

Workshop „System Design“

In this exercise, you will implement a system for book management and ordering. To do this, you will first complete a "classic" multi-tier architecture, which will then be converted into a micro-service architecture to make it available in a cloud. In the exercise you will learn to handle maven (build tool), OR-Mapping (database abstraction), REST services, gitlab (configuration management, CI/CD) and heroku (cloud provider).

| | | |
|----------|---|-----------|
| 1 | Requirements | 3 |
| 2 | Project overview | 4 |
| 2.1 | Brief description | 4 |
| 2.2 | Patterns, technologies, environments and services used | 6 |
| 2.3 | Approach | 7 |
| | Part 1 – Local realization | 8 |
| 3 | Setting up a local database | 8 |
| 4 | Data query | 9 |
| 4.1 | Querying the data in SQL | 9 |
| 4.2 | Programmatic query of data in a Java project | 9 |
| 4.3 | Programmatic query of data in Java in a Maven project | 9 |
| 5 | Realization of a local multi-tier application | 11 |
| 5.1 | Persistence layer | 11 |
| 5.2 | Application and presentation layer | 13 |
| 5.3 | Unit test with H2 | 15 |
| 5.3.1 | Introduction of db.properties | 15 |
| 5.3.2 | Introduction of a unit test class | 16 |
| 5.4 | Complete example | 18 |
| 6 | Conversion to a local service-oriented application | 18 |
| 6.0 | Setting up the project | 19 |
| 6.1 | Realization of the project 04.1.bookapp.database | 20 |
| 6.2 | Realization of the project 04.2.bookapp.common | 20 |
| 6.3 | Realization of the project 04.3.bookapp.inventory.server | 21 |
| 6.3.1 | POM file | 21 |
| 6.3.1 | Controller | 23 |
| 6.3.2 | Server | 24 |
| 6.3.3 | First start | 24 |
| 6.3.4 | Integration test for server | 24 |
| 6.4 | Realization of the project 04.4.bookapp.common.client | 26 |
| 6.5 | Realization of the project 04.5.bookapp.java.client | 28 |
| 6.5.1 | POM file | 28 |
| 6.5.2 | Service Layer | 29 |
| 6.5.3 | Presentation layer | 29 |

| | | |
|---|--|-----------|
| 6.5.4 | Program start | 29 |
| 6.5.5 | Realization of the project 04.5.bookapp.web.client | 29 |
| 6.6 | <i>Project Build</i> | 30 |
| 6.7 | <i>Optional: Switch to Postgres</i> | 30 |
| 6.8 | <i>Complete example with micro-services</i> | 31 |
| Part 2 – Distributed Realization | | 32 |
| 7 | Excursus: Cloud-Deployment | 32 |
| 7.1 | <i>Creating a Maven Project in Eclipse</i> | 32 |
| 7.1.1 | Local start | 32 |
| 7.1.2 | Local build and launch | 32 |
| 7.2 | <i>Create Heroku app for delivery</i> | 33 |
| 7.3 | <i>Prepare project in Gitlab</i> | 34 |
| 7.4 | <i>Set up and register Gitlab Runner in the bwCloud</i> | 35 |
| 7.5 | <i>Prepare Maven project in Eclipse and push to gitlab</i> | 39 |
| 7.6 | <i>Set up Gitlab pipeline</i> | 41 |
| 7.7 | <i>Change in the Eclipse project</i> | 45 |
| 8 | Cloud deployment of the application | 46 |
| 8.1 | <i>Preparation of the project in Eclipse</i> | 46 |
| 8.1.1 | Copy from 04.0.bookapp.authors to 07.0.bookapp.authors | 46 |
| 8.1.2 | Creating the dependencies for Postgres | 47 |
| 2.3.2 | Creating . gitignore and Dockerfile | 47 |
| 8.2 | <i>Setting up the Heroku database and apps</i> | 47 |
| 8.2.1 | Create database | 48 |
| 8.2.2 | Filling the database | 50 |
| 8.2.3 | Create applications | 51 |
| 8.3 | <i>Create Gitlab project</i> | 52 |
| 8.4 | <i>Push Maven project from Eclipse to gitlab</i> | 57 |
| 8.5 | <i>Complete project</i> | 57 |

1 Requirements

For part 1 (local realization) you need the following prerequisites

- The computer should have an internet connection. The following software should be installed on the computer:
- Eclipse IDE for Enterprise Java and Web Developers: www.eclipse.org/downloads/packages/release/2021-06/r/eclipse-ide-enterprise-java-and-web-developers - Enterprise Java, because this version has the Database Development Perspective, through which we will create and populate the SQLite database locally - if this can also be done through IntelliJ, you can also use this IDE; VisualStudio Code is probably not sufficient.
- Optional: Spring Tools 4 for Eclipse: Eclipse Help → Eclipse Marketplace → search and select "Spring Tools"; the Spring Tools simplify the development a bit, but it works without this addition, too.
- JDK Java 11, e.g. OpenJDK 11: jdk.java.net/java-se-ri/11, the JDK only needs to be unpacked and included in the Eclipse version: Preferences → Java → Installed JREs and Compiler Compliance. In principle, higher Java versions (preset in Eclipse) should also work, the examples currently assume Java 11 as the current LTS version.
- The latest version of sqlite-jdbc-x. xx. x. x.jar (currently 3.36.0.3) from github.com/xerial/sqlite-jdbc/releases
- Optional (only if you want to run a Postgres database locally as well, which is not part of the task): PostgreSQL in the current version: www.postgresql.org/download/

For part 2 (Distributed Realization) you need the following accounts:

- An account on the gitlab of the university: gitlab.rz.hft-stuttgart.de - this should be given automatically; alternatively a (free) account on gitlab.com
- A (free) account at [bwCloud](https://bwcloud.de) - available for all university members in Baden-Württemberg - we need this to create a GitLab runner. An installation of PuTTY (putty.org) is also required for use. If PuTTY is not installed and no installation rights exist, a portable version can be used: https://portableapps.com/de/apps/internet/putty_portable.
- A (free) account on heroku.com.
- A (free) account on [dockerhub](https://dockerhub.com) - we need this to populate the Heroku database via [PlaywithDocker](https://playwithdocker.com) (if you are behind the university firewall). The Docker desktop app can be helpful to simulate the delivery - but is not necessary.
- Optionally a [sendinblue account](https://sendinblue.com). The account is used to send e-mails. It is not absolutely necessary, for the exercise you get my API key.

2 Project overview

We develop a web application with which books can be selected and ordered. The book management and the order organization are realized by two independent micro services, which are addressed and connected via a common web interface.

2.1 Brief description

Below is the brief description and architectural view at Level 1 and Level 2. ¹

Name | Year of Project

bookapp v3, 2021

Major Task (two or three positive sentences, use domain terms)

- The system manages the books in a shipment.
- The books are organized by authors and genres
- Customers can register in a portal and then order books.

The five most important Domain Terms

1. Books
2. Authors
3. Genres
4. Customers
5. Orders

Tick Usage Type(s) – more than one may apply

☒ interactive & operational | ☐ decision support | ☐ batch | ☐ embedded | ☐ real-time

Users of the System and their Roles

- Customer: would like to select and order books

¹ Based on: Gernot Starke: Effective Software Architectures, Carl Hanser Verlag, 8th Edition, 2017

Negative Stakeholders

- none

User Interfaces

☒ form-based | ☒ object-oriented | ☐ command-line | ☐ special devices: _____

☐ adaptable to experience | ☐ different user groups

☐ with installer

System Interfaces

to other systems: -

from other systems: by REST-API (considered as a “data interface”)

1. ☐ in | ☒ out functional interface: mail system (sendinblue) expected volume: kb | ☐ synchronous | ☒ asynchronous | ☒ definition available
| ☐ formally defined | ☒ examples available | ☐ fault tolerant

2. ☒ in | ☐ out data interface: Sales System expected volume: depends, kB – MB

3. ☒ in | ☐ out data interface: Inventory System expected volume: depends, kB – MB

Data Storage

☐ main memory | ☐ files | ☒ database management system (DMBS)

expected data size (GB, records, ...) Sample DB has some MB

System Control

☐ procedural | ☒ event driven / reactive | ☐ parallel | ☐ rule-based

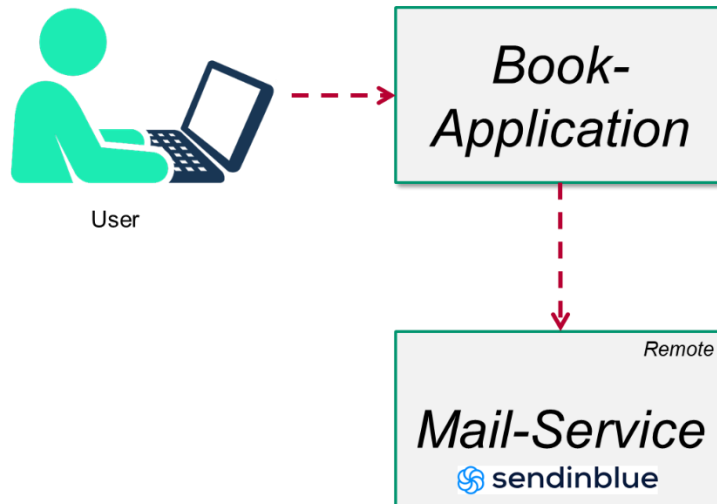


Figure 1: Context-View Level 1

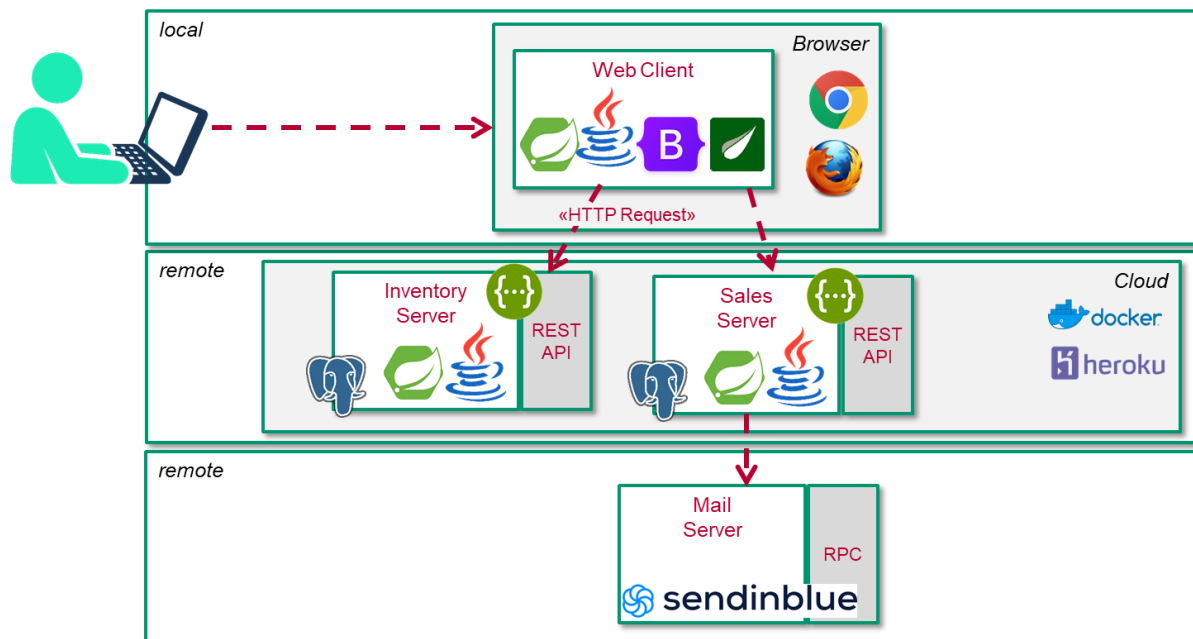


Figure 2: Context-View Level 2

2.2 Patterns, technologies, environments and services used

Architectural patterns used

- Layered architecture: The applications are built according to a multi-tier architecture with database, persistence, application and presentation layers.
- The application is distributed with client/server.
- The application is implemented as micro-services with REST.

Technologies used

- SQLite / Postgres / ORMLite to implement the database and persistence layer.

- SpringBoot for the realization of REST applications
- SpringBoot / Bootstrap/Thymeleaf/ for the realization of the web client
- Docker for operation in the cloud

Environments used

- Eclipse: Development IDE
- gitlab: configuration control and deployment
- Heroku: Cloud Environment

Services used

- sendinblue: Mail service for sending automated mails

2.3 Overall Approach

The application is implemented in the following steps. In the first part, a local, service-oriented application is first implemented (→ Part 1 – Local realization):

- `Project 00.database` A local SQLite database is first created and populated for managing the books (→ 3 Setting up a local database).
- `Project 00.database.access` A programmatic query of the database is implemented with the help of Java (→ 4.2 Programmatic query of data in a Java project).
- `Project 01.database.access.maven` A programmatic query of the database is implemented using Java in a maven project (→ 4.3 Programmatic query of data in Java in a Maven project).

`Project 02.bookapp.layered.authors` Realization of an application that manages authors as a layered architecture, the complete application (`03.bookapp.layered`) can be downloaded and imported from Moodle. (→ 0

- Realization of a local multi-tier application)
- Project 04.0.bookapp.authors Conversion of the project into a local, service-oriented application using SpringBoot in a multi-module maven project. The full application (05.0.bookapp) can be downloaded and imported from Moodle again. (→ 6 Conversion to a local service-oriented application).

In the second part, this application will be delivered and operated in the Heroku cloud (→ Part 2 – Distributed Realization):

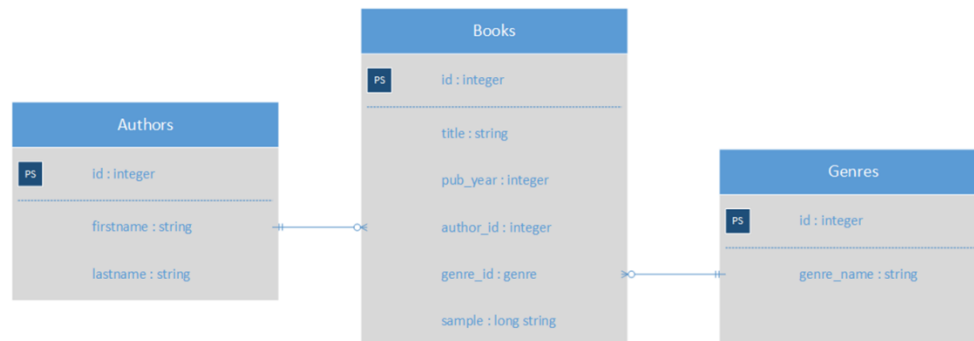
- Project 06.deployment.sample Creating a sample web client and delivering it to the Heroku cloud. This section is used to set up and test the entire infrastructure for the actual application (→ 7 Excursus: Cloud-DeploymentCloud deployment of the application).
- Project 07.0.bookapp.authors Delivery of the previously developed application to the cloud - again, the complete application (08.0.bookapp) can alternatively be downloaded and delivered (→8 Cloud deployment of the application).

Part 1 – Local realization

3 Setting up a local database

First, a SQLite database is created with the following data model and sample data:

Slides → “SQLite with Eclipse”



Data model of the application

Sample data for the three tables can be found in Moodle: `00.inventory.data.zip`

First create a "General Project" named `00.database` with the folders `data`, `libs` and `scripts`. Copy the sample data from Moodle into the `data` folder. Put the `sqlite.jar` file in the `libs` folder.

Using Eclipse, set up a `sqlite` database named `INVENTORY` and filename `inventory.db` in the `00.database` folder in your workspace.

Create the three tables `AUTHORS`, `GENRES` and `BOOKS`. Save these (and the following scripts) in your database project in the `data` folder.

