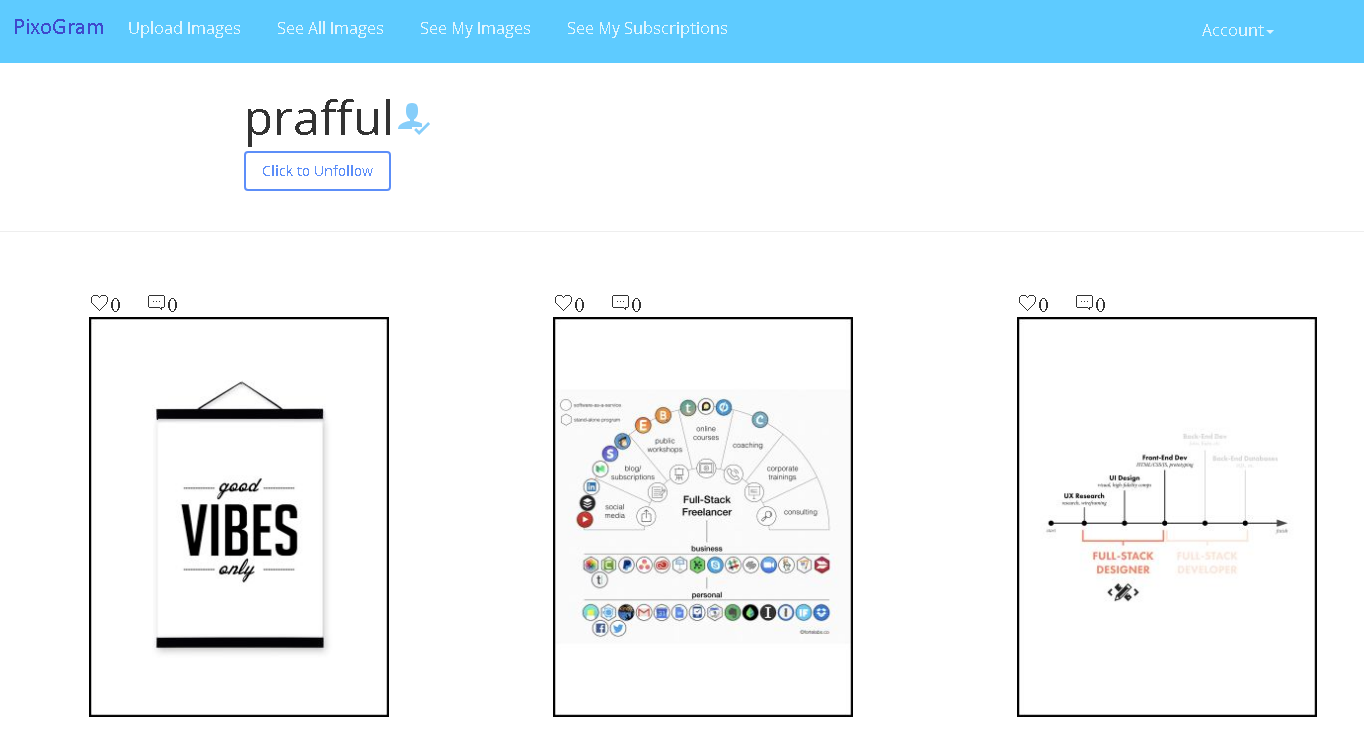
Java Microlayer

Prafful Daga  
IIHT

Micro Credential: Java Microlayer

Duration: 4 to 8 Hour



Java Business Layer for RESPONSIVE Single Page App (SPA) FOR SOCIAL PICTURE SHARING APPLICATION

This page is intentionally left blank.

