Meta

• Team: Maximilian Burr, Valerio Cocco, Daniel Rittershofer

• License: MIT

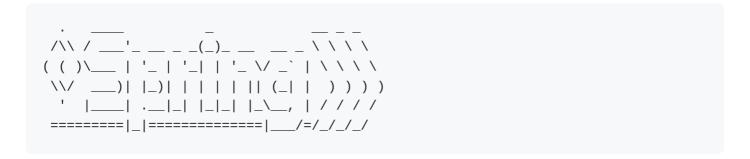
• Language: Java (version 19)

• Framework: Spring Boot 2.7.6

• ORM: JPA with Hibernate and PostgreSQL (spring-boot-starter-jpa)

• Docker image: ghcr.io/derburri/biletado-personal-v1:master

Port: 9000



OpenAPI generated server

Overview

This server was generated by the OpenAPI Generator project. By using the OpenAPI-Spec, you can easily generate a server stub. This is an example of building a OpenAPI-enabled server in Java using the SpringBoot framework.

The underlying library integrating OpenAPI to Spring Boot is springdoc. Springdoc will generate an OpenAPI v3 specification based on the generated Controller and Model classes. The specification is available to download using the following url: http://localhost:9000/v3/api-docs/

Start your server as a simple java application

Change default port value in application.properties

Generate Java Code

```
docker run --rm -v "${PWD}:/local" openapitools/openapi-generator-cli generate \
-i /local/personnel-v1.yml \
-g spring \
-o /local/out/spring
```

Build and Run Docker

Local

Build

- ./build_docker_image.sh
- docker build -t biletado-personal ./

Run

- ./run_docker.sh
- docker run -p 9000:9000 --network="host" --env-file ./env.list biletadopersonal:latest

Build & Run

```
./build_and_run_docker.sh
```

Config

Environment variables

With default values for local development:

```
POSTGRES_PERSONAL_USER=postgres
POSTGRES_PERSONAL_PASSWORD=postgres
POSTGRES_PERSONAL_DBNAME=personal
POSTGRES_PERSONAL_HOST=localhost;POSTGRES_PERSONAL_PORT=5432
BACKEND_URL=http://localhost/api/reservations/
```

```
KEYCLOAK_HOST=localhost
KEYCLOAK_REALM=biletado
```

In env.list are the variables configured that which used in run_docker.sh

Config Files

The default config files are located in src/main/java/resources. There are two profiles:
prod and test.

You can mount .properties files into the container:

```
docker run -v <PATH>:/usr/app/config/ -p 9000:9000 --network="host" --env-file
./env.list biletado-personal:latest
```

docker compose default config files

docker-compose mounts the compose/backend-personal directory with the config files

- ./compose/backend-personal/application.properties
- ./compose/backend-personal/application-prod.properties into the backend-personal container.

Tests

Automated tests are implemented with spring-boot-starter-test.

The tests are located in src/test/java/org/biletado/personal/v1/.

To run tests, execute the PersonalV1ApiBackendApplicationTests class.

CI/CD

CI/CD ist realized with GitHub Actions. The workflows are located in .github/workflows/docker-image.yml.



The tests are run automatically and a Docker image is created and uploaded to the registry.

Authentication

The JWT token is parsed by the Spring Security framework.

The configuration is in the class

org.biletado.personal.v1.configuration.SecurityConfiguration.

Logging

For logging, the capabilities of the Spring framework are used. For example, you can configure logging with the following properties in a properties file:

```
# logging
logging.level.org.hibernate.SQL=INFO
logging.level.org.hibernate=INFO
logging.level.javax.persistence=INFO
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=INFO
logging.level.org.biletado.logging.RequestLoggingFilter=INFO
logging.level.org.springframework.web=TRACE
logging.level.org.springframework.security=TRACE
#logging.level.org.zalando.logbook=TRACE
```

Zalando Logbook

Logbook is an library to enable **complete** request and response logging in JSON for different client- and server-side technologies.

You have to configure logging.level.org.zalando.logbook=TRACE for logging with logbook.

Example log:

```
{
  "origin": "remote",
  "type": "request",
  "correlation": "bd311e6d-1150-48f7-88f4-4c01152c7ab5",
  "protocol": "HTTP/1.1",
  "remote": "127.0.0.1",
  "method": "POST",
  "uri": "http://localhost:9000/personal/employees/",
  "headers": {
    "accept": [
      "application/json"
    ],
    "accept-encoding": [
      "gzip, deflate, br"
    ],
    "accept-language": [
      "de, en-US; q=0.7, en; q=0.3"
    ],
    "connection": [
      "keep-alive"
    "content-length": [
      "77"
    "content-type": [
      "application/json"
    ],
    "cookie": [
      "PGADMIN_LANGUAGE=en; pga4_session=32558f15-d757-4f1a-967a-Z5VY="
    ],
    "host": [
      "localhost:9000"
    ],
    "origin": [
      "http://localhost:9000"
    ],
    "referer": [
      "http://localhost:9000/swagger-ui/index.html"
    ],
    "sec-fetch-dest": [
```

```
"empty"
    ],
    "sec-fetch-mode": [
     "cors"
    ],
    "sec-fetch-site": [
     "same-origin"
    ],
    "user-agent": [
      "Mozilla/5.0 (X11; Linux x86_64; rv:108.0) Gecko/20100101 Firefox/108.0"
    1
  },
  "body": {
    "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "name": "Max Specimeno"
  }
}
```

How to use

Start the docker compose application in the compose directory with: docker-compose up. You can now test the personal backend under localhost.

If you want to test backend locally in IntelliJ you can use the run profile PersonalV1ApiBackendApplicationDev from the .run directory. For local tests, docker-compose needs to be running.