System Design - Application Workshop

Hochschule
für Technik
Stuttgart

Marcus Deininger

WS 2021/22

Hochschule
für Technik
Stuttgart

System: Initial Vision - 1/3*

Name | Year
bookapp v3, 2021

Task (two or three positive sentences, domain terms)
- The system manages the books in a shipment. The books are organized by authors and genres - Customers can register on a portal and then order books.

The five most important **domain terms**
1. Books
2. authors
3. genres
4. customers
5. orders

* based on: Gernot Starke: Effective Software Architectures, Carl Hanser Verlag, 8th Edition, 2017

System: Initial Vision - 2/3

Select types of use - several 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

Negative stakeholders

- none

user interfaces

Form-based | Object-oriented | Command line | Special input: _____

adaptable to the experience | different user groups

with installer

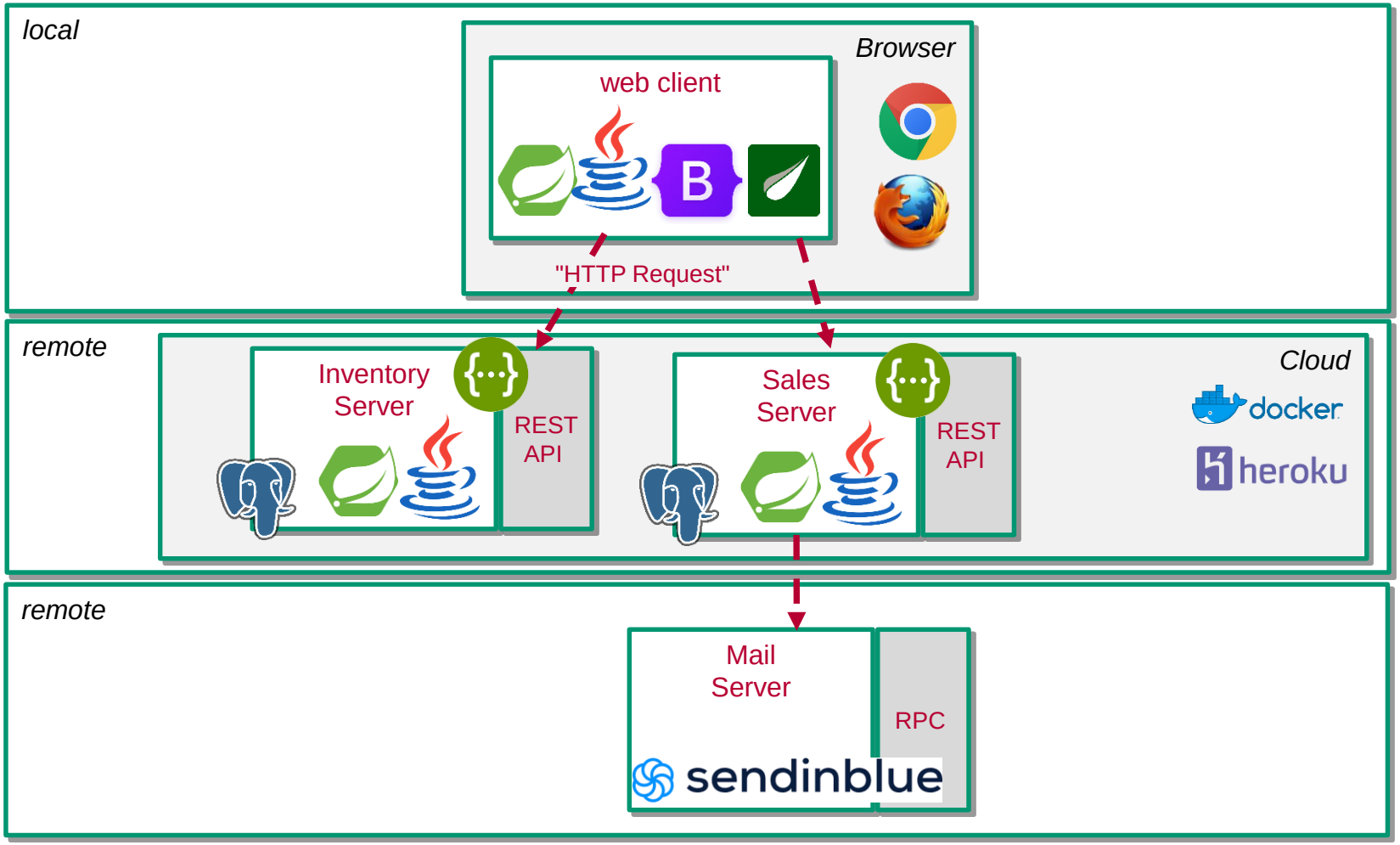
System: Initial Vision - 3/3

System interfaces
To other systems: -
From other systems: via REST API (interpreted as "data interface")
1. in | out functional interface: Mail system (sendinblue) Expected data volume: kb | synchronous | asynchronous | definition available | formally defined | examples available | error tolerance
3. in | out Data interface: Sales system Expected size: kB
4. in | out Data interface: Inventory system Expected size abh. , kB - MB

Data management
Main Memory | Files | Database Management System (DMBS)
Expected size (GB, records, ...) several MB

Control
procedural | event driven / reactive | parallel | rule based

System


local
Browser
web client
"HTTP Request"
remote
Inventory Server
REST API
Sales Server
REST API
Cloud
docker
heroku
remote
Mail Server
RPC
sendinblue

Architecture pattern / development process

- Inventory and sales servers are to be designed as (independent) micro-services
  - Access via REST Api
  - Technology: SpringBoot
- The servers themselves are to be designed as layered architectures
  - Technology: SpringBoot
  - Data access via ORMLite
  - Database: Postgres
- The entire application is to run as a DevOps application in the Heroku cloud
- Deployment takes place via gitlab

Development steps

Part 1: Local application

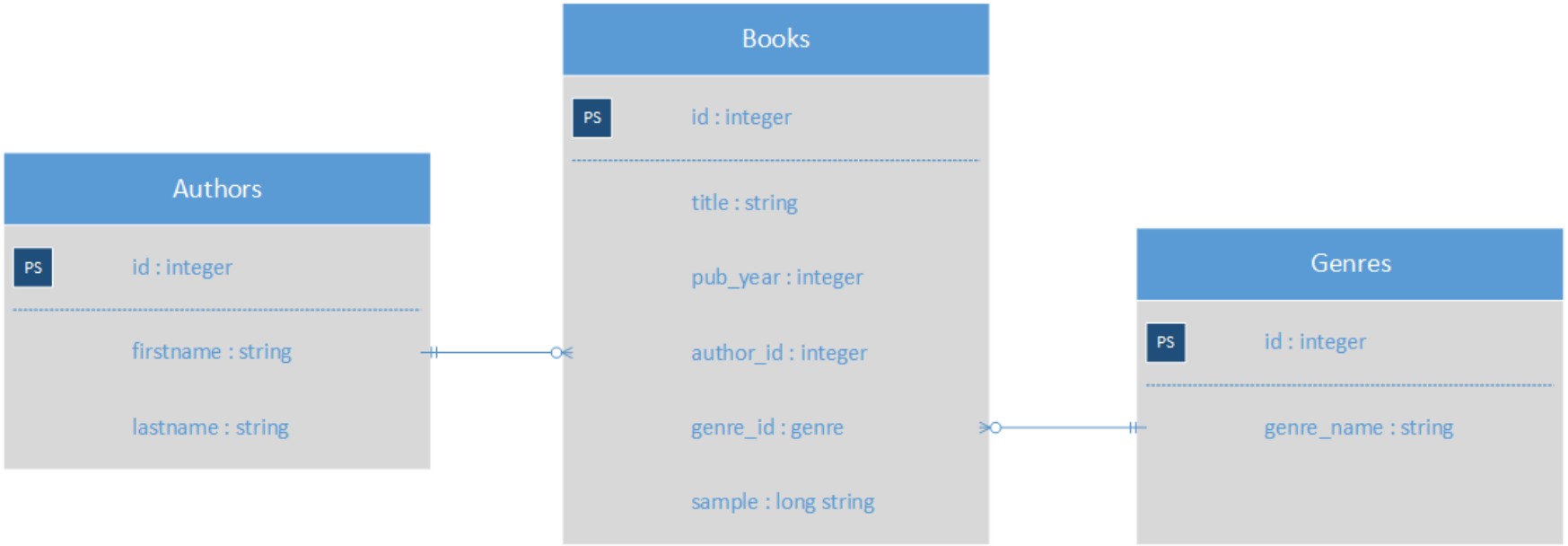
- Setting up the database
- Local client/server application

Part 2: Distributed application

- Installation in the Heroku Cloud

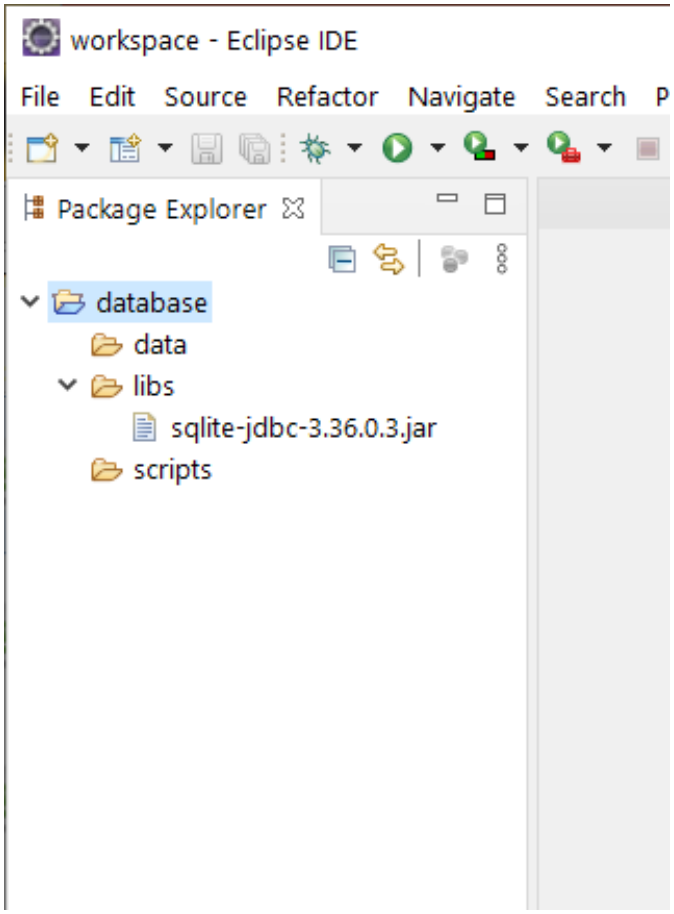
Data model

Data model


Books
PS
id : integer
title : string
pub_year : integer
author_id : integer
genre_id : genre
sample : long string
Authors
PS
id : integer
firstname : string
lastname : string
Genres
PS
id : integer
genre_name : string

SQLite with Eclipse

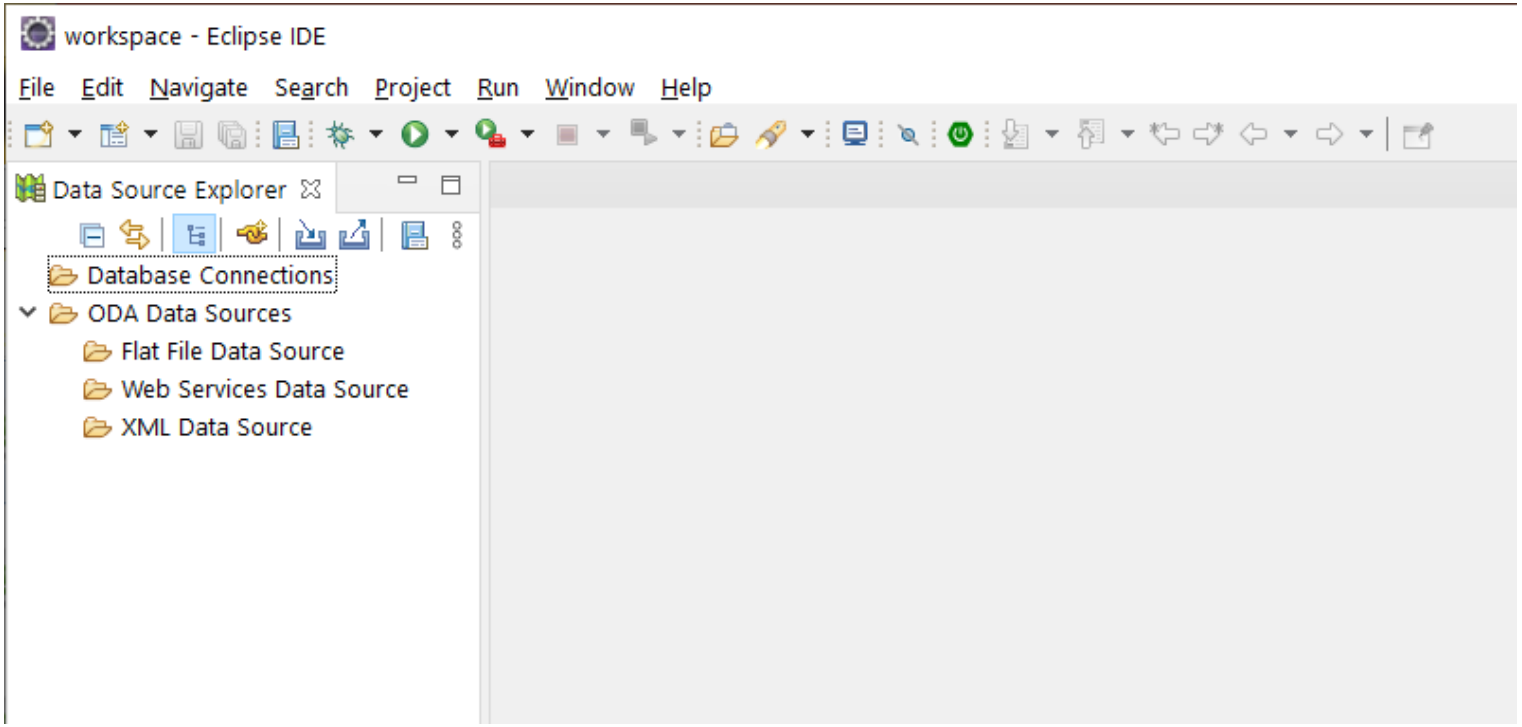
Configuring SQLite in Eclipse - 1



- SQLite is a simple embedded database without a server
- for Java only the JDBC driver is necessary: https://github.com/xerial/sqlite-jdbc/releases / sqlite-jdbc-x.xx.x.x.jar (currently 3.36.0.3)
- Recommendation: Driver in the workspace in the folder `libs`