Contents

[1 Important Instructions 3](#_Toc529801607)

[2 Business-Requirement: 4](#_Toc529801608)

[2.1 Problem Statement: 4](#_Toc529801609)

[2.1.1 Overview of Fields used in User Registration 5](#_Toc529801610)

[2.1.2 Overview of fields used for Add Content 7](#_Toc529801611)

[3 Java Layer 7](#_Toc529801612)

[3.1 Classes for User registration 7](#_Toc529801613)

[3.1.1 Classes: 7](#_Toc529801614)

[3.1.2 Description: 8](#_Toc529801615)

[3.2 Classes for Add Content 9](#_Toc529801616)

[3.2.1 Classes 9](#_Toc529801617)

[3.2.2 Description 9](#_Toc529801618)

[4 Solution Architecture 9](#_Toc529801619)

[4.1 User Registration Architecture 9](#_Toc529801620)

[4.2 Add Content Architecture 10](#_Toc529801621)

[5 Methodology 11](#_Toc529801622)

[5.1 Agile 11](#_Toc529801623)

[5.2 Continuous Integration 12](#_Toc529801624)

[5.2.1 As you code the solution: 12](#_Toc529801625)

[5.2.2 On conclusion of application development (before going to production) 12](#_Toc529801626)

[6 Technical Specification – Java Layer Solution Development Environment 13](#_Toc529801627)

[6.1 Java Layer 13](#_Toc529801628)

[6.2 Editors 13](#_Toc529801629)

[7 Development Workflow 14](#_Toc529801630)

[8 Important Instructions 15](#_Toc529801631)

[9 Assessment Deliverables 15](#_Toc529801632)

[10 Other Full Stack Layers 16](#_Toc529801633)

[10.1 UI Layer (Not Applicable for Present Case Study) 16](#_Toc529801634)

[10.2 UX Layer (Not Applicable for Present Case Study) 16](#_Toc529801635)

[10.3 Back End Layer (Applicable for Present Case Study) 16](#_Toc529801636)

[10.4 Middle Tier Framework Layer (Not Applicable for Present Case Study) 16](#_Toc529801637)

[10.5 ORM & Integration Layer (Not Applicable for Present Case Study) 16](#_Toc529801638)

[10.6 Database Layer (Not Applicable for Present Case Study) 16](#_Toc529801639)

[10.7 Ancillary Layer (Not Applicable for Present Case Study) 16](#_Toc529801640)

[10.8 Deployment & Infra(Not Applicable for Present Case Study) 16](#_Toc529801641)

# Important Instructions

1. Follow the design specifications mentioned in the case study. You are free to improvise certain specifications mentioned in the case-study. But, for each such improvisation, you should keep the concerned POC informed. **POC will get in touch with concerned team at IIHT.**
2. You should stay **motivated** to initiate such and specific communications as it may have positive influence on the evaluation scores.
3. Please make sure that your code does not have any compilation errors while submitting your case study solution.
4. The final solution **should be deployed in docker**.
5. The **docker image should start the console application (developed as per present case study) at command prompt**.
6. Implement the code using best design standards for:
   1. Variable declarations
   2. Class names
   3. Package names
   4. Code Refactoring

# Business-Requirement:

## Problem Statement:

**The PixoGram (Single Page Picture Sharing Application)** allows you to:

1. Register as a user
2. Login as a user
3. Retrieve password
4. Manage your user account
5. Login/Logout to/from your account on PixoGram
6. Add Content
   1. Upload single/multiple pictures, caption and description
   2. Upload single/multiple videos, caption and description
7. Manage Content
   1. Organize Picture in Gallery
   2. Organize Videos in Playlists
   3. Rename Pictures and Videos
   4. Edit Caption, Description, Comment
8. Social Features
   1. Use emojis in comment
   2. Like or Unlike comment, pictures and videos of other users
   3. Follow/Unfollow other users
9. Edit Pictures
   1. Apply effects to pictures (sepia, grayscale, etc.)
10. Hide Pictures/Videos
11. Activity/Newsfeed
    1. View activity log of user-activity on the PixoGram
12. Offline Functionality:
    1. Certain parts of the application should be available in absence of connectivity.
    2. Relevant areas on the screen should display “Connectivity Not Available”
13. BONUS REWARDS/SCORE Feature:
    1. To implement offline image upload functionality such that user can upload content when offline. It will sync with backend when connected.

