

# Exam for PG5100

**11:55am Wednesday 7<sup>th</sup> June – 11:55am Friday 9<sup>th</sup> June**

This assignment is worth 100% of your final grade for PG5100. The assignment has to be submitted on Its Learning by Friday 9<sup>th</sup> of June morning at 11:55am, under the “Exam delivery” folder.

The exam assignment will have to be zipped in a zip file with name pg5100\_<id>.zip, where you should replace <id> with your own student id, eg pg5100\_123456.zip. No “rar”, no “tar.gz”, etc. You need to submit all source codes (eg., .java and .xml), and no compiled code (.class) or IDE files (.idea). In the past, I got a 175MB and a 214MB files as a submission, containing everything, compiled code included... you can guess what grade the student got... an F. You really want to make sure to run a “mvn clean” before submitting.

The delivered project should be compilable with Maven 3.x with commands like “mvn install -DskipTests” directly from your unzipped file. Compilation failures will **heavily** reduce your grade. All tests should run and pass when running “mvn verify -P selenium”.

The assignment is divided into several parts/exercises. The parts are incremental, ie building on each other, for a total of 100 points.

Note: during the exam period, I will NOT answer to any email. However, I will answer questions on the Its Learning forum of the course. I will answer questions regarding possible misunderstanding in the exam's instructions, or more general questions like “can we use library X?”. Questions like “How can we do Y?” will be of course left unanswered... At any rate, during the exam you should check the discussion forum often, as I might post some clarifications or other messages of interest.

You can (and should when appropriate) reuse code from

[https://github.com/arcuri82/testing\\_security\\_development\\_enterprise\\_systems](https://github.com/arcuri82/testing_security_development_enterprise_systems)

for example the pom files. You will also notice that the exam has many similarities with the exercises you have done during the course. You are allowed to re-use / adapt your own code, but of course not the one from other students...

The source code of the solution will be posted on the main Git repository of the course after the exam is finished.

Easy ways to get a straight F with no appeal:

- have production code (not the test one) that is exploitable by SQL injection
- submit a solution with no test at all, even if it is working
- submit your delivery as a rar file instead of a zip (yes, I do hate rar files)
- submit a far too large zip file. Ideally it should be less than 10MB, unless you have (and document) very good reasons for a larger file.

Easy ways to get your grade reduced (but not necessarily an F):

- submit code that does not compile
- do not provide a readme.txt file (more on this later) at all, or with missing parts
- skip/miss any of the instructions in this document (e.g., how to name the zip file)

The exam should NOT be done in group: each student has to write the project on his/her own. During the exam period, you are not allowed to discuss any part of this exam with any other student or person. The only exception is questions to me in the PG5100 discussion forum, as discussed earlier. Failure to comply to these rules will result in an F grade and possible further disciplinary actions.

## The MyCantina Application

In this exam, you will need to create a basic web application called “My Cantina” (the actual name does not really matter...). In short, chefs can register dishes on a database, and create menus for different days based on those existing dishes. Regulars users should be able to see the different menus for the different days.

The application should be implemented on the JEE stack, eg JPA, JTA, EJB, and JSF, as shown in class. For this exam, you do NOT need to write any CSS or JavaScript, although you will have to edit some HTML.

Note: when I show screenshots, it is only to clarify the text. You do not have to perfectly 100% match the layout/size of the widgets.

The project should be structured in 3 Maven submodules, in the same way as in the jsf/jacoco and exam example modules shown in class: “backend” (containing Entity, EJB, and all other needed classes), “frontend” (JSF beans and XHTML) and “report” (for aggregated JaCoCo report). Note, if you copy and paste those pom files, you will have to modify the root pom file of your project to be self-contained (ie no pointing to any parent).

You should use Maven to compile a WAR file that should be deployable on WildFly 10. You need to configure Maven to be able to start WildFly and automatically deploy the WAR when running “mvn wildfly:run” from the “frontend” folder (after doing a “mvn install” on the root folder). You have to use an embedded database (eg H2).

Testing should be based on Arquillian (for EJB) and Selenium (for end-to-end) using Chrome. You can make the assumption that the Chrome drivers are available under the user's home folder (as done in class and in the Git repository). All tests should be automatically started (and connecting to WildFly if necessary) when running “mvn clean install -P selenium” from the root folder.

Tests should be independent, ie they should not fail based on the order in which they are run, and neither they should fail if run more than once (e.g., when run several times from IDE without restarting WildFly).

You need to provide a “readme.txt” file (in same folder as the root pom.xml) where you briefly discuss your solution: eg, how you structured the code (eg, “@Entity classes are in package X”), and any other important info you want to provide.

If you do not attempt to do some of the exercises, state so in the “readme.txt” file (this will make me correct the exam much faster), eg “I did not do exercises X, Y and Z”. Failure to do so will reduce your grade.

## (E1, 20pt) Backend

In the backend module, you need the following 2 entities:

- *Dish* with name and description
- *Menu* with a date (year, month and day) and set of dishes. Dates should be unique. You might want to use a `LocalDate` object to represent them. A valid menu must contain at least one dish.

