Using the CA-3 start code

*The following steps can easily take some time, but when completed, and if you follow the test-first strategy, you know that you always have a (deployed) backend that works.*

*Some of the exam questions will require you to deploy your code. If you use your CA-3 project as the start code for these exercises this should be a 2-minutes (max) operation.*

### Preparing your Digital Ocean Droplet

Before you start you must have a Droplet with Tomcat, MySQL and Nginx as explained [here](https://docs.google.com/document/d/1pP1eLz1r-gxPhzzZcEyhQMKIiv_kxFkQKZu_XC8IjFg/edit?usp=sharing).

If you already have a droplet you are comfortable with, use that. If not, set up a droplet as explained in the document above, and KEEP IT, at least until after the exam.

### Clone the start code for the CA-3

Clone [this projec](https://github.com/cphdat3sem2019spring/semester-seed.git)t as the start code for CA-3. Delete the .git folder and do your own git init

Create a new github project, and push your project to up there.

Important: The first thing you should do is to change the version of mysql-connector-java in the pom-file, according to the version of MySQL you have installed.

#### Run the unit tests

The provided unit-tests use an in-memory database, so they should run right out of the box like this:

mvn test

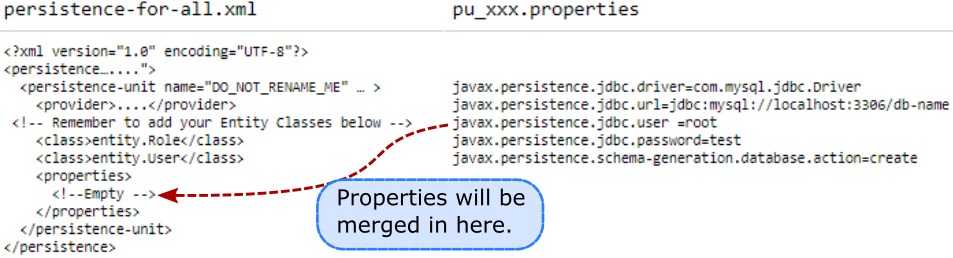
### Investigate the provided code (IMPORTANT INFO)

CA-3 requires a minimum of **five** databases as sketched below:

1. One for your local development (requires you to set up a local MySQL database)
2. One for unit testing (the code is set up to use an in-memory derby database which should run anywhere)
3. One for local integration testing (testing your REST-endpoints) (requires you to set up a local MySQL database)
4. One for integration testing on Travis. The code is set up with support for such a Travis generated database.
5. And finally, your “production” database running on your Digital Ocean Droplet

(*You will never set the last two manually, they will be set by the provided code or maven)*

To minimize the time you have to spend on this part, the code ships with a programmatic way to set up your persistence-files, which uses a replacement for the normal persistence.xml file, called persistence-for-all.xml and a number of property-files with the properties for each “normal” persistence-unit matching the databases listed above. This is sketched in the figure below.



With this setup, you should NEVER request an entity manager the usual way like this

Persistence.createEntityManagerFactory("pu\_name", null);

But always like this:

PuSelector.getEntityManagerFactory("pu")

The pu name must match the file name for the required property-file (pu → pu.properties) to create the matching persistence-unit.

Via Maven (and Travis via Maven) you can override which pu\_xxxx.properties file to use. On your Droplet, you can set a few environment variables to force the code to use your remote database.

See the file PuSelector.class for how this is done.

**Important:** *If you find this a bit strange, remember that properties in a pu\_XX-properties file always are merged into persistence-for-all.xml to give you a “normal” persistence unit.*

### Create local databases (one for development, and one for integration tests)

Usually, we git-ignore the persistence.xml file to allow all members in a group to have their own local database, with database name, user and password to their own liking.

In this setup, the files pu.properties and pu\_integration\_test.properties represents the local persistence-units, and as so *are git-ignored*.

a) Create two local MySQL databases and create the files (in resources/META-INF) pu.properties and pu\_integration\_test.properties that matches these databases.