**In this micro layer you will develop Java layer of the respective application using Java 8. Since, it is not possible to implement Java layer for all modules listed above, you should develop Java layer only for below modules:**

1. **Register as a user**
2. **Login as a user**
3. **Retrieve password**
4. **Manage your user account**
5. **Login/Logout to/from your account on PixoGram**
6. **Add Content**
   1. **Upload single pictures, caption and description**
   2. **Upload multiple pictures, caption and description**

### Overview of Fields used in User Registration

The application will consist of 7 fields. Given below are the fields and validation guidelines (as used in creation of UI. Some of the guidelines given for the fields in this section may not be applicable to the Java layer).

1. First Name:
   1. Should allow alphabets only
2. Last Name:
   1. Should allow alphabets only
3. Username
   1. Should allow mix of alphabets and number
   2. Username must not start with number
   3. Length of username should be between 8 to 12
4. Email
   1. Must allow email in valid email format
   2. Must not allow two @ symbols
5. Password
   1. Must be alphanumeric
   2. Should allow only following special characters- . # % $ !
   3. Length of password should be between 8 to 12
   4. Should contain at-least one capital alphabet
6. Confirm Password
   1. Should be like the above password
   2. Same validation rules should apply
7. Upload profile picture
   1. Upload the profile picture. Picture should be of dimension 200x200 before upload
8. Spreadsheet Wireframe: Empty form (Do not create in project. FYI only.)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| First Name | : |  | | |
|  |  |  |  |  |
| Last Name | : |  | | |
|  |  |  |  |  |
| User Name | : |  | | |
|  |  |  |  |  |
| Email | : |  | | |
|  |  |  |  |  |
| Date Of Birth | : |  | | |
|  |  |  |  |  |
| Password | : |  | | |
|  |  |  |  |  |
| Confirm Password | : |  | | |
|  |  |  |  |  |
| Profile Picture | : |  | Browse |  |
|  |  |  |  |  |
|  | Submit |  | Reset |  |

1. You must implement the back-end of the User Registration application which includes Java files organized in proper packages.
   1. You do not have to implement the REST API or JSP pages.
   2. You must simulate the user registration process using below process:
      1. Use keyboard to input the user data.
      2. Use POJO Setter Methods to initialize instance of User Pojo.
      3. The user data will be validated as per the validation rules before it is saved to the database.
      4. You must use H2 in-memory database which requires only .jar (h2 1.x.x.jar) to be added to the class-path to implement the database.
      5. To browse the H2 in-memory database visually (e.g. the way you use MySQL Workbench to work with MySQL); you may want to use:
         1. Execute Query - <http://executequery.org/index.php>
         2. SQL Workbench - <http://www.sql-workbench.eu/downloads.html>

### Overview of fields used for Add Content

There are two scenarios for content input:

* + - 1. Single Image Input
  1. Title – can be alphanumeric. The length should not go beyond 80 characters.
  2. Description – can be alphanumeric. The length should not go beyond 144 characters.
  3. Image name – can be alphanumeric. You must supply full image name (e.g. imagesample.jpg)
  4. Date – It should take current date and time using Date object.
  5. The program will response with success or failure depending on whether image was saved in the database or not.
  6. If success, program will end.
  7. If failure, program will re-start.
     + 1. Multiple Image Input
  8. Title – can be alphanumeric. The length should not go beyond 80 characters.
  9. Description – can be alphanumeric. The length should not go beyond 144 characters.
  10. Image name – can be alphanumeric. You must supply multiple image names separate by comma “,” (e.g. imagesample1.jpg, imagesample2.jpg etc)
  11. Date – It should take current date and time using Date object.
  12. The program will response with success or failure depending on whether multiple images was saved in the database or not. Here, each image saved will have same title and description as input above.
  13. If success, program will end.
  14. If failure, program will re-start.