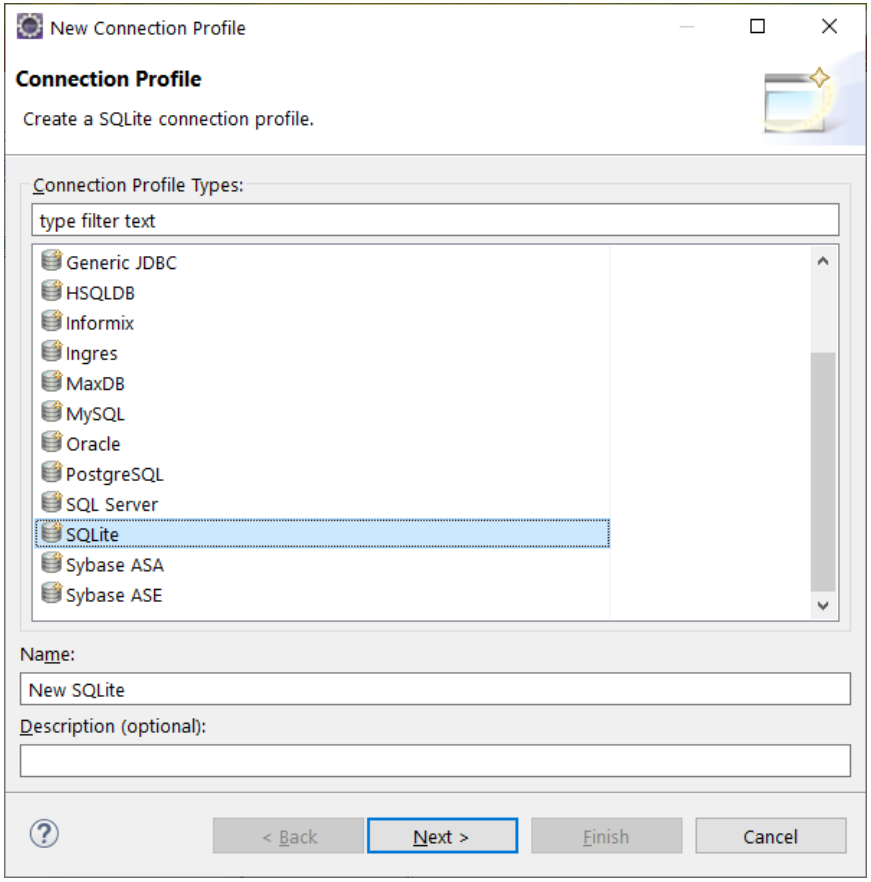
Configuring SQLite in Eclipse - 2

- Eclipse EE IDE for Web Developers*
- Perspective "Database Development



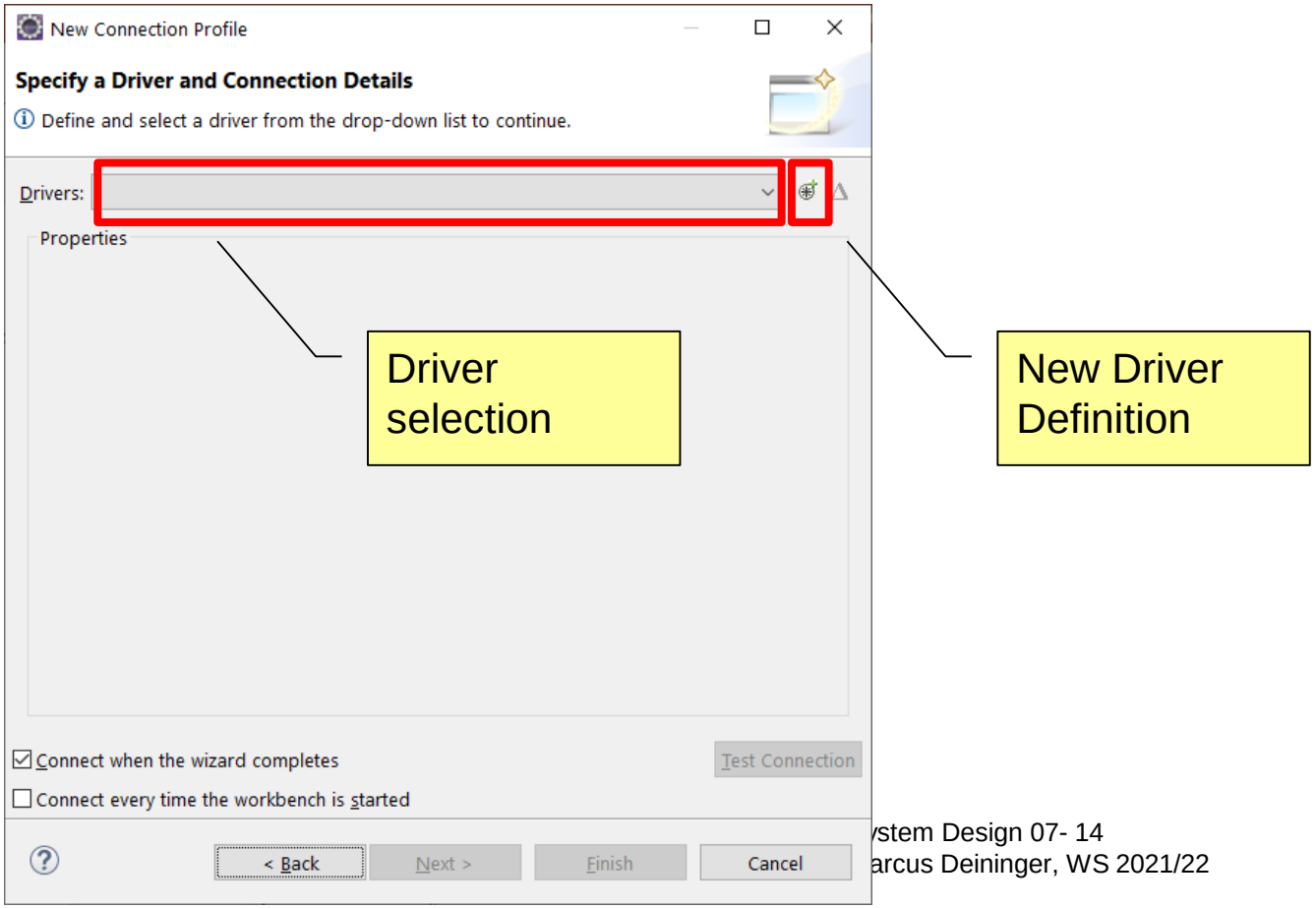
Configuring SQLite in Eclipse - 3

- Database Connections > New... > SQLite



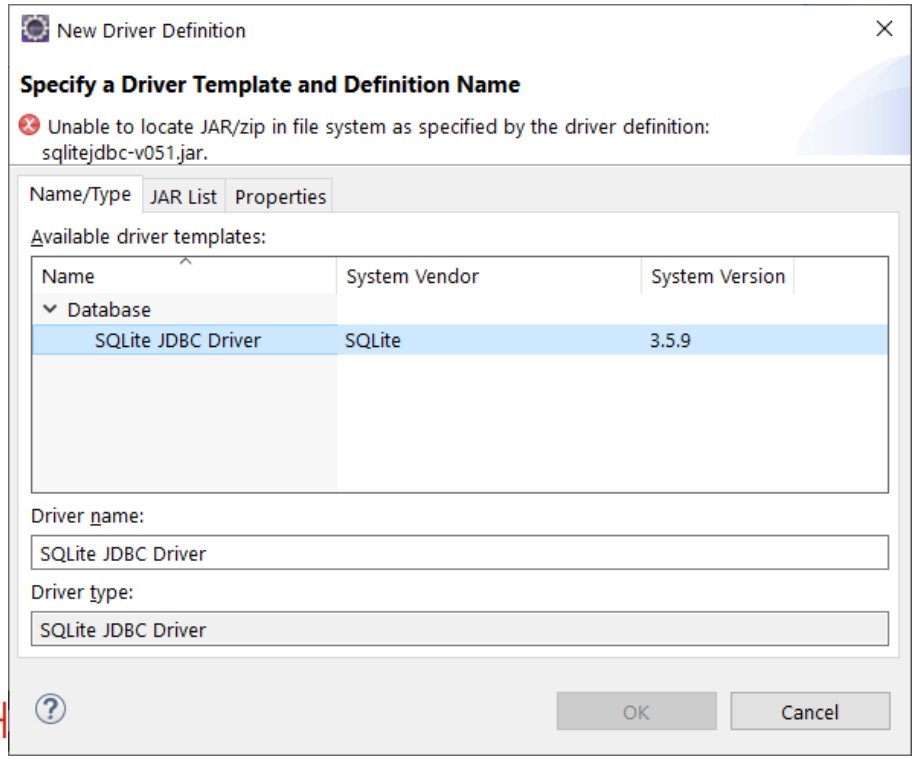
Configuring SQLite in Eclipse - 4

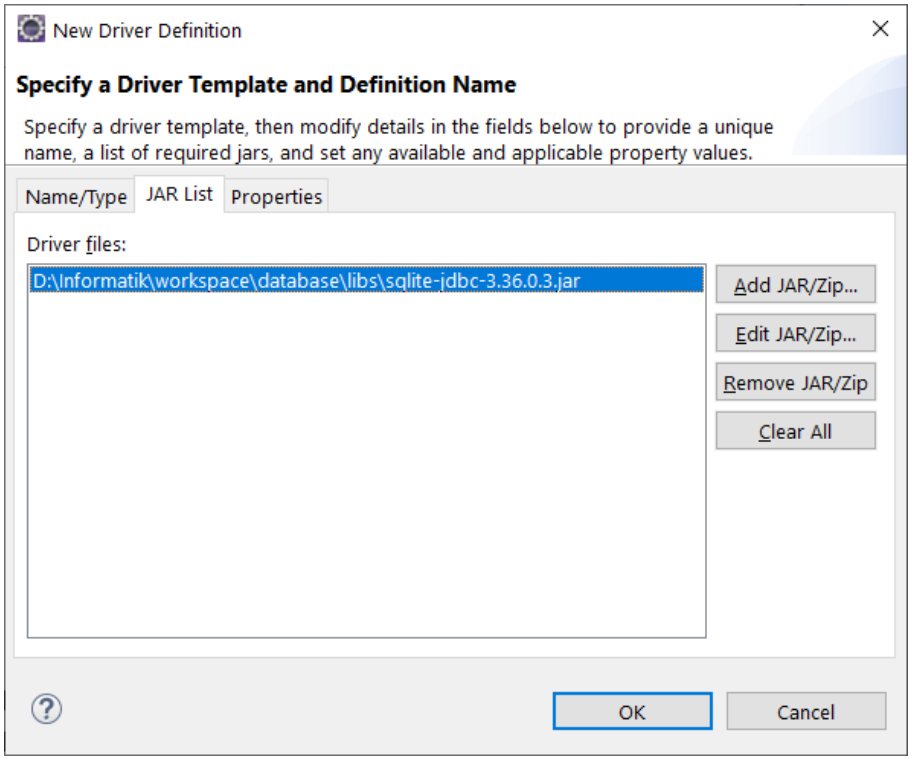
- If no driver is available for selection → New Driver Definition".



