|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Lending Library Management Portal**  **FSE – Business Aligned Project**  **Case Study Specification**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Logeshwaran Mani |  |  | | **Role** | Trainer |  |  | | **Signature** | t-Logesh3 |  |  | | **Date** | 15 October 2022 |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc116039845)

[2.0 Introduction 5](#_Toc116039846)

[2.1 Purpose of this document 5](#_Toc116039847)

[2.2 Project Overview 5](#_Toc116039848)

[2.3 Scope 5](#_Toc116039849)

[3.0 Use Case Diagram 7](#_Toc116039850)

[4.0 System Architecture Diagram 8](#_Toc116039851)

[5.0 Development Phases 8](#_Toc116039852)

[6.0 System Requirements](#_Toc116039853) 8

[**6.1.1** **Module – Book Management**](#_Toc116039854) 8

[**6.1.2** **Module – Reader Management** 1](#_Toc116039855)1

[**6.1.3** **Module – Book Lending Management** 15](#_Toc116039856)

[**6.1.4** **Module – Book Return Management**](#_Toc116039857) 18

[7.0 Deployment requirements 24](#_Toc116039858)

[8.0 Design Considerations 24](#_Toc116039859)

[8.1.1 Most Important and Common rules 24](#_Toc116039860)

[9.0 Reference learning 25](#_Toc116039861)

[10.0 Project Templates 27](#_Toc116039862)

[11.0 Change Log 27](#_Toc116039863)

# Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technolgy Track.
2. Associate must not submit project with compile-time or build-time errors.
3. Being a Full-Stack Developer Project, you must focus on ALL layers of the application development.
4. Unit Testing is Mandatory, and we expect a code coverage of 100%. Use Unit testing and Mocking Frameworks wherever applicable.
5. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only.
6. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file.
7. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the Microservices in cloud must be specified.

(Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)

1. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
2. You are supposed to use an In-memory database or code level data as specified, for the Microservices that should be deployed in cloud. No Physical database is suggested for Microservice.

# Introduction

## Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Lending Library Management Tool” that has to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specificed wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

## Project Overview

Cognizant technology solutions undertakes a number of books for their employees spread across the globe. In order to manage and track the borrowing of books by its employees the company has decided that a Library Management tool needs to be developed by a team of internal developers. The project is aimed at helping the librarian in tracking and managing the books flow.

## Scope

Below are the modules that needs to be developed part of the Project:

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Book Management** module | * This module will be used by Librarian for managing the Books * Librarian can enter details of new book purchased by him * Librarain should Add a new Book, Edit a Book and Remove a Book * Librarian can also update the current status of the book. |
| REQ\_02 | **User (Reader) Management** module | * The Reader management module will be used by Librarian to manage readers i.e. users * This module will allow the Librarians to create a new reader for the books available in the library * The module will also allow the Librarians to edit reader details * Librarians can also remove a reader from DB |
| REQ\_03 | **Book Lending Management** module | * The Librarian will use this module to create a new lending record for each book borrowed from the library * This module will be used to record the details of Lending process, Librarian should be able to enter the details of book borrowed by a reader. It records ReaderId, BookId, BorrowDate, DueDate, AmountToBePaid, PenaltyAmountAfterDueDate etc. * The module will add or manage the book lending in this module |
| REQ\_04 | **Book Return management module** | * The Librarian will use this module to update the lending record for each book borrowed from the library with return date and fee to be paid * The librarian will get the penalty for late return * The Librarian will also be able to see books with high demand * The Librarian should be able to see list books lended by reader * The Librarian should be able to readers details like no.of books borrowed by him and his/her credentials * The Librarian should be able to see a complete report on various angles |

Table 1 : Application Modules

# Use Case Diagram

The following use case diagram shows various users of the system and their responsibilities.

Figure 1 : Use case diagram



# System Architecture Diagram

**Front End (UI)**

Books Service

Reader Service

Book Lending Service

Book Return Service

DB

DB

DB

DB

Figure 2 : Application Architecture Diagram

# Development Phases

* The application will be developed in 2 phase.
* Each phase will have 4 stages followed by a review at the end.
* The phase-1 output will be unit tested core business logic of the application.
* In phase-2 the output will be a functional application with micro-service and the font-end.
* Each stage of the development phase must be completed alongside the learning milestone

# System Requirements

### **Module – Book Management**

The Project Management module will allow Librarians to manage the projects. The module will provide the following features.

