

Cognizant Academy

Data Loader Portal

Case Study Specification

Data Loader Portal

Contents

[1 Problem statement 3](#_Toc115089444)

[2 Architecture Diagram for the Problem Statement 4](#_Toc115089445)

[3 Use case details 5](#_Toc115089446)

[4 Functional/Non-Functional Requirement of the Problem Statement 6](#_Toc115089447)

[4.1 Login 6](#_Toc115089448)

[4.1.1 Acceptance Criteria 7](#_Toc115089449)

[4.1.2 UI-Screen View 7](#_Toc115089450)

[4.1.3 REST End Points 7](#_Toc115089451)

[4.1.4 Entity 7](#_Toc115089452)

[4.1.5 Login Table 8](#_Toc115089453)

[4.1.6 Non-Functional Requirement 8](#_Toc115089454)

[4.1.7 Business Validations 8](#_Toc115089455)

[4.2 Patient Induction 9](#_Toc115089456)

[4.2.1 Acceptance criteria 9](#_Toc115089457)

[4.2.2 UI-Screen View 9](#_Toc115089458)

[4.2.3 REST End Points 9](#_Toc115089459)

[4.2.4 Entity 10](#_Toc115089460)

[4.2.5 Patient Details Table 10](#_Toc115089461)

[4.2.6 Detailed Requirement 11](#_Toc115089462)

[4.2.7 Non-Functional Requirements 11](#_Toc115089463)

[4.2.8 Business Validations 11](#_Toc115089464)

[4.3 Patient Update 12](#_Toc115089465)

[4.3.1 Acceptance criteria 13](#_Toc115089466)

[4.3.2 UI-Screen View 14](#_Toc115089467)

[4.3.3 REST End Points 16](#_Toc115089468)

[4.3.4 Entity 16](#_Toc115089469)

[4.3.5 Detailed Requirement 17](#_Toc115089470)

[4.3.6 Non-Functional Requirements 17](#_Toc115089471)

[4.3.7 Business Validations 17](#_Toc115089472)

[5 Expected Deliverables 18](#_Toc115089473)

[6 Milestone 18](#_Toc115089474)

[7 Skills to develop the project 18](#_Toc115089475)

[8 Implementation Notes 19](#_Toc115089476)

[9 Evaluation rubrics 20](#_Toc115089477)

# Problem statement

The purpose of the requirements document is to systematically capture requirements for the project and the system “**Data Loader Portal**” to be developed. The application should be Cloud Native Architecture with Microservices. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

**About the System**

The client would like to develop a Data Loader Portal application to store and process the Patient and his Prescription details through Data Loader Portal to Downstream Systems.

**Scope of the System**

The scope of the system is explained through its modules as follows

1. Patient Induction – used by admin to register the details of patient into the system. The system stores the details of the Patient and his Prescription Details in the system and able to edit it. The Patient and his prescription details available in the excel sheet.
2. Process Data – The user should be able to send Patient Information to Downstream system. (Just Change the Status to Processed from Inducted)

# Architecture Diagram for the Problem Statement

**Use case Diagram**

**Patient Induction**

Read From File

Patient & Prescription Details

Save to DB

**Process Data**

Send to Downstream system 

Process Patient Data 

Fetch Inducted Patient

***Out Of Scope***

Save to DB

**Update Patient details**

Update only Patient details

Fetch Inducted Patient

Save to DB

# Use case details

|  |  |  |
| --- | --- | --- |
| **User Story #** | **User Story Name** | **User Story** |
| US\_01 | Login | Admin User needs to provide username and password. |
| US\_02 | Patient Induction | Admin User should induct the Patient information and store it in DB.  Acceptance criteria:  Patient and Prescription details should be fetched from Excel sheet, and it should be saved in the database and set the Status as Inducted.  Capture the details like Patient Name, Address, DOB, Email Address, Phone Numbers, Drug Id, Drug Name  Note- Excel sheet should be upload from UI (as a form submit the excel data should move to Db) |
| US\_03 | Update the Patient details | As an Admin User, I should be able to update only the Patient Details not Drug Details |
| US\_04 | Process Data | As an Admin User, I should send the Patient and Prescription details to downstream system and change the status to Processed |