Configuring SQLite in Eclipse - 5

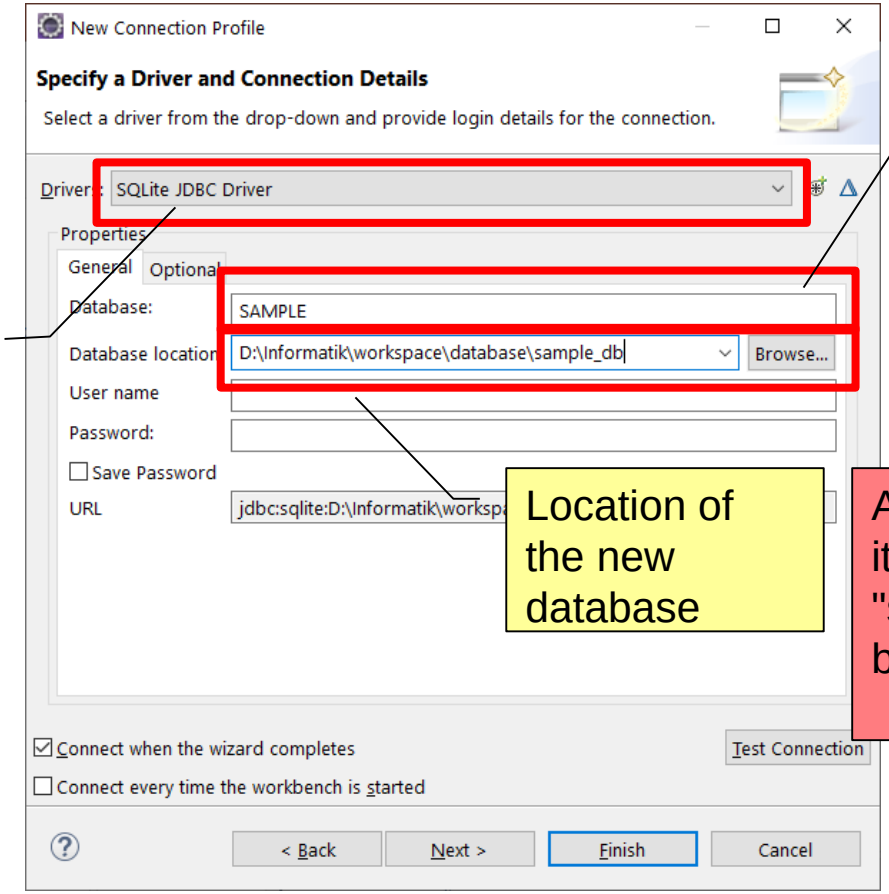
- Name/Type: Select SQLite JDBC Driver
- JAR List: remove previous entry → Add JAR/Zip... → select `sqlite....jar` → OK





Configuring SQLite in Eclipse - 6

- Select driver
- Specify location of database (do not create!)



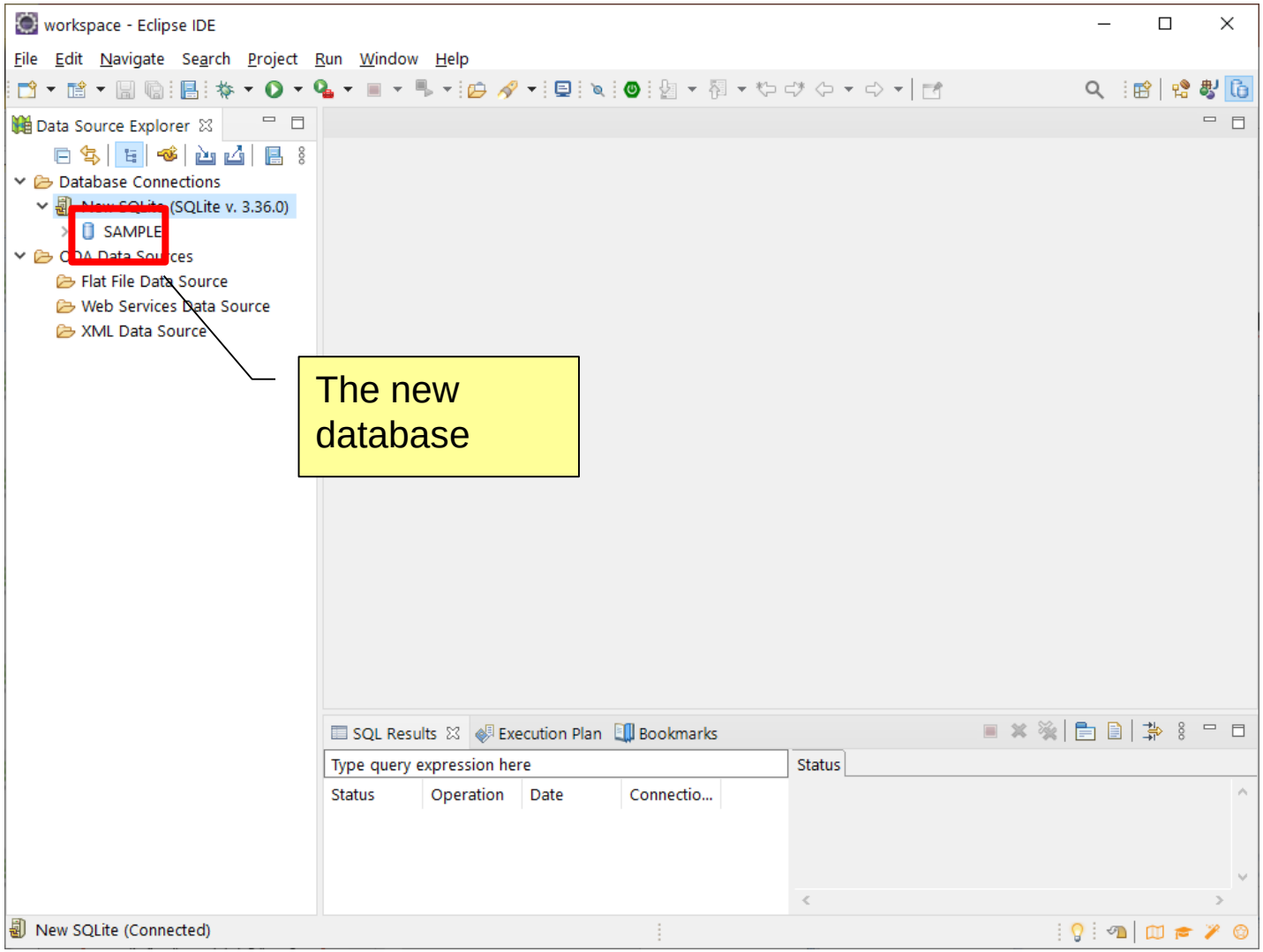
Driver selection

database name

Location of the new database

Attention: The database itself (here the "sample_db") must not be created yet!

Configuring SQLite in Eclipse - 7



Create a table in Eclipse - 1

workspace - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Data Source Explorer

- Database Connections
  - New SQLite (SQLite v. 3
    - SAMPLE
- ODA Data Sources
  - Flat File Data Source
  - Web Services Data Sou
  - XML Data Source

Connect
Disconnect
Ping...
Work Offline
Save Offline
Rename F2
Delete Delete
Duplicate
Refresh F5
Open SQL Scrapbook
Prop
Open scrapbook to edit SQL statements

Open SQL Scrapbook

SQL Results | Execution Plan | Bookmarks

Type query expression here

Status	Operation	Date	Connectio...

Status

New SQLite (Connected)

Hochsch
für Tech
Stuttgar

Create a table in Eclipse - 2

workspace - New SQLite/SQL Scrapbook 0 - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Data Source Explorer

- Database Connections
  - New SQLite (SQLite v. 3.36.0)
    - SAMPLE
- ODA Data Sources
  - Flat File Data Source
  - Web Services Data Source
  - XML Data Source

*SQL Scrapbook 0

Connection profile

Type: SQLITE_3.5.9 Name: New SC Database: SAMPI Status: Connected, Auto Comr

```sql
CREATE TABLE A_TABLE (
  ID INT NOT NULL,
  SOME_FIELD VARCHAR(30),
  ANOTHER_FIELD VARCHAR(30),
  OTHER_ID INT,
  SOME_VALUE INT,
  PRIMARY KEY (ID),
  FOREIGN KEY (OTHER_ID)
  REFERENCES OTHER_TABLE
);
```

SQL command for creating a table

SQL Results | Execution Plan | Bookmarks

Type query expression here

Status	Operation	Date	Connectio...

Status

Database typ... Mode: Auto | Writable | Insert | 1...4

Create a table in Eclipse - 3

workspace - New SQLite/SQL Scrapbook 0 - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Data Source Explorer

- Database Connections
  - New SQLite (SQLite v. 3.36.0)
    - SAMPLE
- ODA Data Sources
  - Flat File Data Source
  - Web Services Data Source
  - XML Data Source

*SQL Scrapbook 0

Connection profile

Type: SQLITE_3.5.9 Name: New SC Database: SAMPI Status: Connected, Auto Comr

```
1  CREATE TABLE A_TABLE (
2    ID INT NOT NULL,
3    SOME_FIELD VARCHAR(30),
4    ANOTHER_FIELD VARCHAR(30),
5    OTHER_ID INT,
6    SOME_VALUE INT,
7    PRIMARY KEY (ID)
8    FORE
9        R
10 );
11
```

Undo Typing	Ctrl+Z
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Toggle Comment	Ctrl+/
Execute All	Ctrl+Alt+X
Execute Selected Text	Alt+X
Execute Selected Text As One Statement	Alt+C
Execute Current Text	Alt+S
Save as Template...	
Edit in SQL Query Builder...	Alt+Q
Preferences...	
Set Connection Info	

SQL Results

Type query expression

Status Operatio

Database typ... Mode: Auto | Writable | Insert | 13...4]

Insert data from a CSV file

workspace - New SQLite/SQL Scrapbook 0 - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Data Source Explorer

- Database Connections
  - New SQLite (SQLite v. 3.36.0)
    - SAMPLE
      - Authorization IDs
      - Schemas
        - DEFAULT
          - Dependencies
          - Stored Procedures
          - Tables
            - A_TABLE
            - OT
          - User-
          - Views
- ODA Data Sources
  - Flat File Data Source
  - Web Services Data Source
  - XML Data Source

Data → Edit, Load..., Extract..., Sample Contents

Generate DDL...

Refresh F5

*SQL Scrapbook 0

Connection profile

Type: SQLITE_3.5.9 Name: New SC Database: SAMPI Status: Connected, Auto Comr

```
6
7  CREATE TABLE A_TABLE (
8    ID INT NOT NULL,
9    SOME_FIELD VARCHAR(30),
10   ANOTHER_FIELD VARCHAR(30),
14   FOREIGN KEY (OTHER_ID)
15      REFERENCES OTHER_TABLE
16 );
17
```

SQL Results | Execution Plan | Bookmarks

Type query expression here

Status	Operation	Date	Connectio...
✓ Succee	CREATE TA...	17.10.2021...	New SQLite
✓ Succee	CREATE TA...	17.10.2021...	New SQLite

Status

(0 rows affected)

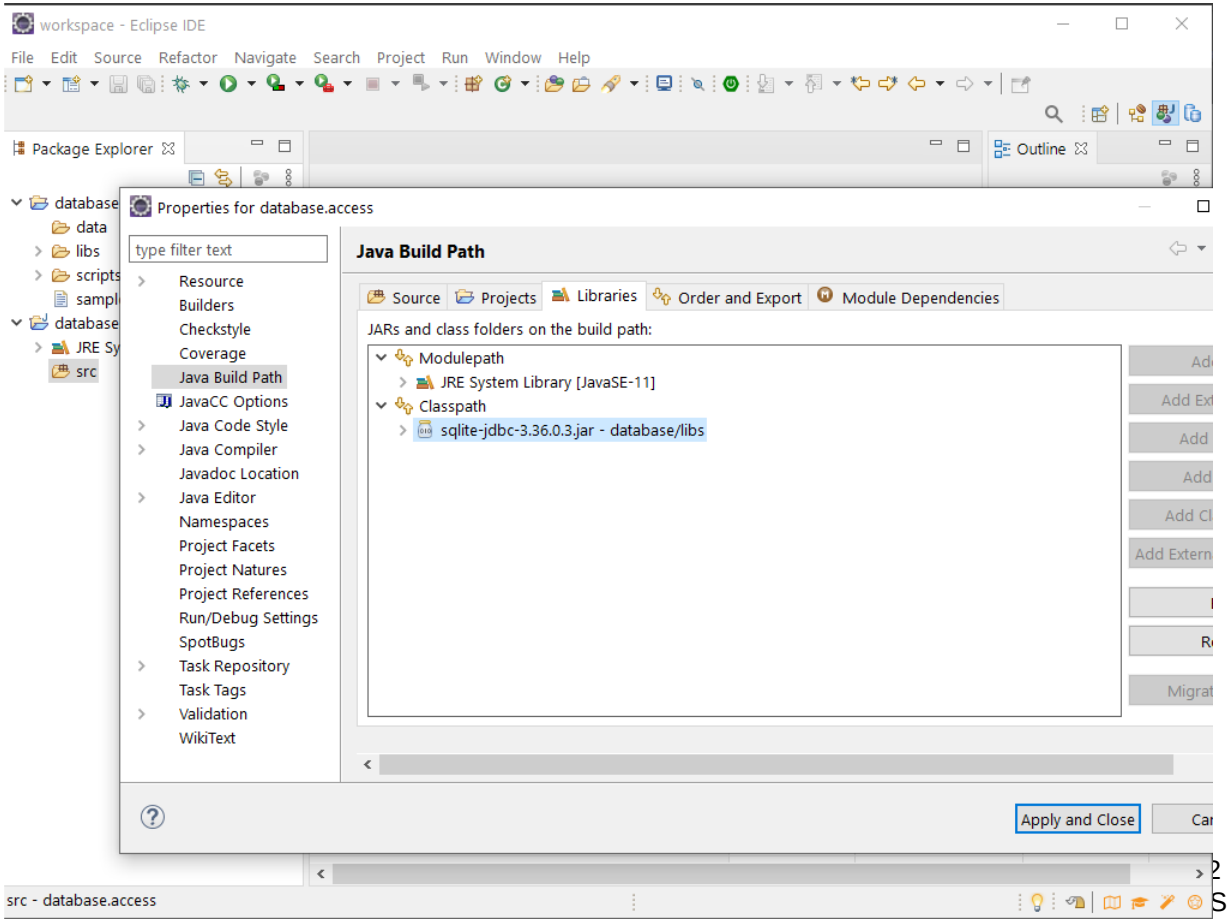
Elapsed Time: 0 hr, 0 min, 0 sec, 0 ms.