**Hint:** use the two XXpu… files as templates for the steps above.

b) Test this setup by running this command which should run all the supplied unit- and integration tests.

mvn verify (ALL tests should pass, if they don’t, try to restart NetBeans)

## Setup Local Users and test the Security Functionality

a) Run the file (in utils): SetupTestUsers.java to set up a few users for your local development.

### Test the Security Functionality (copied from the original [security exercise](https://docs.google.com/document/d/1hC21WzWjNnQLy0eoLJB-HFRmVYpj-KtprHfmgwlyx50/edit?usp=sharing))

**a)** Test, and verify that you cannot access the two security protected endpoints:

* /api/info/user
* /api/info/admin

*Before you continue, check the code (in the rest package) to see what it takes to protect an endpoint*

**b)** Test the **POST**-Login Endpoint: **xx/api/login** via Postman to verify you are connected to the database. Add this JSON to the body, to include username and password with the request:

{"username":"user","password":"test"} **or** {"username":"admin","password":"test"}

Also, test with a non-existing user and a wrong password.

Verify that, for a successful login, the system returns a *token*. This is what you need to provide with all subsequent request to access protected resources.

**c)** Login, via postman as in step b) but this time copy the returned token into the clipboard.

**d)** Test the two endpoints, as in step a) Make sure to include the token with the request in a http-header “x-access-token”

Verify that it’s not enough to have a valid token, you must also have a valid role to be authorized

*Locate the integration test in IntegrationTest.java and verify that these tests does exactly what you did above, that is: verifies the login-behaviour.*

# Safely Deploy to your remote server

*These steps add two additions, both related to security.*

* *they remove the need for hardcoded database credentials for the database located on your droplet*
* *They remove the need for hardcoded Tomcat Credentials necessary for deployment via maven*

### Setup required *environment variables* on your remote Tomcat Server.

**a)** On your droplet, via the MySQL client (or locally via Workbench), create a database for the project called **seed** (if you want a different name, make sure to change into this name in the file pu\_production.properties). Also, unless you already have created (suggested for convenience) a user with access to all your databases as [suggested here](https://docs.google.com/document/d/1pP1eLz1r-gxPhzzZcEyhQMKIiv_kxFkQKZu_XC8IjFg/edit#bookmark=id.kqlbdmcwau5i), create a user for the database (you need the credentials for the next step)

**b)** On your droplet, open the file tomcat8 with nano:

sudo nano /etc/default/tomcat8

Add these lines to the bottom of the file (replace text in yellow):

SERVER="PRODUCTION"

USER="YOUR\_DATABASE\_USER"

PASSWORD="YOUR PASSWORD FOR THE PRODUCTION DB"

c) Login to your droplet via SSH

e) Open the file setenv.sh with nano (this will create, if it does not exist)

sudo nano /user/share/tomcat8/bin/setenv.sh

**d)** Restart Tomcat: sudo systemctl restart tomcat8

*What this did was to set three environment variables, SERVER, USER and PASSWORD, which your code, running on Tomcat (on your droplet) can read. This eliminates the need for hard-coded credentials in the persistence unit. See the class PuSelector.java if you want to know how. When your code requests an EntityManagerFactory via the PuSelector class it will use the file pu\_production.properties and merge the username and password read from the environment variables into the persistence-unit.*

### Deploy the project

We will do this, similar to what we have done before, but *without hardcoded Tomcat Manager Credentials*.

**a)** Open the pom file and locate this line, somewhere near the top:

<remote.server><https://www.sem3.mydemos.dk/manager/text></remote.server>

Replace the URL with the one for your own server.

*Locate the Tomcat7-maven-plugin in the pom-file, and observe how the property you just set is used. Also, observe that that username and password are read the same way, but* ***NOT******set in the file****.* 

**b)** Now let's see how we can supply the username and password, required for Tomcat7-maven-plugin to deploy your project.

Open a terminal and type this line (replace with the credentials you normally use for ../manager/html) :

mvn -Dremote.user=admin -Dremote.password=password tomcat7:deploy

**c)** Verify (manually via ..../manager/html, that it has been deployed)

## Creating the tables and initial users on the production Database

Next step is to create the tables and some initial users on the remote database. For the first part, you could write a script, but since we are using JPA, the simplest way is to let JPA create the tables (using the create or drop-and-create strategies) in the properties file.

