Содержание

1 Описания **2**
1.1 Телеграм-бот . . . 2
1.2 User . . . 2
1.3 Admin . . . 2
1.4 MessageParser . . . 2
1.5 TripHelperWorker . . . 2

2 Схемы **3**
2.1 Сервер . . . 3
2.2 MongoDB . . . 4
2.3 HealthCheckModule . . . 5
2.4 AdminModule . . . 6
2.5 PlannedTripsModule . . . 8
2.6 TripHelperModule . . . 10
2.7 TripHistoryModule . . . 12

3 Описание развертывания приложения **14**
3.1 Подготовка окружения . . . 14
3.2 Конфигурация инфраструктуры . . . 14
3.3 Развертывание бота . . . 14

4 Сборка приложения **14**
4.1 Установка зависимостей . . . 14
4.2 Сборка Fat JAR . . . 14
4.3 Тестирование . . . 14
4.4 Создание Docker-образа . . . 14

5 Деплой приложения **14**
5.1 Загрузка образа в Docker Hub . . . 14
5.2 Развертывание на сервере . . . 15
5.3 Проверка работоспособности . . . 15

6 Запуск приложения **15**
6.1 Локальный запуск (для разработки) . . . 15
6.2 Продуктивный режим . . . 15
6.3 Команды для управления . . . 15

1 Описания

1.1 Телеграм-бот

- Компонент: интерфейс для взаимодействия с пользователем через Telegram.
  - Передача команд и запросов от пользователя на сервер.
  - Отображает ответы сервера в чате бота.

1.2 User

- Роль:
  - Telegram пользователь.
- Функции:
  - Инициализирует запросы через Telegram бота, такие как: запланировать поездку, добавить заметку к путевой точке и другие.
  - Делится геолокацией.
  - Получает сообщения от Telegram бота.
  - Имеет доступ к эндпоинту /healthcheck без авторизации.

1.3 Admin

- Роли:
  - администратор системы.
- Функции:
  - Имеет доступ к эндпоинту /users с авторизацией по API ключу.
  - Имеет доступ к эндпоинту /healthcheck без авторизации.

1.4 MessageParser

- Компонент:
  - обработка сообщений от Telegram бота.
- Функции:
  - Обрабатывает сообщения от Telegram бота.
  - Направляет команды в соответствующий модуль.

1.5 TripHelperWorker

- Компонент:
  - модуль помощник с поездкой.
- Функции:
  - Обрабатывает команду /addnote, добавление заметки к путевой точке одной из поездкой.
  - Обрабатывает команду /markpoint, мануальная отметка путевой точки текущей поездки как посещенной.

2 Схемы

2.1 Сервер

Основной компонент.

- Функции:
  - Обрабатывает все веб запросы и запросы от телеграмм бота.
  - Обеспечение взаимодействия между всеми компонентами.

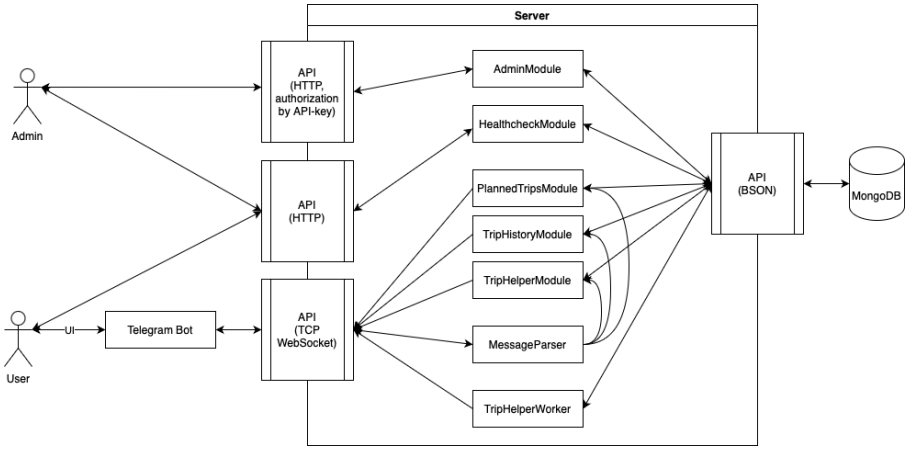


Рис. 1: Схема сервера

2.2 MongoDB

- База данных:
  - документоориентированная база данных.
- Функции:
  - Хранит информацию о администраторах.
  - Хранит информацию о пользователях и их поездках.

