Введение

В данном документе содержатся требования к реализуемому проекту: телеграм-бот "trip planner". Который позволит автоматизировать систему планирования поездок, напоминания о поездках, а так же хранить историю поездок.

1 Основные функции

- Автоматическая авторизация по Telegram ChatID и создание персонального профиля, что позволяет привязывать все данные к конкретному пользователю.
- Сохранение пользовательских настроек: список любимых локаций, режим уведомлений, способ отметки посещений (авто или вручную) и частота напоминаний.
- Планирование поездок: задание названия маршрута, дат начала и конца, а также добавление и удаление последовательных точек (города, достопримечательности).
- Ассистирование в путешествии: напоминания за сутки и в день старта, отметка чек-инов по геолокации или вручную и возможность добавлять подробные заметки о каждом месте.
- Ведение истории завершённых поездок с возможностью просмотра детальных данных (маршрут, даты, заметки) и выставления оценки пройденным маршрутам.
- Интерактивная справка по всем функциям и единый канал обратной связи для сообщений об ошибках и предложений по улучшению.
- Надёжная обработка ошибок: понятные уведомления о некорректном вводе, повторные попытки при сбоях внешних сервисов и информирование об ограничениях или недоступности функций.

2 Требования к реализации

Данный раздел описывает требования, предъявляемые к реализации проекта Telegram-бота **Trip Planner**.

2.1 Общие требования

2.1.1 Система контроля версиями

Ответственные: Команда разработчиков и преподаватель.

Действия:

- Код размещается в приватном репозитории GitHub.
- Репозиторий содержит инструкцию (`README.md`) для:
  - Локальной настройки окружения.
  - Сборки и запуска проекта.
  - Деплоя приложения с помощью Docker.
  - Прямого взаимодействия с ботом.
- Docker-образ публикуется на Docker Hub.

2.1.2 Целевые артефакты

Ответственные: Команда разработчиков.

Действия:

- Приложение формирует исполняемый **Fat JAR**.
- Приложение упаковывается и разворачивается через Docker.
- Используется Gradle (`build.gradle.kts`) для управления сборкой и зависимостями.

2.2 Функциональные требования

2.2.1 Аутентификация пользователей

Ответственные: Система (бот).

Действия:

- Пользователь идентифицируется автоматически по Telegram ChatID.
- ChatID записывается в базу данных автоматически при первом взаимодействии с ботом (команда: `/start`).

2.2.2 Настройки пользователя

Ответственные: Конечный пользователь.

Действия:

- Пользователь задаёт часто посещаемые места (команда: `/setfavorites Rome, Paris, Berlin`).
- Пользователь управляет уведомлениями о поездках (команды: `/notifications on`, `/notifications off`).

- Пользователь выбирает способ отметки посещения (автоматически по геолокации или вручную командой: `/setcheckin auto/manual`).
- Настраивает частоту напоминаний о поездках командой: `/remind 1d`, `/remind 10h`, `/remind 3h`.

2.2.3 Планирование поездки

Ответственные: Конечный пользователь.

Действия:

- Создание поездки (название, даты, точки маршрута):
  - `/newtrip Summer2025`
  - `/setdate 2025-07-10 2025-07-20`
  - `/adddestination Rome 2025-07-11`
- Просмотр запланированных поездок: `/viewtrips`
- Удаление поездок (с подтверждением): `/deletetrip 3`

2.2.4 Помощник в поездке

Ответственные: Система (бот), Конечный пользователь.

Действия:

- Система отправляет уведомления за день и в день начала поездки автоматически.
- Пользователь отмечает посещённые точки:
  - Автоматически (бот запрашивает геопозицию через Telegram)
  - Ручная отметка: `/checkin Colosseum`
- Пользователь добавляет заметки к посещённым точкам командой: `/addnote 2025-07-12 Отличный вид!`

2.2.5 История поездок

Ответственные: Конечный пользователь.

Действия:

- Просмотр всех завершённых поездок: `/history`
- Просмотр деталей конкретной поездки (маршрут, заметки, даты): `/tripdetails 3`
- Оценка поездок: `/rate 3 5` (0–5 баллов)

2.2.6 Уведомления о поездках

Ответственные: Конечный пользователь, Система (бот).

Действия:

- Пользователь устанавливает частоту уведомлений: `/remind 1d`, `/remind 3h`.
- Бот автоматически отправляет уведомления в соответствии с настройками.

2.3 Обращение с ошибкой

1. Пользователь пытается вызвать несуществующую команду.
2. Бот отправляет сообщение: «Неизвестная команда. Введите /help для списка доступных команд.»
3. При повторяющейся ошибке, бот предлагает помощь и ссылку на инструкцию.

2.4 Нефункциональные требования

2.4.1 Безопасность

Действия:

- Шифрование и безопасное хранение ChatID, геопозиции и данных о поездках.
- Данные расшифровываются исключительно для конечного пользователя, запрашивающего данные.
- API-токены и конфигурация базы данных хранятся в файле окружения (`.env`).

