Timesheet RESTful Interface

COMP3910

Assignment 3 Report

Tony Pacheco & Danny Di Iorio

December 5, 2018

Table of Contents

[Purpose 3](#_Toc531786044)

[API Interface Requirements 3](#_Toc531786045)

[Input Data 3](#_Toc531786046)

[Request Requirements 4](#_Toc531786047)

[Use Case Requirements 4](#_Toc531786048)

[Use Case Diagram 5](#_Toc531786049)

[REST API Specification 6](#_Toc531786050)

[Supported Request Calls 6](#_Toc531786051)

[User Authentication 6](#_Toc531786052)

[Example Requests 7](#_Toc531786053)

[Return Representation 8](#_Toc531786055)

[Operating Environment 9](#_Toc531786056)

[Build Instructions 9](#_Toc531786057)

[Configurations 9](#_Toc531786058)

[Importing Project into Eclipse 9](#_Toc531786059)

[Set Up Database and Datasource 9](#_Toc531786060)

[Build and Deploy Project on Server 10](#_Toc531786061)

[Postman 10](#_Toc531786062)

[User Accounts 11](#_Toc531786063)

[Extra Credit 11](#_Toc531786064)

[Class Diagram 12](#_Toc531786065)

[Database Entity Relational Diagram 13](#_Toc531786066)

# Purpose

The Timesheet RESTful service will provide an interface for logged in users to view, edit, add, and remove timesheet and employee data, through various HTTP requests.

# API Interface Requirements

## Input Data

The service will accept and store the following input data in its database:

1. User (Employee) Data for each user in the system:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Required (yes/no) | Description |
| First name | String | Yes | Employee’s first name |
| Last name | String | Yes | Employee’s last name |
| Employee number | Integer | Yes | E number assigned to employee |
| Username | String | Yes | Employee’s username to log in |
| Password | String | Yes | Login password |
| Admin status | Bit (0 or 1) | Yes | Denotes if employee is an admin (1) or not (0) |

1. Timesheet Data:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Required (yes/no) | Description |
| Employee number | Integer | Yes | Employee who owns timesheet |
| Week number | - | - | Week number in year, *not stored in database* |
| Timesheet date | Date | Yes | The date on which the week ends – Friday |
| Timesheet ID | Integer | Yes | ID for timesheet |
| Overtime | Integer | No | Overtime hours worked |
| Flex time | Integer | No | Flex hours for week |

1. Timesheet Row Data:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Required (yes/no) | Description |
| Project number | Integer | Yes | Project worked on by employee |
| Work package ID | String | Yes | WP associated with project |
| Hours for each day (Sat-Fri) | Decimal | No | Number of hours worked on each day |
| Timesheet ID | Integer | Yes | ID for timesheet containing row |

## Request Requirements

* All POST requests do not require the ID to be in the message body as this is generated by the system
* Creating new timesheets requires a POST request for the new timesheet, followed by additional POST requests to add new rows to it
  + The timesheetId that the rows are associated with must be included in the payload
  + As many new rows can be added as desired
* All user/employee related requests will only be available to users with administrator access

# Use Case Requirements

The two types of users which the RESTful web services will support are general users and the system’s administrator(s). General users will be able to create, edit, and delete their own timesheets. Administrators will be able to do anything that general users can do, as well as manage the creation, modification, and deletion of other user accounts. Administrators can also view timesheets from all users in the system.

# Use Case Diagram



# REST API Specification

## Supported Request Calls

|  |  |  |
| --- | --- | --- |
| URI | HTTP Method | Payload Format |
| /auth | POST – send username and password and receive token if it is valid combo | JSON |
| /timesheets | GET – list all timesheets | JSON |
|  | POST – submit a new timesheet | JSON |
| /timesheets/{timesheetId} | GET – get an existing timesheet | JSON |
|  | PUT – update an existing timesheet | JSON |
|  | DELETE – delete an existing timesheet & associated rows | - |
| /row | POST – add a new timesheet row to existing time | JSON |
| /row/{timesheetRowId} | PUT – update an existing timesheet row | JSON |
|  | DELETE – delete a timesheet row | - |
| /employees | GET – list all employees/users | JSON |
|  | POST – create a new employee/user | JSON |
| /employees/{empNumber} | GET – get an employee profile | JSON |
|  | PUT – update an existing employee | JSON |
|  | DELETE – delete an employee | - |

## User Authentication

Logging into the system will require the client to send an auth request with the username and password in the request body. A successful login request will authorize the employee to use the service for up to one hour, and the response will return a user token (string). This token will need to be sent in the header of all subsequent requests until it times out or the user stops using the service.

Once the token expires, the employee’s access is revoked, and they must log in again to retrieve a new one.