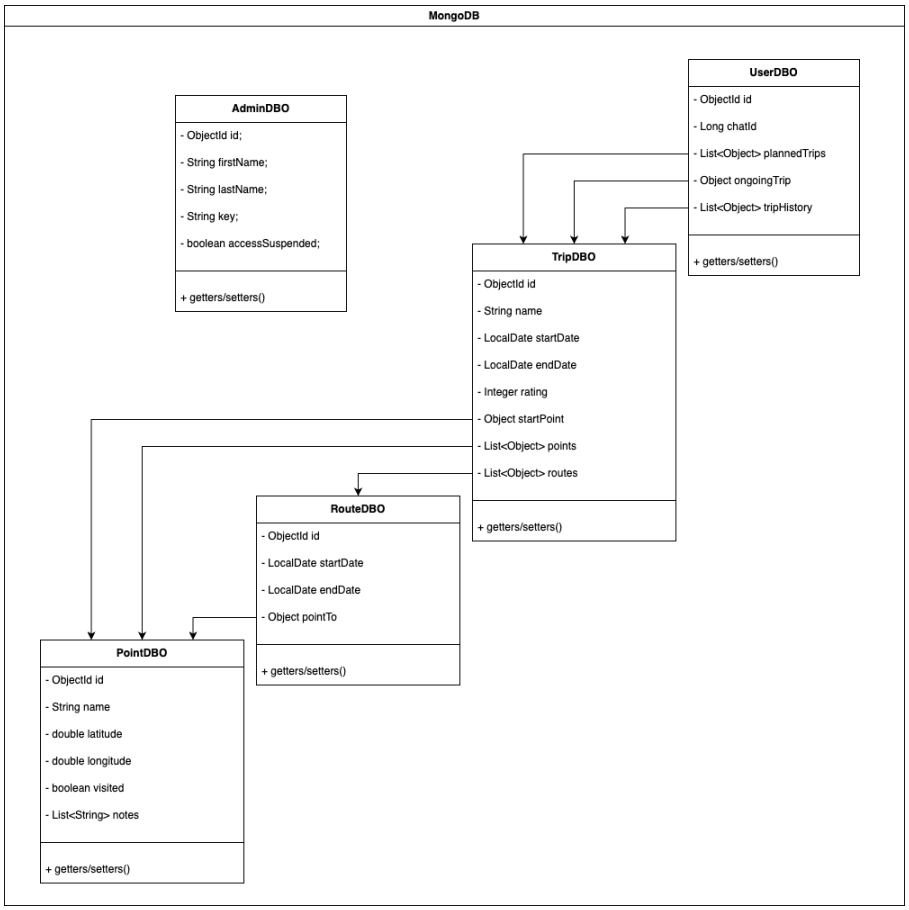


Рис. 2: Схема MongoDB (NoSQL)

2.3 HealthCheckModule

- Компонент:
  - модуль проверки состояния сервера.
- Функции:
  - Реализует эндпоинт /healthcheck для проверки работоспособности сервера и получения списка авторов.
  - Доступен без авторизации.

