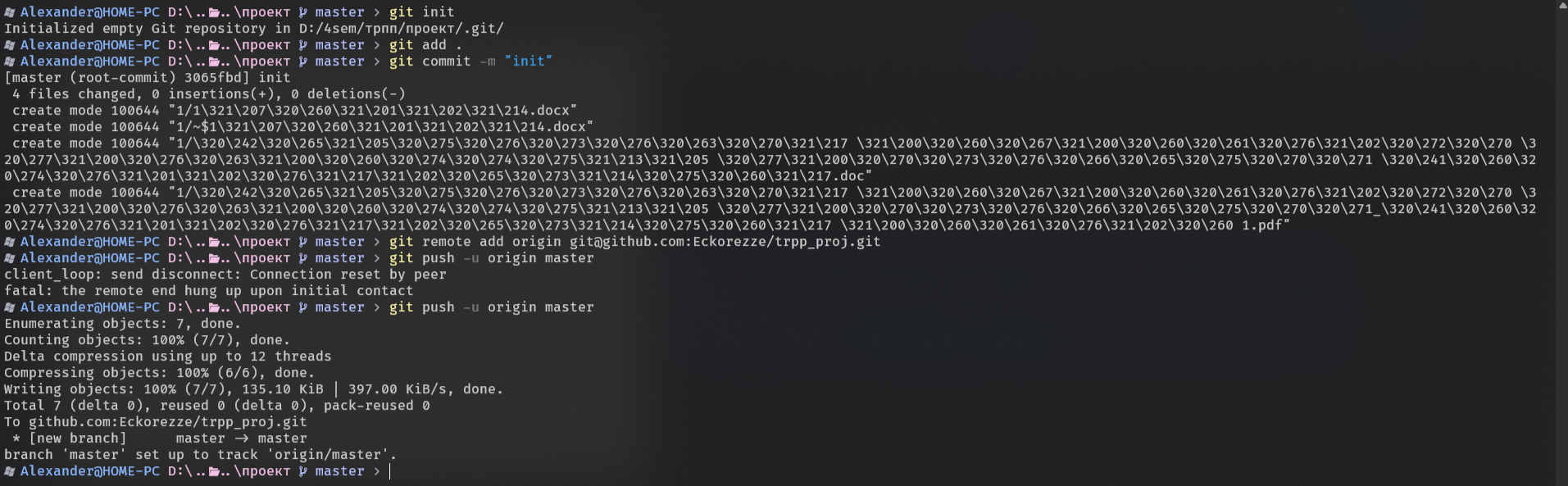


Рисунок 2 – создание репозитория для проекта

## Часть 4

Листинг 1 – файл /services/db.py

|  |
| --- |
| import psycopg2  import config  keepalive\_kwargs = {  "keepalives": 1,  "keepalives\_idle": 60,  "keepalives\_interval": 10,  "keepalives\_count": 5  }  class DataBase:  def \_\_init\_\_(self):  self.connect = psycopg2.connect(config.db\_auth, \*\*keepalive\_kwargs)  self.cursor = self.connect.cursor()  async def add\_users(self, user\_id, user\_name, user\_username):  with self.connect:  self.cursor.execute(  """INSERT INTO users (user\_id, user\_name, user\_username)  VALUES (%s, %s, %s) ON CONFLICT DO NOTHING""",  (user\_id, user\_name, user\_username)  )  async def delete\_user(self, user\_id):  with self.connect:  self.cursor.execute(  "DELETE FROM users WHERE user\_id = %s;",  (user\_id,))  async def user\_exist(self, user\_id):  with self.connect:  self.cursor.execute("""SELECT \* FROM users WHERE user\_id = %s""", (user\_id,))  return self.cursor.fetchall()  async def get\_language(self, user\_id):  with self.connect:  self.cursor.execute("SELECT DISTINCT language FROM users WHERE user\_id = %s", (user\_id,))  return self.cursor.fetchone()[0]  async def get\_notes(self, user\_id):  with self.connect:  self.cursor.execute('SELECT id, note FROM notes WHERE user\_id = %s', (user\_id,))  return self.cursor.fetchall()  async def get\_note(self, note\_id):  with self.connect:  self.cursor.execute('SELECT note FROM notes WHERE id = %s', (note\_id,))  return self.cursor.fetchone()  async def delete\_note(self, note\_id):  with self.connect:  self.cursor.execute('DELETE FROM notes WHERE id = %s', (note\_id,))  async def add\_note(self, user\_id, note\_text):  with self.connect:  self.cursor.execute('INSERT INTO notes (user\_id, note) VALUES (%s, %s)', (user\_id, note\_text))  async def set\_language(self, user\_id, language):  with self.connect:  self.cursor.execute("UPDATE users SET language=%s WHERE user\_id=%s", (language, user\_id))  async def user\_count(self):  with self.connect:  self.cursor.execute("SELECT COUNT(\*) FROM users")  return self.cursor.fetchone()[0]  async def status(self, user\_id):  with self.connect:  self.cursor.execute("SELECT DISTINCT status FROM users WHERE user\_id = %s", (user\_id,))  return self.cursor.fetchone()[0]  async def get\_admins(self):  with self.connect:  self.cursor.execute("SELECT DISTINCT user\_id FROM users WHERE status = 'admin'")  return self.cursor.fetchall()  async def all\_users(self):  with self.connect:  self.cursor.execute("SELECT user\_id FROM users")  return self.cursor.fetchall()  async def user\_update\_name(self, user\_id, user\_name, user\_username):  with self.connect:  self.cursor.execute("UPDATE users SET user\_username=%s, user\_name=%s WHERE user\_id=%s",  (user\_username, user\_name, user\_id))  async def get\_user\_info(self, user\_id):  with self.connect:  self.cursor.execute(  "SELECT user\_name, user\_username, status FROM users WHERE user\_id = %s", (user\_id,))  return self.cursor.fetchone()  async def get\_user\_info\_username(self, user\_username):  with self.connect:  self.cursor.execute(  "SELECT user\_name, user\_id, status FROM users WHERE user\_username = %s", (user\_username,))  return self.cursor.fetchone()  async def ban\_user(self, user\_id):  with self.connect:  self.cursor.execute("UPDATE users SET status=%s WHERE user\_id=%s", ("ban", user\_id))  async def get\_all\_users\_info(self):  with self.connect:  self.cursor.execute(  "SELECT user\_id, user\_name, user\_username, language, status FROM users")  return self.cursor.fetchall()  async def unban\_user(self, user\_id):  with self.connect:  self.cursor.execute("UPDATE users SET status=%s WHERE user\_id=%s", ("user", user\_id)) |