1. Creation of new Book
2. Assigning a Book Number to Book
3. Classify the Book Type and Subject
4. Update the Book status

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

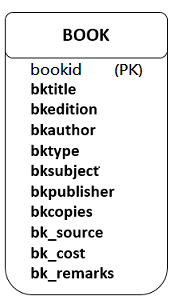


Figure 3 : ER Diagram – Library Management - Book

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. Title, Bookname must be mandatory
   2. DueDate must be greater than start date
   3. BookID should always be 6 characters long

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return list of books
   2. Insert a new book
   3. Update an existing book

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Return list of books
   2. Insert a new book
   3. Update an existing book
5. Following business rules must be implemented as part of the business service class
   1. BookId must be auto-generated.
   2. Phone number should be exactly 10 digits long
   3. Email address should always have @cognizant.com
   4. Firstname and lastname should only have alphabets and last name must be minimum 3 characters long.
   5. Each project is allowed a maximum of 50 developers and 10 testers, if the limit exceeds then raise a user-defined exception as “MaximumResourceLimitReachedException”.
   6. Lending of a damaged books should not be allowed.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 2 : Book Management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/books |
| **Request Type** | GET |
| **User Role** | Librarians |
| **Trigger** | Front end |
| **Description** | Endpoint will be responsible for displaying the list of books present in the database |
| **Inputs** |  |
| **Outputs** | BookDTOs |

Table 3 : Book Management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/books/new |
| **Request Type** | POST |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | Using this endpoint the Librarian will be able to add a new book details into the system |
| **Inputs** | BookDTO |
| **Outputs** | Status code |

Table 4 : Book Management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/books |
| **Request Type** | GET |
| **User Role** | Librarians |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the Librarians to view all the books present in the system |
| **Inputs** |  |
| **Outputs** | BooktDTOs |

Table 5 : Book Management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/books/<bookcode>/update |
| **Request Type** | PUT |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the user will be able to change the status of the book in the system |
| **Inputs** | BookCode |
| **Outputs** | Status code |

Table 6 : Book Management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/books/addbook |
| **Request Type** | POST |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the Librarian can add a new book into the system |
| **Inputs** | BookDTO |
| **Outputs** | Status code and bookid of new book |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. CreateBookComponent
2. Create a component which will be used by Librarians for adding a new book
3. The component should provide a form for the book
4. The book type must be selected from a dropdown list
5. Once all the book details are validated, user should be able to submit the form and get an acknowledgement.

1. AddBookTypeComponent
2. Design a add Book type component and provide a navigation to it via navbar
3. The component must accept the vendor details using the HTML5 form elements.
4. The type of the should be selected using the radio buttons
5. Use a dropdown list to choose a book
6. Before submitting the form ensure that all fields are validated
7. Once the form is submitted successfully, display an acknowledgement along with the newly generated vendor id

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Reader Management**

The Project Management module will allow Librarians to manage the projects. The module will provide the following features.

1. Creation of new Reader
2. Assigning a Reader Number to Reader
3. Classify the Reader Type and Subject
4. Update the Reader status

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

Figure 3 : ER Diagram – Library Management - Reader

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. Title, Readername must be mandatory
   2. DueDate must be greater than start date
   3. Allowed values for lend status are – New/Renewal
   4. ReaderID should always be 6 characters long
   5. Allowed values for the role are Developer/Tester

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return list of Readers
   2. Insert a new Reader
   3. Update an existing Reader