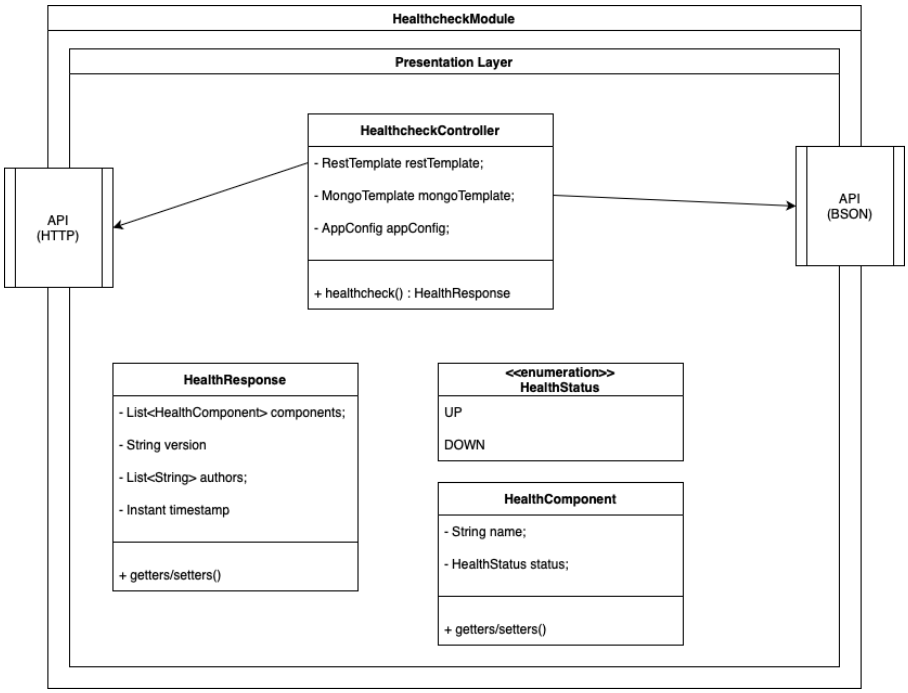
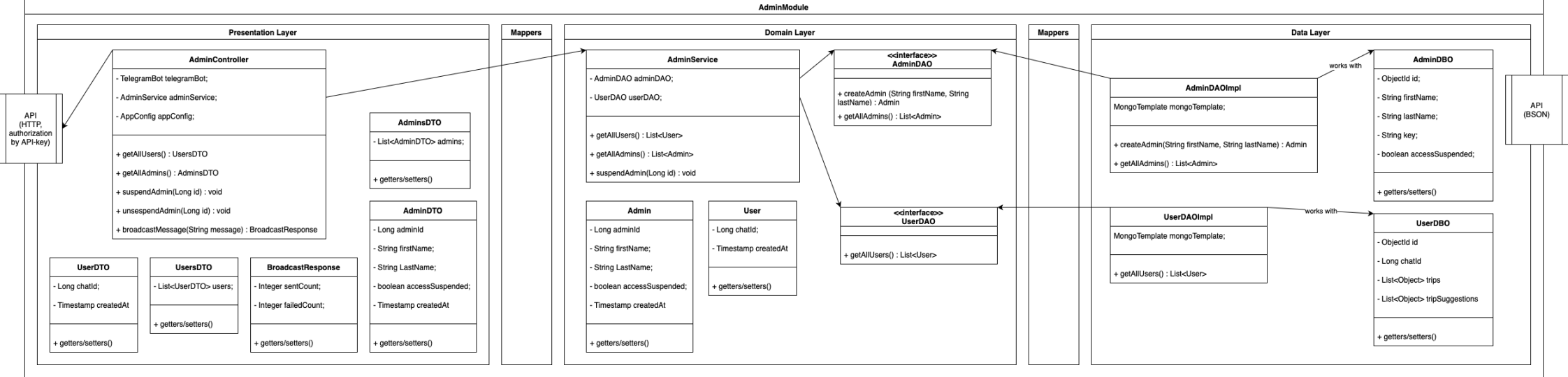


