Пакет: `com.example.auth_app.dto`

`LoginRequest`

Намена: Едноставен DTO (Data Transfer Object) кој прима податоци при најавување.
Полиња:

- `username` — корисничко име внесено од клиентот.
- `password` — лозинка внесена од клиентот.

Опис: Овој објект автоматски се создава од JSON телото на `POST /login` барањето. Контролерот го проследува кон сервисот за автентикација. Класата нема логика — служи само како контејнер за податоци.

Зошто DTO: DTO класите се користат за одделување на влезните податоци од ентитетите што се зачувуваат во базата.

`RegisterRequest`

Намена: DTO објект кој прима податоци за регистрација.
Полиња:

- `username` — корисничко име.
- `email` — адреса за е-пошта.
- `password` — лозинка внесена од корисникот.

Опис: Се користи при `POST /register`. Контролерот ги зема податоците од објектот и ги праќа до сервисот за валидација и зачувување во базата.

Пакет: `com.example.auth_app.model`

`AppUser`

Намена: Ентитет кој го претставува регистрираниот корисник.
Главни полиња:

- `id` — примарен клуч (генериран автоматски).
- `username` — уникатно корисничко име.
- `email` — уникатна адреса за е-пошта.
- `passwordHash` — лозинка хаширана со BCrypt.

Опис:

- Се мапира на табелата `users` во H2 базата.
- Се креира при регистрација во `UserService`.
- Се користи за валидација на логин и за прикажување на податоци за профилот.

Безбедносни белешки:
Лозинките се хашираат и никогаш не се чуваат во чиста форма. Хашот не се испраќа кон клиентот преку API.

`SessionToken`

Намена: Ентитет кој претставува сесија на најавен корисник.
Главни полиња:

- `token` — случаен UUID стринг (служи како идентификатор на сесијата).
- `user` — поврзан `AppUser`.
- `createdAt` — момент кога е креирана сесијата.
- `expiresAt` — момент кога истекува сесијата.

Опис:

- Се создава при успешна најава во `UserService.createSession()`.
- Се чува во табелата `sessions`.
- Се проверува при секое барање во `UserService.validateSession()`.

- Се брише при одјава во `UserService.removeSession()`.

Зошто е потребен:
Постојаната сесија во базата овозможува лесно бришење (logout) и проверка на истекување. Тоа е едноставно и безбедно решение за лабораториска задача.

Пакет: `com.example.auth_app.repository`

`UserRepository`

Намена: Репозиториум за работа со `AppUser` објекти (JPA интерфејс).
Главни методи:

- `findByUsername(String username)` — бара корисник по корисничко име.
- `existsByUsername(String username)` — проверува дали веќе постои корисничкото име.
- `existsByEmail(String email)` — проверува дали веќе постои е-поштата.

Опис:
Сервисот `UserService` го користи овој репозиториум за барање и зачувување на корисници во базата.

`SessionRepository`

Намена: Репозиториум за работа со `SessionToken` ентитети.
Главни методи:

- `findByToken(String token)` — враќа сесија по токен.
- `deleteByToken(String token)` — ја брише сесијата по токен.

Опис:
`UserService` го користи за зачувување, верификација и бришење на сесии.

Пакет: `com.example.auth_app.service`

`UserService`

Намена: Главна класа со бизнис логика — регистрација, логирање и работа со сесии.
Зависности:

- `UserRepository` — за работа со корисници.
- `SessionRepository` — за работа со сесии.

Главни полја:

- `sessionExpiresSeconds` — колку долго трае сесијата.
- `passwordEncoder` — објект од `BCryptPasswordEncoder` за хаширање на лозинки.

Главни методи:

- `register(username, email, password)`
  - Проверува дали полињата се валидни и дали веќе постои корисник.
  - Хашира лозинка со BCrypt и ја зачувува во базата.
- `authenticate(username, password)`
  - Го проверува корисникот и споредува дали лозинката се совпаѓа со хашот.
- `createSession(AppUser user)`
  - Генерира случаен токен и креира сесија која се чува во базата.
- `validateSession(token)`
  - Проверува дали сесијата постои и не е истечена.
- `removeSession(token)`
  - Ја брише сесијата од базата (logout).

Зошто BCrypt:
BCrypt е безбеден и докажан алгоритам за хаширање кој користи вградени соли. Поефикасен е и побезбеден од едноставни алгоритми како SHA-256.

Зошто има SessionToken:
Серверски чувани сесии овозможуваат одјава, контрола и рок на траење — полесно е да се тестира и се добива целосна контрола над најавите.

Пакет: `com.example.auth_app.web`

`AuthController`

Намена: Контролер кој ги обработува HTTP барањата за регистрација, најавување и одјавување.
Ендпоинти:

- `POST /register`
    - Ги зема податоците за регистрација, ги праќа кон сервисот и враќа порака за успех или грешка.
- `POST /login`
    - Проверува кориснички податоци. Ако се точни:
        - Креира токен со `UserService.createSession()`.
        - Го става токенот во **HttpOnly cookie** за да се чува на клиентот.
    - Ако се неточни — враќа 401 (Unauthorized).
- `POST /logout`
    - Ја брише сесијата од базата и го чисти cookie-то.

Зошто HttpOnly cookie:
Ова спречува JavaScript да пристапи до токенот и го прави системот побезбеден од XSS напади.

`UserController`

Намена: Контролер за кориснички операции кои бараат автентикација.
Ендпоинт:

- `GET /user/me`
    - Ја чита cookie вредноста и ја проверува сесијата.
    - Ако е валидна, враќа основни податоци за корисникот (id, username, email).
    - Ако не е валидна — враќа 401 (неавтентицирано).

Зошто постои:
Овој ендпоинт покажува како се проверува логирањето. Во посложени системи тоа би го правел безбедносен филтер (Spring Security), но ова е поедноставна, разбирлива имплементација за лабораторија.

Главна класа

`AuthAppApplication`

Намена: Главна класа која ја стартува Spring Boot апликацијата.
Опис: Ја иницијализира апликацијата, конфигурациите и ја подига вградената web околина. Нема дополнителна логика.

Општи белешки за архитектурата

- **DTO класите** се користат за пренос на податоци од/до клиент без да се мешаат со ентитетите.
- **Ентитетите** (`AppUser`, `SessionToken`) ја претставуваат вистинската состојба во базата.
- **Репозиториумите** се користат за комуникација со базата без пишување SQL рачно.
- **UserService** ја содржи целата логика за валидација, хаширање и управување со сесии.

- **Контролерите** се едноставни — само ги примаат барањата и ги праќаат кон сервисот.

- **SessionToken** класата е создадена за да овозможи зачувување и управување со најавените сесии — што е токму барањето на лабораториската задача.