# Java Layer

## Classes for User registration

### Classes:

The application must have following classes:

* + - 1. User.java (com.social.user.pojo)
      2. UserInput.java (com.social.user.input)
      3. UserRegistration.java (com.social.user.mainRegistration)
      4. UserValidation.java (com.social.user.utilities)
      5. UserService.java (com.social.user.service)
      6. UserDB.java (com.social.user.database)
      7. UserOutput.java (com.social.user.display)

### **Description**:

1. User.java – POJO class
   1. Fields and their data types:
      1. First Name (string)
      2. Last Name (string)
      3. Username (string)
      4. Email (string)
      5. Password (string)
      6. Profile Picture (string)
2. UserRegistration.java
   1. It uses UserInput.java to capture data from default input device (keyboard) and then creates the multiple instances of the User.java. All data input code to capture data from keyboard is implemented in userInput.java.
   2. It creates the instance of UserService.java which is a service layer.
   3. UserService.java will use the instance of UserDB.java to save the user instances to the H2 database.
3. UserValidation.java
   1. Will do the validation checks and fire the exception if the instances do not contain the value as per the validation guidelines:
      1. First Name must contain alphabets only
      2. Last Name must contain alphabets only
      3. Username should allow mix of alphabets and number
      4. Username must not start with number
      5. Length of Username should be between 8 to 12
      6. Must allow an email in valid email format
      7. Must not allow two @ symbols in an email
      8. Password must be alphanumeric
      9. Password should allow only following special characters- . # % $ !
      10. Password length should be between 8 to 12
      11. Password should contain at-least one capital alphabet
4. UserInput.java
   1. It will take the user input from console.
      1. First Name
      2. Last Name
      3. Username
      4. Email
      5. Password
      6. Confirm Password
      7. Profile Picture
   2. Password must be visible as “#” character when typing in the console.
   3. It will perform the validation check using UserValidation.java and save the user to file using UserDao.java
5. UserDB.java
   1. It will save the user instances to the DB.
   2. It will also do the h2 in-memory database IO and save the user data further to h2 in-memory database (<http://www.h2database.com/>)
6. UserOutput.java
   1. It will ask for user input. If input is:
      1. 1 – Sorting will be done by First Name
      2. 2 – Sorting will be done by Last name
   2. It will use UserService.java and UserDB.java to read the h2 in-memory database to which user data is written and write it to the console in sorted manner.
7. Use lambda expression to define functions/methods
8. Use proper package naming best practices
9. Use proper variable naming best practices
10. Use procedures where possible to make the code re-usable.

## Classes for Add Content

### Classes

The application must have following classes organized in different packages as per best practice:

* + - 1. SingleImageInput.Java (com.social.image.controller)
      2. MultipleImageInput.java (com.social.image.controller)
      3. ImageInput.java (used by both files above to read image file/files from folder) (com.social.image.input)
      4. ImageService.java (com.social.image.service)
      5. ImageDB.java (com.social.image.database)
      6. ImageOutput.java (com.social.image.display) – will display the list of all images along with their title, description and image name and ask you for input. It will associate unique number to each row.

1. It will ask you for the input of row number and save the respective image to the folder.

### Description

Description of classes for Add Content is on same terms as that for User Registration.

# Solution Architecture

## User Registration Architecture

Console

UserOutput.java

Entry Point

(Console)

UserService.java  
(Service Layer)

UserRegistration.java

UserInput.java   
(Input from keyboard)

UserValidation.java  
(Validation Rules)

H2 in-memory Database

User.java  
(POJO)

UserDB.java  
(Database IO)

## Add Content Architecture

Console

ImageOutput.java

SingleImageInput.java

Entry Point

(Console)

ImageService.java  
(Service Layer)

H2 in-memory Database

ImageDB.java  
(Database IO)

MultipleImageInput.java

Image.java  
(POJO)

ImageValidation.java  
(Validation Rules)

ImageInput.java   
(Input from keyboard)

# Methodology

## Agile

1. Mentor will ask you about daily progress as you start implementing Java Layer.
2. Communicate with your mentor via email; ideally daily as you develop the Java Layer.
3. Scope of discussion with your mentor:
   1. Q/A
   2. New Ideas and New feature implementations
   3. Any development related challenges
   4. Skill Gaps
   5. Another pointers key to Java Development

## Continuous Integration

### As you code the solution:

1. Check in your code - ideally every hour in the internal GIT repository at 172.18.2.18

### On conclusion of application development (before going to production)

1. Set up Jenkins on the localhost/cloud and integrate with cloud GIT repository.
2. Setup Jenkins to create docker image which can deploy the final solution.
3. Save the Jenkins/build report’s as a part of final assessment deliverable.