Рис. 3: Схема HealthChecker

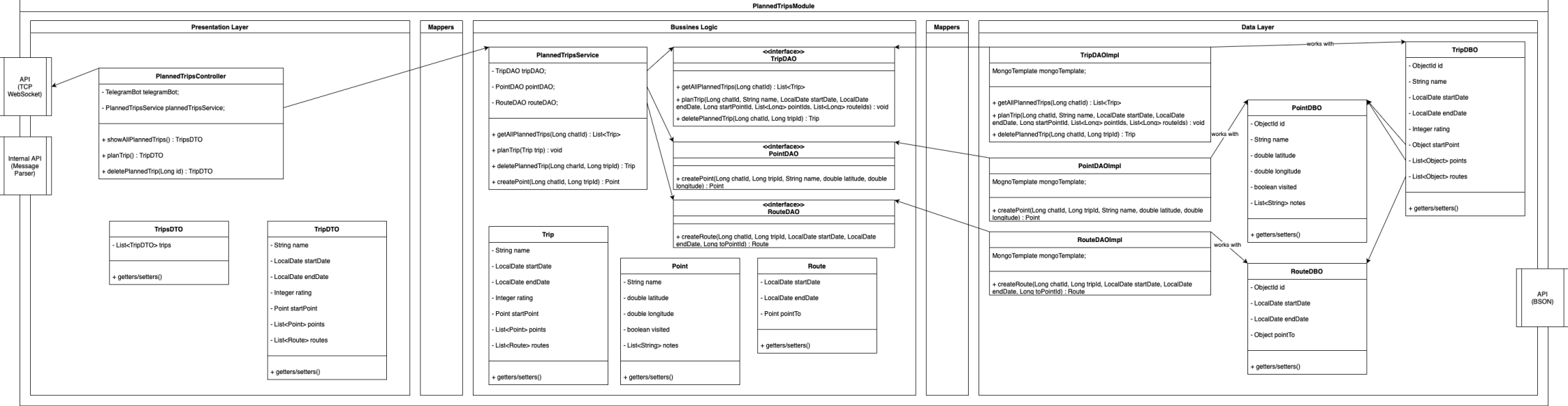
2.4 AdminModule

- Компонент:
  - модуль администрации системы.
- Функции:
  - Реализует эндпоинт /users, отображающий данные о всех пользователях.
  - Доступен администраторам с авторизацией.


AdminModule
Presentation Layer
Mappers
Domain Layer
Mappers
Data Layer
API (HTTP, authorization by API-key)
AdminController
- TelegramBot telegramBot;
- AdminService adminService;
- AppConfig appConfig;
+ getAllUsers() : UsersDTO
+ getAllAdmins() : AdminsDTO
+ suspendAdmin(Long id) : void
+ unsespendAdmin(Long id) : void
+ broadcastMessage(String message) : BroadcastResponse
AdminsDTO
- List<AdminDTO> admins;
+ getters/setters()
AdminDTO
- Long adminId
- String firstName;
- String LastName;
- boolean accessSuspended;
- Timestamp createdAt
+ getters/setters()
UserDTO
- Long chatId;
- Timestamp createdAt
+ getters/setters()
UsersDTO
- List<UserDTO> users;
+ getters/setters()
BroadcastResponse
- Integer sentCount;
- Integer failedCount;
+ getters/setters()
AdminService
- AdminDAO adminDAO;
- UserDAO userDAO;
+ getAllUsers() : List<User>
+ getAllAdmins() : List<Admin>
+ suspendAdmin(Long id) : void
Admin
- Long adminId
- String firstName;
- String LastName;
- boolean accessSuspended;
- Timestamp createdAt
+ getters/setters()
User
- Long chatId;
- Timestamp createdAt
+ getters/setters()
<<interface>>
AdminDAO
+ createAdmin (String firstName, String lastName) : Admin
+ getAllAdmins() : List<Admin>
<<interface>>
UserDAO
+ getAllUsers() : List<User>
AdminDAOImpl
MongoTemplate mongoTemplate;
+ createAdmin(String firstName, String lastName) : Admin
+ getAllAdmins() : List<Admin>
works with
AdminDBO
- ObjectId id;
- String firstName;
- String lastName;
- String key;
- boolean accessSuspended;
+ getters/setters()
UserDAOImpl
MongoTemplate mongoTemplate;
+ getAllUsers() : List<User>
works with
UserDBO
- ObjectId id
- Long chatId
- List<Object> trips
- List<Object> tripSuggestions
+ getters/setters()
API (BSON)