**a)** Navigate to the api/info/all endpoint to trigger a database operation and verify that the tables were created. Use the script below to set up a few users (change usernames and passwords)

USE seed;

INSERT INTO users (user\_name, user\_pass) VALUES ('user', 'test');

INSERT INTO users (user\_name, user\_pass) VALUES ('admin', 'test');

INSERT INTO roles (role\_name) VALUES ('user');

INSERT INTO roles (role\_name) VALUES ('admin');

INSERT INTO user\_roles (user\_name, role\_name) VALUES ('user', 'admin');

INSERT INTO user\_roles (user\_name, role\_name) VALUES ('admin', 'admin');

**b)** Login to your droplet via SSH

**c)** Repeat the steps from the section “Test the Security Functionality” and verify that you can log in via Postman (requires that you have inserted some test users as explained above).

### Maven commands for the project.

Run unit tests only: mvn test

Run unit and integration tests: mvn verify

Run all tests and deploy: mvn -Dremote.user=admin -Dremote.password=password tomcat7:deploy

## Preparing the project for CI and Continuous Deployment.

**a)** If not already done clone and push the project to your own repository

**b)** Login to Travis, locate your project and “pull the switch” to make it available to Travis

***Travis and Databases for testing****. Remember from the previous setup that the tests require two databases. The in-memory database, which will work fine also on Travis, and also a MySQL database used for integration tests. Fortunately, Travis makes it very simple to automatically set up a MySQL database which can be used for our tests. The start code ships with this ready to go (see the files .travis.yml and pu\_travis\_integration\_test.properties). Read more here* ***ONLY*** *if you want all the* [*details*](https://docs.travis-ci.com/user/database-setup/)*.*

### Safely Deploy from Travis

The only thing missing, to repeat all the previous steps via Travis is to find a way to pass in the values for deployment, see this line in .travis.yml:

- mvn -Dremote.user=$REMOTE\_USER -Dremote.password=$REMOTE\_PW tomcat7:deploy

**a)** Open your project on travis-ci.org and define the two variables **REMOTE\_USER** and **REMOTE\_PW** with the values necessary to deploy to your Tomcat as described [here](https://docs.travis-ci.com/user/environment-variables/#defining-variables-in-repository-settings)

**b)** commit and push your project, and verify that Travis will execute your tests, and if all green, deploy your project to your Tomcat Server

## Handle Passwords the right way

Repeat the steps from last weeks security exercise, and change the code to Hash and Salt passwords before they are stored in the database.

## Create the initial users and roles with hashed passwords

We just need one more thing to get started, and that is to add some initial users with hashed/salted passwords.

#### One way to do this is via the steps given below:

1) Locally, from within Netbeans run the main method in SetupTestUsers.java (make sure to change usernames and passwords to real “secret” values)

2) Verify that the hashed passwords were written to your local database

3) DELETE the values for usernames and Password in SetupTestUsers.java BEFORE you commit (or anyone with access to GIT can see your secret credentials)

4) Delete (on your remote database) the existing users from the database (those from before you added hashing)

5) Run the script below up against your remote database, with the user names, and hashed passwords, taken from your local database (created in step-1).

USE seed;

-- Delete all previous data first

DELETE FROM user\_roles WHERE NOT user\_name='xxx';

DELETE FROM roles WHERE NOT role\_name ='xxx';

DELETE FROM users WHERE NOT user\_name ='xxx';

INSERT INTO users (user\_name, user\_pass) VALUES ('user', 'Hashed-from-your-local-db');

INSERT INTO users (user\_name, user\_pass) VALUES ('admin', 'Hashed-from-your-local-db');

INSERT INTO roles (role\_name) VALUES ('user');

INSERT INTO roles (role\_name) VALUES ('admin');

INSERT INTO user\_roles (user\_name, role\_name) VALUES ('user', 'user');

INSERT INTO user\_roles (user\_name, role\_name) VALUES ('admin', 'admin');

6) Remember, JPA uses by default caching, so you need to restart your Tomcat Server to see this take effect.