## Example Requests

### Auth – POST

{

"username": "tonyp",

"password": "pass"

}

***Employee – POST (***empNumber ***not required)***

{

"lastName": "Pacheco",

"firstName": "Tony",

"userName": "tonyp",

"password": "pass",

"isAdmin": true

}

***Employee – PUT***

{

"lastName": "Link",

"firstName": "Bruce",

"userName": "brucel",

"empNumber": 3,

"password": "password",

"isAdmin": false

}

Timesheet – POST (timesheetId not required)

{

"empNumber": 1,

"endWeek": "Nov 23, 2018",

"overtime": 0,

"flextime": 0

}

Timesheet – PUT

{

"empNumber": 1,

"endWeek": "Nov 30, 2018",

"overtime": 0,

"flextime": 0,

"timesheetId": 2

}

Timesheet Row – POST (timesheetRowId not required)

{

"projectId": 45,

"workPackage": "std",

"notes": "some notes",

"sunHours": 0,

"monHours": 8,

"tueHours": 8,

"wedHours": 8,

"thuHours": 8,

"friHours": 8,

"satHours": 0,

"timesheetId": 2

}

Timesheet Row – PUT

{

"projectId": 45,

"workPackage": "std",

"notes": "some notes",

"sunHours": 0,

"monHours": 8,

"tueHours": 8,

"wedHours": 8,

"thuHours": 8,

"friHours": 8,

"satHours": 0,

"timesheetId": 2,

"timesheetRowId": 20

}

## Return Representation

Successful GET methods will return HTTP status code 200 (OK). If the resources cannot be found, the method will return 404 (Not Found).

Successful POST methods to add a new resource will return HTTP status code 201 (Created) along with a representation of the newly added resource in the response body. If the client puts invalid data into the request, the server will return HTTP status code 400 (Bad Request).

Successful PUT methods to edit a resource will return HTTP status code 200 (OK) along with a representation of the newly edited resource in the response body. If there is no result to return, ie no resources are updated, the method will return HTTP status code 204 (No Content) with no response body. If the client puts invalid data into the request, the server will return HTTP status code 400 (Bad Request).

Successful DELETE methods will return HTTP status code 204, indicating that the process has been successfully handled. If the resource doesn't exist, the server will return HTTP status coded 404 (Not Found).

# Operating Environment

The operating environment will be split up into three layers: *middleware* *tier*, *business objects tier*, and *persistence tier*.

The *middleware tier* consists of a RESTful interface, which provides request services for various GET, PUT, POST, and DELETE requests that interact with data stored in the database. Users will be required to successfully send a login request and receive a valid token prior to any interaction with the REST services.

The *business objects tier* will use JPA session and entity beans to handle the RESTful interface functionality and represent the data within the database, respectively. We have designed entity beans to represent each database table, each having overridden toString methods to print as JSON objects using the GSON library. GSON builder is used in PUT and POST methods to create EJB objects from the JSON payload.

The *persistence tier* consists of a MySQL relational database designed by us. The database will be connected to the other layers with a data source running on JBoss Wildfly 13 server.

# Build Instructions

## Configurations

* Java Development Kit 8
* Eclipse Java EE IDE
* Wildfly 13 and JBOSS\_HOME defined
* MySQL 8
* MySQL user “timesheetuser” with password “password”
* MySQL Workbench
* Postman desktop application (<https://www.getpostman.com/apps>)

## Importing Project into Eclipse

1. Unzip submitted zip file into your chosen directory
2. In Eclipse, select “File -> Import…”
3. Select “Existing Projects into Workspace”, select next
4. Select “Browse” beside “Select root directory” dropdown and locate the project directory
5. Select the project directory and press “Select Folder”
6. Select the check box for our project “COMP-3910-Assignment-3”
7. Select “Finish”

## Set Up Database and Datasource

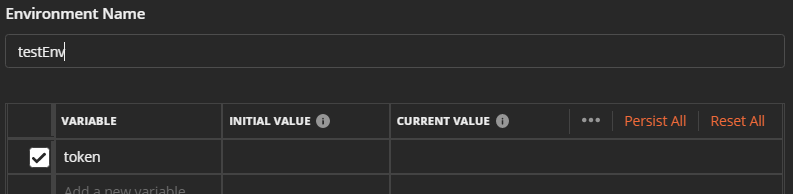
1. Log in to your MySQL database with root user access
2. Create the database using the create.sql files, ***file is located in ‘Documentation’ project folder***
   1. Open MySQL Workbench and open the create.sql file, OR
   2. Run the command “source create.sql” using command line
   3. This creates the timesheet database and grants timesheetuser all access
   4. Verify that the database has been created with some sample data inserted
3. Add datasource to Wildfly with the following configuration:
   1. JNDI: java:jboss/datasources/timesheetDB
   2. Choose MySQL as driver
   3. JDBC Connection URL: jdbc:mysql://localhost:3306/timesheet
   4. Enter username “timesheetuser” and password “password”

## Build and Deploy Project on Server

1. In Eclipse, stop WildFly server and delete any previously deployed versions of this project
2. Start the WildFly server
3. Run “Clean” on the server to clear any past versions
4. Right click on the project folder in Eclipse
5. Select “Run As” -> “Run on Server”

