Project Beginning

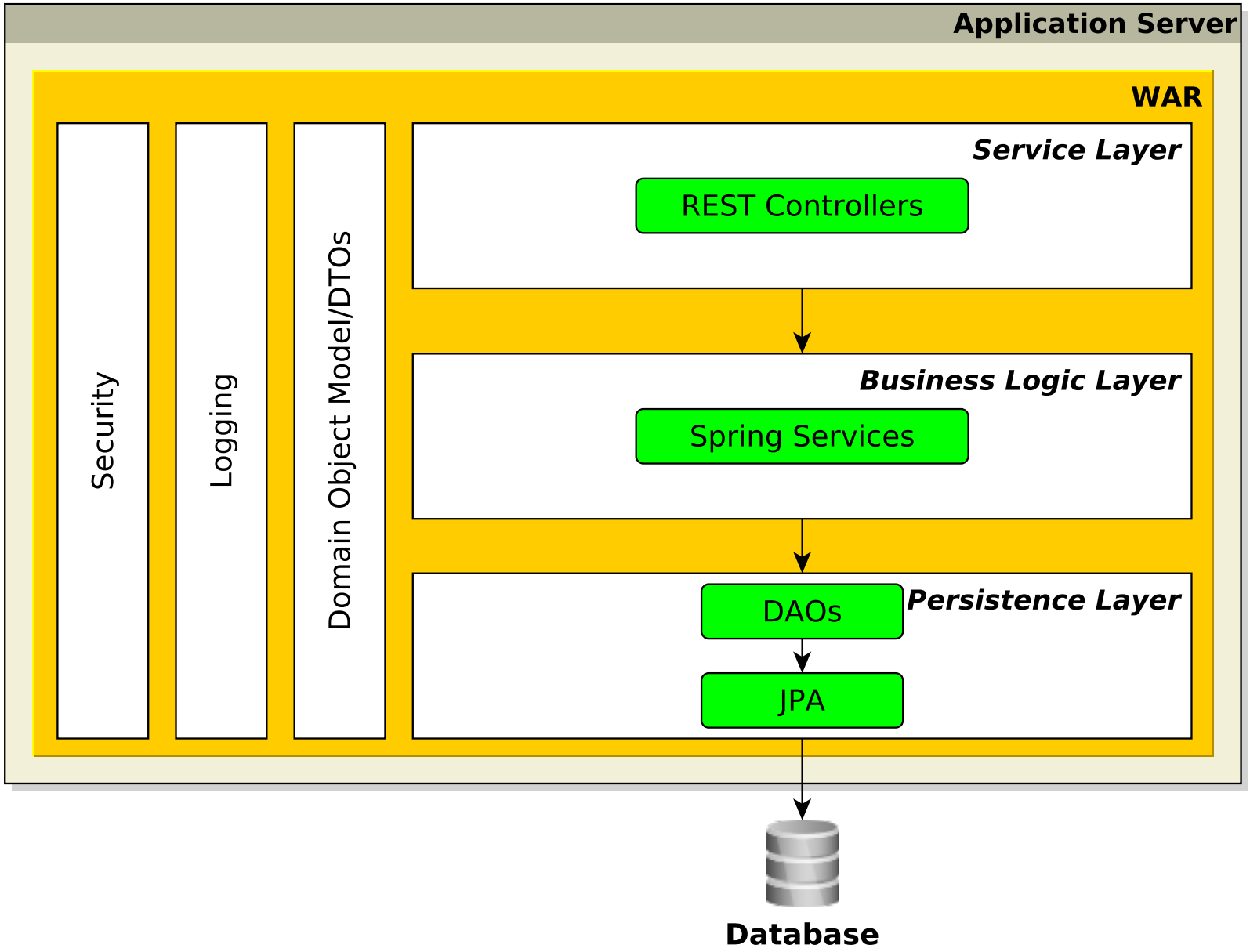
Martin Ledvinka

martin.ledvinka@fel.cvut.cz

Winter Term 2019

Contents

Architecture



Deploy

WAR

- *Web Archive*
- Format of deployable Java web application artefacts
  - **EAR** for full-blown Java EE artefacts

