|  |  |
| --- | --- |
| **Date:** | 10th September 2015 (Thursday) |
| **Time:** | 2000 |
| **Venue:** | SMUX Meeting Room 2-2 |
|  |  |
| **Attendees:** | Jennifer, Chu Qian, Remy, Shing Hei |
|  |  |
| **Agenda:** | 1. Learning framework |

**1. Learning framework**

Click on new file > Maven > Project with existing POM

There will be pom.xml which is a xml file with all dependencies that we need

\*Note: Look at ‘Files’ tab for Maven project to access the raw directory structure.

When Running application: main source files stored at src /main/Java/netgloo. We will be using java version 1.7 for our project. OpenShift cartridges operates on 1.7 by default. Netgloo is just the name that u have created for yourself.

**-Pom.xml-**

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

<finalName>ROOT</finalName>

</build>

<packaging>war</packaging>

**//specified under build**

**//allow your applications to build into a jar or war file . In this case, it is a war file listed under the packaging tab**

<repositories>

<repository>

<id>spring-milestone</id>

<url>https://repo.spring.io/libs-release</url>

</repository>

</repositories>

**//the place where we download the jar files from (our dependencies)**

Introducing annotations: @ symbol

**Application.java (src/main/java/netgloo)**: @SpringBootApplication – indicates the starting file to be run, similar to index.jsp page

# Controllers:

@Controller: Stereotype for controller files

@RequestMapping (value = “/XXX”): the method will be called when XXX is called in the URL

@RequestMapping(value = "/tester", method = RequestMethod.GET) : an option of POST or GET for RequestMethod

@ReponseBody: Sending messages back

\*Note: If there is no @ReponseBody, it will search for view pages which is under resources/template

=> **Main/Resources/Templates: View files**

@Autowired

private UserDao userDao;

@Autowired: the instance will be created for use directly, unlike in OOAD. (Auto-instantiation)

# Config:

@Configuration: (main/netgloo/configs/Config.java)

**//Spring will run it from the start**

@Configuration

@EnableWebMvc

public class Config extends WebMvcConfigurerAdapter{

@Bean

public ViewResolver getViewResolver() {

InternalResourceViewResolver resolver = new InternalResourceViewResolver();

resolver.setPrefix("/resource/templates");

resolver.setSuffix(".html");

return resolver;

}

@Override

public void configureDefaultServletHandling(

DefaultServletHandlerConfigurer configurer) {

configurer.enable();

}

}

**//let Spring understand where to find the file; only specified for view files**

**//setSuffix: the file type**

Sequence of Flow: (/hex) User Request -> Controller (find the mapping of /hex)

**Spring Framework searches through controller NOT the view files.**

@Configuration: inside **DatabaseConfig.java** (combination of database, hibernate configurations- where exactly will the database function be called)

// An EntityManager will be automatically injected from entityManagerFactory

// setup on DatabaseConfig class.

@PersistenceContext

private EntityManager entityManager;

entityManagerFactory method found in **DatabaseConfig.java**

**(Java Persistence API) JPA**: A standard of ORMs.

**Object Relations Mapping (ORM**): e.g. Hibernate- don’t need to create SQL queries; let you use codes, instead of SQL statements; contains functions that have SQL queries mapped to it

# DAO:

@Repository: DAO files

@Transactional: for you to do commits to database; a way for Spring to interact with database

@PersistenceContext : whatever you do, will always be committed to the database

**Main/Resources/Application.properties**

# Hibernate

hibernate.dialect: org.hibernate.dialect.MySQL5Dialect

hibernate.show\_sql: true

**//will show the SQL statements**

hibernate.hbm2ddl.auto: update

**//update the schema; telling the database to keep the data there but add on to the current records.**

**//or** ‘create’**: creates the schema, destroying previous data (when we are accessing it for the first time)**

entitymanager.packagesToScan: netgloo

**//netgloo: the database name that you have named**

Calls method from **entityManager: contains**, **persist** (throwing the object as is inside the parameter; sequence doesn’t matter; insert based on the column name), **remove, merge** (for updating the input data)

@SuppressWarnings: to suppress the error messages – but NOT using it for project

**-UserDAO.java-**

public User getByEmail(String email) {

return (User) entityManager.createQuery(

"from User where email = :email")

.setParameter("email", email)

.getSingleResult();

}

**:email – values not specified, until setParameter (otherwise SQL injections)**

public User getById(long id) {

return entityManager.find(User.class, id);

}

**User.class -> specify the table that you are going to search from, because only id is given**

# Entity:

@Entity

@Table(name = “users”): Plural of the entity class name (Person -> People)

@Id : Primary Key

* composite keys? Set multiple @id

@NotNull

Let Hibernate create the tables for us. Table will be reflected in our SQL.

**ThymeLeaf**

<p **th**: text=”’Hello, ‘ + ${name} + ‘!’”/>

If using thymeleaf, need to specify its dependencies at **pom.xml**. **pom.xml is situated at the root folder.**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

Dependencies on thymeleaf need to be included before use.

**Bootstrap:**

1. app, app-lookup, demo

2. location, location-hist

3. delete-location

**Additional Data:**

App, + demo, + location, + location-delete

The meeting was adjourned at 10.30 pm. These minutes will be circulated and adopted if there are no amendments reported in the next three days.

Prepared by,

Tang Shing Hei

Vetted by,

**Remy Ng Zheng Yao**