# Functional/Non-Functional Requirement of the Problem Statement

## Login

|  |  |
| --- | --- |
| US-01 | Login |
| Acceptance Criteria Admin User needs to provide Username and Password | |
| UI-Screen View **Chart  Description automatically generated** | |
| REST End Points POST - /login  **Input**   * Username * Password in encrypted format   **Output**   * The service should response 200 ok along with success message. * Error code 400 for any business validation error. * Error code 500 for internal prog. error | |
| Entity **Login**   * **Username**   + Variable name as <username> * **Password**   + Variable name as <password> | |
| Login Table  |  |  | | --- | --- | | **Username** | **Password** | | [**xxxxxx@gmail.com**](mailto:xxxxxx@gmail.com) | **Springboot@123** | | [**yyyyyy@gmail.com**](mailto:yyyyyy@gmail.com) | **Gautham1@31** | | [**rajesh@gmail.com**](mailto:rajesh@gmail.com) | **Rajesh111@111** | | [**farook@gmail.com**](mailto:farook@gmail.com) | **Farookshaik@121** | | [**pavan@gmail.com**](mailto:pavan@gmail.com) | **Pavanmuppala@543** | | |
| Non-Functional Requirement  * Authentication should be mandatory & Admin User would load data (username, PW) & would share with users & the role should be USER / ADMIN * Application should allow as much as external configuration rather than deploying for any small change. * Proper error message & logging should be in-place.  Business Validations  * All fields are mandatory. * Email id should be in valid email pattern, containing a single @. * Password contains at least 8 characters and at most 20 characters.   + It contains at least one digit.   + It contains at least one upper case alphabet.   + It contains at least one lower case alphabet.   + It contains at least one special character which includes! @#$%&\*()-+=^.   + It doesn’t contain any white space. | |

## Patient Induction

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| US\_02 | | Patient Induction | | | | | |
| Acceptance criteria Patient and Prescription details should be fetched from Excel sheet, and it should be saved in the database and set the Status as Inducted. | | | | | | | |
| UI-Screen View   **Graphical user interface  Description automatically generated** | | | | | | | |
| REST End Points POST - /load/patientdata  **Output**   * The service should response 200 ok along with success message. * Error code 400 for any business validation error. * Error code 500 for internal prog. error | | | | | | | |
| Entity **Patient Details**   * **ID**   + Variable name as <patientId> * **Patient Name**   + Variable name as <patientName> * **Address**   + Variable name as <patientAddress> * **DOB**   + Variable name as <patientDateofBirth> * **Email Id**   + Variable name as <patientEmail> * **Phone Number**   + Variable name as **<**patientContactNumber> * **Drug Id**   + Variable name as **<**patientDrugId> * **Drug Name**   + Variable name as **<**patientDrugName> | | | | | | | |
| Patient Details Table | | | | | | | |
| **Patient Name** | **Address** | | **DOB** | **Email ID** | **Phone Number** | **Drug Id** | **Drug Name** |
| Test Patient | Building Name, Street, Place, District, State | | MM-DD-YYYY | email id | numbers | xxxxx-xxxx-xx | desc |
| Test PatientA | Building Name, Street, Place, District, State | | MM-DD-YYYY | email id | numbers | xxxxx-xxxx-xx | desc |
| Test PatientB | Building Name, Street, Place, District, State | | MM-DD-YYYY | email id | numbers | xxxxx-xxxx-xx | desc |
| Detailed Requirement  * Needs to call API call and this method should read the File from the shared folder and saved the Patient and Prescription details in the db. * When the details are saved successfully, the service should response 200 ok along with success message. * If there are any exceptions while connecting/saving to DB, the service should throw corresponding error with error status as 500. * Mandatory fields should be validated as mentioned in the rules above and should be set the status as FAILED with the missing/validation field details. * While persist the data, have column “Status” which should be defaulted to “Pending” during upload & should be updated to “Approved” / “rejected” during Update (UC\_3). | | | | | | | |
| Non-Functional Requirements  * Authentication should be mandatory & Admin / Users can be able to login with username and password & the role should be USER / ADMIN. * Application should allow as much as external configuration rather than deploying for any small change. * Proper error message & logging should be in-place.  Business Validations  * All fields are mandatory. * Patient id should be generated automatically during the time of registration and should be shown in the success message. * The Patient Name should contain only alphabets and space. min 5 and max 30 characters * Contact number should be min 10 and max 10 character and all must be numeric * Email id should be in valid email pattern, containing a single @. * Drug id should be in the format of ‘XXXXX-XXXX-XX’. Should be in Numbers. * DOB should be in MM-DD-YYYY format, and it should not be less than system date. * The Status should be in INDUCTED after successful Induction. * If any one Patient details is not in valid condition, should store other patient details. Shouldn’t reject remaining patient details if those are in valid. * Validation failed patients should be stored in another table with validation failed message with Status as FAILED. | | | | | | | |