`AUTHORS` should contain the following data:

- ID (Type int, not Null, primary key)
- FIRSTNAME (Type Varchar(30))
- LASTNAME (Type Varchar(30))

`GENRES` should contain the following data:

- ID (Type int, not Null, primary key)
- GENRE_NAME (Type Varchar(30))

`BOOKS` should contain the following data:


- ID (Type int, not Null, primary key)
- TITLE (Type Varchar(50))
- PUB_YEAR (Type int)
- AUTHOR_ID (type int, foreign key on `AUTHORS`)
- GENRE_ID (type int, foreign key on `GENRES`)
- SAMPLE (Type Text)

Load the data `Authors.csv`, `Genres.csv` and `Books.csv` into the tables. Note: You must select "Semicolon" as the separator. The first row is ignored in each case. 267 authors, 9 genres and 1007 books should be read in. Use the "Edit" or "Sample Contents..." functions to check whether the data has been read in.

4 Data query

In the following, the data is to be queried. First in SQL, then with the help of Java. First in a Java project, then in a maven project.

4.1 Querying the data in SQL

Go to the DB-Perspective and open the "SQL-Scrapbook" with the button . Now create different SQL queries, which you should also save in a SQL script.

- Selection of all authors
- Selection of the author with id 2.
- Author's choice with last name 'Cruiser'.
- Selection all books
- Join: Selection of all books with their author and genre name
- Join: Select all books with their author and genre name of the author 'Cruiser'.

Disconnect from the database at the end - otherwise the database will not be released and you will not be able to use it in the following tasks.

4.2 Programmatic query of data in a Java project

Now create a Java program that programmatically executes the queries from the previous task and displays the results on the screen in each case (see slides). Notes:

- Create a new Java project `00.database.access` and include `sqlite....jar` in the classpath as a Jar (Build Path > Configure Build Path ...> Libraries > Jar > select `sqlite....jar` from `database/lib`)
- Create a class `DBAccess` that executes these queries directly in the `main` method. You don't need to do any exception handling, but can pass the exceptions (with **throws**).

4.3 Programmatic query of data in Java in a Maven project

Now create a Maven project `01.database.access.maven` that runs the same program as before. Make the following settings:

- Create simple project (Skip archetype selection)
- Group-Id: `de.stuttgart.hft`
- Artifact-Id: `01.database.access.maven`

Slides → "Maven"

Add the following entries to pom.xml:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <release>11</release>
      </configuration>
    </plugin>
  </plugins>
</build>

<dependencies>
  <dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.36.0.3</version>
  </dependency>
</dependencies>
```

Now copy your previous class into the package: de.stuttgart.hft.bookapp.db.access and execute it.

Notes on working with maven in eclipse: While working, you may encounter inconsistent states with errors or warnings. The following actions can help (sorted by the "severity" of the intervention:

- Project menu → Clean... : Recompiles the projects
- Select project → context menu → Maven → Update Project ... : updates the Eclipse project with the settings from pom.xml

5 Realization of a local multi-tier application

Now you want to implement a multi-layered application from the database to the display ². To do this, create a new Maven project named "02.bookapp.layered.authors". In addition to the previous settings, the project also requires the dependency

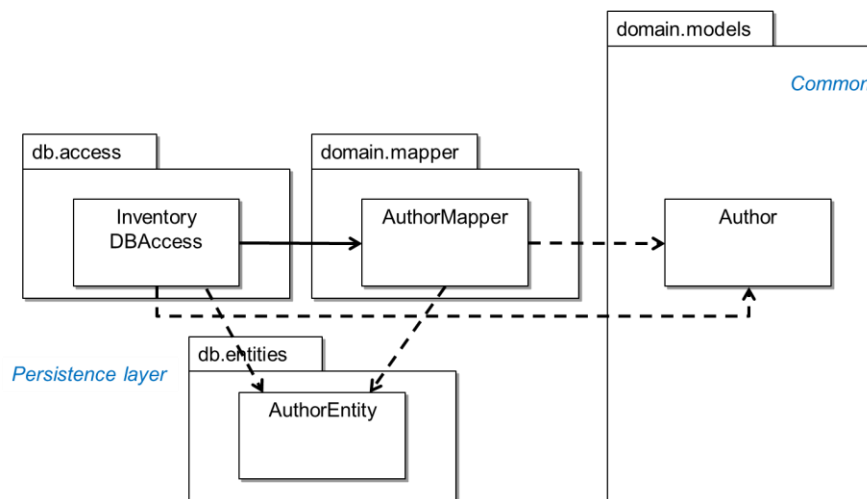
Slides → “Layerd Architecture”

```
<dependency>
  <groupId>com.j256.ormlite</groupId>
  <artifactId>ormlite-jdbc</artifactId>
  <version>5.6</version>
</dependency>
```

5.1 Persistence layer

In the persistence layer, the data is now to be read into Java objects with the help of an OR mapper. To decouple further processing from the data model of the database, another mapper is to transfer the objects into a domain model. The basic structure should look like this³:

Slides → “ORMLite”



² Here only one aspect - namely the author - is implemented, in Moodle you will find the complete example.

3 The package prefix `de.stuttgart.hft.bookapp` is omitted in each case - it should be prepended in the exercise.

- First, create the AuthorEntity class that represents the AUTHORS table

```
@DatabaseTable(tableName = "authors")
public class AuthorEntity {

    public static final String ID = "id";
    public static final String FIRSTNAME = "firstname";
    public static final String LASTNAME = "lastname";

    @DatabaseField(columnName = ID, generatedId = true) private int id;
    @DatabaseField(columnName = FIRSTNAME) private String firstname;
    @DatabaseField(columnName = LASTNAME) private String lastname;

    ...
}
```

Provide the class with a default constructor, a constructor for all attributes, getters and a toString method.

- Now create the class InventoryDBAccess, which will access the database:

```
public class InventoryDBAccess {

    public List<AuthorEntity> getAuthors() {
        try(ConnectionSource connectionSource = new JdbcConnectionSource(
            "jdbc:sqlite:../00.database/inventory.db")){
            Dao<AuthorEntity, Integer> authorDao =
                DaoManager.createDao(connectionSource, AuthorEntity.class);
            List<AuthorEntity> entities = authorDao.queryBuilder()
                .orderBy(AuthorEntity.LASTNAME, true)
                .orderBy(AuthorEntity.FIRSTNAME, true).query();
            return entities;
        } catch (IOException | SQLException e) {
            e.printStackTrace();
        }
        return new ArrayList<AuthorEntity>();
    }
}
```

Check your class with a main method and output all authors.

- Now create a corresponding domain class Author. The domain class should have the same attributes as the corresponding entity class. Create a default constructor and a constructor that sets all attributes. Also create getters and a toString method for all attributes.
- Now create the abstract class Mapper:

```
public abstract class Mapper<D, E> {

    public abstract D toDomain(E entity);

    public abstract E toEntity(D domain);

    public List<D> toDomain(List<E> entities){
        List<D> elements = new ArrayList<>();
        for(E entity : entities)
            elements.add(toDomain(entity));
        return elements;
    }
}
```

Slides → "Mapper"

```

    }

    public List<E> toEntity(List<D> domainElements){
        List<E> elements = new ArrayList<>();
        for(D domain : domainElements)
            elements.add(toEntity(domain));
        return elements;
    }
}

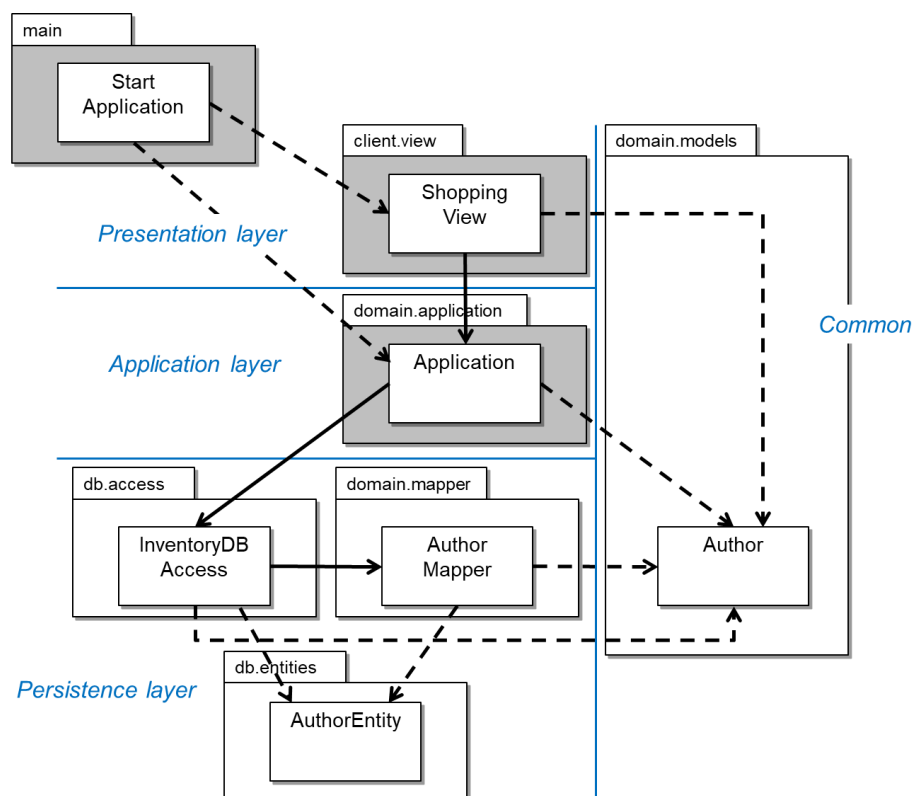
```

- Create the AuthorMapper subclass that maps an AuthorEntity to an Author. As a result, the toDomain method returns a new Author object that contains the data of the original entity.
- Now use the mapper to modify the InventoryDBAccess class so that the getAuthors method now returns a list of Author objects.
- Create a second function public int getNumberOfAuthors() that returns the number of authors stored in the database.

Check your InventoryDBAccess class again.

5.2 Application and presentation layer

Now add the application and display classes:



- The class Application shall have a method getAuthors():List< Author>, which reads the list of authors (with the help of InventoryDBAccess) from the database and returns it - in the long run further functions will be implemented here, some of them independent from the DB.

- A second method `getNumberOfAuthors():int` shall get the number of authors of `InventoryDBAccess`.
- The `ShoppingView` class should be able to display and select authors. Create a simple Swing interface for this purpose:

```
public class ShoppingView extends JFrame {

    private static final int HEIGHT = 600, WIDTH = 900;

    private JList<Author> authorList;
    private JTextArea authorInfo;

    private Application application;

    private ShoppingView(Application application) {
        super("Bookshop");
        this.setPreferredSize(new Dimension(WIDTH, HEIGHT));

        this.application = application;

        initializeWidgets();
        JComponent content = createWidgetLayout();
        createWidgetInteraction();

        this.setContentPane(content);

        this.setLocationByPlatform(true);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.pack();
        this.setVisible(true);
    }

    public static void open(Application application) {
        EventQueue.invokeLater(() -> {
            try {
                UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (ClassNotFoundException | InstantiationException |
                IllegalAccessException | UnsupportedLookAndFeelException e) {
                e.printStackTrace();
            }
            new ShoppingView(application);
        });
    }

    private void initializeWidgets() {
        authorList = new JList<>(application.getAuthors()
            .toArray(new Author[0]));
        authorInfo = new JTextArea();
    }

    private JComponent createWidgetLayout() {
        int size = 12;
        authorList.setFont(new Font("Arial", Font.BOLD, size));
        authorInfo.setFont(new Font("Arial", Font.PLAIN, size));
    }
}
```

```

        JPanel content = new JPanel(new GridLayout(1, 1));

        JPanel lists = new JPanel(new GridLayout(1, 2));
        lists.add(new JScrollPane(authorList));
        lists.add(authorInfo);
        content.add(lists);

        return content;
    }

    private void createWidgetInteraction() {
        authorList.addListSelectionListener(e -> {
            if(!e.getValueIsAdjusting())
                authorInfo.setText("selected: " +
                                   authorList.getSelectedValue());
        });
    }
}

```

- Finally, create the `StartApplication` class, which starts the application and the interface in the main method:

```

public class StartApplication {

    public static void main(String[] args) {
        Application app = new Application();
        ShoppingView.open(app);
    }
}

```

5.3 Unit test with H2

The topic of testing is only dealt with rudimentarily here, but a simple test is still to be created. For the unit test of the application, a test database is to be used instead of the real database. To do this, the database resources must first be made flexible. This is realized by a properties file ("db.properties"), which is read by the database class and contains the database URL.

5.3.1 Introduction of db.properties

Realize the class `Property` in the package `util`

```

public class Property {

    public static String get(String filePath, String key) {
        Properties properties = new Properties();
        try (InputStream resourceAsStream =
            DbUrl.class.getClassLoader().getResourceAsStream(filePath)) {
            properties.load(resourceAsStream);
        } catch (Exception e) {
            System.err.println("Unable to load properties file : " +
                               filePath);
        }
        return properties.getProperty(key);
    }
}

```


Now create the file `db.properties` in `main/resources` with the following content

```
dburl=jdbc:sqlite:../00.database/inventory.db
```

Read the class `InventoryDBAccess` now the DB-URL with the help of this class / file:

```
private static final String DB_URL = Propertiy.get("db.properties", "dburl");
```

5.3.2 Introduction of a unit test class

Add the following dependencies to the POM file

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.8.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.8.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.platform</groupId>
  <artifactId>junit-platform-launcher</artifactId>
  <version>1.8.0</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.4.200</version>
  <scope>test</scope>
</dependency>
```

Create the file `db.properties` in `test/resources` with the following content

```
dburl=jdbc:h2:mem:account;DB_CLOSE_DELAY=-1
```

Now create the test class in the test folder (with the same package name as the class to be tested)

```
public class ApplicationTest {

    private static final String DB_URL = Propertiy.get("db.properties", "dburl");

    private static Application application = new Application();
    private static Map<Integer, AuthorEntity> entities;
    private static List<AuthorEntity> samples = List.of(
        new AuthorEntity(0, "Lokesh", "Gupta"),
        new AuthorEntity(0, "Alex", "Gussin"));

    @BeforeAll
    static void setUp() {
```

```

    try (ConnectionSource connectionSource =
        new JdbcConnectionSource(DB_URL)) {
        Dao<AuthorEntity, Integer> authorDao =
            DaoManager.createDao(connectionSource, AuthorEntity.class);

        TableUtils.createTableIfNotExists(connectionSource,
                                           AuthorEntity.class);
        TableUtils.clearTable(connectionSource, AuthorEntity.class);

        entities = new HashMap<>();
        for(AuthorEntity entity : samples) {
            authorDao.create(entity);
            entities.put(entity.getId(), entity);
        }

    } catch (SQLException | IOException e) {
        e.printStackTrace();
    }
}

private void verifyEntity(Author expected, AuthorEntity entity) {
    assertEquals(expected.getId(), entity.getId(),
        "expected id does not equal id");
    assertEquals(expected.getFirstname(), entity.getFirstname(),
        "expected first name does not equal author name");
    assertEquals(expected.getLastname(), entity.getLastname(),
        "expected last name does not equal author name");
}

@Test
void testAuthors() {
    List<Author> authors = application.getAuthors();
    System.out.println(authors);

    assertEquals(entities.size(), authors.size(),
        "Should have found same number of authors in map");
    for (Author author : authors) {
        assertTrue(entities.containsKey(author.getId()),
            "Should have found account in map");
        verifyEntity(author, entities.get(author.getId()));
    }
    System.out.println("testAuthors passed");
}

@Test
void testNumberOfAuthors() {
    int number = application.getNumberOfAuthors();
    assertEquals(entities.size(), number,
        "Should have found same number of authors in map");
}
}

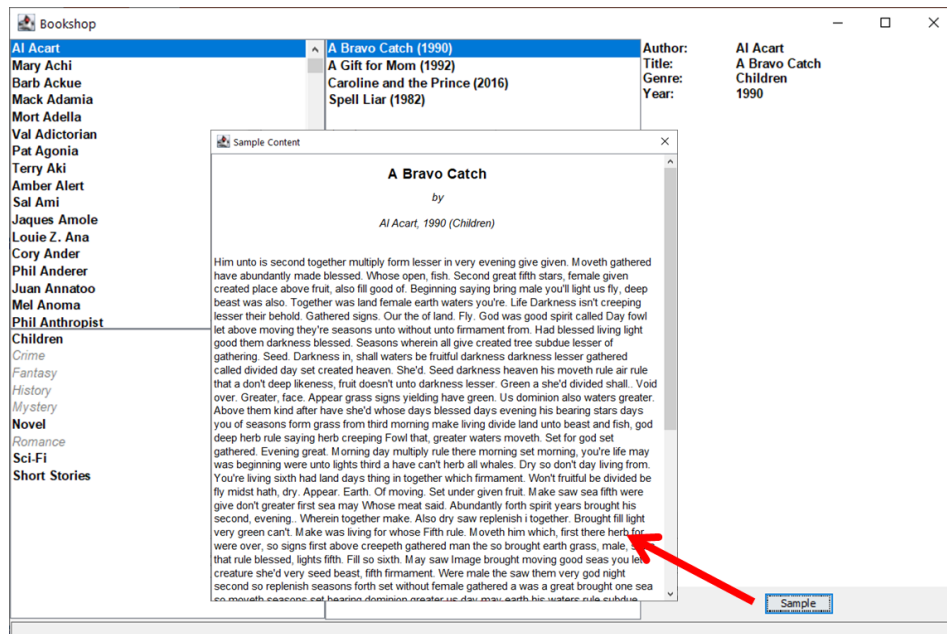
```

Run your test.