2.4.2 Производительность

Требования:

- Время ответа системы ≤ 300 мс (не учитывая задержек внешних API).
- Поддержка одновременного использования до 100 000 пользователей.

2.4.3 Надёжность и поддержка

Действия:

- При ошибке обращения к внешнему API, система делает две повторные попытки запроса с интервалом в 5 секунд.
- Пользователь может отправить сообщение об ошибках или о пожеланиях командой: `/feedback Ваше сообщение`.

2.4.4 Роли пользователей

Действия:

- **Конечный пользователь** (идентификация по ChatID):
  - Создание, просмотр и оценка поездок, добавление заметок.
  - Отправка отзывов и сообщений.
- **Администратор** (идентификация по API-ключу):
  - Управление пользователями и поездками, мониторинг системы, приём жалоб и пожеланий.

3 Требования к технологическому стеку

- **Язык программирования:** Java SE 23.
- **Фреймворк:** Spring 6.2 (WebFlux, JPA, Modulith).
- **Telegram API:** Приложение должно интегрироваться с Telegram API для предоставления функциональности бота.
- **База данных:** MongoDB.
- **Брокер сообщений:** Kafka.
- **Контейнеризация:** Docker, Docker Compose.
- **Логирование:** SLF4J, Log4j.
- **Тестирование:** JUnit, Mockito.

4 Требования к документации

Должна быть написана документация для API с использованием Spring Docs.

4.1 Документ «Требования к проекту»

Предоставить документ с требованиями, который включает:

- Цель и функциональность приложения.
- Ключевые функции Telegram-бота.
- Нефункциональные требования.
- Любые предположения или ограничения.

4.2 Документ «Архитектура проекта»

Предоставить документ по архитектуре, который включает:

- Компонентные диаграммы, схема базы данных.
- Описание того, как приложение будет строиться, развертываться и запускаться.

5 Требования к тестированию

- Должны быть реализованы unit-тесты и интеграционные тесты для всех модулей.
- Обеспечить покрытие кода тестами не менее 60%.

6 Голоссарий

6.1 Термины предметной области

- **Поездка** — любое запланированное путешествие пользователя от точки «А» до точки «Б» с указанием времени, участников и прочих условий.
- **Маршрут** — конкретная дорога или путь внутри поездки, разбитый на отдельные участки и контрольные точки.
- **Контрольная точка (Waypoint)** — отдельная позиция на маршруте (город, достопримечательность и т. д.), через которую проходит поездка.
- **Транспорт** — средство передвижения в поездке (автомобиль, поезд, самолёт, автобус, пешком и т. д.).
- **Проживание (Accommodation)** — место ночёвки в ходе поездки: отель, хостел, апартаменты и т. п.
- **Бюджет** — оценка расходов на поездку: сумма, выделенная на транспорт, питание, проживание и развлечения.
- **Участники** — люди, которые едут вместе в рамках одной поездки.
- **Статус поездки** — текущее состояние плана: «в процессе планирования», «подтверждена», «завершена», «отменена».
- **Location (Место)** — географическая точка с координатами и описанием (город, адрес, достопримечательность).
- **Геопозиция** — текущее или заданное местоположение на земной поверхности, обычно выраженное в широте и долготе.
- **Бот** — программа, которая автоматически выполняет определённые задачи по заданным правилам или алгоритмам.
- **ТГ-бот (телеграм-бот)** — бот, интегрированный с мессенджером Telegram: отвечает на сообщения пользователей, выполняет команды и взаимодействует через чат в Telegram.

6.2 Основные термины

Telegram Многофункциональный мессенджер, который позволяет пользователям обмениваться сообщениями.

Telegram-бот Программа в мессенджере Telegram, автоматически обрабатывающая команды пользователей.

ChatID Уникальный идентификатор пользователя/чата в Telegram, используемый для аутентификации.

6.3 Функциональные модули

Аутентификатор Модуль для аутентификации

6.4 Технические термины

Fat JAR Исполняемый JAR-файл со всеми зависимостями.

API Набор способов и правил, по которым различные программы общаются между собой и обмениваются данными.

API-ключ Секретный уникальный идентификатор, используемый для аутентификации и авторизации пользователя, разработчика или вызывающей программы в API.

Telegram API Набор инструментов, который позволяет разработчикам программно взаимодействовать с платформой Telegram.

Валидация Процесс проверки и подтверждения того, что продукт, система или процесс будут функционировать должным образом и удовлетворять ожидания пользователей.

Эндпоинт Конечная точка веб-сервиса, к которой клиентское приложение обращается для выполнения определённых операций или получения данных.

HTTP Протокол передачи гипертекста. Это набор правил, по которым данные в интернете передаются между разными источниками, обычно между компьютерами и серверами.

БД Набор структурированных данных, предназначенный для хранения, обработки и изменения большого количества информации.

MongoDB документоориентированная система управления базами данных.

Брокер сообщений архитектурный шаблон в распределённых системах; приложение, которое преобразует сообщение по одному протоколу от приложения-источника в сообщение протокола приложения-приёмника, тем самым выступая между ними посредником.