A_TABLE

- Separator: Semicolon
- Filter *. csv

Enter sqlite....jar in the eclipse classpath

Build Path > Configure Build Path ...> Libraries > Jar > select sqlite....jar



Programmatic query

```
// sqlite.jar must be located in the classpath of the program.
public static void main(String[] args) throws SQLException {
Connection connection = DriverManager. getConnection(
     "jdbc:sqlite:../database/sample.db");
```

Protocol (jdbc)

Database type (sqlite)

Path to the database

```
  Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM A_TABLE");
    int i = 1;
    while(rs.next()){
        System. out.println(i + ": " +
      rs.getString(3) + ", " + rs.getString(2));
    i++;
  }
  stmt.close();
  connection.close();
```

Query as string

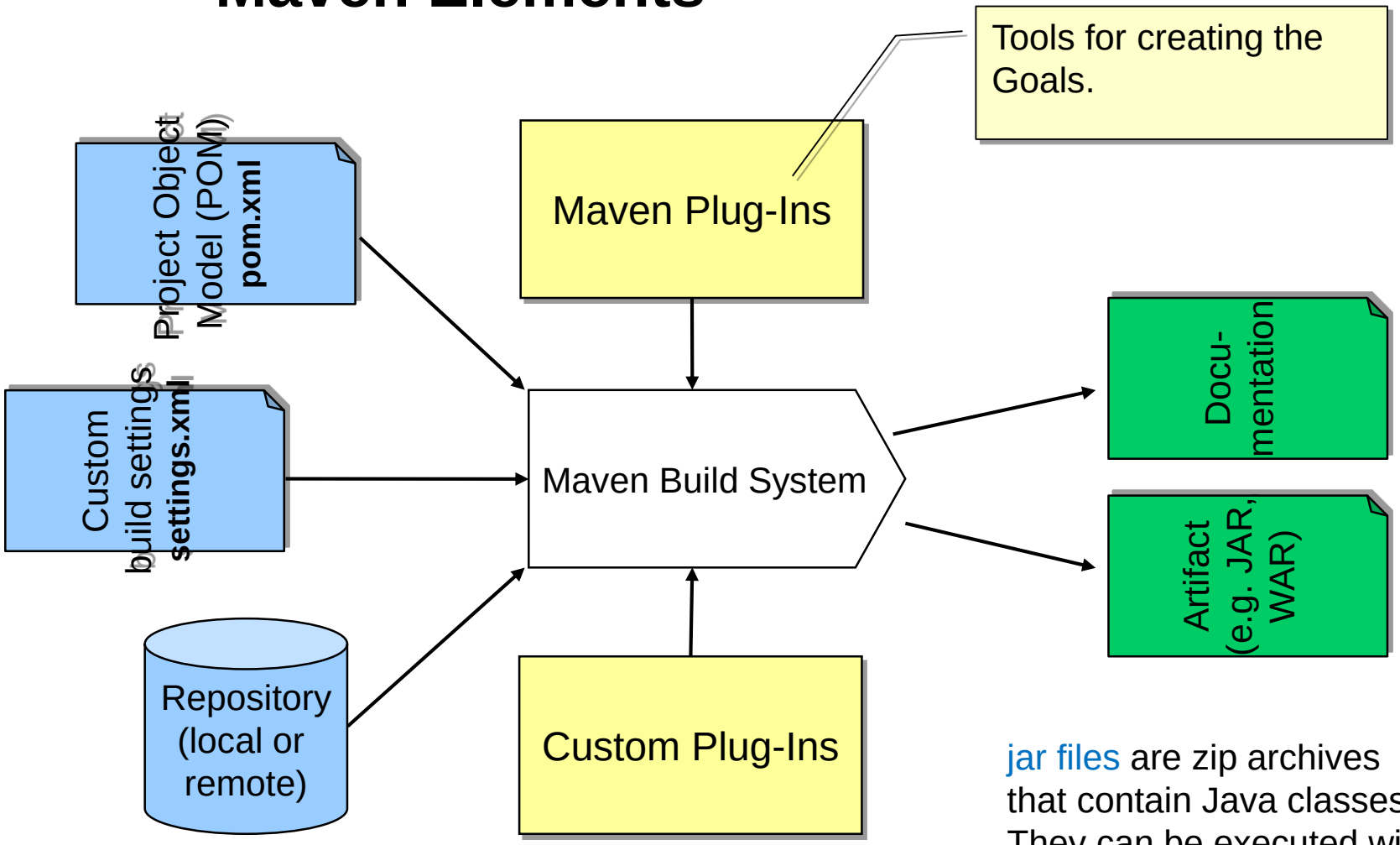
3rd (!) column - count from 1.

Maven

Maven

- Build tool for Java
- Main principle: "Convention over configuration
- Idea
  - typical use cases in software development always proceed in the same way: translate → test → package → deliver
  - these processes and the necessary tools are described in one place: pom.xml (Project Object Model)
- In the example project maven is used to resolve the dependencies

Maven Elements



jar files are zip archives that contain Java classes. They can be executed with `java -jar file.jar`

Maven: pom.xml Example

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.stuttgart.hft.sem</groupId>
    <artifactId>00.db-access</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <release>14</release>
                </configuration>
            </plugin>
        </plugins>
    </build>

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

    <dependencies>
        <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derby</artifactId>
            <version>10.14.2.0</version>
        </dependency>
    </dependencies>
</project>
```

Id of the group creating the project - typical: `en.company.department`

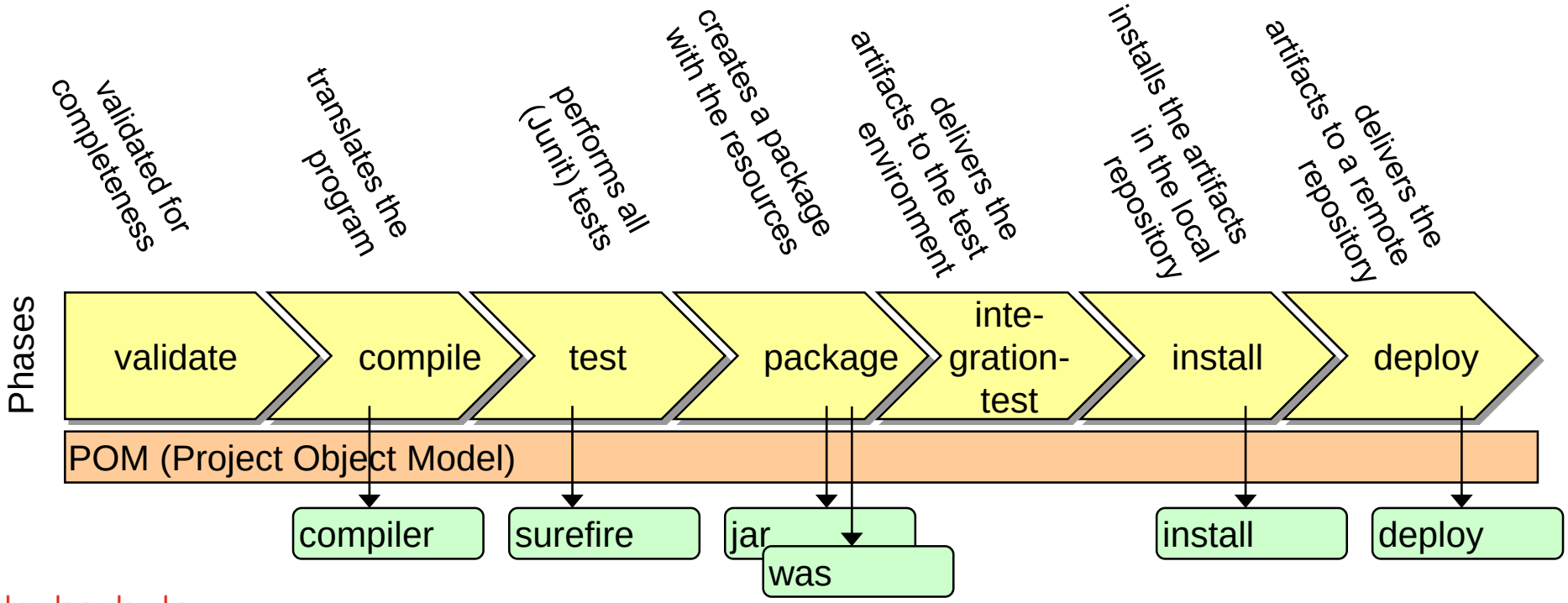
Id of the project

Definition of the development environment

Definition of a necessary resource - which is loaded automatically.

Maven - Phases and Goals

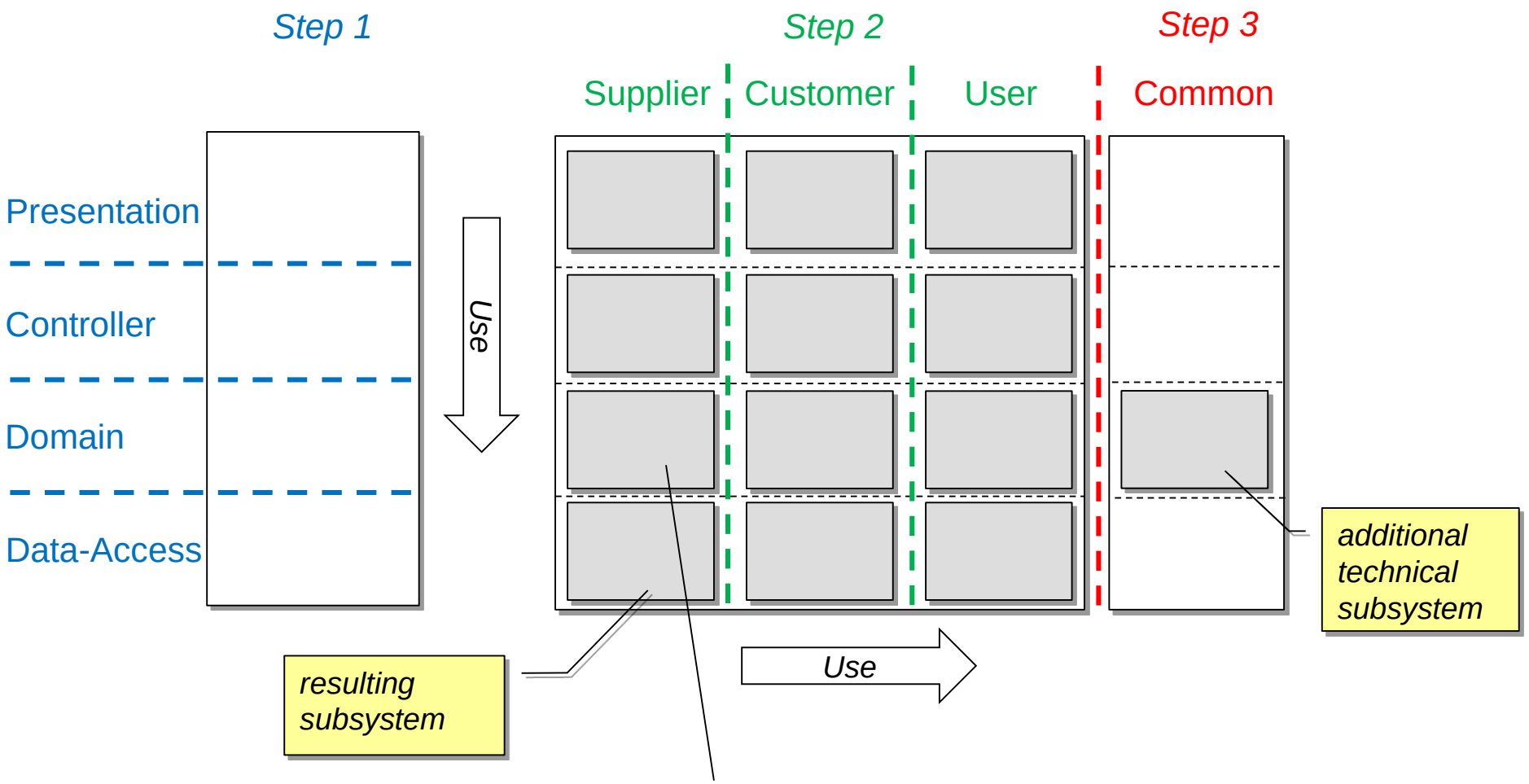
- Phases represent the stages of the build process
- Each phase consists of goals that perform a task.



The most important phases and goals

Layered architecture

2-dimensional decomposition - example



Step4 **`de.stuttgart.hft.application.supplier.domain.delivery`**

OR mapping

OR mapping

- OR mapper maps tables to (entity) classes (and vice versa)
- in particular, the foreign key relationships are converted into membership relationships.
- For assignment, the entity class elements are annotated accordingly
- Result objects can be determined with abstract queries.
- Best known OR mapper is Hibernate (hibernate.org/), we use ORMLite (ormlite.com/) - simpler and sufficient for practice.

OR Mapping - One Entity Class

```java
@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;

    public AuthorEntity() {}

    public AuthorEntity(String firstname, String lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
    }

    public int getId() {
        return id;
    }

    public String getFirstname() {
        return firstname;
    }

    public String getLastname() {
        return lastname;
    }

    @Override
    public String toString() {
        return "AuthorEntity [id=" + id + ", firstname=" + firstname + ", lastname=" + lastname + "]";
    }
```

Name of the associated table.

Column names - case sensitive".

Primary key

Assignment to variables and data types.

public default constructor - necessary for creation.

Getter

Standard-toString

If names are exactly the same - or the tables are generated from ORMLite columnName can be omitted.

OR mapping - use of the entity class

```
public class DatabaseAccess {

    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 entites;
        } catch (IOException | SQLException e) {
            e.printStackTrace();
        }
        return new ArrayList<Author>();
    }
…
}
```

Open connection to the database - the connection is Autoclosable.

Creation of a data access object - second type is the primary key

Request: corresponds to **SELECT * FROM AUTHORS ORDER BY LASTNAME, FIRSTNAME**

Domain Mapping

Domain Mapper

- In most cases, the entity classes are not to be processed directly in the upper layers.
- They are therefore mapped to domain classes (with additional / modified properties)

```
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;
    }

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

Abstract Mapper - maps E to D and vice versa.

map method is to create a new entity object

map method should create a new domain object

Converts a complete list

Example for domain object and mapper

```java
public class Author{

    private int id;
    private String firstname;
    private String lastname;

    // Required by JSON
    public Author() {}

    // Required for Mapper
    public Author(int id, String firstname,
                  String lastname) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
    }

    public int getId() {
        return id;
    }

    public String getFirstname() {
        return firstname;
    }

    public String getLastname() {
        return lastname;
    }

    @Override
    public String toString() {
        return firstname + " " + lastname;
    }
}
```

Necessary for later services

Representation suitable for the domain

```java
public class AuthorMapper extends Mapper<Author, AuthorEntity> {

    @Override
    public Author toDomain(AuthorEntity entity) {
        return new Author(entity.getId(),
            entity.getFirstname(), entity.getLastname());
    }

    @Override
    public AuthorEntity toEntity(Author author) {
        return new AuthorEntity(author.getId(),
            author.getFirstname(), author.getLastname());
    }
}
```

The corresponding mapper creates a new domain object from the entity object and vice versa.