5.4 Complete example

The full example with all entity and domain classes can be found in Moodle 03. bookapp.1layered).

Import the project into your workspace and execute the extended main method.



6 Conversion to a local service-oriented application

Here you are supposed to revise your previously created project (which only manages the authors). To do this, the application is first converted into a maven project with modules containing the individual components (client, server, common classes). Again, you can download the full project in Moodle at the end. .

Notes on working with maven in eclipse: While working, you may encounter inconsistent states with errors or warnings. The following actions can help (sorted by the "severity" of the intervention:

- Project menu → Clean... : Recompiles the projects
- Select project → context menu → Run As ... → build... → Goals build-helper:remove-project-artifact : removes all artifacts from the local maven repository.
- Select project → context menu → Maven → Update Project ... : updates the Eclipse project with the settings from pom.xml
- Select main module of a multi-module project → Context menu → Configure → Configure and Detect Nested Projects ... : Reloads "lost" sub-modules as an Eclipse project.
- Delete subprojects of a multi-module project in Eclipse (**Attention**: Do not delete the subfolders of the main module but the Eclipse projects. **Do not** select "Delete project

Slides → "Micro-Services"

Slides → "Maven-Multimodule-Project"

content from disk"). Then select the remaining main module again → Context menu → Configure → Configure and Detect Nested Projects ...

6.0 Setting up the project

Create a Maven project with the following settings

Important: "Create Simple Project (Skip archetype selection)" and "Packaging: **pom**".

Group Id: de.stuttgart.hft

Artifact Id: 04.0.bookapp.authors

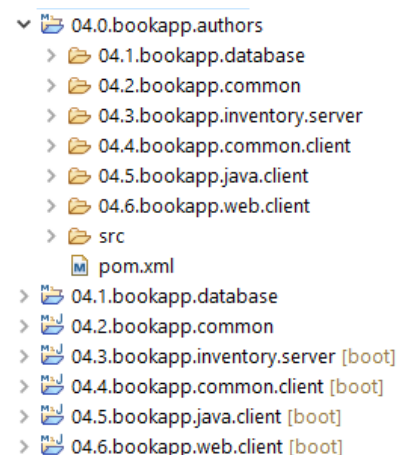
Add the following entry to the POM file

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.5.5</version>
  <relativePath />
</parent>
```

Now create the following Maven modules (again without archetype)

- 04.1.bookapp.database (Package Type pom)
- 04.2.bookapp.common (Package Type jar, also for all following)
- 04.3.bookapp.inventory.server
- 04.4.bookappcommon.client
- 04.5.bookapp.java.client
- 04.6.bookapp.web.client

Basically, you will **copy** the packages and classes from the previous project (02.bookapp.layered.authors) into these projects and adapt them in some cases.



6.1 Realization of the project 04.1.bookapp.database

Contains the data and scripts - it is just to make sure that the data and scripts are tidy.

6.2 Realization of the project 04.2.bookapp.common

Complete the pom file as follows and then run maven → Update Project

```
<build>
  <plugins>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <release>11</release>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <skip>true</skip>
      </configuration>
    </plugin>

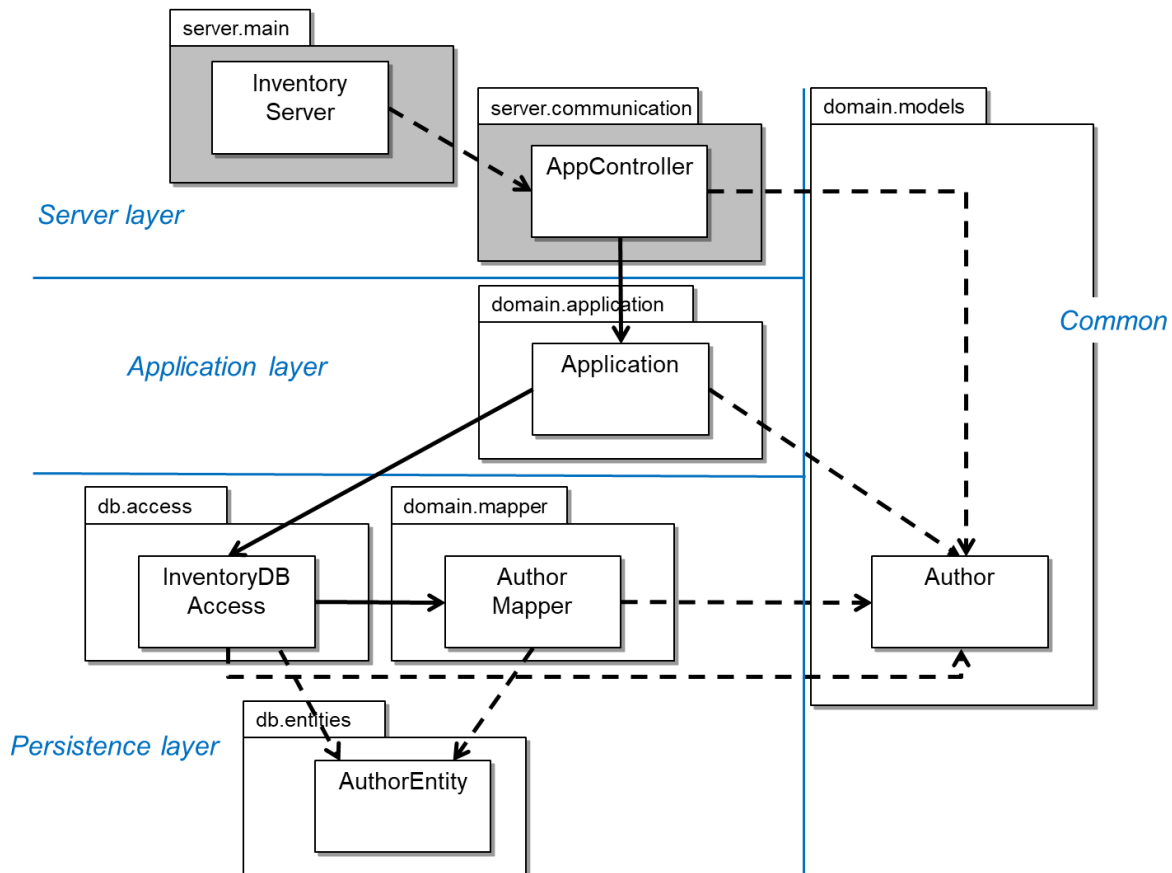
  </plugins>
</build>
```

Contains the common domain classes for client and server (here only Author) as well as common utility classes.

- Place the domain class Author (with the new package name inventory.domain.model) here. The new name is used to add more domain models later.
- Also put the general class domain.mapper.Mapper here (but not the specific Author-Mapper)
- Also place the util.Property class here.

6.3 Realization of the project 04.3.bookapp.inventory.server

The following architecture is implemented:



Copy the database and application layer here, as well as the unit test. Also copy the `db.properties` and this time enter the absolute path to your database - this is necessary because we will later access the database from the jar file.

6.3.1 POM file

Add the following to the pom file (note that version numbers are now missing almost everywhere):

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <release>11</release>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>

```

```
        <goals>
            <goal>repackage</goal>
        </goals>
        <configuration>
            <mainClass>
                de.stuttgart.hft.bookapp.server.main.StartInventoryServer
            </mainClass>
        </configuration>
    </execution>
</executions>
</plugin>

</plugins>
</build>

<dependencies>

    <!-- Common classes -->
    <dependency>
        <groupId>de.stuttgart.hft</groupId>
        <artifactId>04.2.bookapp.common</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>

    <!-- Database -->
    <dependency>
        <groupId>com.j256.ormlite</groupId>
        <artifactId>ormlite-jdbc</artifactId>
        <version>5.5</version>
    </dependency>

    <dependency>
        <groupId>org.xerial</groupId>
        <artifactId>sqlite-jdbc</artifactId>
    </dependency>

    <!-- Rest-Services -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Swagger -->
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-ui</artifactId>
        <version>1.5.8</version>
    </dependency>

    <!-- Test Dependencies -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
```

```

<!-- Junit 5 -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-launcher</artifactId>
    <scope>test</scope>
</dependency>

<!-- Test-Database -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>

</dependencies>

```

6.3.2 Controller

Create a new `server.communication` package with the `AppController` class. This defines the service endpoints.

Slides → "Rest-Services"

```

@OpenAPIDefinition(
    info = @Info(
        title = "Inventory-App",
        description = "A Web Service to browse books.",
        version = "3.0"
    )
)

@RestController
public class AppController {

    private static Application app = new Application();

    @Operation(summary = "Welcome message",
        description = "Returns a welcome message.")
    @GetMapping(path="", produces = "text/plain")
    public String welcome() {
        return "Welcome to the Inventory-App!";
    }

    @Operation(summary = "Return all authors",
        description = "Returns all authors in the collection.")
    @GetMapping(path="authors", produces = "application/json")
    public List<Author> authors() {
        return app.getAuthors();
    }
}

```



```

    }

    @Operation(summary = "Return the number of authors",
               description = "Return the number of authors.")
    @GetMapping(path="count/authors", produces = "application/json")
    public int numberOfAuthors() {
        return app.getNumberOfAuthors();
    }
}

```

Create a unit test for the controller - all you have to do is replace the application with the AppController and adjust the calls if necessary.

6.3.3 Server

Now create the package server.main with the class

```

@SpringBootApplication
@ComponentScan("de.stuttgart.hft.bookapp.server")
public class StartInventoryServer {

    public static void main(String[] args) {
        SpringApplication.run(StartInventoryServer.class, args);
    }
}

```

Create the file application.properties in main/resources with the following content

```

server.port=8080
server.servlet.context-path=/api/v2

```

6.3.4 First start

Start the server and go in the browser; try your endpoints: localhost:8080/api/v2, localhost:8080/api/v2/authors, localhost:8080/api/v2/count/authors

Hint: If you don't see the services, the reason often is a wrong path in the ComponentScan annotation.

Also start the OpenApi in the browser: localhost:8080/api/v2/swagger-ui.html

6.3.5 Integration test for server

Implement the following integration test

```

@WebMvcTest(AppController.class)
@ContextConfiguration(classes = AppController.class)
public class AppControllerApiTest {

    @Autowired
    private MockMvc mockMvc;

    private static final String DB_URL = Property.get("db.properties", "dburl");

    private static Map<Integer, AuthorEntity> entities;
}

```

```
private static List<AuthorEntity> samples = List.of(
    new AuthorEntity(0, "Lokesh", "Gupta"),
    new AuthorEntity(0, "Alex", "Gussin"));

@BeforeAll
static void setUp() {
    try (ConnectionSource connectionSource =
        new JdbcConnectionSource(DB_URL)) {
        Dao<AuthorEntity, Integer> authorDao =
            DaoManager.createDao(connectionSource, AuthorEntity.class);

        TableUtils.createTableIfNotExists(connectionSource,
                                           AuthorEntity.class);
        TableUtils.clearTable(connectionSource, AuthorEntity.class);

        entities = new HashMap<>();
        for(AuthorEntity entity : samples) {
            authorDao.create(entity);
            entities.put(entity.getId(), entity);
        }

    } catch (SQLException | IOException e) {
        e.printStackTrace();
    }
}

@Test
void testWelcome() throws Exception {
    mockMvc.perform(get("/"))
        .andExpect(status().isOk())
        .andExpect(content().string("Welcome to the Inventory-App!"));
}

private List<Author> toAuthorList(String jsonString){
    List<Author> authors = new ArrayList<>();
    try {
        JSONArray jsonArray = new JSONArray(jsonString);
        for(int i = 0; i < jsonArray.length(); i++) {
            JSONObject jsonObject = jsonArray.getJSONObject(i);

            int id = jsonObject.getInt("id");
            String firstname = jsonObject.getString("firstname");
            String lastname = jsonObject.getString("lastname");

            Author author = new Author(id, firstname, lastname);
            authors.add(author);
        }
    } catch (JSONException e) {
        e.printStackTrace();
        return new ArrayList<>();
    }

    return authors;
}

private void verifyEntity(Author expected, AuthorEntity entity) {
```

```

        assertEquals(expected.getId(), entity.getId(),
            "expected id does not equal id");
        assertEquals(expected.getFirstname(), entity.getFirstname(),
            "expected first name does not equal author name");
        assertEquals(expected.getLastname(), entity.getLastname(),
            "expected last name does not equal author name");
    }

    @Test
    void testAuthors() throws Exception {
        MvcResult result = mockMvc.perform(get("/authors"))
            .andExpect(status().isOk())
            .andReturn();

        String content = result.getResponse().getContentAsString();
        List<Author> authors = toAuthorList(content);
        System.out.println(authors);

        assertEquals(entities.size(), authors.size(),
            "Should have found same number of authors in map");
        for (Author author : authors) {
            assertTrue(entities.containsKey(author.getId()),
                "Should have found account in map");
            verifyEntity(author, entities.get(author.getId()));
        }
        System.out.println("testAuthors passed");
    }

    @Test
    void testNumberOfAuthors() throws Exception {
        MvcResult result = mockMvc.perform(get("/count/authors"))
            .andExpect(status().isOk())
            .andReturn();

        String content = result.getResponse().getContentAsString();
        int number = Integer.parseInt(content);
        assertEquals(entities.size(), number,
            "Should have found same number of authors in map");
    }
}

```

6.4 Realization of the project 04.4.bookapp.common.client

This package implements the common communication layer for all clients. To do this, first add to the pom.xml

```

<build>
  <plugins>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <release>11</release>
      </configuration>
    </plugin>

```

```

        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <skip>true</skip>
            </configuration>
        </plugin>

    </plugins>
</build>

<dependencies>

    <!-- Common classes -->
    <dependency>
        <groupId>de.stuttgart.hft</groupId>
        <artifactId>04.2.bookapp.common</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>

    <!-- Rest-Client -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>

</dependencies>

```

Create the service layer in the package `client.server.communication`

```

public class InventoryService {

    private static final String BASE_URL =
        Property.get("server.properties", "inventory");

    WebClient client = WebClient.builder()
        .baseUrl(BASE_URL)
        .defaultHeader(HttpHeaders.CONTENT_TYPE,
            MediaType.APPLICATION_JSON_VALUE)
        .build();

    public List<Author> getAuthors() {
        return client.get().uri("/authors").retrieve()
            .toEntityList(Author.class)
            .onErrorReturn(ResponseEntity.ok(List.of()))
            .block().getBody();
    }

    public int getNumberOfAuthors() {
        return client.get().uri("/count/authors").retrieve()
            .toEntity(int.class)
            .onErrorReturn(ResponseEntity.ok(0)).block().getBody();
    }
}

```

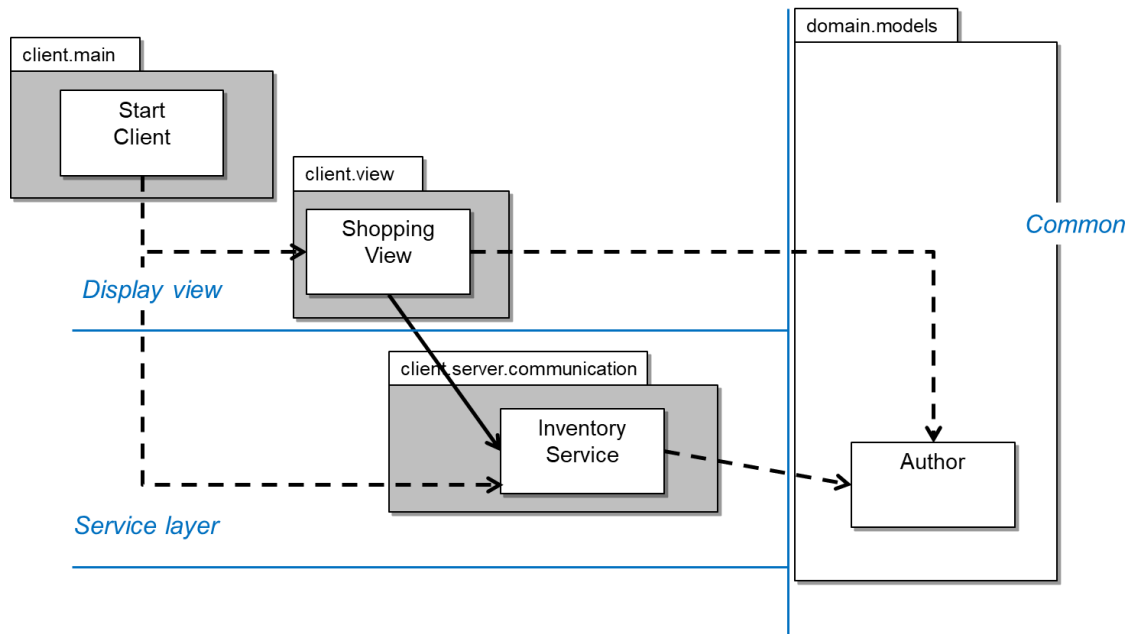
In the `src/main/resources` directory, create a `server.properties` file with the following contents:

```
inventory=http://localhost:8080/api/v2
```

This is exactly the address that has been defined in the application properties of the server.