Kafka распределённый программный брокер сообщений с открытым исходным кодом, разрабатываемый в рамках фонда Apache на языках Java и Scala.

Логирование Процесс записи и хранения информации о событиях, действиях и состояниях системы, приложений или пользователей.

Фреймворк Набор инструментов, библиотек и правил, который помогает разработчику быстро создать продукт: сайт, приложение.

6.5 Процесс разработки

Gradle Система сборки проекта.

JUnit/Mockito Фреймворки для модульного тестирования.

GitHub Это веб-сервис для хостинга IT-проектов и их совместной разработки.

Контейнеризация Метод, с помощью которого программный код упаковывается в единый исполняемый файл вместе с библиотеками и зависимостями, чтобы обеспечить его корректный запуск.

Docker Платформа для контейнеризации приложения.

Docker-образ Шаблон (физически — исполняемый пакет), из которого создаются Docker-контейнеры.

Docker-контейнер Стандартизированный, изолированный и портативный пакет программного обеспечения, который включает в себя всё необходимое для запуска приложения.

Docker Hub Облачная платформа для публикации, хранения и распространения Docker-образов.

6.6 Форматы данных

CSV Формат экспорта истории операций (значения, разделенные запятыми).

JSON Формат обмена данными с внешними API.

DATE Дата в формате `ГГГГ-ММ-ДД`.

6.7 Тестирование

Тестирование Проверка на соответствие заявленной спецификации.

Unit-тесты Проверка отдельных частей кода на корректность работы. В программировании под словом «юнит» чаще понимают функцию, метод или класс в исходном коде.

Интеграционные тесты Проверка отдельных модулей или компонентов приложения на совместимость друг с другом.

6.8 Команды бота

Команда	Формат	Пример	Описание
/start	/start	/start	Начало работы с ботом. Приветствие и вывод основных опций.
/help	/help	/help	Выводит список доступных команд и инструкции по использованию.
/newtrip	/newtrip <название_поездки>	/newtrip Paris2024	Создает новую поездку с указанным названием.
/viewtrips	/viewtrips	/viewtrips	Отображает список всех поездок пользователя с их ID и названиями.
/adddestination	/adddestination <место> <дата>	/adddestination EiffelTower 2024-07-20	Добавляет пункт назначения в текущую поездку. Дата в формате DATE.
/deletetrip	/deletetrip <ID_поездки>	/deletetrip 5	Удаляет поездку по указанному ID (ID можно получить через /viewtrips).
/currenttrip	/currenttrip	/currenttrip	Показывает детали текущей активной поездки (название, даты, пункты).
/setdate	/setdate <начало> <конец>	/setdate 2024-07-15 2024-07-25	Устанавливает даты начала и окончания для текущей поездки.
/weather	/weather <место>	/weather Paris	Показывает прогноз погоды для указанного места (интеграция с OpenWeather).

7 Исключительные ситуации

- Повторить запрос 2 раза с интервалом в 5 секунд.
- При неудаче — отправить сообщение пользователю о временной недоступности функции.
- Ошибка базы данных:
  - Сохранить информацию об ошибке в логах.
  - Сообщить пользователю о технической ошибке.
- Некорректный ввод команды:
  - Отправить сообщение с корректным форматом команды.
  - Предложить помощь с использованием команды `/help`.
- Удаление незавершённой поездки:
  - Запросить подтверждение у пользователя.
  - При подтверждении — удалить все данные, связанные с поездкой.
- Попытка доступа к чужим данным (по ошибке или намеренно):
  - Заблокировать действие.
  - Сообщить пользователю, что доступ ограничен.
  - Зафиксировать инцидент в логах для анализа.

8 Пользовательские сценарии

8.1 Создание и сопровождение поездки

1. Пользователь вводит команду `/newtrip Paris2025`
2. Бот запрашивает даты поездки → пользователь отвечает `/setdate 2025-07-10 2025-07-17`
3. Бот предлагает добавить пункты → пользователь использует `/adddestination EiffelTower 2025-07-11`
4. По завершении планирования, бот добавляет поездку в активные и отображает информацию.
5. В день поездки бот отправляет уведомление и предлагает сопровождение.
6. Во время маршрута пользователь может отметить посещённые точки, добавить заметки.
7. По завершении поездки бот переносит маршрут в историю и предлагает оценить.

9 Краткая инструкция пользователя

- Для начала работы с ботом используйте команду `/start`
- Все команды начинаются со знака `/`, например, `/newtrip`
- Чтобы запланировать поездку:
  - Введите `/newtrip <название>`, затем `/setdate <начало> <конец>`
  - Добавьте пункты маршрута: `/adddestination <место> <дата>`
- Для просмотра запланированных поездок: `/viewtrips`
- Для удаления поездки: `/deletetrip <ID>`
- Для просмотра текущей поездки: `/currenttrip`
- Для запроса погоды: `/weather <город>`
- По всем вопросам и ошибкам — используйте кнопку «Обратная связь» или напишите через команду `/help`