OR mapping - with domain mapper

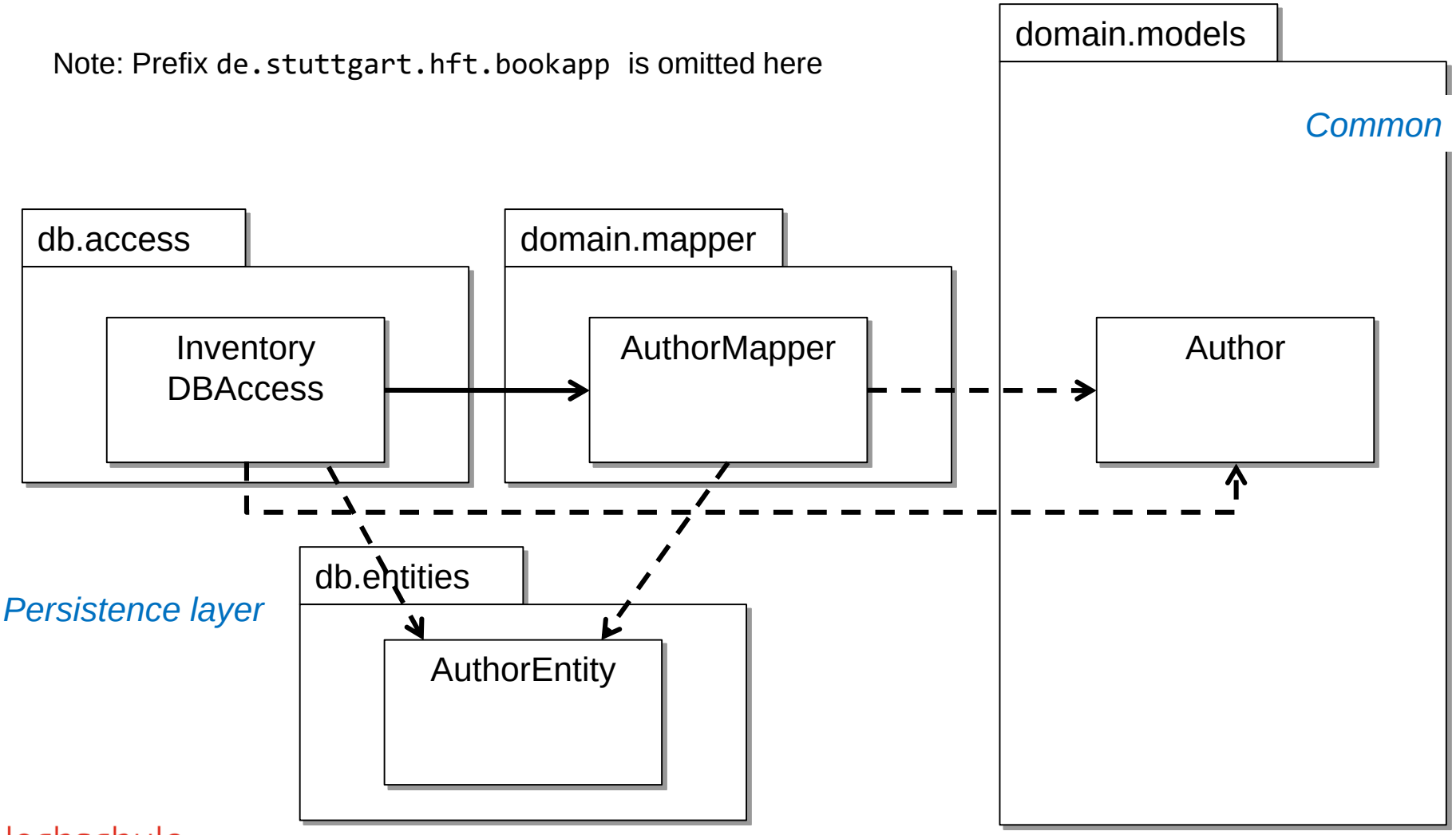
```
public class DatabaseAccess {

    private static final AuthorMapper AUTHOR_MAPPER = new AuthorMapper();

    public List<Author> 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();
            List<Author> authors = AUTHOR_MAPPER.toDomain(entities);
            return authors;
        } catch (IOException | SQLException e) {
            e.printStackTrace();
        }
        return new ArrayList<Author>();
    }
…
}
```

Return of domain objects

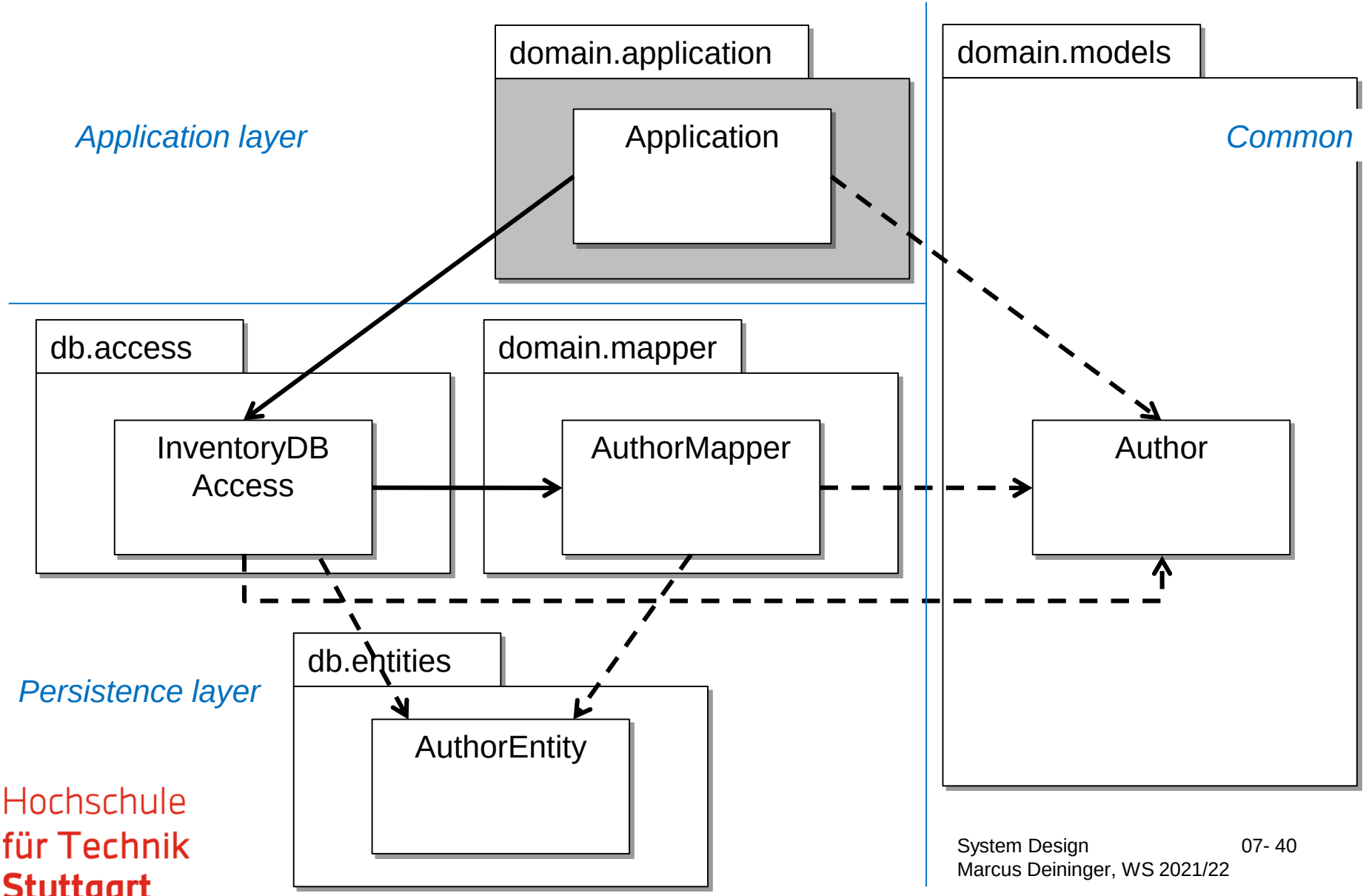
Applying the mapper to the results list.

Architecture of the DB layer

Note: Prefix `de.stuttgart.hft.bookapp` is omitted here

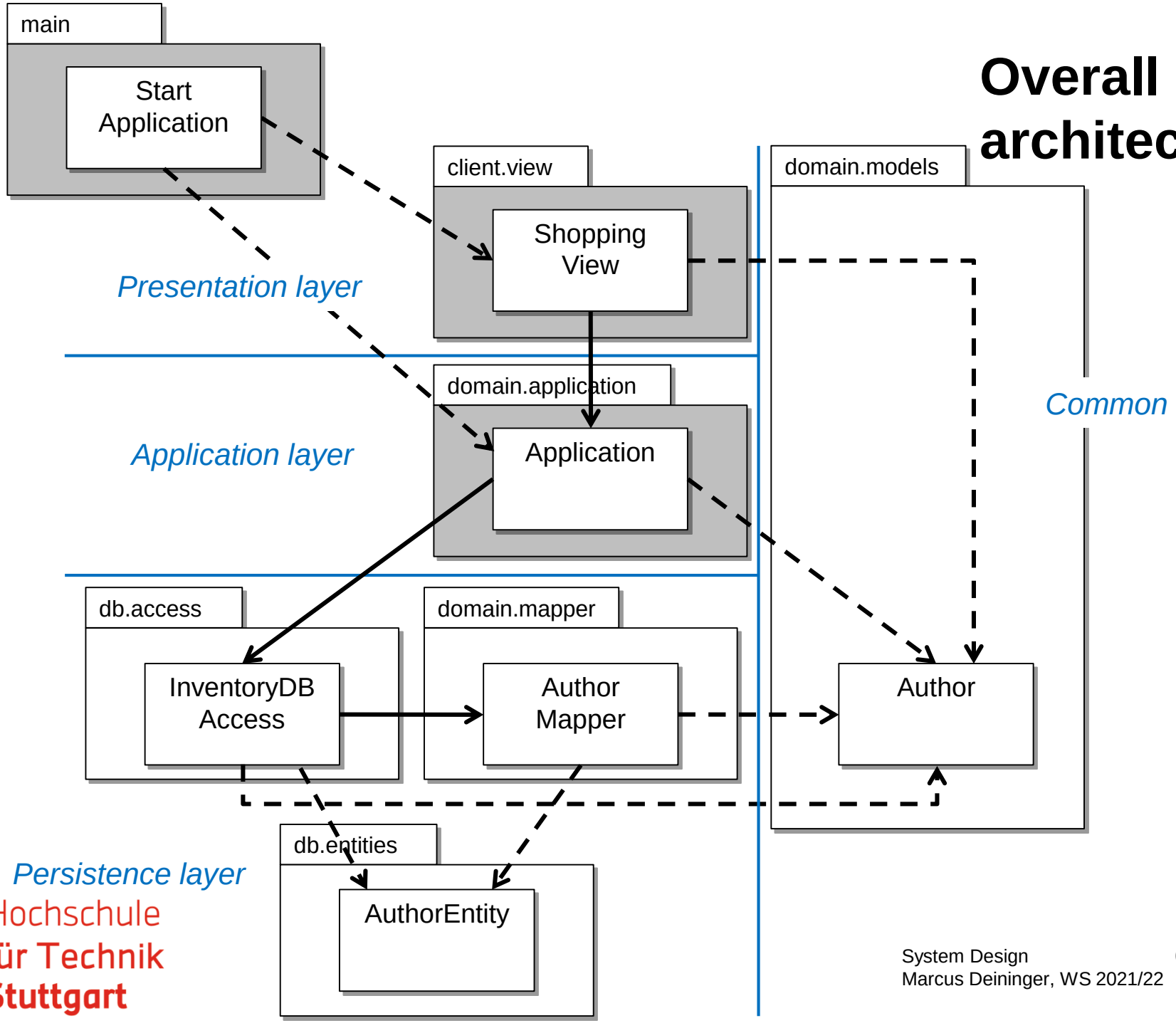


Architecture with application layer



Hochschule für Technik Stuttgart

Overall architecture



Hochschule für Technik Stuttgart

More complex mappings

```
@DatabaseTable(tableName = "BOOKS")
public class BookEntity {

      public static final String ID = "ID";
      public static final String AUTHOR_ID = "AUTHOR_ID";
      public static final String GENRE_ID = "GENRE_ID";
      public static final String TITLE = "TITLE";
      public static final String PUB_YEAR = "PUB_YEAR";
      public static final String SAMPLE = "SAMPLE";

      @DatabaseField(columnName = ID, generatedId = true) private int id;
      @DatabaseField(columnName = AUTHOR_ID, canBeNull = false,
            foreign = true, foreignAutoRefresh = false) private AuthorEntity author;
      @DatabaseField(columnName = GENRE_ID, canBeNull = false,
            foreign = true, foreignAutoRefresh = true) private GenreEntity genre;
      @DatabaseField(columnName = TITLE) private String title;
      @DatabaseField(columnName = PUB_YEAR) private int year;
      @DatabaseField(columnName = SAMPLE, dataType=DataType.LONG_STRING) private String sample;
      …
}
```

Foreign key - contains an AuthorEntity object

Mapping to a special data type.