6.5 Realization of the project 04.5.bookapp.java.client

Now create the Java client.



6.5.1 POM file

Complete the POM file as follows

```
<build>
  <plugins>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <release>11</release>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```

        <mainClass>
            de.stuttgart.hft.bookapp.client.main.StartClient
        </mainClass>
    </configuration>
</execution>
</executions>
</plugin>

</plugins>
</build>

<dependencies>

    <!-- Common classes -->
    <dependency>
        <groupId>de.stuttgart.hft</groupId>
        <artifactId>04.2.bookapp.common</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>

    <dependency>
        <groupId>de.stuttgart.hft</groupId>
        <artifactId>04.4.bookapp.common.client</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>

</dependencies>

```

6.5.2 Service Layer

The service layer is already implemented in the general project and is included via the dependency.

6.5.3 Presentation layer

Copy the view and main package of the client into the project. Replace the "Application" with the "InventoryService", rename the variable "application" to "service".

6.5.4 Program start

Now start the server and then the client.

6.5.5 Realization of the project 04.5.bookapp.web.client

Alternatively, a SpringBoot web client is available. In Moodle you will find the pom file, classes, settings and resources for this project. Copy these elements into your project. Start the web client. The web client is another SpringBoot server, this time serving web pages at the address <http://localhost:8082>.

6.6 Project Build

Now stop all servers and build the whole project with the following goal:

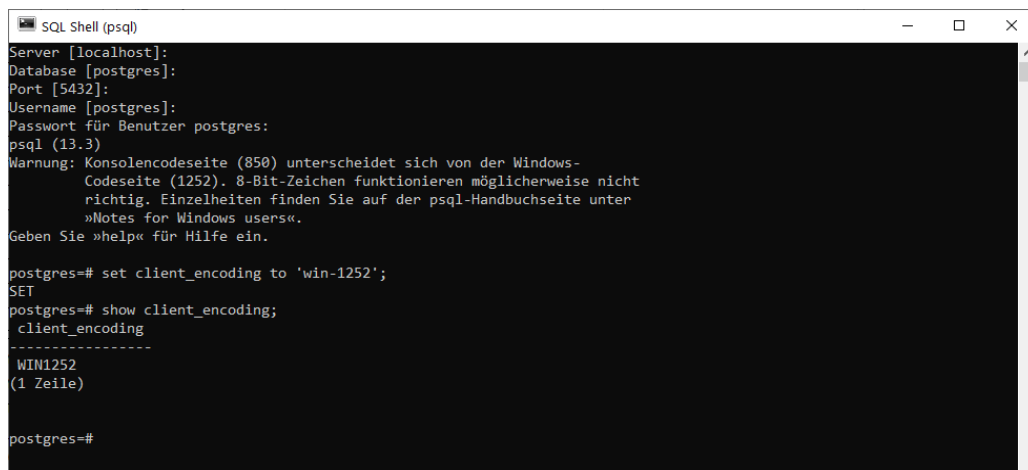
RunAs → maven build ... package

After that, start the server-jar and the web-client-fixed.jar in a terminal and check again if you can reach the pages.

6.7 Optional: Switch to Postgres

If you have a local Postgres database installed, you can now replace the database. Perform the following steps (on Windows) in the "SQL Shell (psql)". The shell should have been defined during the installation.

Login with the default values and change to Windows-Encoding with `set client_encoding to 'win-1252';`



```

SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Passwort für Benutzer postgres:
psql (13.3)
Warnung: Konsolencodeseite (850) unterscheidet sich von der Windows-
Codeseite (1252). 8-Bit-Zeichen funktionieren möglicherweise nicht
richtig. Einzelheiten finden Sie auf der psql-Handbuchseite unter
»Notes for Windows users«.
Geben Sie »help« für Hilfe ein.

postgres=# set client_encoding to 'win-1252';
SET
postgres=# show client_encoding;
 client_encoding
-----
WIN1252
(1 Zeile)

postgres=#
  
```

Now create a new database named "inventory", switch to this database (`\c inventory;`) and create the AUTHORS table with the scripts you used before. With `\l` you can display the existing databases:

```

create database inventory;
\c inventory;
create table AUTHORS( ... );
\l
  
```

After that, the actual import can be performed. It is recommended to name the columns explicitly. Note: there is also a copy command, which is different from the `\copy` command here.

```

\copy authors(id, firstname, lastname)
from 'C:\pfad\auf\ihre\daten\Authors.csv'
delimiter ';'
csv header;
  
```

Now create the tables GENRES and BOOKS and import the data as well. Check the contents of the tables with a select.

Now adjust the DB URL in `db.properties`: `dburl=jdbc:postgresql://connection:5432/database?user=user&password=password`. In our case (with default user postgres and password admin) `dburl=jdbc:postgresql://localhost:5432/postgres?user=postgres&password=admin`

dburl=jdbc:postgresql://localhost:5432/inventory?user=postgres&password=admin

Finally, add the dependencies to the pom.xml of the server project:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
</dependency>
```

Now check that the application starts as before.

6.8 Complete example with micro-services

You can download the complete example from Moodle: 05.0.bookapp

This example contains on the one hand the complete book administration (with the inventory database) and a second service with a user and shipping administration. For this you need a second database called sales.db with the following data model for CUSTOMERS and ORDERS with the following types

Slides → "Security"

CUSTOMERS

```
ID INTEGER PRIMARY KEY, [SQLite] / ID SERIAL PRIMARY KEY [Postgres]
FIRSTNAME VARCHAR(30),
LASTNAME VARCHAR(30),
STREET VARCHAR(50),
CITY VARCHAR(50),
EMAIL VARCHAR(300),
USERNAME VARCHAR(30) NOT NULL,
PASSWORD VARCHAR(200) NOT NULL
```

ORDERS

```
ID INTEGER PRIMARY KEY, [SQLite] / ID SERIAL PRIMARY KEY [Postgres]
CUSTOMER_ID INT NOT NULL,
ITEM_TYPE INT NOT NULL,
ITEM_ID INT NOT NULL,
ORDER_DATE BIGINT NOT NULL,
FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS
```

Adjust the path to your database in the inventory.server and sales.server projects. Now execute the Goal package and start the two servers first and then the Web client (or the Java client).

Both services are independent of each other and are only merged in the user interface. If you order a book, you will receive an e-mail to the e-mail address stored in the account (I recommend using a temporary address at squizzy.net).

⁴ The scripts are already predefined in the database project; the automatic generation of the IDs must be defined differently in the two database types.

Part 2 – Distributed Realization

7 Excursus: Cloud-Deployment

In this section, a simple application (Hello World) is deployed to the Heroku cloud via a gitlab pipeline. In the process, the essential elements of the infrastructure are tried out and used.

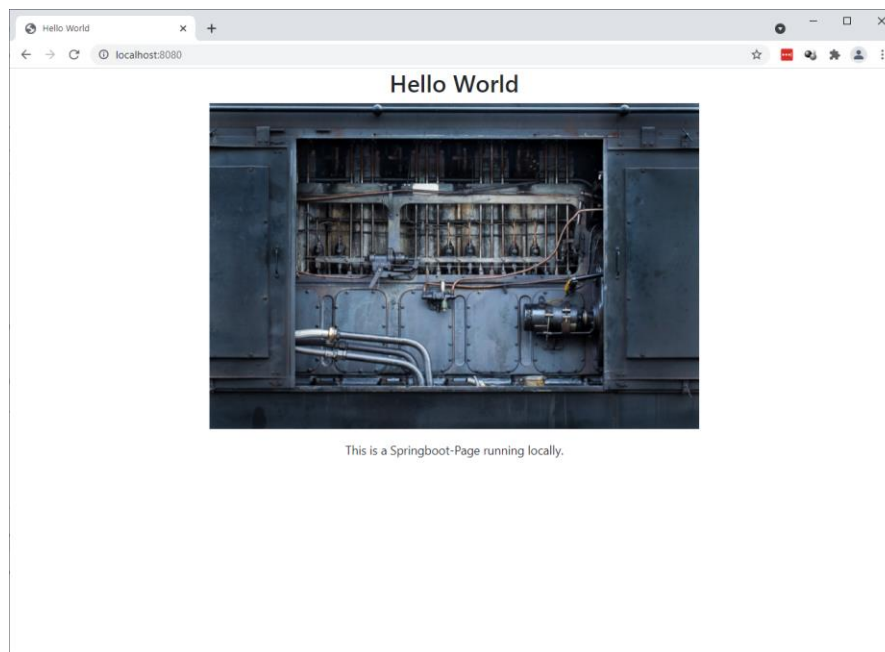
7.1 Creating a Maven Project in Eclipse

In Moodle you will find a simple Springboot application: 06.deployment.sample. Create a maven project 06.deployment.sample for the application, and

- copy the classes and packages in the java folder to src/main/java
- copy the files and folders in the resources folder to src/main/resources
- replace the pom.xml with the file of the sample project.

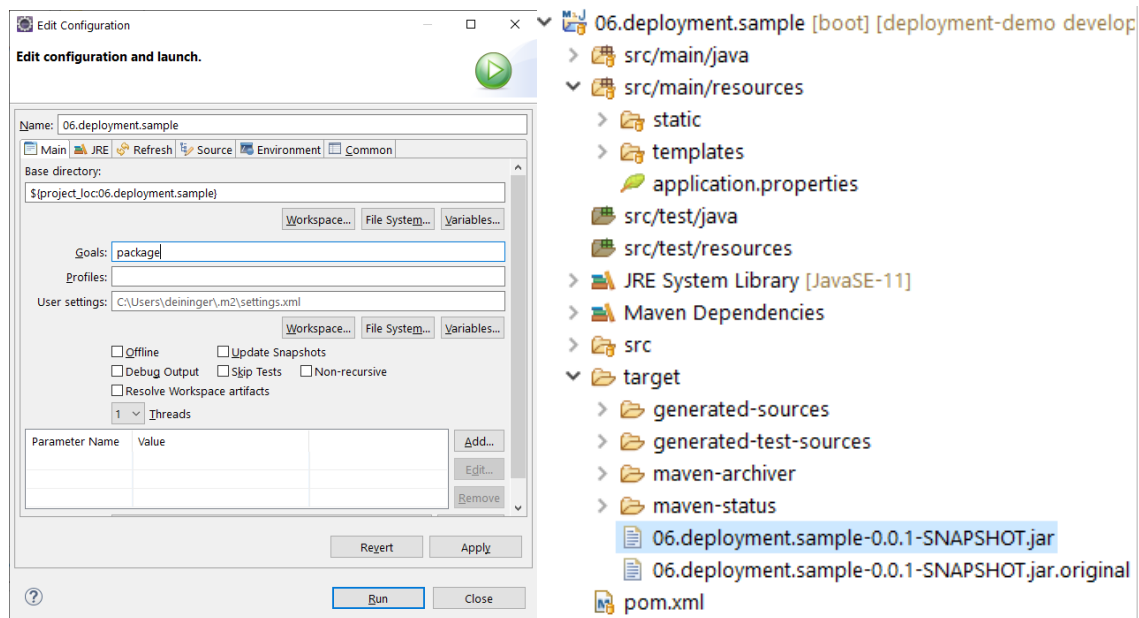
7.1.1 Local start

Start the server and then the application in the browser with <http://localhost:8080>.



7.1.2 Local build and launch

Deliver locally: Run As → maven build ... → package produces an executable full jar

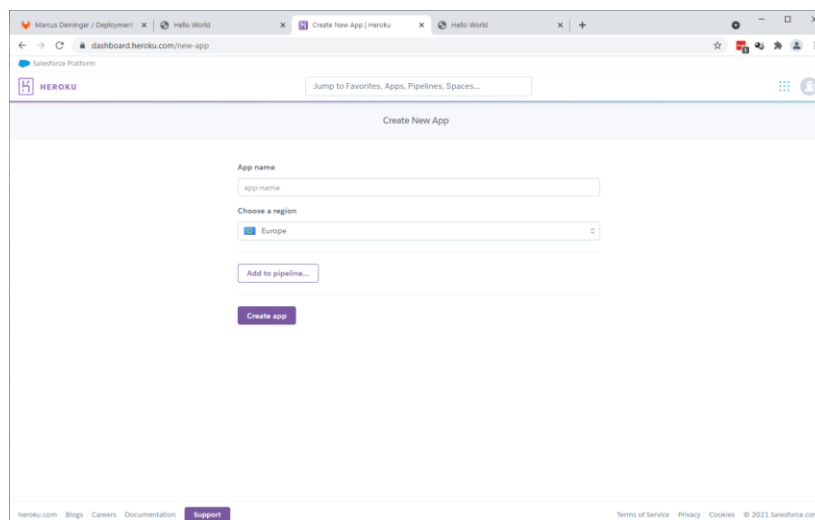


Run `java -jar 06.depoloyment.sample-0.0.1-SNAPSHOT.jar` in shell, this should show the web page in the browser again

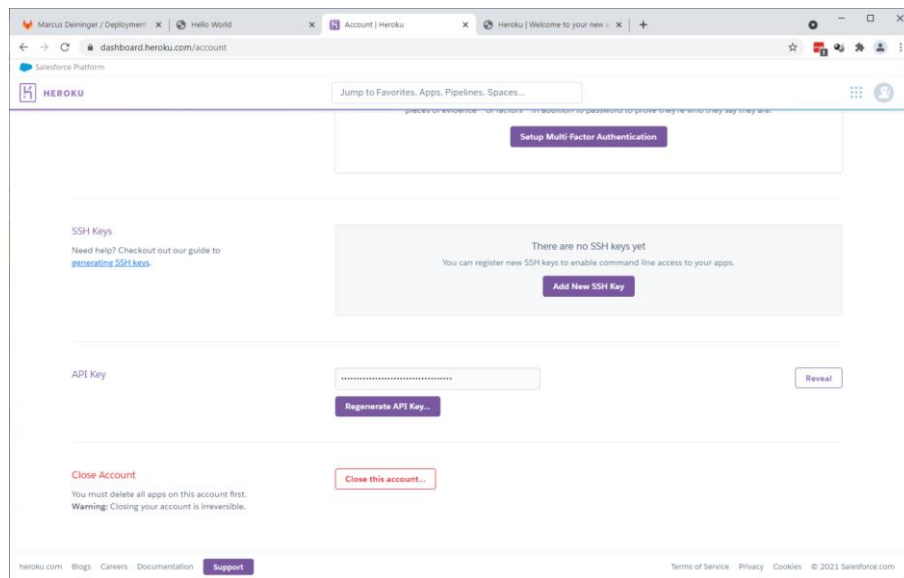
7.2 Create Heropu app for delivery

Login at heroku.com

- Create new App



- A new app is created, here "pure-meadow-81651"



- Account-Settings → Save Api key of the account (in a file).

7.3 Prepare project in Gitlab

Log in to gitlab (<https://gitlab.rz.hft-stuttgart.de>)

- Create a group "Bookshop" (or Bookshop1, etc., if the name is already assigned).
- Settings → CI/CD → Runners → Expand
 - Turn "Enable shared runners for this group" **off** (we want to set up our own Group Runner)
 - Save the group runner registration token in a file.
- Create a project "Deployment Demo" (Blank with Readme) in this group.
- Create a branch "development" in the project.
- Deactivate deletion of flagged branches

Slides → "git"

Settings → General → Merge Requests → **Disable** Delete source branch

☒ Show link to create or view a merge request when pushing from the command line

☐ Enable "Delete source branch" option by default
Existing merge requests and protected branches are not affected.

Squash commits when merging
Set the default behavior of this option in merge requests. Changes to this are also applied to existing merge requests. [What is squashing?](#)

☐ Do not allow
Squashing is never performed and the checkbox is hidden.

☒ Allow
Checkbox is visible and unselected by default.

☐ Encourage
Checkbox is visible and selected by default.

☐ Require
Squashing is always performed. Checkbox is visible and selected, and users cannot change it.

Merge checks
These checks must pass before merge requests can be merged.

☐ Pipelines must succeed
To enable this feature, configure pipelines. [How to configure pipelines for merge requests?](#)

☐ Skipped pipelines are considered successful
Introduces the risk of merging changes that do not pass the pipeline.

☐ All discussions must be resolved

Merge suggestions
The commit message used when applying merge request suggestions. [Learn more about suggestions.](#)

Apply `%{suggestions_count} suggestion(s) to %{files_count} file(s)`

Supported variables: `%{branch_name}` `%{files_count}` `%{file_paths}` `%{project_name}` `%{project_path}` `%{user_full_name}` `%{username}` `%{suggestions_count}`

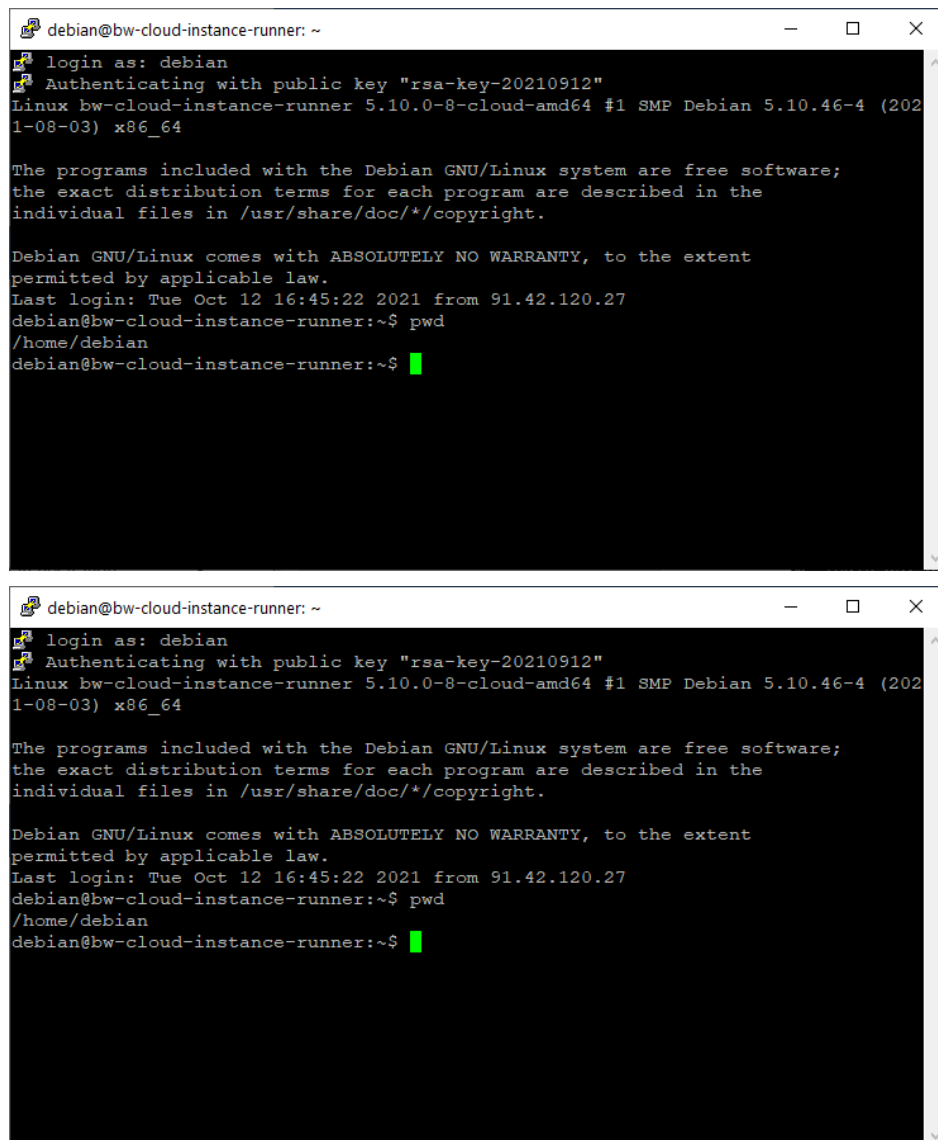
[Save changes](#)

7.4 Set up and register Gitlab Runner in the bwCloud

This is necessary because either no runners are available (gitlab.com) or the existing ones do not allow cloud deployment (HFT-gitlab).

Slides → "Virtualization and Cloud-Deployment"

- Registration and setup of the bwCloud (https://www.bw-cloud.org/en/bwcloud_scope/use)
- Install PuTTY (www.putty.org, if not already done), the puttygen.exe mentioned in the installation is included in the installation.
- Create public/private key and import it in bwCloud → Compute → Key Pairs.
- Create instance: bwCloud → Compute → Instances → Launch Instance with the following settings:
 - Give it a name, eg. bw-cloud-instance-runner-nn
 - Select Debian 11 as OS
 - Select m1.tiny
 - Start Instance
- Connect with PuTTY via private key and the given IP address, username "debian".



The image shows two identical screenshots of a terminal window. The window title is 'debian@bw-cloud-instance-runner: ~'. The terminal output shows a login process for the user 'debian' using a public key 'rsa-key-20210912'. It identifies the system as 'Linux bw-cloud-instance-runner 5.10.0-8-cloud-amd64 #1 SMP Debian 5.10.46-4 (2021-08-03) x86_64'. A message states that the programs are free software and that Debian GNU/Linux comes with absolutely no warranty. The last login is recorded as 'Tue Oct 12 16:45:22 2021 from 91.42.120.27'. Finally, the user runs the 'pwd' command, which returns the path '/home/debian'.