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Return list of Readers
   2. Insert a new Reader
   3. Update an existing Reader
5. Following business rules must be implemented as part of the business service class
   1. ReaderId must be auto-generated.
   2. Phone number should be exactly 10 digits long
   3. Email address should always have @cognizant.com
   4. Firstname and lastname should only have alphabets and last name must be minimum 3 characters long.
   5. Each project is allowed a maximum of 50 developers and 10 testers, if the limit exceeds then raise a user-defined exception as “MaximumResourceLimitReachedException”.
   6. Lending of a damaged Readers should not be allowed.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 2 : Reader Management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/Readers |
| **Request Type** | GET |
| **User Role** | Librarians |
| **Trigger** | Front end |
| **Description** | Endpoint will be responsible for displaying the list of Readers present in the database |
| **Inputs** |  |
| **Outputs** | ReaderDTOs |

Table 3 : Reader Management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/Readers/new |
| **Request Type** | POST |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | Using this endpoint the Librarian will be able to add a new Reader details into the system |
| **Inputs** | ReaderDTO |
| **Outputs** | Status code |

Table 4 : Reader Management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/Readers |
| **Request Type** | GET |
| **User Role** | Librarians |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the Librarians to view all the Readers present in the system |
| **Inputs** |  |
| **Outputs** | ReadertDTOs |

Table 5 : Reader Management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/Readers/<Readercode>/update |
| **Request Type** | PUT |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the user will be able to change the status of the Reader in the system |
| **Inputs** | ReaderCode |
| **Outputs** | Status code |

Table 6 : Reader Management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/Readers/addReader |
| **Request Type** | POST |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the Librarian can add a new Reader into the system |
| **Inputs** | ReaderDTO |
| **Outputs** | Status code and Readerid of new Reader |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. CreateReaderComponent
2. Create a component which will be used by Librarians for adding a new Reader
3. The component should provide a form for the Reader
4. The Reader type must be selected from a dropdown list
5. Once all the Reader details are validated, user should be able to submit the form and get an acknowledgement.

1. AddReaderTypeComponent
2. Design a add Reader type component and provide a navigation to it via navbar
3. The component must accept the vendor details using the HTML5 form elements.
4. The type of the should be selected using the radio buttons
5. Use a dropdown list to choose a Reader
6. Before submitting the form ensure that all fields are validated
7. Once the form is submitted successfully, display an acknowledgement along with the newly generated vendor id

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API

Valid error messages should be shown based on various response status codes received form the API

### **Module – Book Lending management**

This module will provide various features to the Librarian for Sprint management activities. Below are the features provided by this module

1. Create a new record for each book borrowed
2. Send notification to registered readers that new book is added
3. Update the Vendor status

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

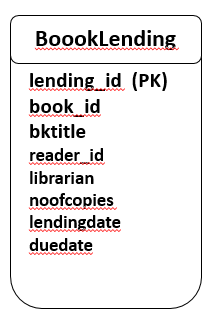


Figure 4 : ER Diagram – Book Lending management

1. Apply the following constraints apart from primary keys and foreign keys on the database
   1. Vendor created on date must be taken as today by default.
   2. Values allowed for the vendor status are – string

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return list of vendors
   2. Insert new vendor
   3. Update a vendor

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes using the single responsibility principle which perform the given operations as follows
   1. Get all vendors
   2. Add a new vendor

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 9 : Book Lending management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/book/lending |
| **Request Type** | GET |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | This endpoint will be used to provide a list of books lended by the user from the database |
| **Inputs** |  |
| **Outputs** | BookDTOs |

Table 10 : Book Lending management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/book/new |
| **Request Type** | POST |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | Using this endpoint the Librarian will create a new lending record |
| **Inputs** | BookDTO |
| **Outputs** | Status code along with book name and reader name |

Table 11 : Book Lending management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/books |
| **Request Type** | GET |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | Librarian will use the endpoint to get a list of books borrowed by readers in the database |
| **Inputs** |  |
| **Outputs** | BookDTOs |

Table 13 : Book Lending management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/book/<bookid> |
| **Request Type** | GET |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | Using this endpoint the librarian will be able to see the details of a book borrowed by the user |
| **Inputs** | BookID |
| **Outputs** | BookDTO |

Table 14 : Book Lending management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/books/<bookid>/penalty |
| **Request Type** | PUT |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | The Librarians will use this endpoint to see the details of penalty for the late return of books |
| **Inputs** | BookID, BookDTO |
| **Outputs** | Status code |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. AddLendingComponent
   1. Create lend a book which can be navigated to from the menu of the application.
   2. The component will provide a form to accept a new sprint details and save it into the system after validating all the details.