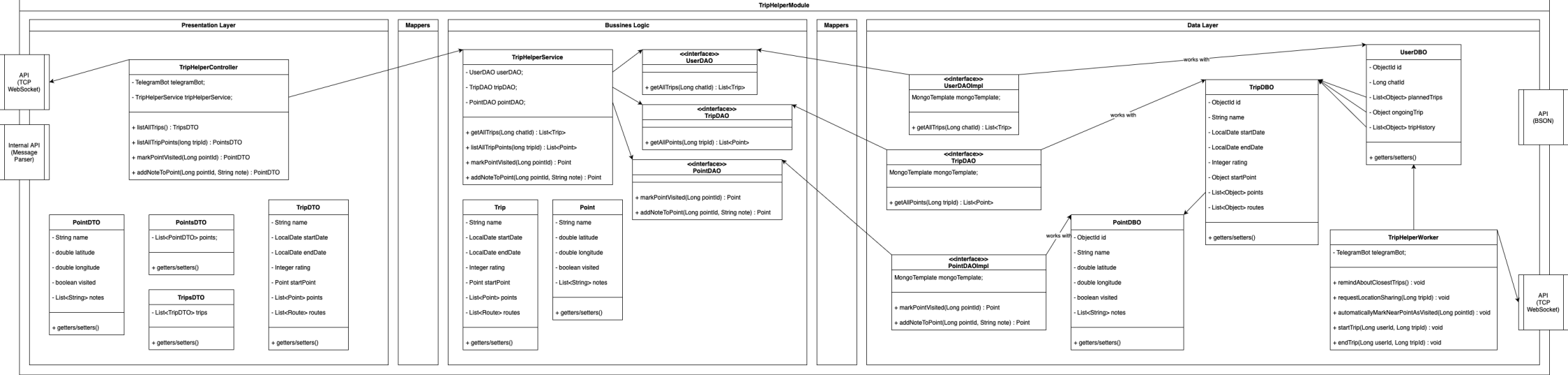
2.5 PlannedTripsModule

- Компонент:
  - модуль планирования поездок.
- Функции:
  - Обрабатывает команду /plantrip, запланирование новой поездки.
  - Обрабатывает команду /showplanned, получение списка запланированных поездок.
  - Обрабатывает команду /deleteplanned, удаление запланированной поездки.