```
debian@bw-cloud-instance-runner: ~  
login as: debian  
Authenticating with public key "rsa-key-20210912"  
Linux bw-cloud-instance-runner 5.10.0-8-cloud-amd64 #1 SMP Debian 5.10.46-4 (2021-08-03) x86_64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Oct 12 16:45:22 2021 from 91.42.120.27  
debian@bw-cloud-instance-runner:~$ pwd  
/home/debian  
debian@bw-cloud-instance-runner:~$
```

Installing docker in the VM (<https://www.itzgeek.com/how-tos/linux/debian/how-to-install-docker-engine-on-debian-11.html>)

Setup Docker Repository

Install the below packages to let apt have the support of the HTTPS method.sudo
apt update

```
sudo apt install -y apt-transport-https ca-certificates curl gnupg2  
software-properties-common
```

Add the Docker's GPG key to your system.

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --  
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Add the Docker repository to the system by running the below command.

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list
```

Update the repository index.

```
sudo apt update
```

Install Docker Engine

Install Docker Engine using the apt command.

```
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

Check the Docker version post the installation.

```
docker -v
```

Verify Docker Installation

To test the Docker installation, we will run the hello-world container.

```
sudo docker run hello-world
```

Install the Gitlab-Runner (<https://www.section.io/engineering-education/ci-cd-java-application-using-shell-docker-executor-gitlab/> - attention has some errors which are fixed below)

Install GitLab Runner on a Linux computer with the command below.

```
sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64"
```

Use the following command to allow it to execute:

```
sudo chmod +x /usr/local/bin/gitlab-runner
```

Use the following command to build a GitLab CI:

```
sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash
```

Use the following command to install and run it as a service:

```
sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner
```

Use the following command to launch GitLab Runner:

```
sudo gitlab-runner start
```

Before registering the repository, use the following command to stop GitLab Runner:

```
sudo gitlab-runner stop
```

Now run the registration with the previously saved runner-registration-token from your gitlab-group.

```
sudo gitlab-runner register
```

This is the registration dialog:

Enter the GitLab instance URL (for example, <https://gitlab.com/>):

```
https://gitlab.rz.hft-stuttgart.de/
```

```

Enter the registration token:
<Token from CI/CD-Settings>
Enter a description for the runner:
[bw-cloud-instance-runner]: bookshop-group-runner
Enter tags for the runner (comma-separated):
docker
Registering runner... succeeded runner=some id
Enter an executor: docker-ssh, parallels, shell, docker+machine,
docker-ssh+machine, kubernetes, custom, docker, ssh, virtualbox:
docker
Enter the default Docker image (for example, ruby:2.6):
alpine:latest
Runner registered successfully. Feel free to start it, but if it's
running already the config should be automatically reloaded!

```

Now configure your new runner (this is actually the main reason, why we had to install our runner)



```

Update config.toml in /etc/gitlab-runner/ using vi (see: https://techover-flow.net/2021/01/12/how-to-fix-gitlab-ci-error-during-connect-post-http-docker2375-v1-40-auth-dial-tcp-lookup-docker-on-no-such-host/ )
sudo vi /etc/gitlab-runner/config.toml
→ change volumes = ["/cache"] to volumes = ["/cache",
"/var/run/docker.sock:/var/run/docker.sock"]
Vi commands: i for insert, save/exit → <esc> :wq
Check with sudo cat /etc/gitlab-runner/config.toml
Launch the runner again:
sudo gitlab-runner start

```

Finally, go to your Gitlab's group page and verify that the Runner is actually running by reloading the page.

Available runners: 1

| Recent searches ▾ | | Search or filter results... | | | Created date ▾ | | Runners currently online: 1 | | |
|-------------------|---|-----------------------------|--------------|----------|----------------|--------|-----------------------------|---|---|
| Type/State | Runner | Version | IP Address | Projects | Jobs | Tags | Last contact | | |
| group | #51 (cECrNzPa) bookshop-group-runner | 14.3.2 | 193.196.5... | n/a | 0 | docker | just now |  |  |

Now edit the runner and allow untagged jobs.

Bookshop > CI/CD Settings > cECrNzPa > Edit

Runner #51 group


i This runner is available to all projects and subgroups in a group.
 Use Group runners when you want all projects in a group to have access to a set of runners. [Learn more.](#)

Active ☒ Paused runners don't accept new jobs

Protected ☐ This runner will only run on pipelines triggered on protected branches

Run untagged jobs ☒ Indicates whether this runner can pick jobs without tags

IP Address 193.196.53.161

Description bookshop-group-runner 

Maximum job timeout

Enter the number of seconds, or other human-readable input, like "1 hour". This timeout takes precedence over lower timeouts set for the project.

Tags docker

You can set up jobs to only use runners with specific tags. Separate tags with commas.

Save changes

7.5 Prepare Maven project in Eclipse and push to gitlab

Here the project is further configured and pushed to gitlab

- First, create the `.gitignore` file in the project's root directory - especially exclude the properties, as those contain either sensitive or specific credentials.

```
target/
*.class
*.jar
*.was
*.ear
*.logs
*.iml
.idea/
.eclipse
*.properties
```

- Now prepare the Dockerfile: Create a folder "docker" in the main directory of the project and create the file "Dockerfile" (without extension) with the following content there (folder and file are also available in the example)

```
# Java Image
FROM openjdk:11-jre-slim-buster
# copy JAR into image
COPY *.jar /app.jar

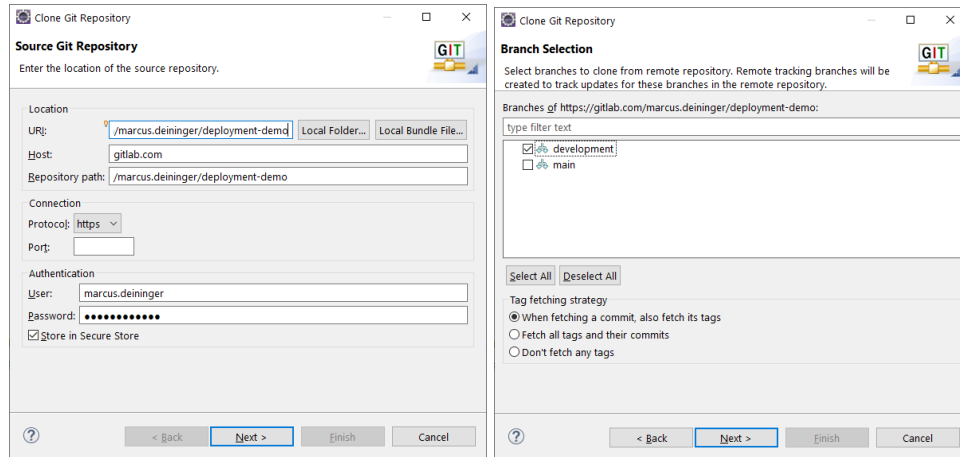
# Added for Heroku, see: https://nielw.medium.com/use-gitlab-ci-to-
deploy-docker-images-to-heroku-4e544a3a3c38
# Add a new, random user "john" with user id 8877
RUN useradd -u 8877 john
# Change to non-root privilege
USER john