1. AddReturnComponent
   1. Create a add meetings component which is accessible to Librarian by navigation from return component
   2. The component should contain a form to show the borrow details and save it.
   3. The return must be selected form a dropdown list.
   4. Use a set of radio buttons for the type of meeting
   5. Once all details are validated and saved then an acknowledgement must be displayed
2. RenewBookLendedComponent
   1. Develop a component which is accessible from the menu bar for Librarians.
   2. The component should accept a Book ID in a textbox and a search button should fetch and display the borrow details in a form
   3. Once the user updates the book and return date then an acknowledgement must be displayed.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Book Return Management**

The developers and Librarian in the agile teams will be able to work on book return which consists of books details and a set of users who borrowed it. This module will provide various features for developer and Librarians to manage them.

1. A Librarian can create product backlog which consists of borrow details and the user who borrowed it.
2. Developers can work on the returning on book and update their status accordingly
3. The Librarians can also see a the complete list books borrowed and its return date reports.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

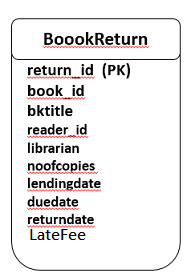


Figure 5 : ER Diagram - Book Return Management

1. Apart from primary and foreing keys implement the following additional constraints
   1. Return date must be taken by default as today
   2. Borrow status should be Active/Overdue/Extension. The default value must be InProgress when new epic is created
   3. The values allowed for user borrow status are – New/Active/Extension/Overdue. The default value must be New.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Insert new book along with it’s user details
   2. Borrow details
   3. Return Details
   4. OverDue/Extention details

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Add a new book addition
   2. Add new borrow details
   3. Record the returns
   4. Get all overdue details
   5. Get Penalty details
   6. Perform Extension of borow

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All the exceptions must be handled and logged using a logging library.
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 16 : Book Return Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/book/return/<bookid>,<readerid> |
| **Request Type** | POST |
| **User Role** | Librarians |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the Librarians to return of books which includes bookid and readerid |
| **Inputs** | BookReturnDTO |
| **Outputs** | Status code |

Table 17 : Book Return Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/book/renew/<bookid>,<userid> |
| **Request Type** | GET |
| **User Role** | Librarian |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the Librarians to renew of book borrow which includes bookid and readerid |
| **Inputs** | BookId, ReaderID |
| **Outputs** | BookRenewDTO |

Table 18 : Book Return Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/book/overdue/<bookid>,<userid> |
| **Request Type** | PUT |
| **User Role** | Developers |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the Librarians to send overdue notice mail to reader with details of book(s) he borrowed. Also mention the penalty reader need to pay |
| **Inputs** | BookOverDueDTO |
| **Outputs** | Status code |

Table 19 : Book Return Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/book/overduePenalty/<bookid>,<userid> |
| **Request Type** | POST |
| **User Role** | Librarians |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the Librarians to find overdue days collect penalty/late dee from reader with details of book(s) reader borrowed. |
| **Inputs** | ReaderId |
| **Outputs** | BookPenaltyDTO |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. CreateBookReturnComponent
2. Develop a component to be used by Librarian which contains a form to return books.
3. Use the book details layout for creating the form.
4. Form should display the each book, it lending fee, return date, late fees if any
5. There should be a submit button which should save the return details and all the other information together after all the validation are passed.
6. Librarians must select book which was borrowed by a reader
7. RenewBookComponent
8. Design a component which can be used by librarians to view the list of books to renewed or returned to them in the form of table.
9. Each row in table must contain a button to view details which should redirect to reader details component by passing in the readerid as parameter.
10. OverDueMailComponent
    1. Create a component which will send a notification mail to the reader mentioning details of ovedue of the book.
    2. Component should also provide a button to send mail.
11. CollectLateFeeComponent
    1. Develop a component which will be used by the Librarian to collect late fees for the overdue days during which the reader has to return
    2. Provide a navigation to the component in the application menu
    3. Component should contain a textbox to accept the readerId and a “Show Report” button. Once the users select a readerId and clicks the show report button then display the report to the user in a bootstrap table