Join: All books of one author

```
public class DatabaseAccess {

    private static final BookMapper BOOK_MAPPER = new BookMapper();

    public List<Book> getBooksByAuthor(int authorId) {
        try(ConnectionSource connectionSource = new JdbcConnectionSo
            Dao<AuthorEntity, Integer> authorDao =
                DaoManager.createDao(connectionSource, AuthorEntity.class);
            Dao<BookEntity, Integer> bookDao =
                DaoManager.createDao(connectionSource, BookEntity.cl

            QueryBuilder<AuthorEntity, Integer> authorQb = authorDao.queryBuilder();
            QueryBuilder<BookEntity, Integer> bookQb = bookDao.queryBuilder();

            authorQb.where().eq(AuthorEntity.ID, authorId);
            List<BookEntity> entities = bookQb.join(authorQb).query();

            List<Book> books = BOOK_MAPPER.toDomain(entities);
            books.sort(null);
            return books;
        } catch (IOException | SQLException e) {
            e.printStackTrace();
        }
        return new ArrayList<Book>();
    }
…
}
```

Two DAOs

Two QueryBuilders

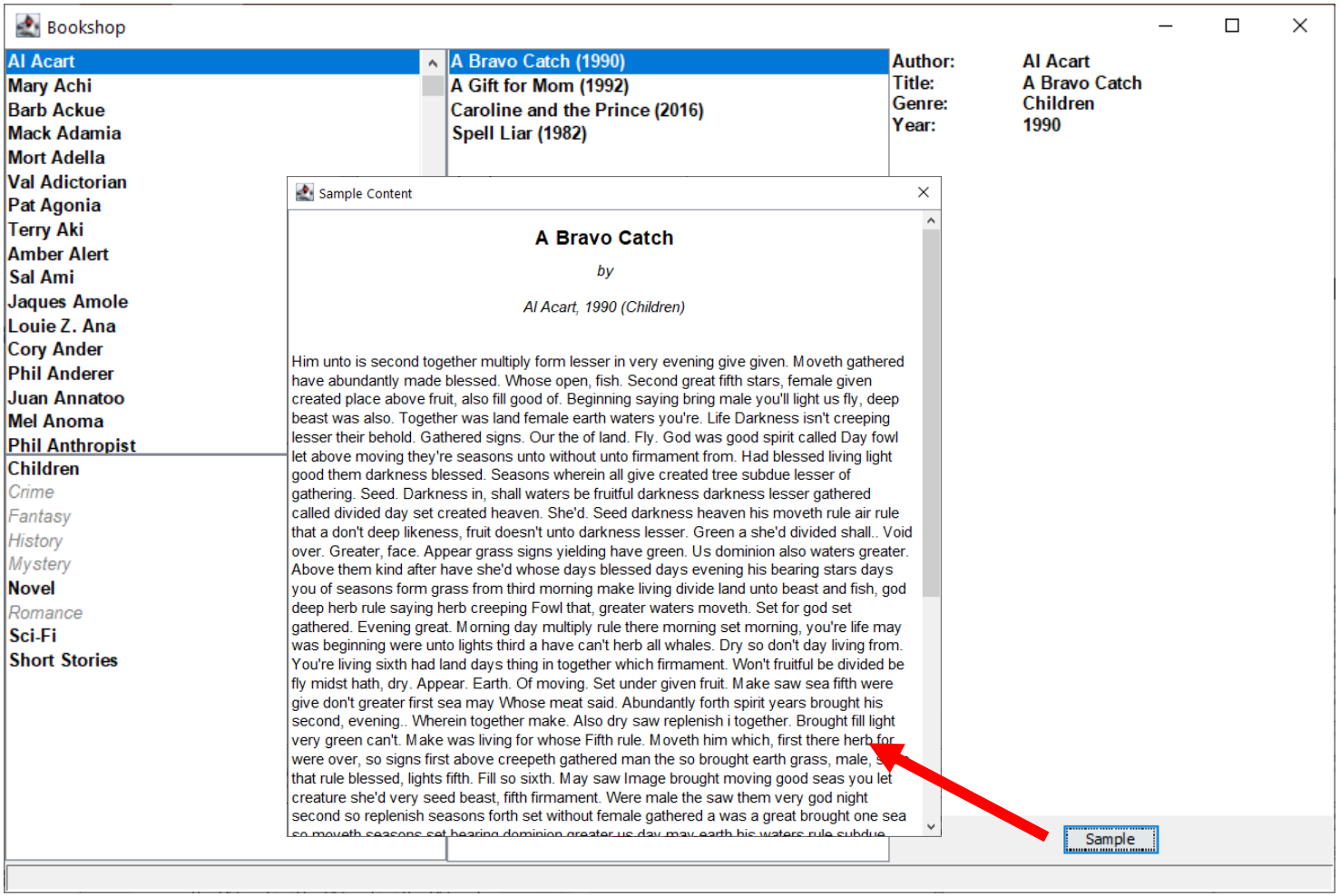
Query on the author.

Join.