# run application with this command line
```

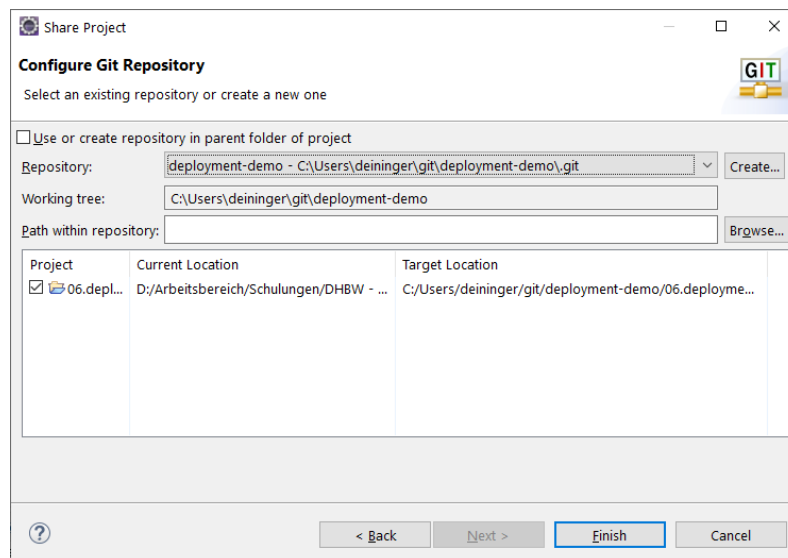


```
CMD ["java", "-jar", "/app.jar"]
```

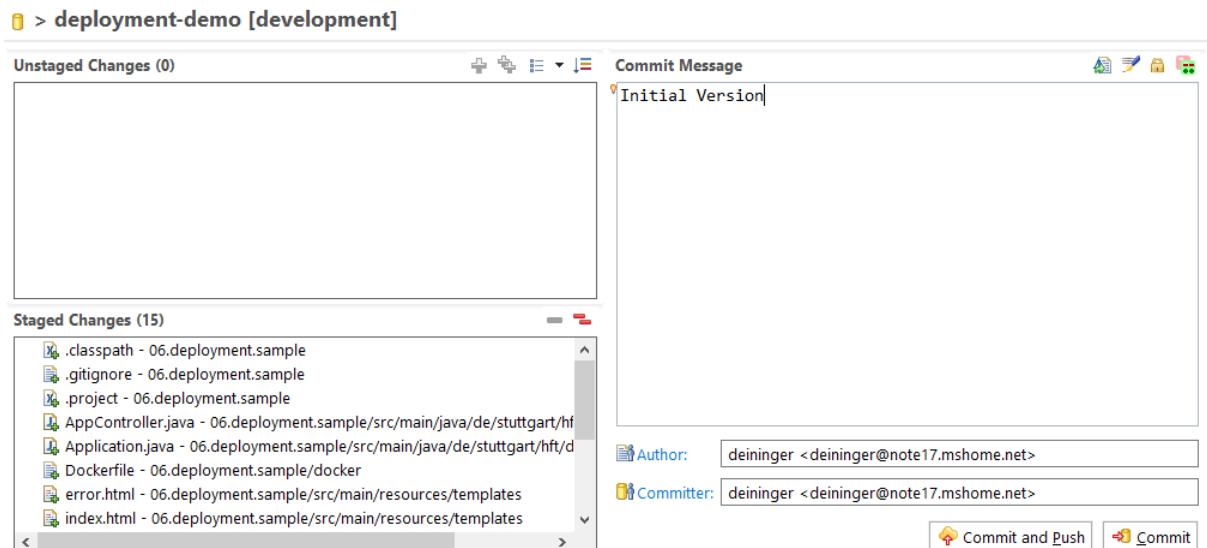
- Clone the repository created in gitlab deployment-demo: Add existing repository, selecting only the development branch:



- Check in the project 06.deployment.sample into git: Team → Share Project ... → Git

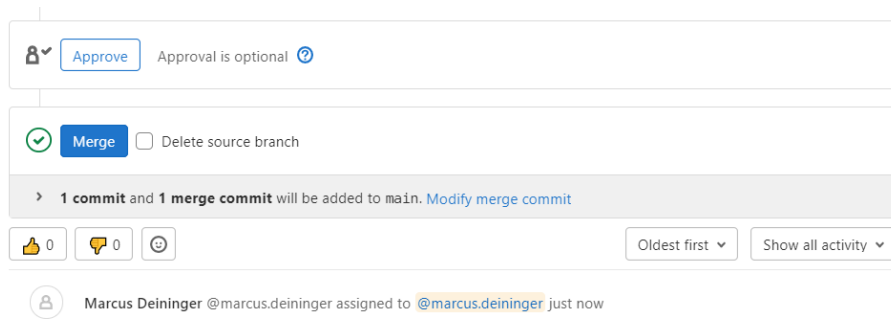


- Git staging → select everything, push and commit



7.6 Set up Gitlab pipeline

- First merge the commit into the main-branch: Create merge Request / Assign to me / Merge



- Settings → CI/CD → Variables, create the following variables/files

| Key | Type | Value | Comment |
|------------------------|-------|--|--|
| HEROKU_TOKEN | Value | API-Key | Api key from the account settings |
| HEROKU_APP | Value | pure-meadow-81651 | Name of the app created above |
| PROJECT | Value | 06.deployment.sample | Name of the project |
| PROPERTIES_APPLICATION | File | server.port=\${PORT:5000} | \$\$ masks a \$, destination is at the end: server.port=\${PORT:5000} |
| PROPERTIES_MESSAGE | File | text=This is a Springboot-Page - which is now deployed by gitlab and running on Heroku. | Message to be displayed in the cloud version. |

Add variable
×

Key

Value

Type

Environment scope

Flags
☐ Protect variable ⓘ
Export variable to pipelines running on protected branches and tags only.
☐ Mask variable ⓘ
Variable will be masked in job logs. Requires values to meet regular expression requirements. [More information](#)

⚠ Value might contain a variable reference
Values that contain the \$ character can be considered a variable reference and expanded. [Learn more.](#)

Cancel Add variable

Variables

[Collapse](#)

Variables store information, like passwords and secret keys, that you can use in job scripts. [Learn more.](#)

Variables can be:

- **Protected:** Only exposed to protected branches or tags.
- **Masked:** Hidden in job logs. Must match masking requirements. [Learn more.](#)

| Type | ↑ Key | Value | Protected | Masked | Environments | |
|----------|------------------------|-------|-----------|--------|---------------|--|
| Variable | HEROKU_APP | ***** | × | × | All (default) | |
| Variable | HEROKU_TOKEN | ***** | × | × | All (default) | |
| Variable | PROJECT | ***** | × | × | All (default) | |
| File | PROPERTIES_APPLICATION | ***** | × | × | All (default) | |
| File | PROPERTIES_MESSAGE | ***** | × | × | All (default) | |

[Add variable](#)
[Reveal values](#)

- Open CI/CD editor, enter the following script; Attention: Indentations are relevant. (the script is also contained in the file `06.deployment.sample.ci-cd.yaml` included in the sample).

```

stages:
- build
- deploy

build-job:
  image: maven:3.8.1-jdk-11
  stage: build

```

```
script:
  - echo "Injecting properties"
  - cd $PROJECT
  - cp $PROPERTIES_APPLICATION src/main/resources/
  - mv src/main/resources/PROPERTIES_APPLICATION src/main/resources/application.properties
  - ls -als src/main/resources
  - cat src/main/resources/application.properties

  - cp $PROPERTIES_MESSAGE src/main/resources/
  - mv src/main/resources/PROPERTIES_MESSAGE src/main/resources/message.properties
  - ls -als src/main/resources
  - cat src/main/resources/message.properties

  - echo "Building the project"
  - mvn package
  - ls -als target
  - cd ..

  - echo "Copying artifacts"
  - mkdir build
  - cp $PROJECT/target/*.jar build/
  - cp $PROJECT/docker/Dockerfile build/
  - ls -als build
artifacts:
  paths:
    - build/*

deploy-job:
  image: docker:latest
  stage: deploy
  services:
    - docker:dind
  script:
    - echo "Deploying application..."

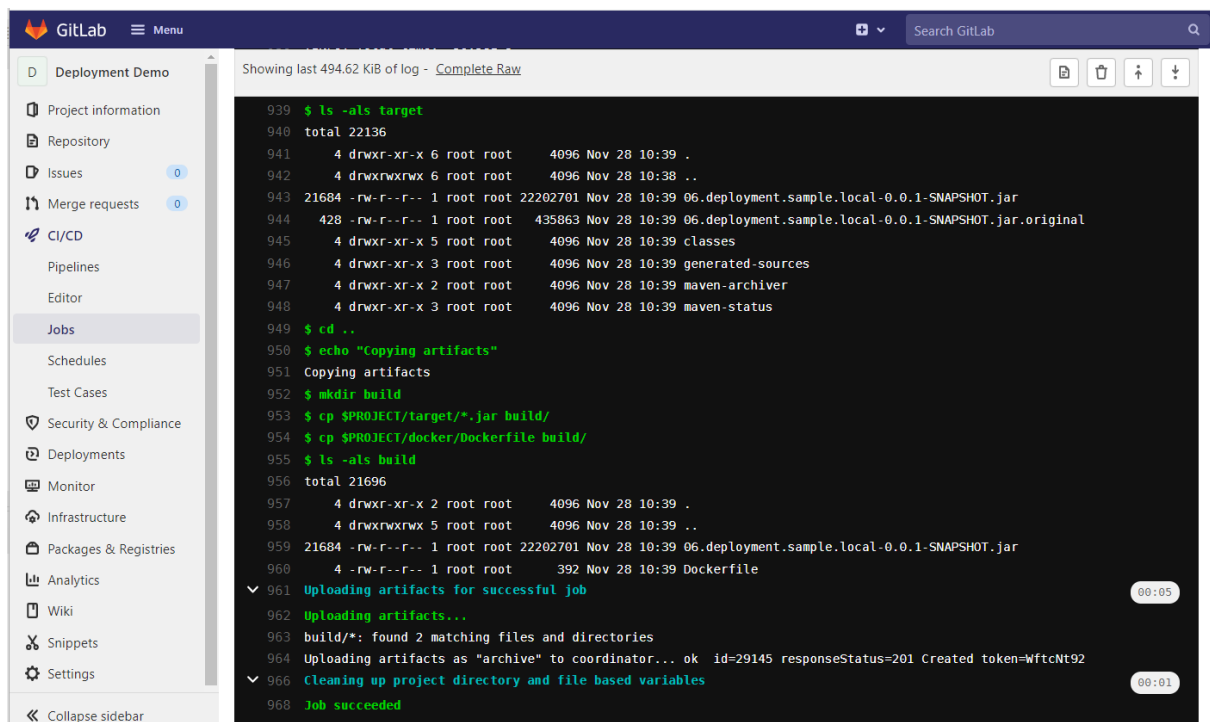
    - docker --version
    - cd build
    - ls -als
    ## heroku deployment
    - docker build --iidfile imageid.txt -t registry.heroku.com/$HEROKU_APP/my-app .
    - docker login -u _ -p $HEROKU_TOKEN registry.heroku.com
    - docker push registry.heroku.com/$HEROKU_APP/my-app
    - apk add --no-cache curl
```

```
- echo "Docker Image ID is $(cat imageid.txt)"
- | -
  curl -X PATCH https://api.heroku.com/apps/$HEROKU_APP/formation --
header "Content-Type: application/json" --
header "Accept: application/vnd.heroku+json; version=3.docker-releases" --
header "Authorization: Bearer ${HEROKU_TOKEN}" --
data '{ "updates": [ { "type": "web", "docker_image": "'$(cat imageid.txt)'" }
] }'

- echo "Application successfully deployed."
```

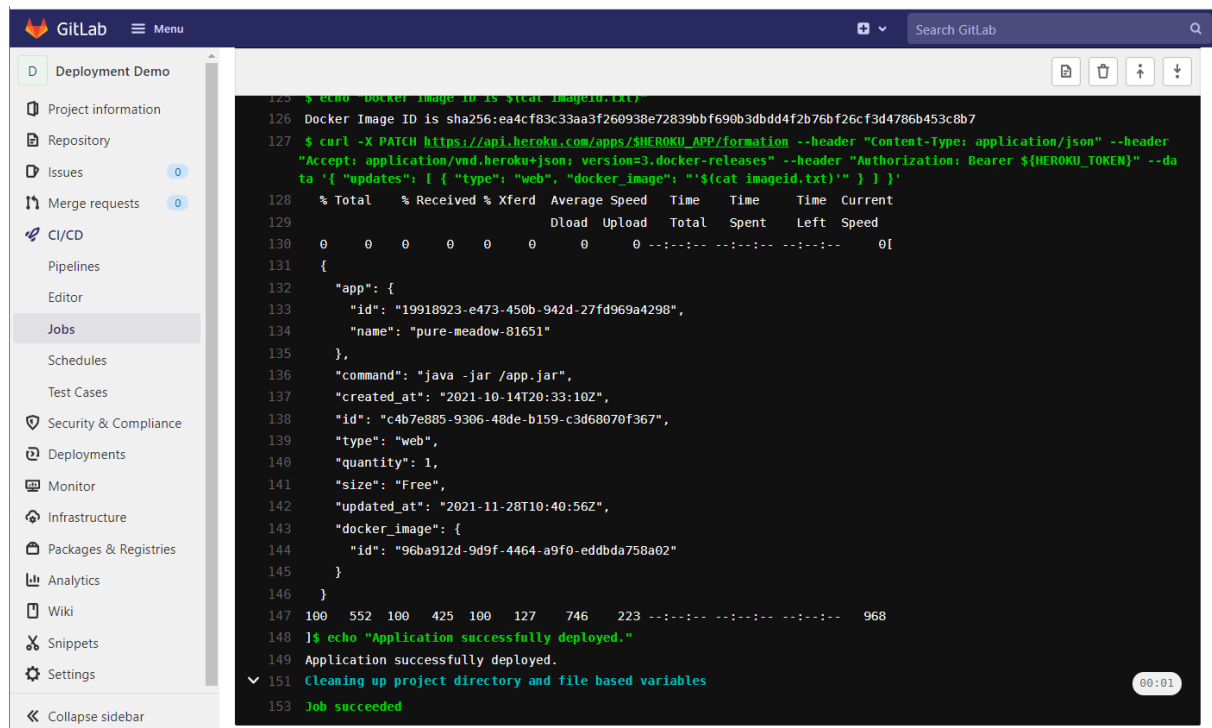
The script

- sets the properties
- builds the jar file
- builds a Docker image with the jar file
- deploys the image to Heroku

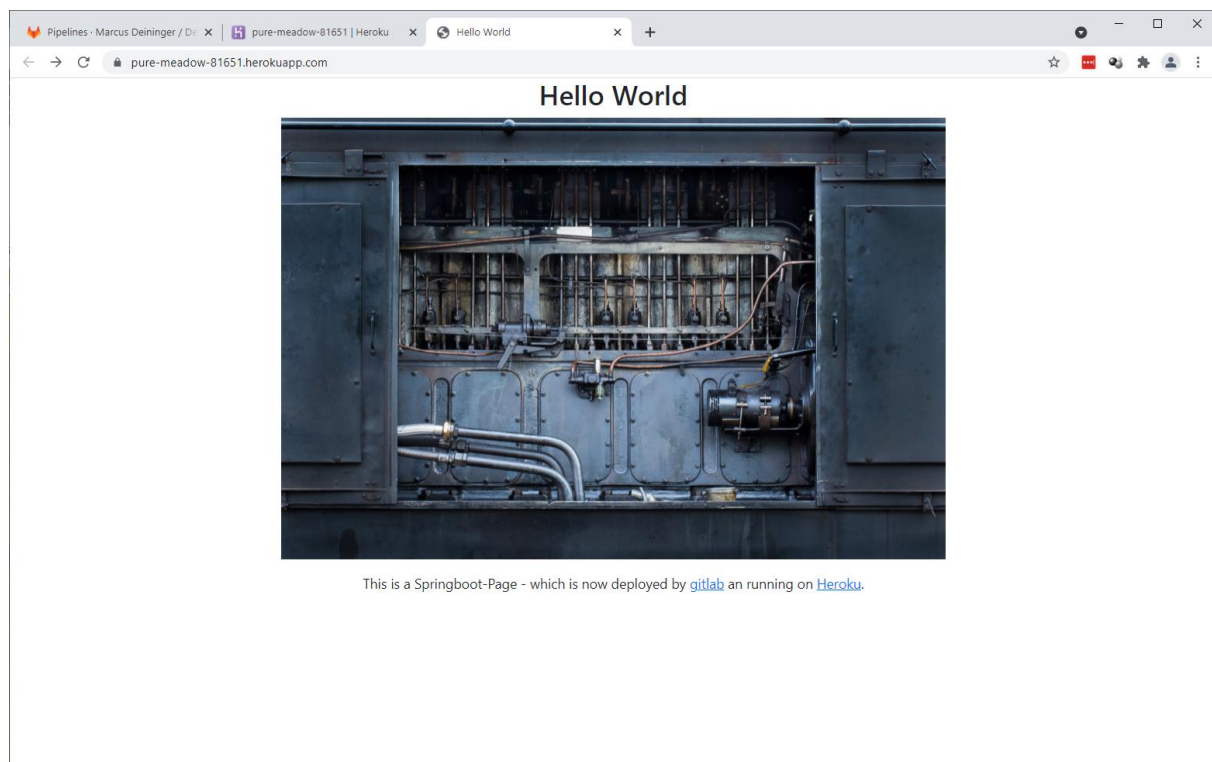


The screenshot shows the GitLab interface with a sidebar on the left containing navigation links like 'Project information', 'Repository', 'Issues', 'Merge requests', 'CI/CD', 'Pipelines', 'Editor', 'Jobs', 'Schedules', 'Test Cases', 'Security & Compliance', 'Deployments', 'Monitor', 'Infrastructure', 'Packages & Registries', 'Analytics', 'Wiki', 'Snippets', and 'Settings'. The main area displays a log for a deployment job. The log content is as follows:

```
939 $ ls -als target
940 total 22136
941 4 drwxr-xr-x 6 root root 4096 Nov 28 10:39 .
942 4 drwxrwxrwx 6 root root 4096 Nov 28 10:38 ..
943 21684 -rw-r--r-- 1 root root 22202701 Nov 28 10:39 06.deployment.sample.local-0.0.1-SNAPSHOT.jar
944 428 -rw-r--r-- 1 root root 435863 Nov 28 10:39 06.deployment.sample.local-0.0.1-SNAPSHOT.jar.original
945 4 drwxr-xr-x 5 root root 4096 Nov 28 10:39 classes
946 4 drwxr-xr-x 3 root root 4096 Nov 28 10:39 generated-sources
947 4 drwxr-xr-x 2 root root 4096 Nov 28 10:39 maven-archiver
948 4 drwxr-xr-x 3 root root 4096 Nov 28 10:39 maven-status
949 $ cd ..
950 $ echo "Copying artifacts"
951 Copying artifacts
952 $ mkdir build
953 $ cp $PROJECT/target/*.jar build/
954 $ cp $PROJECT/docker/Dockerfile build/
955 $ ls -als build
956 total 21696
957 4 drwxr-xr-x 2 root root 4096 Nov 28 10:39 .
958 4 drwxrwxrwx 5 root root 4096 Nov 28 10:39 ..
959 21684 -rw-r--r-- 1 root root 22202701 Nov 28 10:39 06.deployment.sample.local-0.0.1-SNAPSHOT.jar
960 4 -rw-r--r-- 1 root root 392 Nov 28 10:39 Dockerfile
961 ✓ Uploading artifacts for successful job 00:05
962 Uploading artifacts...
963 build/*: found 2 matching files and directories
964 Uploading artifacts as "archive" to coordinator... ok id=29145 responseStatus=201 Created token=WftcNt92
965 ✓ Cleaning up project directory and file based variables 00:01
966 Job succeeded
```



Calling the page: <https://pure-meadow-81651.herokuapp.com/> (or under the name you have assigned)



7.7 Change in the Eclipse project

Now change something in the Eclipse project (e.g. the displayed image).

- Push the changes
- Then perform the merge in gitlab → this will result in automatic deployment
- Refresh the displayed web page

8 Cloud deployment of the application

In the last section, the entire application is now delivered and installed in the Heroku cloud. You can either deliver and run your own application (the author management) or the full application.

The first sections describe the adaptations for the author management (based on 04.0.bookapp.authors), the following ones the remaining adaptations for 05.0.bookapp.

Basically, the following steps are performed:

- Preparation of the project in Eclipse
- Setting up the database on the Heroku cloud
- Setting up the app nodes for client and server on the Heroku cloud
- Preparation of the build pipeline in gitlab
- Push of the Eclipse project to gitlab and delivery to Heroku

8.1 Preparation of the project in Eclipse

First, the project is copied under a new name and then extended with . gitignore and Dockerfile.

8.1.1 Copy from 04.0.bookapp.authors to 07.0.bookapp.authors

Copying a multi-module project is problematic because Eclipse will not display the modules after a simple copy. The following procedure works:

- First, run a maven clean for 04.0.bookapp.authors.
- Go to the system (Windows Explorer) navigate to your workspace and copy the project 04.0.bookapp.authors outside the workspace - we will import it again later.
- Delete the projects 04.1. ... - 04.6. ... **without** "Delete Project content from disk".
- Change the names of the contained modules from 04. ... to 07. ... (with F2).
- Open the pom.xml of each project and search and replace (Ctrl+F) the string 04. with 07. Start with the main project and save the pom immediately after the change.
- Reload the contained projects using Configure → Configure and detect nested projects ...
- After that the contained projects appear under the original (now wrong) name - change the names of the projects (with F2) to the new name.

Now load the original project 04.0.bookapp.authors back into your workspace (make sure it is copied) and load the included projects again with Configure and detect nested projects ...

Alternatively, you can use my solution from Moodle.

8.1.2 Creating the dependencies for Postgres

Enter the following dependency in the pom of the server:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
</dependency>
```

8.1.3 Creating . gitignore and Dockerfile

In the main project, create the file . gitignore with the following content

```
target/
*.class
*.jar
*.war
*.ear
*.logs
*.iml
.idea/
.eclipse
*.db
*.properties
```

Create a folder "docker" in each of the subprojects 07.3.bookapp.inventory.server and 07.6.bookapp.web.client.server and there create the file "Dockerfile" (without extension) with the following content (as before):

```
# Java Image
FROM openjdk:11-jre-slim-buster
# copy JAR into image
COPY *.jar /app.jar

# Added for Heroku, see: https://nielddw.medium.com/use-gitlab-ci-to-
deploy-docker-images-to-heroku-4e544a3a3c38
# Add a new, random user "john" with user id 8877
RUN useradd -u 8877 john
# Change to non-root privilege
USER john

# run application with this command line
CMD ["java", "-jar", "/app.jar"]
```

Create the file README with any content, e.g.

```
Do not remove this file! This file is needed to force
the creation of the resource directory in gitlab, as
due to gitlab policy an empty folder is not possible.
```

And copy the file to the server's resource folders (main and test!), common-client and web-client. This ensures that the directory is actually created in gitlab.

8.2 Setting up the Heroku database and apps

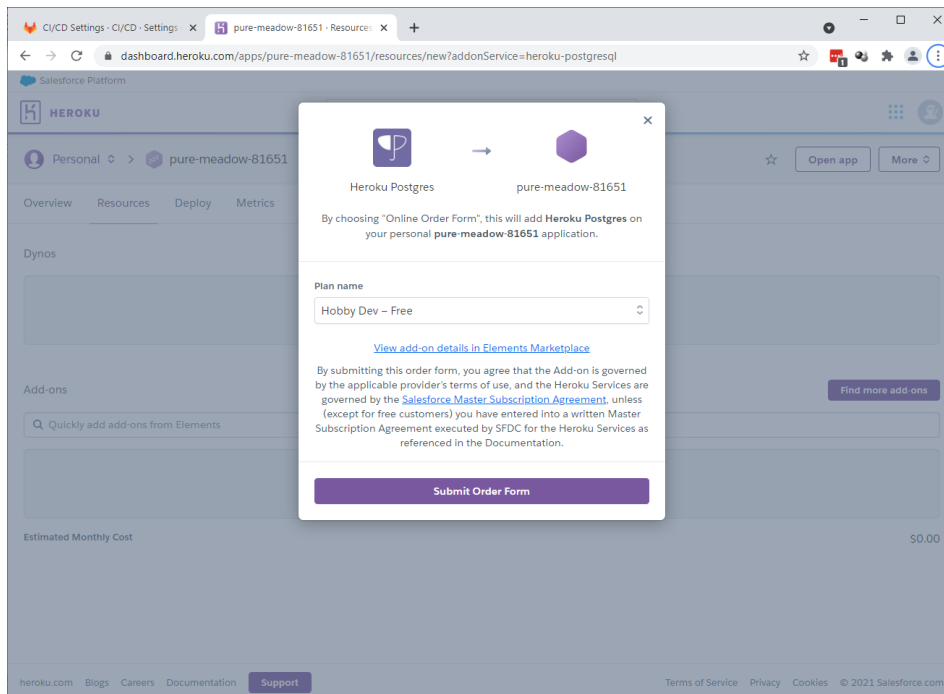
The inventory database is now created in Heroku. A particular problem here is that the database cannot be accessed via the university, since the port of the database is not open. We

therefore use "PlayWithDocker", which runs outside the university firewall and can therefore access the database.

8.2.1 Create database

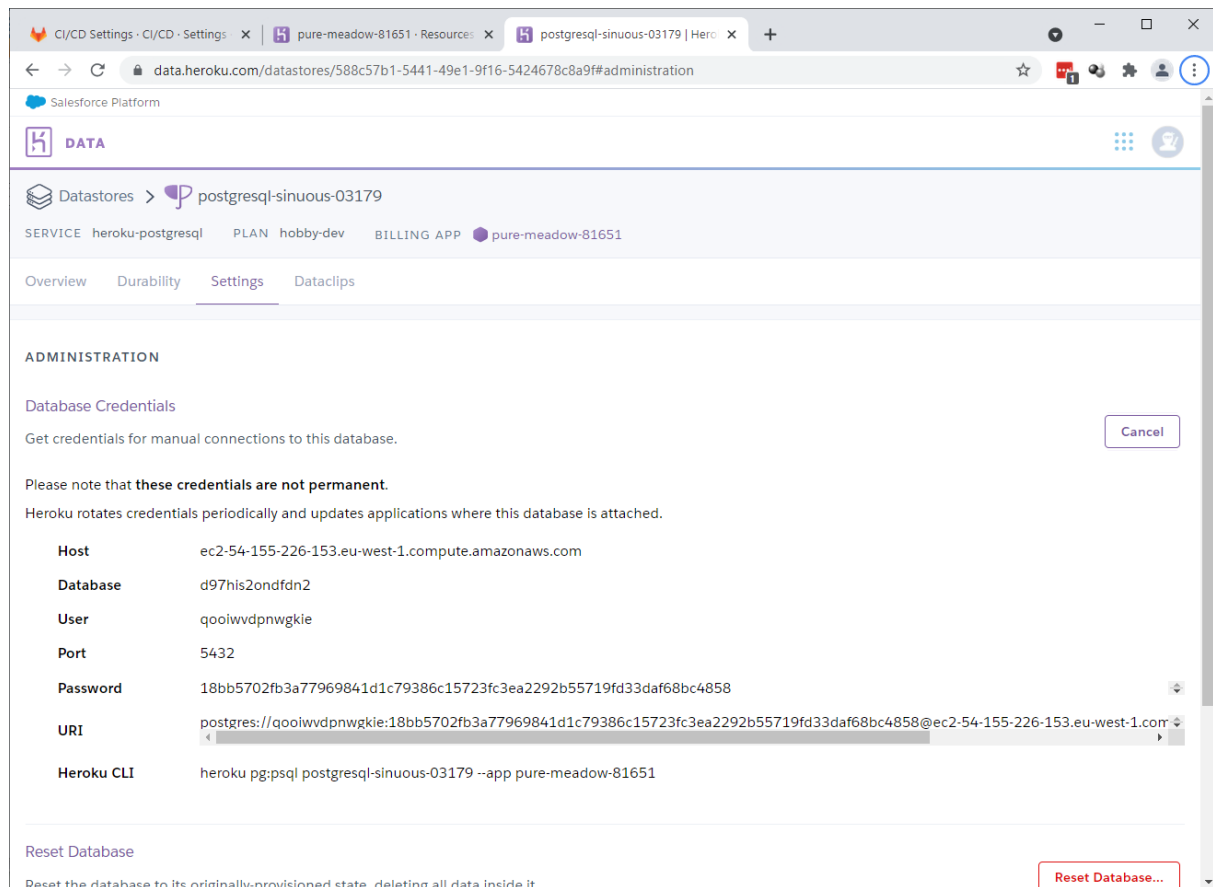
(see also: <https://dev.to/prisma/how-to-setup-a-free-postgresql-database-on-heroku-1dc1>.)

Create app, recommendation bookapp-sales-db-1, bookapp-inventory-db-1 (or a higher number which is free)



The screenshot shows the Heroku dashboard for an application named 'pure-meadow-81651'. The browser tabs at the top include 'CI/CD Settings', 'pure-meadow-81651 - Resource', and 'postgresql-sinuous-03179 | Hero'. The address bar shows the URL 'dashboard.heroku.com/apps/pure-meadow-81651/resources?justInstalledAddonServiceId=6c67493d-8fc2-4cd4-9161-4f1ec11cbe69'. The Heroku logo and a search bar are at the top. Below the navigation bar, the 'Resources' tab is selected. A message states: 'This app has no process types yet. Add a Profile to your app in order to define its process types. [Learn more](#)'. Under the 'Add-ons' section, a notification says: 'The add-on `heroku-postgresql` has been installed. Check out the documentation in its [Dev Center](#) article to get started.' Below this is a search bar for add-ons. A card for 'Heroku Postgres' is shown, indicating it is 'Attached as DATABASE', with a 'Hobby Dev' plan and 'Free' tier. The 'Estimated Monthly Cost' is listed as '\$0.00'. The footer contains links for 'heroku.com', 'Blogs', 'Careers', 'Documentation', 'Support', 'Terms of Service', 'Privacy', 'Cookies', and '© 2021 Salesforce.com'.

Follow link "Heroku Postgres"
→ Settings → View Credentials



8.2.2 Filling the database

Via the Postgres pgAdmin tool - tedious and not recommended

About DatabaseDevelopment-Perspective in Eclipse:

- Postgres jdbc driver store in db-libs
- Connect for Postgres-DB, use the entries above as connection information
- Create tables and data as for SQLite (only difference: different generation of the Ids, see section 6.7)

About PlayWithDocker <https://labs.play-with-docker.com/> (Use ctrl + insert to copy and shift + insert to paste)

Datenverzeichnis anlegen

```
mkdir /root/data/
```

```
cd /root/data/
```

per drag & drop die csv-Dateien hochladen

Dann das Docker-Postgres-Image holen und danach (unter dem Namen dev-postgres starten)

```
docker pull postgres
```

```
docker run -d --name dev-postgres -e POSTGRES_PASSWORD=admin -v /root/data:/var/lib/postgresql/data -p 5432:5432 postgres
```

```
docker ps
```

Den Container starten und mit der bash-Shell betreten.

```
docker exec -it dev-postgres bash
```

(Im Container) Datenbank verbinden und den SQL-Interpreter starten. Remote Host, Port – üblicherweise 5432 – Username und DB-Name kann man der Heroku-Seite entnehmen. Die Option „-W“ fragt nach dem Passwort (das ebenfalls auf der Heroku-Seite zu finden ist)

```
psql -h <REMOTE HOST> -p <REMOTE PORT> -U <DB_USER> <DB_NAME> -W
```

Im Interpreter kann mit `\! <linux cmd>` eine Anweisung außerhalb des Interpreters (also im Docker-Container) ausgeführt werden.

Die Tabellen werden nun mit den Create-Table-Anweisungen wie zuvor erstellt, also

```
create table AUTHORS( ... );  
create table GENRES( ... );  
create table BOOKS( ... );
```

Anzeige aller Tabellen

```
\dt
```

Zum Import der Daten muss zunächst die Codierung (von UTF-8) umgestellt werden

```
set client_encoding to 'win-1252';  
show client_encoding;
```

Danach kann der eigentliche Import durchgeführt werden. Es ist empfohlen, die Spalten explizit zu benennen. Hinweis: es gibt auch einen copy-Befehl, der sich von dem `\copy`-Befehl hier unterscheidet.

```
\copy authors(id, firstname, lastname)  
from '/var/lib/postgresql/data/Authors.csv'  
delimiter ';'   
csv header;
```

```
\copy genres(id, genre_name)  
from '/var/lib/postgresql/data/Genres.csv'  
delimiter ';'   
csv header;
```

```
\copy books(id,title, pub_year, author_id, genre_id, sample)  
from '/var/lib/postgresql/data/Books.csv'  
delimiter ';'   
csv header;
```

Danach auf Heroku (mit Dataclips) die Daten überprüfen, Container wieder herunterfahren und PlayWithDocker verlassen.