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Defects management**

All projects at cognizant undergo thorough testing to identify the defects. This module will facilitate in reporting and fixing the defects as follows

1. A tester can report a defect
2. A developer will be able to view the defects reported and fix them and provide a resolution
3. Librarians can see a report on defects as to how many days each one of them are pending from.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

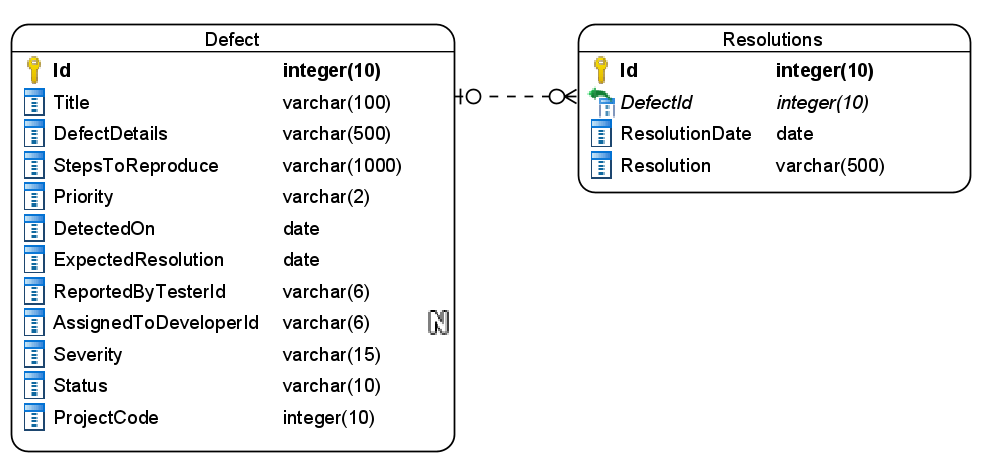


Figure 6 : ER Diagram - Defects Management

1. Enfore the following constraints along with primary and foreign keys
   1. DetectedOn must be taken as today by default
   2. Allowed values for priority is P1, P2 and P3
   3. Allowed values for severity are – Blocker, Critical, Major, Minor and Low
   4. ExpectedResolutionDate must be a future or today’s date.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Insert new defect
   2. Return defects assigned to developer
   3. Return a defect by id
   4. Update a defect
   5. Return a defects report

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Add new defect
   2. Update a defect along with resolution
   3. Fetch defects assigned to developer
   4. Fetch defects report
   5. Fetch defect by id
   6. Calculate ExpectedResolution
5. Following business rules must be implemented as part of the business service class
   1. Steps to reproduce and defects details must be atleast 10 words
   2. ExpectedResolutions must be calculated based on the following
      1. Severity=Blocking, Priority=P1 – To be resolved in 2 days
      2. Severity=Blocking, Priority=P2 – To be resolved in 3 days
      3. Severity = Critical, Priority=P1 – To be resolved in 1 day
      4. Severity = Critital, Priority=P2 – To be resolved in 2 days
      5. Rest all to be resolved in 5, 8 and 10 days based on Priority P1, P2 and P3 respectively.
   3. Each developer can be assigned 5 bugs per day maximum

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. Create the following end-points and test them using postman and export the requests into a json file.

Table 22 : Defects Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/defects/new |
| **Request Type** | POST |
| **User Role** | Testers |
| **Trigger** | Front end |
| **Description** | Using this endpoint the users will be able to add a new defect into the record |
| **Inputs** | DefectDTO |
| **Outputs** | Status code |

Table 23 : Defects Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/defects/assignedto/<developerid> |
| **Request Type** | GET |
| **User Role** | Developers |
| **Trigger** | Front end |
| **Description** | Developers will use this endpoint to view all the defects assigned to him/her |
| **Inputs** | DeveloperId |
| **Outputs** | DefectDTOs |