Листинг 2 – файл /messages/bot\_messages.py

|  |
| --- |
| from main import \_  def choose\_lan(language):  if language == "ru":  return """Вы выбрали русский язык!  Вы всегда можете изменить язык написав / language  Теперь снова нажмите / start  """  elif language == "en":  return """You have selected English!  You can always change the language by writing /language  Now click /start again  """  def send\_info():  return \_("""🤖This bot with various functions acts as a TRPP project, Here is a list of available functions:  ️/start - start interaction with the bot  QR code - generate a QR code  🔢Random number - generate a random number  🔐Generate password - generate a random password  🎯Tasks - create and schedule tasks  🌦Weather - view the weather  🎮Entertainments - the entertainment section  \*If you want to make an offer or ask a question, click the Feedback💬 button!\*""")  def admin\_panel(user\_count):  return \_("""Hello, this is the admin panel.  🪪Number of bot users: \*{user\_count}\*  Admin commands:  /get\\_users - download table with all users info""").format(user\_count=user\_count)  def return\_user\_info(user\_name, user\_id, user\_username, status):  return \_("""\*USER INFO\*  \_Name\_: \*{user\_name}\*  \_ID\_: \*{user\_id}\*  \_Username\_: \*{user\_username}\*  \_Status\_: \*{status}\*""").format(user\_name=user\_name, user\_id=user\_id, user\_username=user\_username, status=status)  def finish\_mailing():  return \_("Mailing is complete!")  def start\_mailing():  return \_("Starting mailing...")  def mailing\_message():  return \_('Enter the message to send:')  def not\_groups():  return \_("This command cannot be used in a group!")  def canceled():  return \_("Action canceled")  def ban\_message(reason):  return \_("🚫You have been banned, contact @mak5er for more information!\nReason: {reason}").format(reason=reason)  def unban\_message():  return \_("🎉You have been unbanned!")  def successful\_ban(banned\_user\_id):  return \_("User {banned\_user\_id} successfully banned!").format(banned\_user\_id=banned\_user\_id)  def successful\_unban(unbanned\_user\_id):  return \_("User {unbanned\_user\_id} successfully unbanned!").format(unbanned\_user\_id=unbanned\_user\_id)  def feedback\_message\_send(user, feedback\_message):  return \_("\*New message\* from user: \*{user}\*\n\*Message:\* `{feedback\_message}`").format(user=user,  feedback\_message=feedback\_message) |

# Вывод

В ходе работы над проектом были успешно выполнены следующие ключевые этапы:

* Формулировка функциональных требований: Разработаны и оформлены в виде User Story, что обеспечило основу для понимания и описания программного продукта.
* Визуализация требований: User Story были оформлены в виде наглядных карточек с макетом, что способствовало лучшему визуальному пониманию проекта и облегчило процесс планирования.
* Создание репозитория: Создан репозиторий для проекта с обеспечением удаленного доступа для всех членов команды, что легло в основу совместной и координированной работы.
* Начало разработки: Разработка проекта начата с акцентом на читаемость и документирование кода, а также на использование системы контроля версий для эффективного управления разработкой и изменениями в проекте.

Эти шаги позволили заложить прочный фундамент для дальнейшей разработки и успеха проекта.