Add *reasonable* constraints to all the fields of those entities (e.g., a name must not be blank or too long).

You need to have EJBs to provide the following functionalities:

- create a dish
- get all existing dishes
- create a menu
- get the “current” menu. This is defined as: if there is a menu for today, get that. If not, look at the closest menu in the future. If none in the future exists, get the closest in the past.
- get the closest menu in the future after a given date
- get the closest menu in the past before a given date

## (E2, 15pt) Arquillian Tests

Create Arquillian tests for the EJBs you developed. You need the following tests (names are important, and also their order in the files, so that I can easily find them when marking your exam). Names should be hopefully self explanatory...the actual content is up to you (unless otherwise stated), as long as it is somehow related to the name of the test:

- `testCreateDish`
- `testCreateTwoDishes`

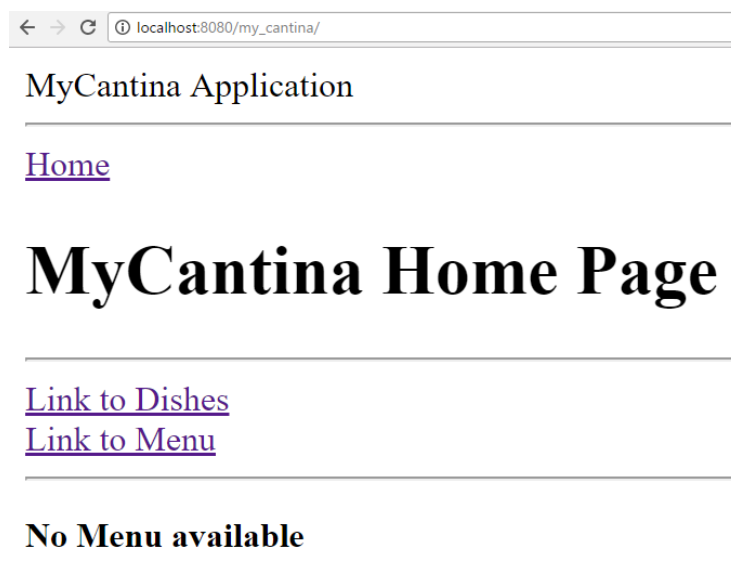
- testCreateMenuWithNoDish
- testGetCurrentMenu
- testGetAbsentPreviousMenu (ie try to retrieve a menu from a previous date when there is none)
- testGetAbsentNextMenu
- testGetPreviousMenu
- testThreeMenus
  - create 3 menus, for today, tomorrow and yesterday
  - verify that today has tomorrow as next, and yesterday as previous
  - verify that tomorrow has no next, and today as previous
  - verify that yesterday has no previous, and today as next

## (E3, 5pt) Home Page

After starting WildFly, the developed application should be accessible at

[http://localhost:8080/my\\_cantina](http://localhost:8080/my_cantina)

Note: this URL is **VERY** important (it makes my marking much easier/faster). Misspelling it will negatively impact your grade. The home page should look like:

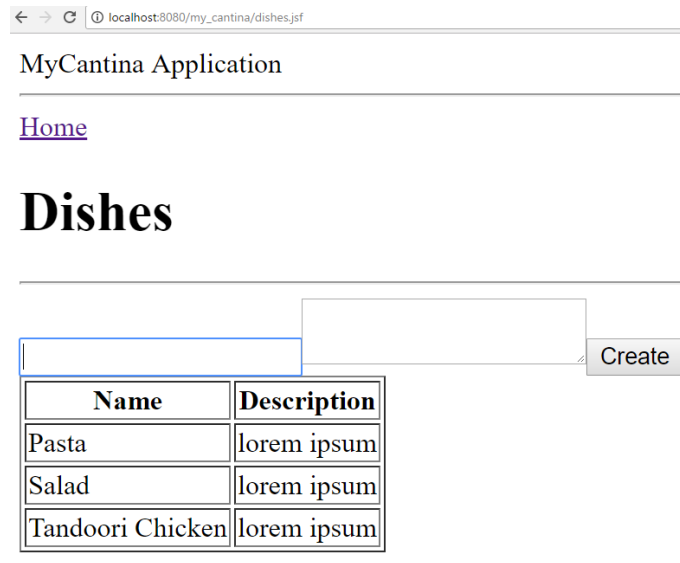


The link “Home” should come from a layout.xhtml template, which must be reused on all the pages. As the application is just started, no menu should be displayed yet. You need a text message to specify it, eg “No Menu available”.

## (E4, 5pt) Dishes

From the home page, clicking to the link for Dishes should lead you to a page where you can create

a new dish (name and description), plus showing all the existing ones. For example:



MyCantina Application

[Home](#)

## Dishes

Name	Description
Pasta	lorem ipsum
Salad	lorem ipsum
Tandoori Chicken	lorem ipsum

### (E5, 5pt) Menu

From home page, the link to Menu should lead to a page where you can set a new menu. The date is going to be passed as a string (there are better ways to do it, but let's keep it simple...). All existing dishes should be visible, with a checkbox to decide whether they should be included or not in the menu. For example:



MyCantina Application

[Home](#)

## Menu

2017-01-01

Name	Include
Pasta	<input checked="" type="checkbox"/>
Salad	<input type="checkbox"/>
Tandoori Chicken	<input checked="" type="checkbox"/>

If the menu is successfully created (i.e., when clicking the Create button), the browser should be redirected to the home page. If not, stay on current one.