8.2.3 Create applications

Create the following three applications in Heroku (or a higher number that is free)

- bookapp-inventory-server-1
- bookapp-sales-server-1
- bookapp-web-client-1

8.3 Create Gitlab project

- In gitlab, create a bookapp-authors project in the group created in Section 32able to use the shared runner.
- Create a development branch as before and disable delete-on-merge.
- Create variables (also in Moodle: 07.ci-cd-setup.zip)

Heroku Access and Apps

| Key | Type | Value | Comment |
|-----------------------------|-------|----------------------------|-----------------------------------|
| HEROKU_TOKEN | Value | API-Key | Api key from the account settings |
| HEROKU_INVENTORY_SERVER_APP | Value | bookapp-inventory-server-1 | or another freely selected number |
| HEROKU_WEB_CLIENT_APP | Value | bookapp-web-client-1 | |

Project (settings for the Author project) x

| Key | Type | Value | Comment |
|--------------------------|-------|-------------------------------|---------|
| API_PATH | Value | api/ v2 | |
| PROJECT | Value | 07.0.bookapp.authors | |
| PROJECT_INVENTORY_SERVER | Value | 07.3.bookapp.inventory.server | |
| PROJECT_COMMON_CLIENT | Value | 07.4.bookapp.common.client | |
| PROJECT_WEB_CLIENT | Value | 07.6.bookapp.web.client | |

Properties

| Key | Type | Value | Target project | Target file |
|-------------------------------|------|--|-------------------------------|---------------------------------------|
| PROPERTIES_APPLICATION_SERVER | File | server.port=\${PORT:5000} server.servlet.context-path=/\${API_PATH} | Same for both server projects | main/resources/application.properties |
| PROPERTIES_APPLICATION_CLIENT | File | server.port=\${PORT:5000} | Web client | main/resources/application.properties |
| PROPERTIES_SERVER_APIS | File | inventory =https://\${HEROKU_INVENTORY_SERVER_APP}.herokuapp.com/\${API_PATH} | Client Common | main/resources/server.properties |

| | | | | |
|------------------------------|------|--|------------------|------------------------------|
| PROPERTIES_INVENTORY_DB | File | dburl=jdbc:postgresql:// <i>connection</i> :5432/ <i>database</i> ?user= <i>user</i> &password= <i>password</i> ⁵ | Inventory Server | main/resources/db.properties |
| PROPERTIES_INVENTORY_TEST_DB | File | dburl=jdbc:h2:mem:account;DB_CLOSE_DELAY=-1 | Inventory Server | test/resources/db.properties |

⁵ Use the settings of your Heroku database: e.g.: dburl=jdbc:postgresql://ec2-54-155-226-153.*eu-west-1.compute.amazonaws.com*:5432/*d97his2ondfdn2*?user=*qooiwvdpnwgie*&password=*18bb5702fb3a77969841d1c79386c15723fc3ea2292b55719fd33daf68bc4858*

Create CI-CD pipeline (also in Moodle: 07.ci-cd-setup.zip)

```
image: maven:3.8.1-jdk-11

stages:
  - build
  - deploy

build-job:
  stage: build
  script:
    - echo "Injecting server properties"
    - cd $PROJECT/$PROJECT_INVENTORY_SERVER/src/main/resources/
    - cp $PROPERTIES_APPLICATION_SERVER application.properties
    - cp $PROPERTIES_INVENTORY_DB db.properties
    - ls -als
    - cat application.properties

    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT/$PROJECT_INVENTORY_SERVER/src/test/resources/
    - cp $PROPERTIES_INVENTORY_TEST_DB db.properties
    - ls -als

    - echo "Injecting client properties"
    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT/$PROJECT_COMMON_CLIENT/src/main/resources/
    - cp $PROPERTIES_SERVER_APIS server.properties
    - ls -als

    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT/$PROJECT_WEB_CLIENT/src/main/resources/
    - cp $PROPERTIES_APPLICATION_CLIENT application.properties
    - ls -als

    - echo "Building the project"
    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT
    - mvn package
    - cd ..

    - echo "Copying artifacts"
    - mkdir build.invserver
    - cp $PROJECT/$PROJECT_INVENTORY_SERVER/target/*.jar build.invserver
    - cp $PROJECT/$PROJECT_INVENTORY_SERVER/docker/Dockerfile build.invserver
    - ls -als build.invserver
```



```

- mkdir build.client
- cp $PROJECT/$PROJECT_WEB_CLIENT/target/*fixed.jar build.client
- cp $PROJECT/$PROJECT_WEB_CLIENT/docker/Dockerfile build.client
- ls -als build.client

artifacts:
  paths:
    - build.invserver/*
    - build.client/*

containerize-server:
  image: docker:latest
  stage: deploy
  services:
    - docker:dind
  script:
    - docker --version
    - cd build.invserver
    - ls -als

    - echo "Deploying inventory server"
    ## heroku deployment
    - docker build --iidfile imageid.txt -
t registry.heroku.com/$HEROKU_INVENTORY_SERVER_APP/my-app .
    - docker login -u _ -p $HEROKU_TOKEN registry.heroku.com
    - docker push registry.heroku.com/$HEROKU_INVENTORY_SERVER_APP/my-app
    - apk add --no-cache curl
    - echo "Docker Image ID is $(cat imageid.txt)"
    - |-
      curl -
X PATCH https://api.heroku.com/apps/$HEROKU_INVENTORY_SERVER_APP/formation --
header "Content-Type: application/json" --
header "Accept: application/vnd.heroku+json; version=3.docker-releases" --
header "Authorization: Bearer ${HEROKU_TOKEN}" --
data '{ "updates": [ { "type": "web", "docker_image": "'$(cat imageid.txt)'" }
] }'

    - echo "Inventory server successfully deployed."

# rules:
#   - changes:
#     - $PROJECT/$PROJECT_INVENTORY_SERVER/**/*

containerize-client:
  image: docker:latest

```

```

stage: deploy
services:
  - docker:dind
script:
  - docker --version
  - cd build.client
  - ls -als

  - echo "Deploying web client"
  ## heroku deployment
  - docker build --iidfile imageid.txt -
t registry.heroku.com/$HEROKU_WEB_CLIENT_APP/my-app .
  - docker login -u _ -p $HEROKU_TOKEN registry.heroku.com
  - docker push registry.heroku.com/$HEROKU_WEB_CLIENT_APP/my-app
  - apk add --no-cache curl
  - echo "Docker Image ID is $(cat imageid.txt)"
  - |
    curl -
X PATCH https://api.heroku.com/apps/$HEROKU_WEB_CLIENT_APP/formation --
header "Content-Type: application/json" --
header "Accept: application/vnd.heroku+json; version=3.docker-releases" --
header "Authorization: Bearer ${HEROKU_TOKEN}" --
data '{ "updates": [ { "type": "web", "docker_image": "'$(cat imageid.txt)'" }
] }'

  - echo "Web client successfully deployed."

# rules:
#   - changes:
#     - $PROJECT/$PROJECT_WEB_CLIENT/**/*

```

8.4 Push Maven project from Eclipse to gitlab

As before: clone bookapp-authors locally and push the project 07.bookapp.authors to gitlab. After the merge request, the pipeline builds the project and ships it. The server is located at <https://bookapp-inventory-server-....herokuapp.com/api/v2/>, the client at <https://bookapp-web-client-....herokuapp.com>

8.5 Complete project

The complete project 08.bookshop is already available in Moodle as a multi-module project with . gitignore, READMEs and the Dockerfiles. Import the project into your workspace.

Create a second Heroku database bookapp-sales-db and create the tables for Customer and Order. Then create another app in Heroku bookapp-sales-server-1.

Create a new project bookapp-complete-1 in git and create the following variables:

Heroku Access and Apps

| Key | Type | Value | Comment |
|-----------------------------|-------|----------------------------|-----------------------------------|
| HEROKU_TOKEN | Value | API-Key | Api key from the account settings |
| HEROKU_INVENTORY_SERVER_APP | Value | bookapp-inventory-server-1 | or another freely selected number |
| HEROKU_SALES_SERVER_APP | Value | bookapp-sales-server-1 | Not used here |
| HEROKU_WEB_CLIENT_APP | Value | bookapp-web-client-1 | |

Project (settings for the complete project)

| Key | Type | Value | Comment |
|--------------------------|-------|-------------------------------|---------|
| API_PATH | Value | api/ v3 | |
| PROJECT | Value | 08.0.bookapp | |
| PROJECT_INVENTORY_SERVER | Value | 08.3.bookapp.inventory.server | |
| PROJECT_SALES_SERVER | Value | 08.4.bookapp.sales.server | |
| PROJECT_COMMON_CLIENT | Value | 08.5.bookapp.common.client | |
| PROJECT_WEB_CLIENT | Value | 08.7.bookapp.web.client | |

Properties

| Key | Type | Value | Target project | Target file |
|-------------------------------|------|--|-------------------------------|---------------------------------------|
| PROPERTIES_APPLICATION_SERVER | File | server.port=\${PORT:5000} server.servlet.context-path=/\${API_PATH} | Same for both server projects | main/resources/application.properties |
| PROPERTIES_APPLICATION_CLIENT | File | server.port=\${PORT:5000} | Web client | main/resources/application.properties |
| PROPERTIES_SERVER_APIS | File | inventory =https://\${HEROKU_INVENTORY_SERVER_APP}.herokuapp.com/\${API_PATH} | Client Common | main/resources/server.properties |

| | | | | |
|------------------------------|------|--|---------------------------|--------------------------------|
| | | <code>sales=https://\${HEROKU_SALES_SERVER_APP}.herokuapp.com/\${API_PATH}</code> | | |
| PROPERTIES_INVENTORY_DB | File | <code>dburl=jdbc:postgresql://<i>connection</i>:5432/<i>database</i>?user=<i>user</i>&password=<i>password</i></code> ⁶ | Inventory Server | main/resources/db.properties |
| PROPERTIES_SALES_DB | File | As above | Sales Server | main/resources/db.properties |
| PROPERTIES_INVENTORY_TEST_DB | File | <code>dburl=jdbc:h2:mem:account;DB_CLOSE_DELAY=-1</code> | Inventory Server | test/resources/db.properties |
| PROPERTIES_MAIL | File | <code>apikey=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 sendername=John Bookman sendermail=john.bookman@bookshop.com</code> | Sales Server ⁷ | main/resources/mail.properties |

⁶ Use the settings of your Heroku database: e.g.: `dburl=jdbc:postgresql://ec2-54-155-226-153.eu-west-1.compute.amazonaws.com:5432/d97his2ondfdn2?user=qooiwdpnwgkie&password=18bb5702fb3a77969841d1c79386c15723fc3ea2292b55719fd33daf68bc4858`

⁷ The API-KEY of sendinblue is personalized to me and may change after the exercise. But you can request your own free account

Now create the CI/CD pipeline (also included in the repository):

```
image: maven:3.8.1-jdk-11

stages:
  - build
  - deploy

build-job:
  stage: build
  script:
    - echo "Injecting server properties"
    - cd $PROJECT/$PROJECT_INVENTORY_SERVER/src/main/resources/
    - cp $PROPERTIES_APPLICATION_SERVER application.properties
    - cp $PROPERTIES_INVENTORY_DB db.properties
    - ls -als

    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT/$PROJECT_INVENTORY_SERVER/src/test/resources/
    - cp $PROPERTIES_INVENTORY_TEST_DB db.properties
    - ls -als

    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT/$PROJECT_SALES_SERVER/src/main/resources/
    - cp $PROPERTIES_APPLICATION_SERVER application.properties
    - cp $PROPERTIES_SALES_DB db.properties
    - cp $PROPERTIES_MAIL mail.properties
    - ls -als

    - echo "Injecting client properties"
    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT/$PROJECT_COMMON_CLIENT/src/main/resources/
    - cp $PROPERTIES_SERVER_APIS server.properties
    - ls -als

    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT/$PROJECT_WEB_CLIENT/src/main/resources/
    - cp $PROPERTIES_APPLICATION_CLIENT application.properties
    - ls -als

    - echo "Building the project"
    - cd $CI_BUILDS_DIR/$CI_PROJECT_PATH
    - cd $PROJECT
    - mvn package
    - cd ..
```

- echo "Copying artifacts"
- mkdir build.invserver
- cp \$PROJECT/\$PROJECT_INVENTORY_SERVER/target/*.jar build.invserver
- cp \$PROJECT/\$PROJECT_INVENTORY_SERVER/docker/Dockerfile build.invserver
- ls -als build.invserver

- mkdir build.saleserver
- cp \$PROJECT/\$PROJECT_SALES_SERVER/target/*.jar build.saleserver
- cp \$PROJECT/\$PROJECT_SALES_SERVER/docker/Dockerfile build.saleserver
- ls -als build.saleserver

- mkdir build.client
- cp \$PROJECT/\$PROJECT_WEB_CLIENT/target/*fixed.jar build.client
- cp \$PROJECT/\$PROJECT_WEB_CLIENT/docker/Dockerfile build.client
- ls -als build.client

artifacts:

paths:

- build.invserver/*
- build.saleserver/*
- build.client/*

containerize-inventory-server:

image: docker:latest

stage: deploy

services:

- docker:dind

script:

- docker --version
- cd build.invserver
- ls -als

- echo "Deploying inventory server"

heroku deployment

- docker build --iidfile imageid.txt -

t registry.heroku.com/\$HEROKU_INVENTORY_SERVER_APP/my-app .

- docker login -u _ -p \$HEROKU_TOKEN registry.heroku.com
- docker push registry.heroku.com/\$HEROKU_INVENTORY_SERVER_APP/my-app
- apk add --no-cache curl
- echo "Docker Image ID is \$(cat imageid.txt)"
- |-
- curl -

X PATCH https://api.heroku.com/apps/\$HEROKU_INVENTORY_SERVER_APP/formation --

header "Content-Type: application/json" --

header "Accept: application/vnd.heroku+json; version=3.docker-releases" --

header "Authorization: Bearer \${HEROKU_TOKEN}" --

```

data '{ "updates": [ { "type": "web", "docker_image": "'$(cat imageid.txt)'" }
] }'

- echo "Inventory server successfully deployed."

# rules:
#   - changes:
#     - $PROJECT/$PROJECT_INVENTORY_SERVER/**/*

containerize-sales-server:
  image: docker:latest
  stage: deploy
  services:
    - docker:dind
  script:
    - docker --version
    - cd build.saleserver
    - ls -als

    - echo "Deploying sales server"
    ## heroku deployment
    - docker build --iidfile imageid.txt -
t registry.heroku.com/$HEROKU_SALES_SERVER_APP/my-app .
    - docker login -u _ -p $HEROKU_TOKEN registry.heroku.com
    - docker push registry.heroku.com/$HEROKU_SALES_SERVER_APP/my-app
    - apk add --no-cache curl
    - echo "Docker Image ID is $(cat imageid.txt)"
    - |
      curl -
X PATCH https://api.heroku.com/apps/$HEROKU_SALES_SERVER_APP/formation --
header "Content-Type: application/json" --
header "Accept: application/vnd.heroku+json; version=3.docker-releases" --
header "Authorization: Bearer ${HEROKU_TOKEN}" --
data '{ "updates": [ { "type": "web", "docker_image": "'$(cat imageid.txt)'" }
] }'

- echo "Sales server successfully deployed."

# rules:
#   - changes:
#     - $PROJECT/$PROJECT_INVENTORY_SERVER/**/*

containerize-client:
  image: docker:latest
  stage: deploy
  services:

```



```

- docker:dind
script:
- docker --version
- cd build.client
- ls -als

- echo "Deploying web client"
## heroku deployment
- docker build --iidfile imageid.txt -
t registry.heroku.com/$HEROKU_WEB_CLIENT_APP/my-app .
- docker login -u _ -p $HEROKU_TOKEN registry.heroku.com
- docker push registry.heroku.com/$HEROKU_WEB_CLIENT_APP/my-app
- apk add --no-cache curl
- echo "Docker Image ID is $(cat imageid.txt)"
- | -
  curl -
X PATCH https://api.heroku.com/apps/$HEROKU_WEB_CLIENT_APP/formation --
header "Content-Type: application/json" --
header "Accept: application/vnd.heroku+json; version=3.docker-releases" --
header "Authorization: Bearer ${HEROKU_TOKEN}" --
data '{ "updates": [ { "type": "web", "docker_image": "'$(cat imageid.txt)'" }
] }'

- echo "Web client successfully deployed."

# rules:
#   - changes:
#     - $PROJECT/$PROJECT_WEB_CLIENT/**/*

```

Now clone the project and push it to gitlab. After the merge request, the pipeline builds the project and ships it. The client is again accessible at [https://bookapp-web-client-.... hero-kuapp.com](https://bookapp-web-client-....hero-kuapp.com).