Task361. Интеграция поддержки Security в REST API

Дано:

- Разрабатываемая система обрабатывает сущности **User**, **Tweet**, **Label** и **Message**, которые логически связаны отношениями (см. предыдущие задачи)
  - один-ко-многим (**User** и **Tweet**, **Tweet** и **Message**)
  - многие-ко-многим (**Tweet**, **Label**).

Цель: Разработать и интегрировать систему аутентификации и авторизации в существующий REST API.

Описание:

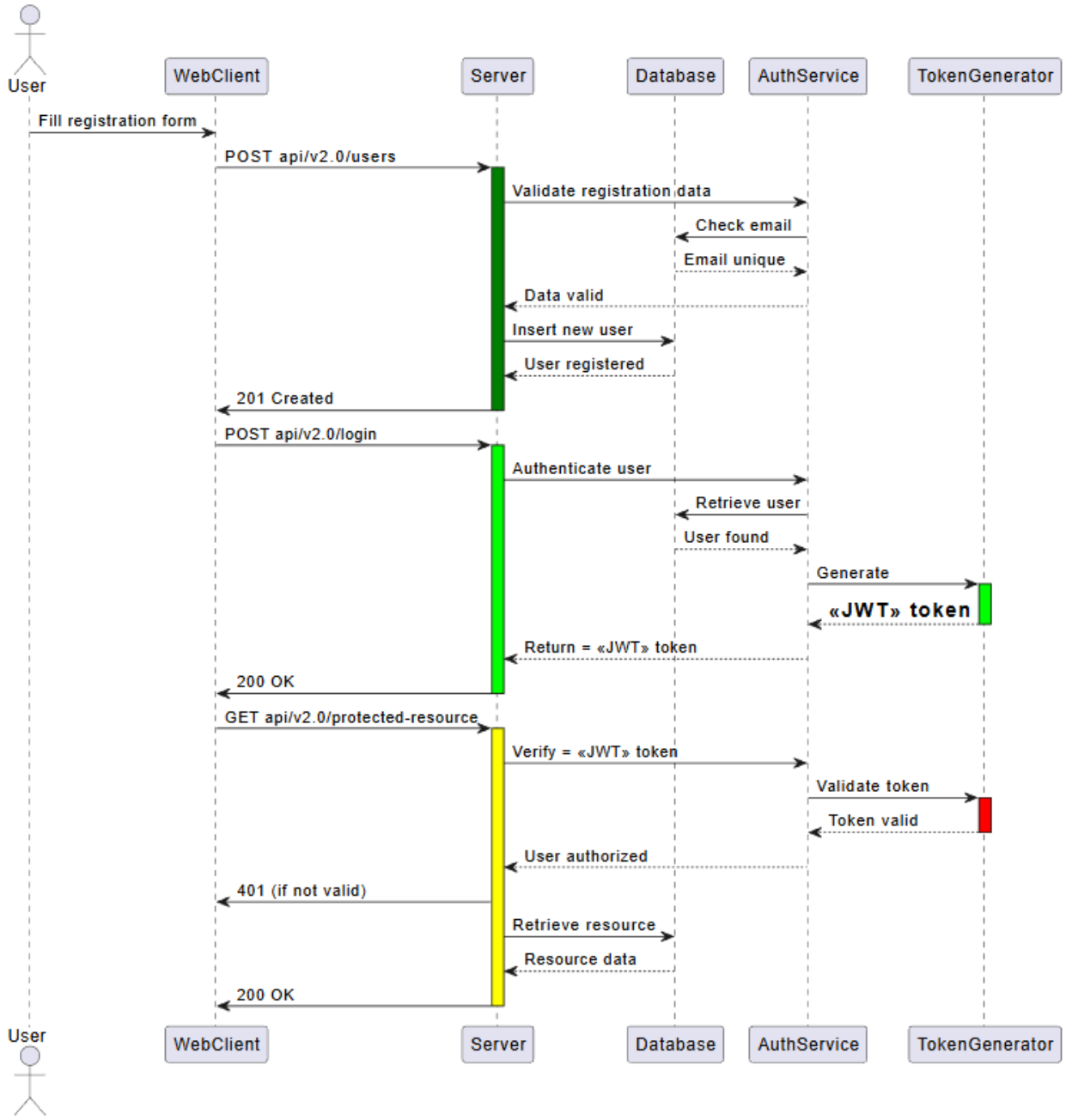
У вас есть готовый REST API, который сейчас работает без аутентификации и авторизации. Ваша задача - добавить поддержку Security для защиты API с другой версией в endpoint **/api/v2.0**