# Technical Specification – Java Layer Solution Development Environment

## Java Layer

|  |  |
| --- | --- |
| **Framework(s)/SDK/Libraries** | **Version** |
| Java 8 | - |
| GIT Basics | - |
| Jenkins Basics | - |
| Docker Basics | - |

## Editors

|  |  |
| --- | --- |
| **Name** | **Version** |
| Eclipse OR STS | - |

# Development Workflow

1. You must follow following process while creating Java classes
   1. \*\*Approval includes:
      1. Communicating with mentor via email to showcase the progress.
      2. Progress must be shared with mentor as each Java layer is crossed.
      3. It is recommended to avoid moving to next stage until feedback from the mentor is received.
      4. Pushing assets/code to GIT repository.
      5. Last three stages do not need any approval as they will test your Business Logic implementation skills.
   2. \*Business Logic Plan includes:
      1. Declaring methods you plan to use for data validation, file io, data formatting or any other utility classes/methods.
      2. Documenting methods you plan to use for data validation, file io, data formatting or any other utility classes/methods.
   3. \*\*\*Business Logic includes:
      1. Data Validation
      2. DB IO
      3. Data Formatting for DB IO
      4. Checking proper input from keyboard

# Important Instructions

1. Follow the design specifications mentioned in the case study. You are free to improvise certain specifications mentioned in the case-study. But, for each such improvisation, you should keep the concerned POC informed. **POC will get in touch with concerned team at IIHT.**
2. You should stay **motivated** to initiate such and specific communications as it may have positive influence on the evaluation scores.
3. Please make sure that your code does not have any compilation errors while submitting your case study solution.
4. The final solution **should be deployed in docker**.
5. The **docker image should start the console application (as per the present case study) at command prompt**.
6. Implement the code using best design standards for:
   1. Variable declarations
   2. Class names
   3. Package names
   4. Code Refactoring

# Assessment Deliverables

Please make sure that your code does not have any compilation/console errors while being deployed using docker.

1. The final solution should consist of:
2. Docker Image which can be executed to capture user input from keyboard.
3. Dump of command “git log”
4. Jenkins report about number of times it pulled the code from GIT to create compiled file.
5. Docker File used to create docker image from Jenkins.
6. Word document with step by step instructions on:
   1. how to run the project

# Other Full Stack Layers

## UI Layer (Not Applicable for Present Case Study)

|  |  |
| --- | --- |
| HTML5 | - |
| CSS3 | - |
| Bootstrap/Material | - |
| Typescript | - |

## UX Layer (Not Applicable for Present Case Study)

|  |  |
| --- | --- |
| Angular | 6 or 7 |
| Javascript & JQuery | - |
| Typescript | - |

## Back End Layer (Applicable for Present Case Study)

|  |  |
| --- | --- |
| Java SE | 8 or above |

## Middle Tier Framework Layer (Not Applicable for Present Case Study)

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Spring Framework | Spring MVC | 5.0 or above |

## ORM & Integration Layer (Not Applicable for Present Case Study)

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Spring | Spring Boot | 2.0 |
| Java JPA | Hibernate | 5.0 or above |
| Spring Data | 2.0 |

## Database Layer (Not Applicable for Present Case Study)

|  |  |  |
| --- | --- | --- |
| MySQL | MySQL | 7.x + |

## Ancillary Layer (Not Applicable for Present Case Study)

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Source Code Management Tool | GIT | 2.18 |
| Build Tool/JAVA Stack | Maven | 3.5.x |
| Testing Tool/JAVA Stack | Junit/Spring Test | 4.x/5.x |
| Javascript Dependency Management Tool | NPM | 6.x.x |

## Deployment & Infra(Not Applicable for Present Case Study)

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Docker | - | 17.06.2 |
| Apache Tomcat | - | 9.0 |
| Jenkins | - | 2.121.2 |