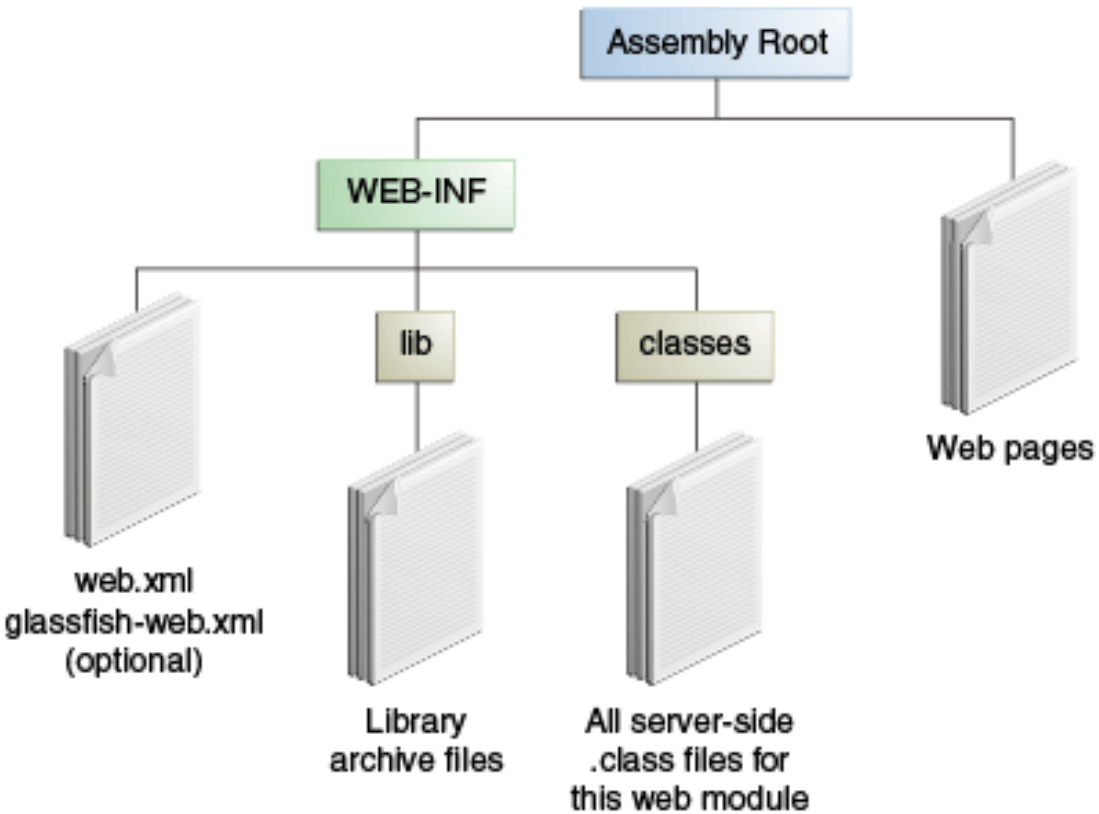


Figure : WAR structure. Source:
`https://docs.oracle.com/javaee/7/tutorial/packaging003.htm`

WAR cont.

- `web.xml` optional since Servlet API 3
  - All configuration can be done in source using Java + annotations
  - We won't be using it in our projects
- `WEB-INF` is not part of the public document tree of the application
  - Not accessible by clients
  - But accessible by servlet code – on classpath
  - Contains application code
- `lib` for required libraries, e.g., Spring, JDBC driver

Deployment

The following applies to Apache Tomcat!

- `webapps` folder for deployed web applications
- Can deploy exploded war (unpacked)
  - Tomcat will otherwise unpack WARs automatically
- Tomcat watches for changes in `webapps`
  - Copy into folder – *deploy*
  - Remove WAR from folder – *undeploy*
  - Application context
    - WAR file name
    - `META-INF/context.xml` in deployed WAR
    - `context.xml` in server configuration

Demo

- Demo of servlet application from lecture one to Tomcat
- Demo of servlet application from lecture one deployment in IntelliJ IDEA via a Run configuration

Maven

Apache Maven

- Software project management and comprehension tool
- Manage project dependencies, build, reporting, documentation
- Repository with libraries
  - Maven central at `maven.org` (web UI at `http://search.maven.org`)
  - Possible to have own repository, see e.g. `http://kbss.felk.cvut.cz/m2repo`
  - Local repository – cache