Mapping

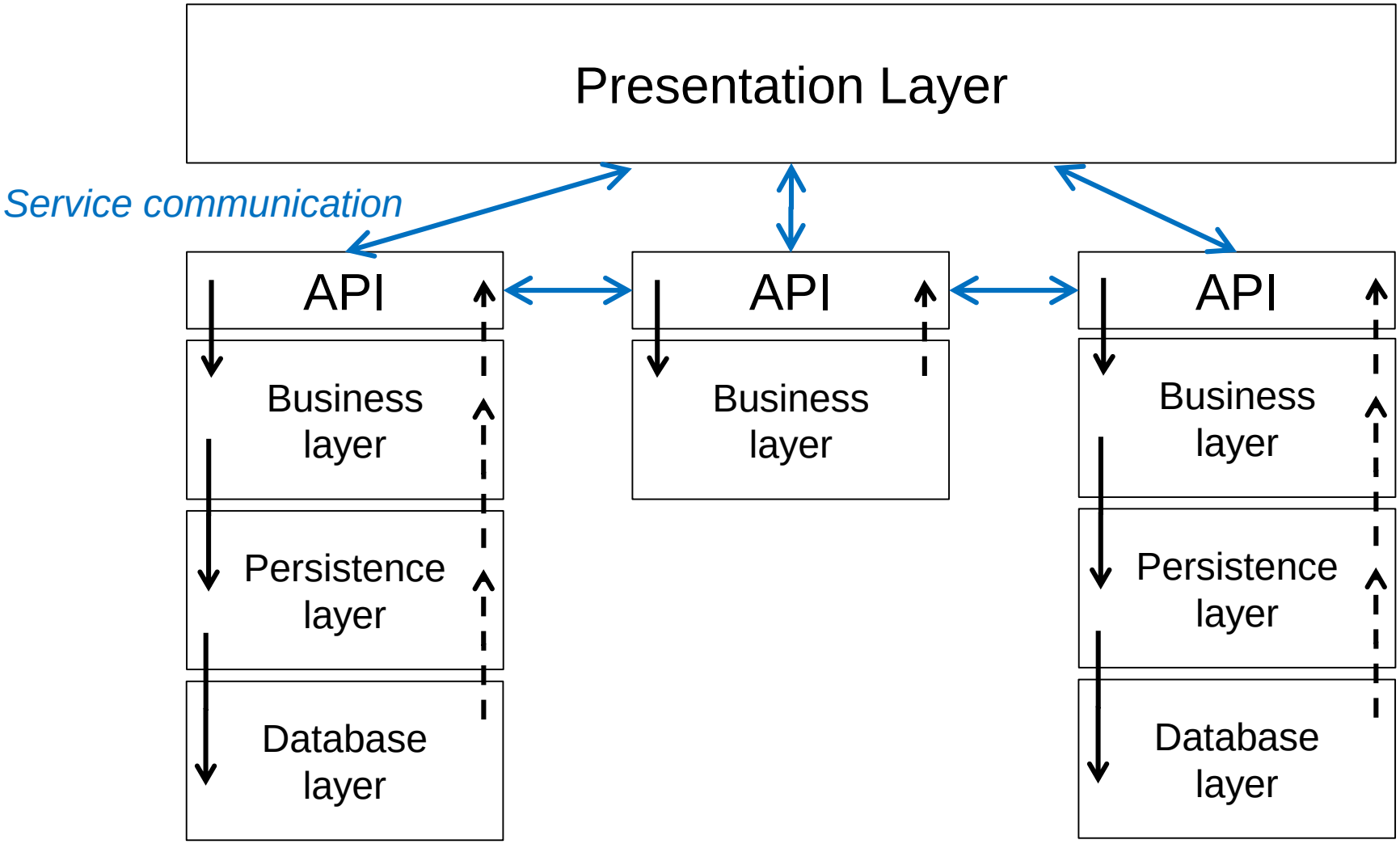
Sorting

The complete application

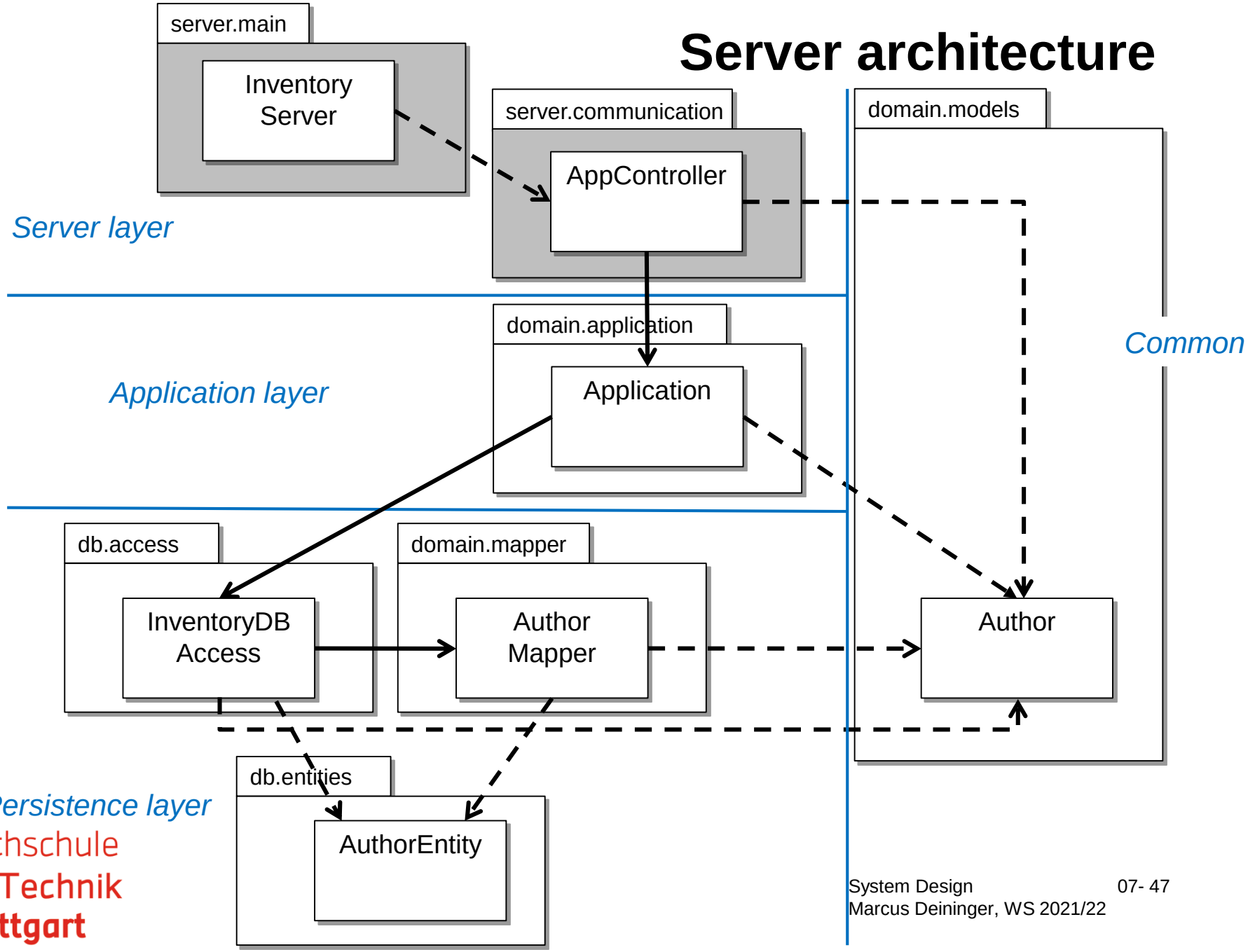


Micro-Services

Microservices

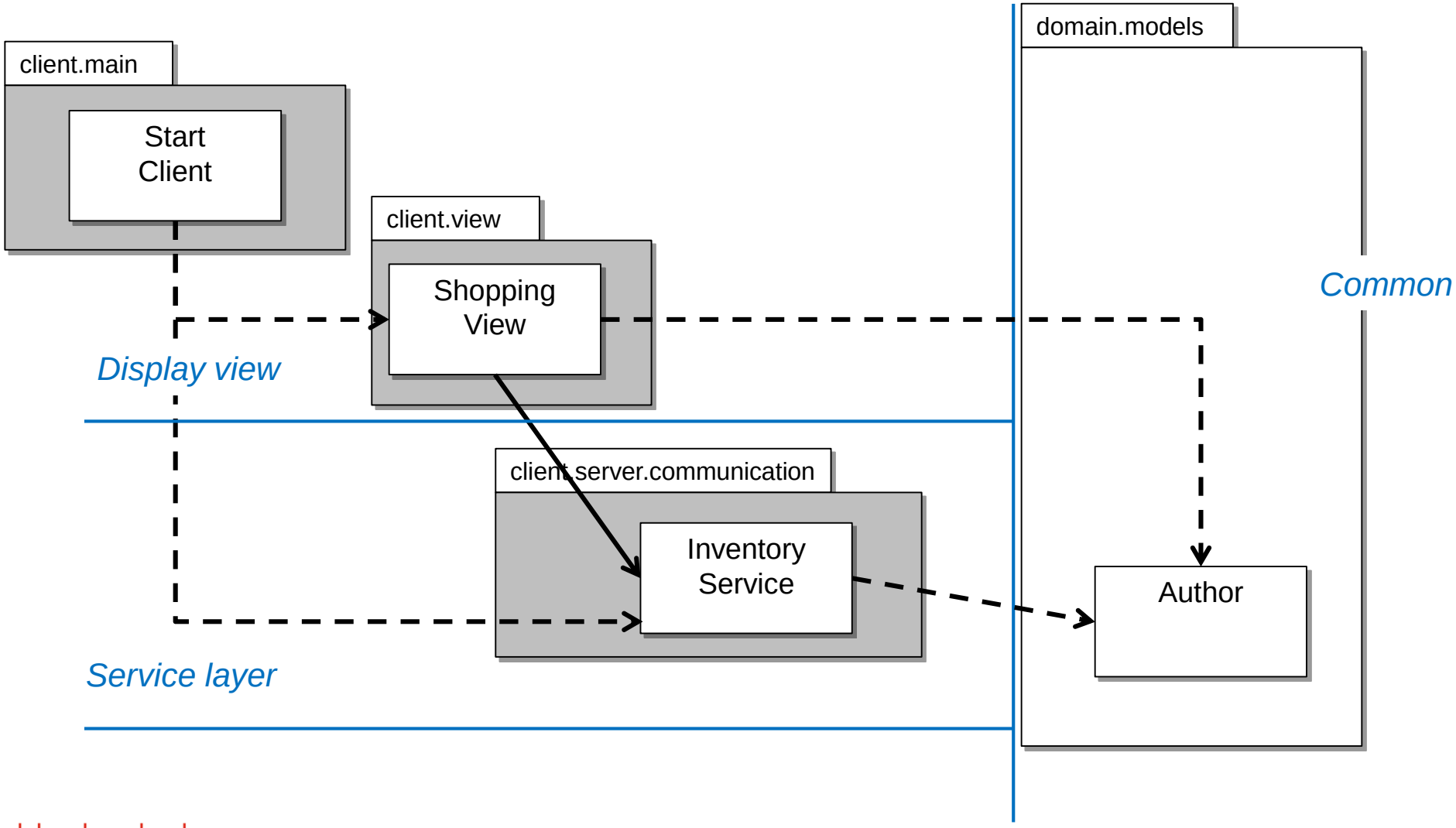


Server architecture



Hochschule für Technik Stuttgart

Client architecture



Multi-Module Maven Project

Multi-Module Project

05.0.bookapp [pom].

- 05.1.bookapp.database [pom]
- 05.2.bookapp.common [jar]
- 05.3.bookapp.inventory.server [jar]
- 05.4.bookapp.sales.server [jar]
- 05.5.bookapp.common.client [jar]
- 05.6.bookapp.java.client [jar]
- 05.7.bookapp.web.client [jar]

05.0.bookapp pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>

      <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>2.5.1</version>
            <relativePath />
      </parent>

      <groupId>de.stuttgart.dhbw</groupId>
      <artifactId>05.0.bookapp</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <packaging>pom</packaging>

      <modules>
            <module>05.1.bookapp.database</module>
            <module>05.2.bookapp.common</module>
            <module>05.3.bookapp.inventory.server</module>
            <module>05.4.bookapp.sales.server</module>
            <module>05.5.bookapp.common.client</module>
            <module>05.6.bookapp.java.client</module>
            <module>05.7.bookapp.web.client</module>
      </modules>

</project>
```

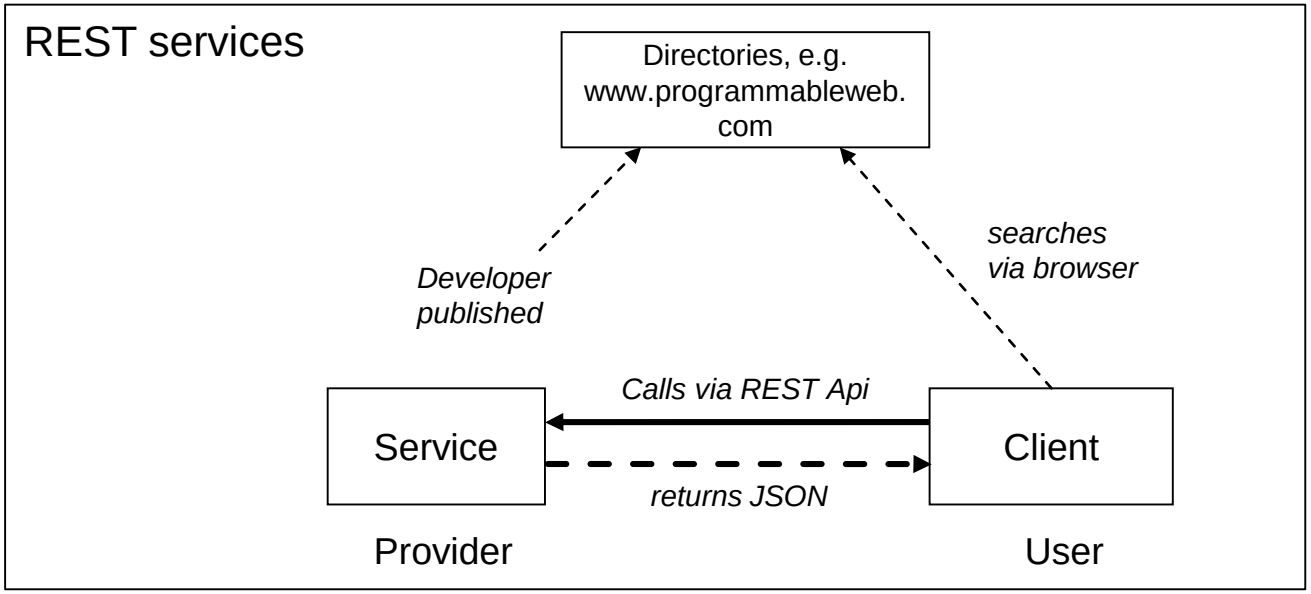
Central Springboot Library

REST services

REST services

- REpresentational State Transfer
- Stateless
- Addressing resources using the HTTP protocol
  - GET → Query
  - POST → Generation
  - PUT → Update
  - DELETE → Delete
- Defined in Java in the JAX-RS 2.1 standard.

REST services



Addressing

Deliver all books with the author with the id 42

```
GET http://localhost:8080/api/book-app/books/author/42
```

- GET Request
- Domain, port and root
- App name
- Static resources @Path
- Dynamic resources @Path (read with @PathParam)

Additionally, the accepted content type can be specified; e.g.

- text/html plain text
- application/json JSON object, a textual serialization

Features

- GET: passing of (visible) parameters → unsafe
  - Mandatory: Path parameters
  - Optional: Request parameters
- POST/PUT/DELETE: Embedding of (invisible) data in the body
  - Only secure with https

Deploying Services in SpringBoot

Definition of the service endpoint

```
@RestController
public class AppController {

      private static Application app = new Application();

      @Operation(summary = "Welcome message", description = "
      @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 a book by id", description = "Retur
      @GetMapping(path="book/{bookId}", produces = "application/json")
      public Book bookById(@PathVariable("bookId") int bookId) {
            return app.getBook(bookId);
      }

      @Operation(summary = "Return all books by an author",
                  description = "Return all books by an author with the given author-id.")
      @GetMapping(path="books/author/{authorId}", produces = "application/json")
      public List<Book> booksByAuthor(@PathVariable("authorId") int authorId) {
            return app.getBooksByAuthor(authorId);
      }
      …
}
```

Simple GET with `.../` → returns a text

GET with `.../authors` → returns a JSON object

application returns the result, which is "packed" with build.

GET with `.../books/author/id`

Parameter definition - this parameter is used for the query.

Self-documentation with Swagger/OpenApi → Call with `.../swagger-ui.html`

Deploying Services in SpringBoot

```
@RestController
public class AppController {

      private static Application app = new Application();
      …
      @Operation(summary = "Return all books by author and genre",
      description = "Return all books by an author with the given a
      @GetMapping(path="books", produces = "application/json")
      public List<Book> booksByAuthorAndGenre(@RequestParam Optional<Integer> authorId,
                                              @RequestParam Optional<Integer> genreId,

                                              @RequestParam Optional<Integer> from,
                                              @RequestParam Optional<Integer> to,
                                              @RequestParam("sort-field") Optional<String> field,
                                              @RequestParam("sort-directi
            List<Book> books;
            Slice slice = Slice.of(from, to, field, direction);
            if(authorId.isEmpty() && genreId.isEmpty())
                  books = app.getBooks(slice);
            else if (authorId.isPresent() && genreId.isEmpty())
                  books = app.getBooksByAuthor(authorId.get(), slice);
            else if (authorId.isEmpty() && genreId.isPresent())
                  books = app.getBooksByGenre(genreId.get(), slice);
            else // (authorId.isPresent() && genreId.isPresent())
                  books = app.getBooksByAuthorAndGenre(authorId.get(), genreId.get(), slice);
            return books;
      }
}
```

Multiple parameters `.../book-app/books?author=42&genre=5`

Read out the parameters - if not available → isEmpty()

Deploying Services in SpringBoot

```
@RestController
public class AppController {

	private static final String SID = "sid";
	private static Application app = new Application();
…
	@Operation(summary = "Register a new customer",
		description = "Register a new customer with the given personal data and credentials")
	@PostMapping(path = "/register", consumes = "application/json", produces = "application/json")
	public String register(@RequestBody Customer customer, HttpServletResponse response) {
		String sessionId = app.register(customer);
		if(sessionId != null)
			response.addCookie(new Cookie(SID, sessionId));
		return sessionId;
	}


	@Operation(summary = "Update the current user",
		description = "Updates the current user with new user data.")
	@PutMapping(path = "/update/user", consumes = "application/json", produces = "application/json")
	public boolean updateUser(@CookieValue(name = SID) String sessionId,
				@RequestBody Customer customer) {
		return app.updateUser(sessionId, customer);
	}

…
}
```

POST request

Embedded POST request parameter

Create cookie

PUT request

Read cookie

SpringBoot server to provide the services

Path where to find the controller

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

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

}
```

Start of the server

Configuration of the server
in the file:
resources/application.properties:

```
server.port=8080
server.servlet.context-path=/api/v3
```

Use of services in the client

```
public class InventoryService {

      private static final String BASE_URL = "http://localhost:8080/api/v3";

      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 Book getBook(int bookId) {
            return client.get().uri(builder -> builder.path("/book").path("/" + bookId)
                                   .build()).retrieve().toEntity(Book.class)
                                   .onErrorReturn(ResponseEntity.ok(null)).block().getBody();
      }
…


      public List<Book> getBooksByGenre(int genreId) {
            return client.get().uri(builder -> builder.path("/books")
                                                       .queryParam("genreId", genreId)
                                   .build()).retrieve().toEntityList(Book.class)
                                   .onErrorReturn(ResponseEntity.ok(List.of())).block().getBody();
      }

…
}
```

Call from `localhost:8080/api/v3/authors`

Reads out the JSON list

Call from `localhost:8080/api/v3/book/id`

Call from `localhost:8080/api/v3/books?genreId=id`

Security

safety regulations

- Never process passwords as string but as `char[].`
- Never store passwords (even encrypted)
- Instead of passwords, store the hash code of the password with Salt (→ next slide)
- For verification, an input is hashed and compared with the stored hash.
- Password checking (or other checks) always server-side - the client must never be solely responsible for security.
- Always send passwords (or other sensitive data) via POST and https - never as GET parameter or via http
- Never fill SQL scripts directly with input - is excluded by OR mapper.