PlannedTripsModule
Presentation Layer
Mappers
Bussines Logic
Mappers
Data Layer
API (TCP WebSocket)
Internal API (Message Parser)
PlannedTripsController
- TelegramBot telegramBot;
- PlannedTripsService plannedTripsService;
+ showAllPlannedTrips() : TripsDTO
+ planTrip() : TripDTO
+ deletePlannedTrip(Long id) : TripDTO
TripsDTO
- List<TripDTO> trips
+ getters/setters()
TripDTO
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Point startPoint
- List<Point> points
- List<Route> routes
+ getters/setters()
PlannedTripsService
- TripDAO tripDAO;
- PointDAO pointDAO;
- RouteDAO routeDAO;
+ getAllPlannedTrips(Long chatId) : List<Trip>
+ planTrip(Trip trip) : void
+ deletePlannedTrip(Long charId, Long tripId) : Trip
+ createPoint(Long chatId, Long tripId) : Point
<<interface>>
TripDAO
+ getAllPlannedTrips(Long chatId) : List<Trip>
+ planTrip(Long chatId, String name, LocalDate startDate, LocalDate endDate, Long startPointId, List<Long> pointIds, List<Long> routeIds) : void
+ deletePlannedTrip(Long chatId, Long tripId) : Trip
<<interface>>
PointDAO
+ createPoint(Long chatId, Long tripId, String name, double latitude, double longitude) : Point
<<interface>>
RouteDAO
+ createRoute(Long chatId, Long tripId, LocalDate startDate, LocalDate endDate, Long toPointId) : Route
Trip
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Point startPoint
- List<Point> points
- List<Route> routes
+ getters/setters()
Point
- String name
- double latitude
- double longitude
- boolean visited
- List<String> notes
+ getters/setters()
Route
- LocalDate startDate
- LocalDate endDate
- Point pointTo
+ getters/setters()
TripDAOImpl
MongoTemplate mongoTemplate;
+ getAllPlannedTrips(Long chatId) : List<Trip>
+ planTrip(Long chatId, String name, LocalDate startDate, LocalDate endDate, Long startPointId, List<Long> pointIds, List<Long> routeIds) : void
+ deletePlannedTrip(Long chatId, Long tripId) : Trip
works with
PointDAOImpl
MognoTemplate mongoTemplate;
+ createPoint(Long chatId, Long tripId, String name, double latitude, double longitude) : Point
works with
RouteDAOImpl
MongoTemplate mongoTemplate;
+ createRoute(Long chatId, Long tripId, LocalDate startDate, LocalDate endDate, Long toPointId) : Route
works with
TripDBO
- ObjectId id
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Object startPoint
- List<Object> points
- List<Object> routes
+ getters/setters()
PointDBO
- ObjectId id
- String name
- double latitude
- double longitude
- boolean visited
- List<String> notes
+ getters/setters()
RouteDBO
- ObjectId id
- LocalDate startDate
- LocalDate endDate
- Object pointTo
+ getters/setters()
API (BSON)

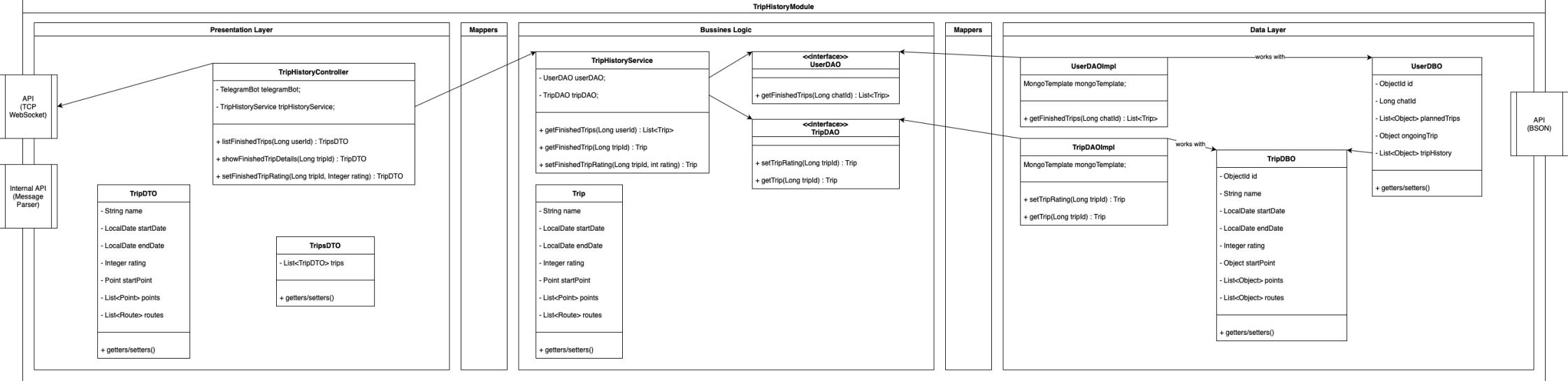
2.6 TripHelperModule

- Компонент:
  - модуль помощник с поездкой.
- Функции:
  - Обрабатывает команду /addnote, добавление заметки к путевой точке одной из поездкой.
  - Обрабатывает команду /markpoint, мануальная отметка путевой точки текущей поездки как посещенной.