### (E6, 5pt) Show of Menu

The home page should show the content of the current menu. For example:

[Home](#)

# MyCantina Home Page

---

[Link to Dishes](#)

[Link to Menu](#)

---

[Show default](#)

## Menu for 2017-01-01

Name	Description
Pasta	lorem ipsum
Tandoori Chicken	lorem ipsum

---

If there is any menu after the displayed one, and only then, there should be a “next” link to display it (in the current page). Samewise, you need a “previous” link if there is a previous menu before the currently displayed one. The “default” link will display the “current” menu. For example (given only 3 menus in the database):



The screenshot shows the web application running in a browser at localhost:8080/my\_cantina/menujsf. The page layout is identical to the one above, but with additional navigation links. Below the 'Show default' link, there are two more links: 'Show next (2017-11-11)' and 'Show previous (2016-03-03)'. The table and menu title remain the same.

MyCantina Application

---

[Home](#)

# MyCantina Home Page

---

[Link to Dishes](#)  
[Link to Menu](#)

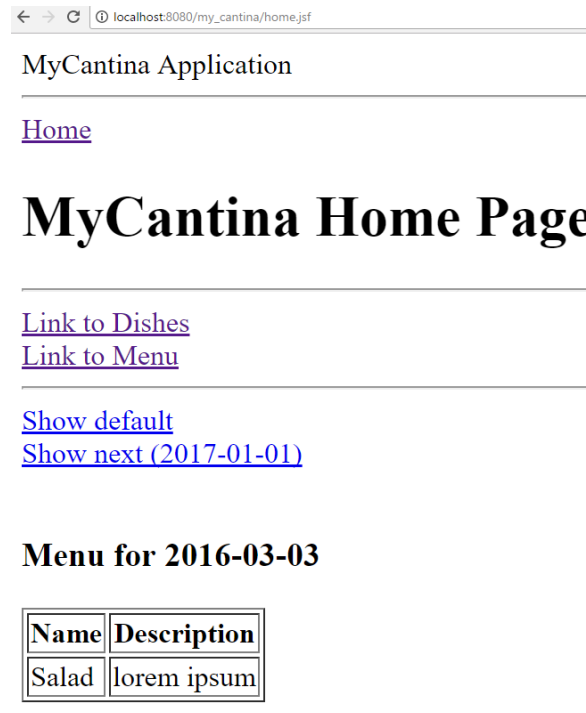
---

[Show default](#)  
[Show next \(2017-11-11\)](#)  
[Show previous \(2016-03-03\)](#)

## Menu for 2017-01-01

Name	Description
Pasta	lorem ipsum
Tandoori Chicken	lorem ipsum

---



## (E7, 10pt) Selenium

For each of the 3 web pages, create a Page Object. Create a test file called MyCantinaIT with the following Selenium tests. Selenium tests should be run from maven only when the “-P selenium” profile is activated.

- testHomePage
  - go on home page and verify it is indeed the home page
- testCreateDish
  - from home page, go to the dishes page
  - choose a new unique name
  - verify no dish exists on the page with that given name
  - create a dish with that name
  - verify that dish is now displayed
- testMenu
  - create 3 new dishes
  - go on the Menu creation page
  - verify that those 3 dishes are selectable
  - select 2 of them

- create a menu for today
- on home page, click “show default” to make sure that the created menu for today is displayed
- verify that the date of the displayed menu is correct (ie today)
- verify that the 2 selected dishes are displayed in the menu, and only those 2.
- testDifferentDates
  - create 3 menus: one for today, one for tomorrow, and one for yesterday
  - on home page click show default to visualize the menu of today, and verify it
  - click next, and verify that the menu of tomorrow is displayed
  - click previous, and verify that the menu of today is displayed
  - click previous, and verify that the menu of yesterday is displayed

## **(E8, 5pt) JaCoCo**

Configure JaCoCo in such a way that, when running “mvn clean verify -P selenium” from the root folder, a test report should be generated under the report/target folder. You need an average instruction coverage of AT LEAST 90%. If it is lower, add more tests. Note: the score for this exercise is awarded if, and only if, all the previous exercises are (mostly) completed.

## **(E9, 30pt) Extra**

In the eventuality of you finishing all of the above exercises, and only then, if you have extra time left you should add functionalities/features to your project. Those extra functionalities need to be briefly discussed/listed in the “readme.txt” file (e.g., as bullet points). Each new functionality must have at least one Selenium test for it. Note: in the marking, I might ignore functionalities that are not listed in the readme. What type of functionalities to add is completely up to you. Example of possible functionality and description: “possibility to delete dishes from the GUI, but only if such dish is not used in any existing menu. The Selenium test for it is called X and can be found in file Y”.

**THIS MARKS THE END OF THE EXAM TEXT**