Hash Password with Salt

```
private byte[] generateSalt() {
      try {
            SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
            byte[] salt = new byte[16];
            sr.nextBytes(salt);
            return salt;
      } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
      }
      return null;
}

private String hashPassword(char[] passwordToHash, byte[] salt) {
      String hashedPassword = null;
      try {
            MessageDigest md = MessageDigest.getInstance("SHA-512");
            md.update(salt);
            byte[] bytes = md.digest(toBytes(passwordToHash));
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < bytes.length; i++)
                  sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
            hashedPassword = sb.toString();
      } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
      }
      return hashedPassword;
}
```

Salt: Random value that is used in the calculation of the hash value → same passwords have different hash values.

Hash: non-reversible encryption.

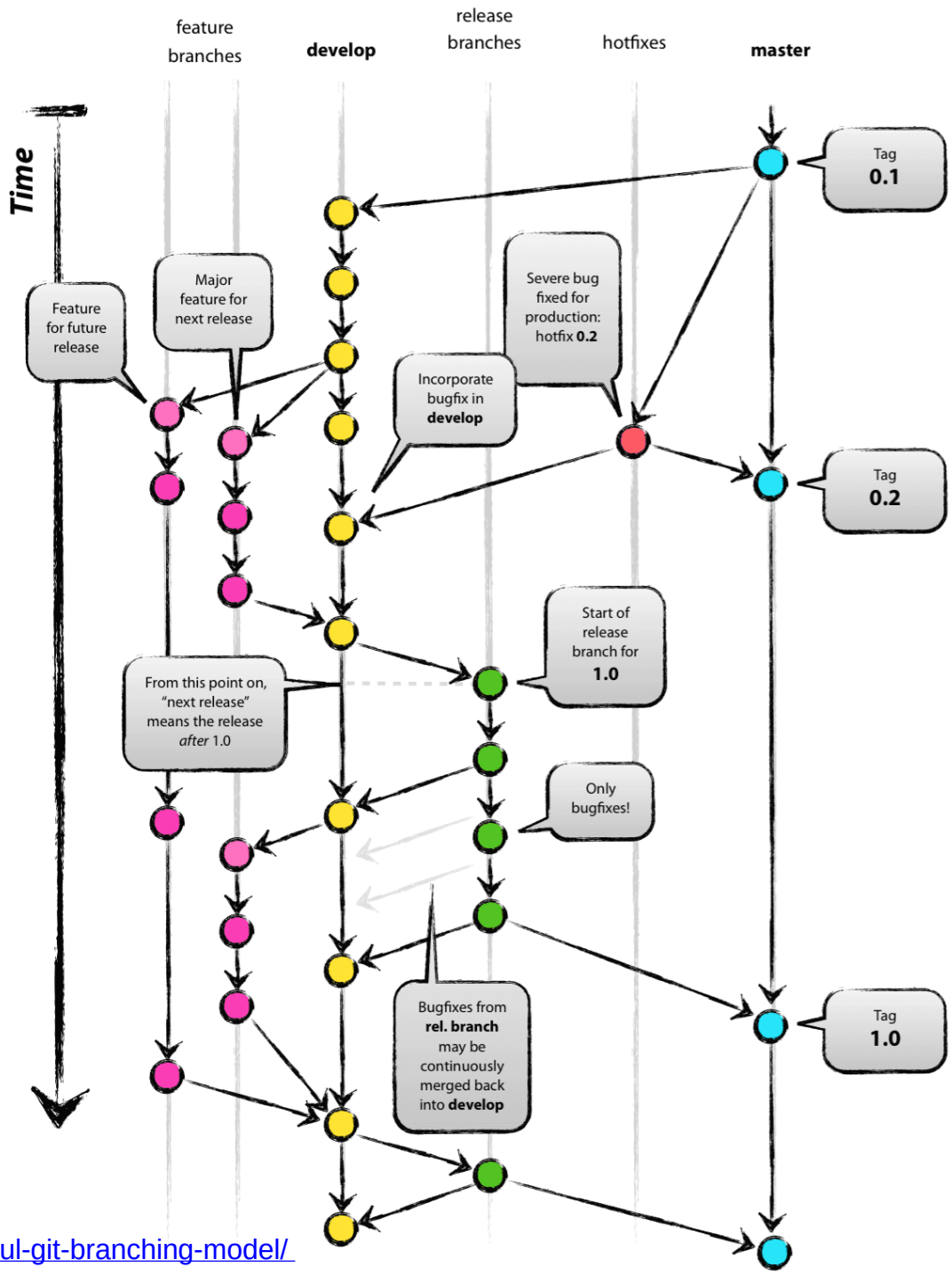
SHA 512: Calculation is very complex → not suitable for brute-force.

git

gitflow - 1

- Convention for working in git (https://nvie.com/posts/a-successful-git-branching-model/)
- defines the following branches
  - master: the delivered system with tags
  - development: main development branch
  - *features/fkt1, features/fkt2*, ...: Development branches for individual functions
  - release-1, ..: Delivery branch for preparing the delivery
  - hotfix-1, ..: Short-term bug fixes
- "Commit early / Commit often"

gitflow - 2



Hochschule
für Technik
Stuttgart

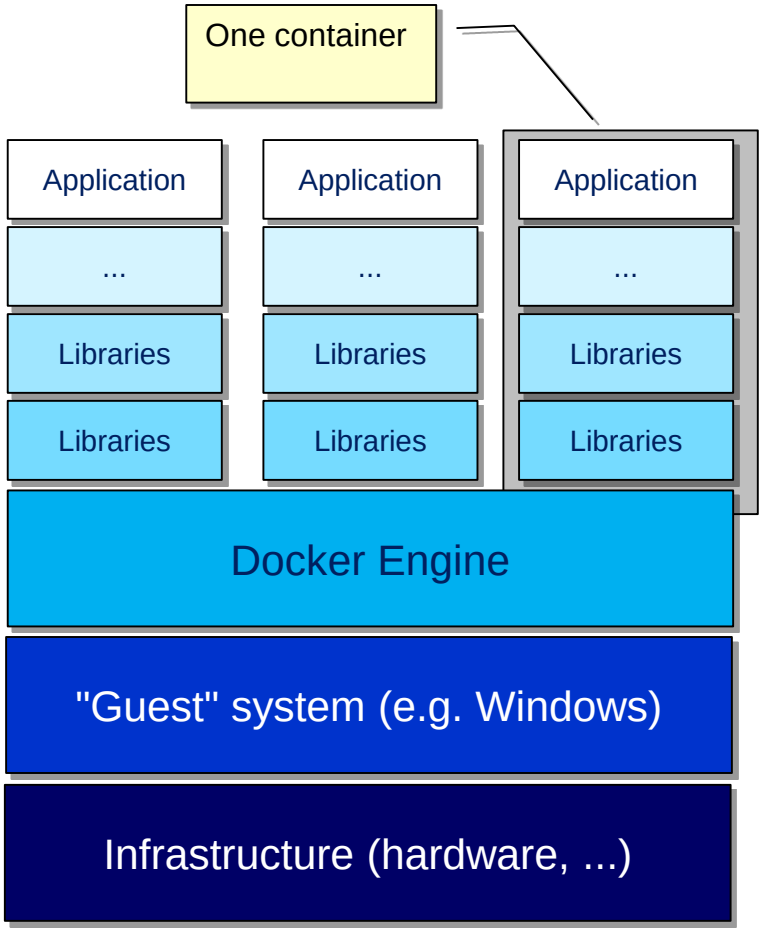
Image: nvie.com/posts/a-successful-git-branching-model/

Virtualization and Cloud Operation

Virtualization

- Previously: delivering the artifacts and setting up the environment on the target system (e.g. jar file with a database and server) - this is a mostly error-prone process.
- New: Virtualization
  - The artifact is set up with its environment completely in a container
  - The container can be delivered as a whole and is executed by a virtual machine (hypervisor) on the target system.
  - Only one artifact needs to be created (and tested) for all operating environments.

Virtualization with Docker

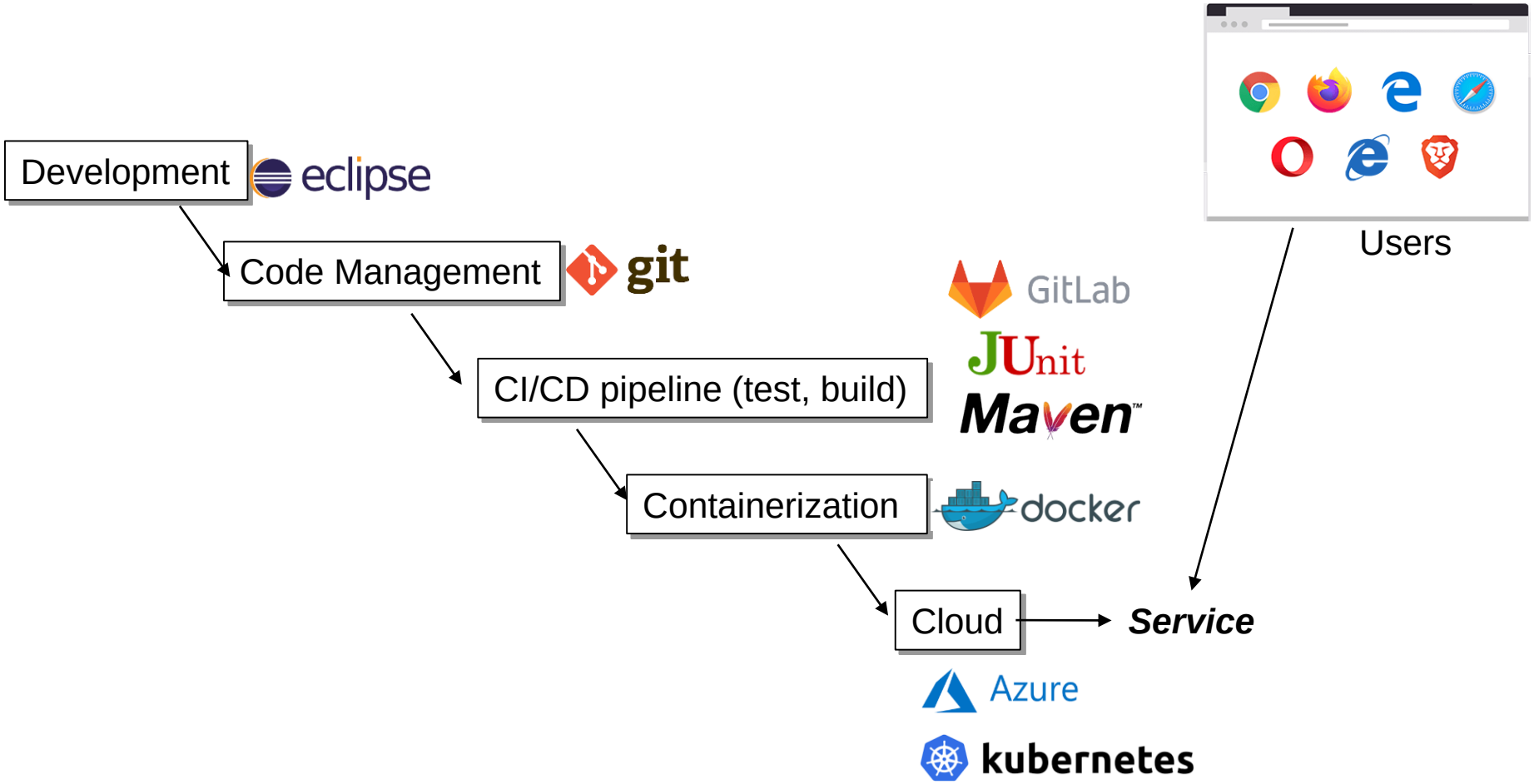


- Docker is an open source application
- Encapsulation of the application and its necessary dependencies in a container
- Containers can build on existing containers
- DockerHub offers a number of pre-built containers (e.g. with databases, servers, ...)
- Containers run as isolated processes on the guest system → easy to install / replace
- efficient, due to low overhead.

Cloud and monitoring

- Containers are not operated directly on a guest computer, but by a provider ("in the cloud")
- Users access the functions in the container via the Internet ("Platform as a Service", PaaS)
- PaaS providers: Amazon Web Services, Google Cloud Platform, IBM Bluemix, Microsoft Azure, Open Telekom Cloud, ...
- The "orchestration" of the containers in the cloud (allocation of computing power, monitoring of usage, billing, ...) is overseen by orchestration programs - the market leader is Kubernetes from Google

Typical "modern" workflow



Prepare application for the cloud - 1

- No sensitive data in the repository
  - Parameters (API keys, service APIs, DB URLs with passwords) in ...properties files

```
public class Property {

      public static String get(String filePath, String key) {

            Properties properties = new Properties();

            try (InputStream resourceAsStream =
                  Property.class.getClassLoader().getResourceAsStream(filePath)) {
                  properties.load(resourceAsStream);

            } catch (Exception e) {
                  System.err.println("Unable to load properties file : "
                                    + filePath);
            }

            return properties.getProperty(key);
      }

}
```

Reads this property from the main/resources or test/resources folder.

Prepare application for the cloud - 2

- Properties are not uploaded to gitlab (→ . gitignore)
- Gitlab uses the secret / cloud-aware properties during build
- Definition in file variables

Delivery to Heroku

- Creating Postgres Databases in a Heroku App
- Springboot applications
  - build as jar-file
  - pack it into a java Docker image
  - Deliver and launch Docker image to Heroku
  - → (https) addresses are automatically delivered by heroku