POM

- *Project Object Model*
- `pom.xml` file
  - Central XML-based configuration of Maven projects
  - Hierarchical project identification
    - `groupId`
    - `artifactId`
    - `version`
  - Manage dependencies – `dependencies` section
  - Manage build process – `build` section – using plugins – `plugins` section

Directory Structure

- `src`
  - `/main`
    - `/java`
    - `/resources`
    - `/webapp`
  - `/test`
    - `/java`
    - `/resources`
- `pom.xml`

Project Build Phases

1. *validate* - validate the project structure and configuration
2. *compile* - compile the source code of the project
3. *test* - test the compiled source code using a suitable testing framework
4. *package* - take the compiled code and package it in its distributable format, such as a JAR
5. *verify* - run any checks on results of integration tests to ensure quality criteria are met
6. *install* - install the package into the local repository
7. *deploy* - copy the final package to the remote repository

Dependency Scopes

- *compile* – default, dependency available on classpath
- *provided* – expected to be provided at runtime – by JDK, application server etc.
- *runtime* – not required for compilation, but is for execution
- *test* – required for test compilation and execution
- *system* – similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
- *import* – used when specifying dependencies in parent projects

Gradle

Maven	**Gradle**
XML	Groovy
Maven repo	Maven repo
Plugins	Plugins, direct code
Recompile everything on build	Incremental build

Tasks

Task – Together

Inception of a Spring Boot project.

- Spring Boot by default packaged as JAR
- Use `spring-boot-starter-parent` Maven project parent to inherit dependencies easily
  - Various `spring-boot-starter-*` Maven projects pulling in groups of related dependencies
  - `spring-boot-starter-data-jpa` for JPA, transaction API
  - `spring-boot-starter-web` for Jackson, Spring Web, MVC and embedded Tomcat
- Build with `spring-boot-maven-plugin` to package dependecies into JAR automatically
- We can use the `SeminarTwoMain.java` class to check that the JAR can be executed

Task – 1 point

1. Fork project `https://gitlab.fel.cvut.cz/ear/b191-eshop`
2. Clone your fork of the B191-eshop project
3. Checkout branch *b191-seminar-02-task*
4. Create a Maven project using the `HelloWorld.java`, `HelloWorldTest.java`, and `logback.xml` files

Acceptance Criteria

- Project has groupId, artifactId, name and description
- Project can be packaged as **WAR** using Maven
- Tests are run during build (and they pass)
- When the resulting WAR is deployed to application server (e.g., Tomcat), the servlet is accessible through a web browser
- Servlet access is logged based on the provided Logback configuration

Syncing Your Fork

1. Add upstream remote to the local clone of your fork
   - `git remote add upstream git@gitlab.fel.cvut.cz:ear/b191-eshop.git`
2. Fetch branches and commits from the upstream repository (EAR/B191-eshop)
   - `git fetch upstream`
3. Check out local branch corresponding to the task branch
   - `git checkout -b b191-seminar-02-task`
4. Merge changes from the corresponding upstream branch
   - `git merge upstream/b191-seminar-02-task`
5. Do your task
6. Push the solution to your fork
   - `git push origin b191-seminar-02-task`

Task – Notes

- You will need an application server for this task
- For example, Apache Tomcat
  1. `http://tomcat.apache.org/`
  2. Download 9.0.26
  3. Unpack
  4. Start by calling `bin/startup.sh` (Linux) or `bin/startup.bat` (Windows)
  5. Deploy by copying WAR file into `webapps` directory

Task – Hints

- Use Google or Maven central to find exact dependency identifiers
- `logback.xml` should go into `src/main/resources`

Useful dependencies

- SLF4J, Logback-classic
- Servlet API (scope *provided*)
- JUnit, Mockito-core (scope *test*)

Useful Maven plugins

- Maven compiler plugin
- Maven WAR plugin
- Maven Surefire plugin

The End

Thank You

Resources

- http://maven.apache.org/guides/
- https://docs.oracle.com/javaee/7/tutorial/packaging003.htm
- https://spring.io/guides/gs/spring-boot/