TripHelperModule
Presentation Layer
Mappers
Bussines Logic
Mappers
Data Layer
API (TCP WebSocket)
Internal API (Message Parser)
TripHelperController
- TelegramBot telegramBot;
- TripHelperService tripHelperService;
+ listAllTrips() : TripsDTO
+ listAllTripPoints(long tripId) : PointsDTO
+ markPointVisited(Long pointId) : PointDTO
+ addNoteToPoint(Long pointId, String note) : PointDTO
TripHelperService
- UserDAO userDAO;
- TripDAO tripDAO;
- PointDAO pointDAO;
+ getAllTrips(Long chatId) : List<Trip>
+ listAllTripPoints(long tripId) : List<Point>
+ markPointVisited(Long pointId) : Point
+ addNoteToPoint(Long pointId, String note) : Point
<<interface>> UserDAO
+ getAllTrips(Long chatId) : List<Trip>
<<interface>> TripDAO
+ getAllPoints(Long tripId) : List<Point>
<<interface>> PointDAO
+ markPointVisited(Long pointId) : Point
+ addNoteToPoint(Long pointId, String note) : Point
<<interface>> UserDAOImpl
MongoTemplate mongoTemplate;
+ getAllTrips(Long chatId) : List<Trip>
works with
UserDBO
- ObjectId id
- Long chatId
- List<Object> plannedTrips
- Object ongoingTrip
- List<Object> tripHistory
+ getters/setters()
<<interface>> TripDAO
MongoTemplate mongoTemplate;
+ getAllPoints(Long tripId) : List<Point>
works with
TripDBO
- ObjectId id
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Object startPoint
- List<Object> points
- List<Object> routes
+ getters/setters()
<<interface>> PointDAOImpl
MongoTemplate mongoTemplate;
+ markPointVisited(Long pointId) : Point
+ addNoteToPoint(Long pointId, String note) : Point
works with
PointDBO
- ObjectId id
- String name
- double latitude
- double longitude
- boolean visited
- List<String> notes
+ getters/setters()
TripHelperWorker
- TelegramBot telegramBot;
+ remindAboutClosestTrips() : void
+ requestLocationSharing(Long tripId) : void
+ automaticallyMarkNearPointAsVisited(Long pointId) : void
+ startTrip(Long userId, Long tripId) : void
+ endTrip(Long userId, Long tripId) : void
API (BSON)
API (TCP WebSocket)
PointDTO
- String name
- double latitude
- double longitude
- boolean visited
- List<String> notes
+ getters/setters()
PointsDTO
- List<PointDTO> points;
+ getters/setters()
TripsDTO
- List<TripDTO> trips
+ getters/setters()
TripDTO
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Point startPoint
- List<Point> points
- List<Route> routes
+ getters/setters()
Trip
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Point startPoint
- List<Point> points
- List<Route> routes
+ getters/setters()
Point
- String name
- double latitude
- double longitude
- boolean visited
- List<String> notes
+ getters/setters()

2.7 TripHistoryModule

- Компонент:
  - модуль истории поездок.
- Функции:
  - Обрабатывает команду /triphistory, предоставляя список всех поездок с возможностью получения подробностей об отдельной поездке или оценки отдельной поездки.