## Postman

1. Open Postman desktop application
2. Select “Import” -> “Import File” -> “Choose Files”
3. Navigate to the project directory, go to Documentation -> Postman
4. Select “IS Assignment 3.postman…” json file
5. Confirm the entire collection of saved requests has loaded (14 requests)
6. Click the eye button  in the top right beside “No Environment” dropdown
7. In the menu that opens, select “Add” to add an Environment
8. Enter an environment name – can be anything
9. Below under “VARIABLE” enter “token” (without the quotes)
10. Leave the rest of the fields empty



1. Select “Add”
2. Close the next dialog
3. From the “No Environment” dropdown, select the environment name you just added
4. Now that the environment and environment variable is created, when you log in using the “Login + Get Token” request, the returned token is stored in the environment and will be sent in the header of the other 13 requests automatically
5. All supplied tests can now be run in order from top to bottom with data supplied

# User Accounts

Admin:

* Username: tonyp
* Password: pass
* Username: dannyd
* Password: password

User:

* Username: brucel
* Password: pass

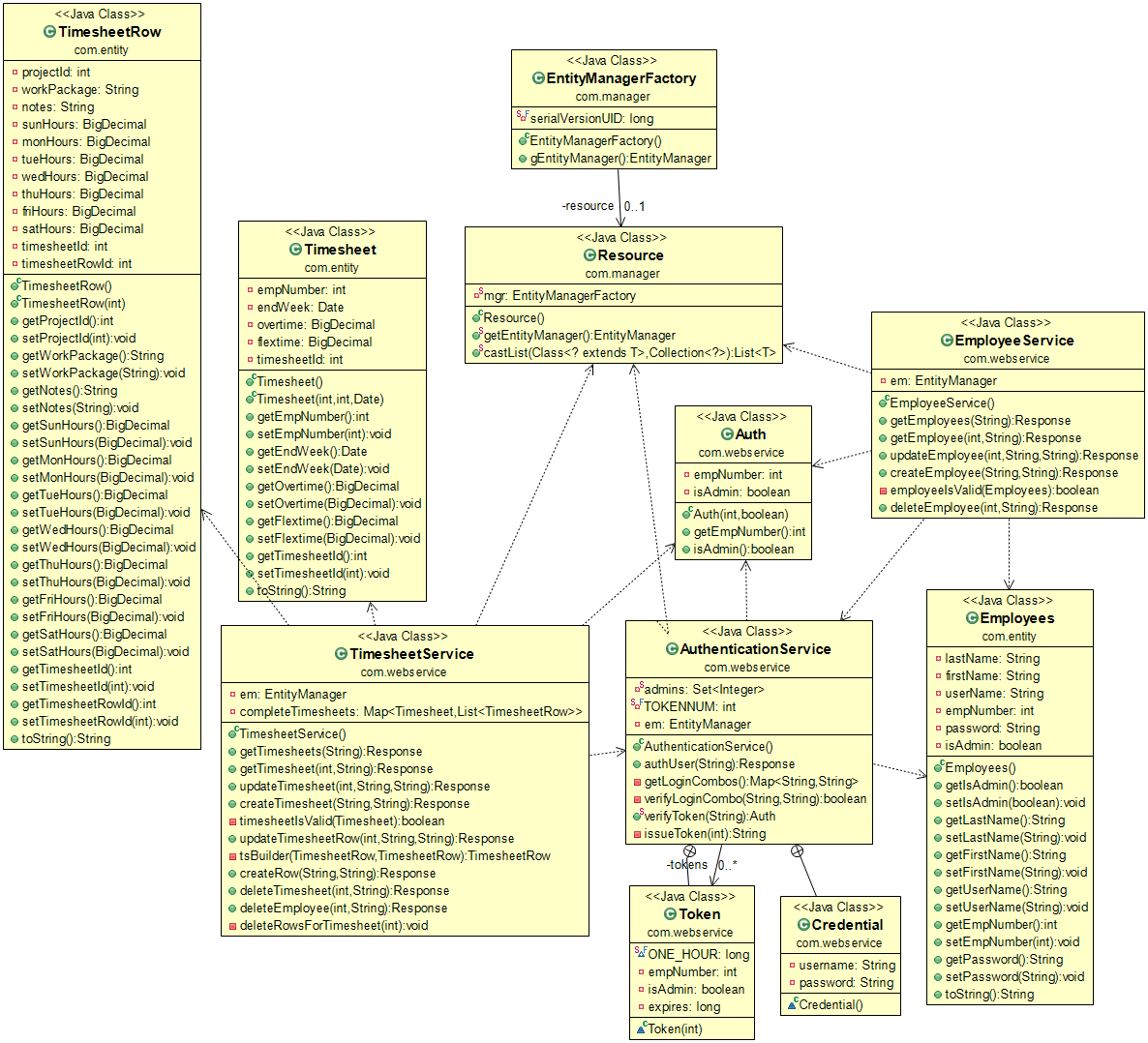
# Extra Credit

* Requirements and design document submitted Sunday Dec. 2nd
* Wrote and submitted full request collection in Postman that exercises the entire interface
  + Included documentation on how to set up and run
  + Created environment variable for authentication token so it does not need to be manually added to every request header

# Test Cases/Results

See PDF file accompanying this report for a detailed list of test cases and results

# Class Diagram



# Database Entity Relational Diagram