Порядок выполнения:

1. Добавить необходимые зависимости в свой проект.
2. Настроить ВРЕМЕННО базовую аутентификацию с использованием firstname/password из **User**.
3. Реализовать защиту REST API.
4. Добавить функционал ролей пользователей (admin, customer).
5. Реализовать механизм аутентификации через JWT (JSON Web Tokens).
6. Реализовать тестирование защиты.
7. Теперь можно отключить базовую аутентификацию.

Требования:

Схема регистрации и использования защищенных ресурсов:



1. Использовать Spring Security или аналогичное решение для реализации аутентификации и авторизации.
2. Использовать контроллер для регистрации новых пользователей.
3. Защитить все endpoints, требующие аутентификации.
4. Реализовать методы для получения текущего пользователя и его роли.
5. Добавить обработку ошибок аутентификации и авторизации.
6. Реализовать генерацию и валидацию JWT токенов.

Технические требования

1. **Префикс и порт:**
   - Все **незащищенные** endpoints должны использовать префикс `/api/v1.0/` и продолжать работу без мер безопасности.
   - Все **защищенные** endpoints должны использовать префикс `/api/v2.0/` и дублировать работу но с безопасностью.
   - Приложение должно работать на порту `24110`.
2. **Модель данных:**
   - Идентификаторы (`id`) всех сущностей должны быть типа `bigint`.
   - Типы полей должны соответствовать Task310 (текст, дата/время ISO_8601).
   - Поле `login` в **User** должно использоваться как уникальный идентификатор для аутентификации.
3. **Аутентификация и авторизация:**
   - Аутентификация должна использовать `login` и `password` из сущности **User**. В базе данных пароль должен храниться в закодированном виде BCrypt.
   - Роли пользователей: `ADMIN`, `CUSTOMER`.
   - Защита эндпоинтов должна соответствовать ролям
     - `ADMIN` - имеет доступ ко всем операциям
     - `CUSTOMER` - к ограниченным: полный доступ только на свои данные - профиль, контент и комментарии, все иное - только чтение
4. **JWT:**
   - JWT токен должен содержать поля: `sub` (login пользователя), `iat` (время выдачи), `exp` (время истечения), `role` (роль пользователя).
5. **Регистрация:**
   - Endpoint: `POST /api/v2.0/users`.
   - Запрос: JSON с полями `login`, `password`, `firstName`, `lastName`, `role`
6. **Аутентификация:**
   - Endpoint: `POST /api/v2.0/login`.
   - Запрос: JSON с полями `login`, `password`.
   - Ответ: JSON с полем `access_token` - обязательно, остальные - по желанию, например `type_token` ...
7. **Защищенные ресурсы:**
   - Пример: `GET /api/v2.0/messages`.
   - Запрос: JWT токен в заголовке `Authorization: Bearer <access_token>`.
8. **Ошибки:**
   - Все ошибки должны содержать `errorMessage` и `errorCode` (пятизначный код, первые три цифры - HTTP код).
9. **CRUD операции:**
   - Все CRUD операции должны быть доступны через соответствующие endpoints (например, `/api/v2.0/users`, `/api/v2.0/messages`).
   - Операции должны соответствовать требованиям Task310 (создание, получение, обновление, удаление).

Пример endpoint для получения автора:

- `GET /api/v2.0/users/{id}`
- Ответ: `UserResponseTo` в формате JSON (`user` объект).

Пример endpoint для создания контента:

- `POST /api/v1.0/tweet`
- Запрос: `TweetRequestTo` в формате JSON (`tweet` объект).
- Ответ: `TweetResponseTo` в формате JSON (`tweet` объект).