## Patient Update

|  |  |
| --- | --- |
| US\_03 | Patient Update |
| Acceptance criteria Patient Details excluding Prescription details should be fetched that can be modified. | |
| UI-Screen View **Graphical user interface  Description automatically generated with medium confidence**  **A picture containing graphical user interface  Description automatically generated**  **Note – Above 2 screens can be combined. In the first screen have the “Approve” “Reject” buttons to update the status of a patient record to “Approved” / “Rejected”. Admin would use Save button for any edit (During edit approve should be disabled) & would use “Approve” / “Reject” buttons for status update.** | |
| REST End Points GET - /retrieve/{patientName}  **Output**   * The service should response 200 ok along with success message & record details * Error code 400 for any business validation error. * Error code 500 for internal prog. Error   PUT - /updatepatient  **Input**   * Patient object   **Output**   * The service should response 200 ok along with success message. * Error code 400 for any business validation error. * Error code 500 for internal prog. error | |
| Entity **Patient Details**   * **ID**   + Variable name as <patientId> * **Patient Name**   + Variable name as <patientName> * **Address**   + Variable name as <patientAddress> * **DOB**   + Variable name as <patientDateofBirth> * **Email Id**   + Variable name as <patientEmail> * **Phone Number**   + Variable name as **<**patientContactNumber> * **Drug Id**   + Variable name as **<**patientDrugId> * **Drug Name** * Variable name as **<**patientDrugName> | |

|  |
| --- |
| Detailed Requirement  * To retrieve the Patient details on entering the Patient name or Id field, a GET method should be implemented to fetch the details. * Once the user enters the details, they should be sent to the PUT method and saved in the db. * Mandatory fields should be validated as mentioned in the rules above and 400 exception response should be sent with the missing field details. * When the details are saved successfully, the service should response 200 ok along with success message. * If there are, any exceptions while connecting/saving to DB. The service should throw corresponding error with error status as 500. |
| Non-Functional Requirements  * Authentication should be mandatory & Admin User alone will be able to edit & the role should be ADMIN. * Application should allow as much as external configuration rather than deploying for any small change. * Proper error message & logging should be in-place. |
| . Business Validations  * Able to fetch only Inducted Patient Details not processed Patient Details * Should be able to update only Patient mail id, address, contact number and DOB. * Email id should be in valid email pattern, containing a single @. * Contact number should be min 10 and max 10 character and all must be numeric. * DOB should be in MM-DD-YYYY format, and it should not be less than system date * Patient Name and Drug Information shouldn’t be updated. |

# Expected Deliverables

The following deliverables are expected as outcomes:

* Application Code base
* Readme document on the complete application
  + Setup of the application
  + How to run the application
  + Any inference
  + Screenshot of UI results
* Reports:
  + Unit/Functional Test Report

# Milestone

The milestone for the project use is given below

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Duration (in weeks)** | **Topic** |
| Milestone -1 | 1 | * Develop the UI, APIs for the use Cases Login & Patient Induction along with the Integration, unit testing, sonar validation & demo. |
| Milestone – 2 | 1 | * Develop the UI, APIs for the use Cases Patient update along with the Integration, unit testing, sonar validation & demo. * Cloud deployment & complete case study demo. |

# Skills to develop the project

List the Technology based on your respective technology stack, that will be used to development the project.

|  |  |
| --- | --- |
| **Layer** | **Tech Stack** |
| Front end | * React / Angular * JavaScript / Type script * Karma/ Cypress/ Jest |
| Back End | * JDK1.8 / .net core * Spring Boot / Asp.net core * JPA/Hibernate / Entity Framework * Maven / NuGet * SonarQube * Junit / NUnit |
| Database | * MongoDB / MySQL / PostgreSQL |
| Deployment Infra | * Cloud / Local Machine |

# Implementation Notes

As per the project requirement modification can be done in the below table.

|  |  |
| --- | --- |
| API | Create UI & REST Microservices to perform login & upload Operation.   * Use Microservices Architecture * Follow coding standards * Follow Standard project structure * Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant * Database connections and web service URLs should be configurable. * Use browser / POST Man to invoke APIs * Swagger implementation. * Run SonarQube for code quality. * Implement unit testing. |
| UI | * Implement user-stories using any one of the UI frameworks [React / Angular] * Implement Forms, databinding, validations * Use appropriate unit test framework. * Follow coding standards * Follow Standard project structure |
| Deployment | * Cloud (Virtual Machines) / Local Machine |

# Evaluation rubrics

|  |  |
| --- | --- |
| Angular | 1. Associate must have used Angular Components, Modules, Databinding, data validation, CLI commands. 2. Associate must have used Forms and Forms validation 3. Associate must have used Directives 4. Associate must have developed Reusable Components 5. Associate must have followed coding standards |
| REACT | 1. Associate must have used Component, Databinding, data validation, CLI commands. 2. Associate must have used Forms and Forms validation 3. Associate must have defined React state 4. Associate must have followed coding standards |
| Microservices | 1. REST controller 2. Follow Std. MVC pattern 3. Follow Entity and Model classes / Dao Pattern 4. Appropriate logging statements 5. Exception handling 6. Make use of latest features from the prog languages. |
| Containerization | 1. Containerize the application 2. Build docker containers 3. Push your Docker images to an Amazon ECR repository with the docker push command |
| AWS / Azure | 1. Associate may use the cloud services for code commit, Virtual Machines, API GW, DB instances / services from AWS / Azure. |