TripHistoryModule
Presentation Layer
Mappers
Bussines Logic
Mappers
Data Layer
API (TCP WebSocket)
Internal API (Message Parser)
TripHistoryController
- TelegramBot telegramBot;
- TripHistoryService tripHistoryService;
+ listFinishedTrips(Long userId) : TripsDTO
+ showFinishedTripDetails(Long tripId) : TripDTO
+ setFinishedTripRating(Long tripId, Integer rating) : TripDTO
TripDTO
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Point startPoint
- List<Point> points
- List<Route> routes
+ getters/setters()
TripsDTO
- List<TripDTO> trips
+ getters/setters()
TripHistoryService
- UserDAO userDAO;
- TripDAO tripDAO;
+ getFinishedTrips(Long userId) : List<Trip>
+ getFinishedTrip(Long tripId) : Trip
+ setFinishedTripRating(Long tripId, int rating) : Trip
Trip
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Point startPoint
- List<Point> points
- List<Route> routes
+ getters/setters()
<<interface>> UserDAO
+ getFinishedTrips(Long chatId) : List<Trip>
<<interface>> TripDAO
+ setTripRating(Long tripId) : Trip
+ getTrip(Long tripId) : Trip
UserDAOImpl
MongoTemplate mongoTemplate;
+ getFinishedTrips(Long chatId) : List<Trip>
works with
TripDAOImpl
MongoTemplate mongoTemplate;
+ setTripRating(Long tripId) : Trip
+ getTrip(Long tripId) : Trip
works with
UserDBO
- ObjectId id
- Long chatId
- List<Object> plannedTrips
- Object ongoingTrip
- List<Object> tripHistory
+ getters/setters()
TripDBO
- ObjectId id
- String name
- LocalDate startDate
- LocalDate endDate
- Integer rating
- Object startPoint
- List<Object> points
- List<Object> routes
+ getters/setters()
API (BSON)

3 Описание развертывания приложения

3.1 Подготовка окружения

- Установка Docker и Docker Compose на сервер.
- Настройка переменных окружения (‘.env‘) для подключения к Telegram API, MongoDB и Kafka.

3.2 Конфигурация инфраструктуры

- Запуск MongoDB и Kafka через Docker Compose.
- Настройка репликации и шардинга для MongoDB (если требуется).
- Проверка доступности брокера сообщений Kafka.

3.3 Развертывание бота

- Сборка Docker-образа приложения (‘docker build‘).
- Публикация образа в Docker Hub.
- Запуск контейнера с ботом на сервере (‘docker-compose up‘).

4 Сборка приложения

4.1 Установка зависимостей

- Запуск ‘gradle build‘ для загрузки зависимостей (Spring Boot, Telegram API, MongoDB Driver и т.д.).
- Проверка совместимости версий Java (SE 23) и Spring (6.2).

4.2 Сборка Fat JAR

- Генерация исполняемого JAR-файла с помощью Gradle (‘./gradlew bootJar‘).
- Проверка наличия всех зависимостей в ‘build/libs/‘.

4.3 Тестирование

- Запуск unit-тестов (‘./gradlew test‘).
- Проверка покрытия кода (минимум 60%).

4.4 Создание Docker-образа

- Написание ‘Dockerfile‘ с базой на ‘openjdk:23‘.
- Копирование JAR-файла и запуск приложения в контейнере.

5 Деплой приложения

5.1 Загрузка образа в Docker Hub

- Авторизация (‘docker login‘).

- Пуш образа (‘docker push username/trip-planner-bot:latest‘).

5.2 Развертывание на сервере

- Запуск MongoDB и Kafka через ‘docker-compose.yml‘.
- Развертывание бота в отдельном контейнере.

5.3 Проверка работоспособности

- Тестирование команд бота через Telegram.
- Мониторинг логов на предмет ошибок (‘docker logs <container_id>‘).

6 Запуск приложения

6.1 Локальный запуск (для разработки)

- Запуск MongoDB и Kafka через Docker Compose.
- Запуск бота через ‘./gradlew bootRun‘.

6.2 Продуктивный режим

- Запуск всех сервисов через ‘docker-compose up -d‘.
- Настройка авто-рестарта при падении (‘restart: always‘).

6.3 Команды для управления

- Остановка: ‘docker-compose down‘.
- Обновление: ‘docker-compose pull && docker-compose up -d‘.