Table 24 : Defects Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/defects/<defectid> |
| **Request Type** | GET |
| **User Role** | Developers |
| **Trigger** | Front end |
| **Description** | A developer will be able to view all details of a defects using this endpoint |
| **Inputs** | DefectID |
| **Outputs** | DefectDetailsDTO |

Table 25 : Defects Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/defects/resolve |
| **Request Type** | PUT |
| **User Role** | Developers |
| **Trigger** | Front end |
| **Description** | Using this endpoint the developers can provide a resolution and update the status of a defect |
| **Inputs** | UpdateDefectDTO |
| **Outputs** | Status code |

Table 26 : Defects Management - End point - 5

|  |  |
| --- | --- |
| **URL** | /api/defects/report/<projectid> |
| **Request Type** | GET |
| **User Role** | Librarians |
| **Trigger** | Front end |
| **Description** | The Librarians will use endpoint to view a report on defects recorded in the system according to the project |
| **Inputs** | ProjectId |
| **Outputs** | DefectReportDTO |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. NewDefectComponent
2. Create a component which contains a form to accept details of a defect.
3. Priority must be accepted using radio buttons
4. The severity should be selected from a dropdown list
5. Once all the details are validated then users must be allowed to submit the form and get an acknowledgement.

1. DefectsListComponent
2. Create a component which is accessible to developers from the nav bar
3. The component should display the new defects assigned to the developer in the form of a table
4. Each row should have view details button which should navigate to Defect details component by passing in the defect id.
5. DefectDetailsComponent
6. Design a new component for developers to view the details of a defect
7. The component must also have a form to provide the resolution and close the defect.
8. Once the details are submitted an acknowledgement must be displayed.
9. DefectsReportComponent
10. Develop a component which is accessible to the Librarian from the application nav bar
11. Provide a textbox to enter project code and a button to fetch the defects report.
12. The report should be displayed in a tabular format.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

# Deployment requirements

1. All the Microservices must be deployed on a local web server like IIS or Apache Tomcat
2. All the Microservices must be independently deployable.
3. These services must be consumed from an front-end app running in a local environment.

# Design Considerations

Java and Dotnet specific design considerations are attached here. These design specifications, technology features have to be strictly adhered to.



Refer this link for the coding standards. <https://cognizantonline.sharepoint.com/:w:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/Coding%20standards/Effective%20coding%20standards.docx?d=w6430574d9db5478bbbe37c25b16e68e2&csf=1&web=1&e=84lTVf>

### Most Important and Common rules

|  |  |
| --- | --- |
| **Category** | **Rule** |
| Database | Table names in database must be pascal cased and plural. All primary keys must be named as Pk\_<table>. All foreign keys must be named as FK\_<PrimaryKeyTable>\_<ForeignKeyTable> |
| Database | Column names must be pascal cased. Multi-word column must be split using \_ (underscore) |
| Coding | Follow pascal casing for naming classes, interfaces, methods, properties and other public members |
| Coding | Use camel casing for method parameter name, backing fields for properties and private variables. Consts must be capitalized |
| Coding | All exceptions must be handled and logged using a logging library |
| Coding | For communication between micro-services use the HttpClient class available in .Net and Java |
| Unit testing | Each method in services classes in business logic must be unit tested using nUnit/jUnit |
| Unit testing | Use a mocking library to mock the repositories while performing tests for business logic layer |
| Code Coverage | Should be minimum 90% |
| Front-end(Angular/React ONLY) | Use pascal casing for the component names |
| Front-end(Angular/React ONLY) | Create all components and data services in Angular/React project in dedicated folders |
| GitHub | Create ONLY Private Repositories.  No password should be stored.  DO NOT Mention in the Profile that You work for Cognizant |

Any deviation from the high level design must be approved by trainer, mentor and solutions team

# Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**Other References:**

|  |  |
| --- | --- |
| Java 8 Parallel Programming | <https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8> |
| Feign client | [https://dzone.com/articles/Microservices-communication-feign-as-rest-client](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Swagger (Optional) | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](https://dzone.com/articles/centralized-documentation-in-microservice-spring-b) |
| ECL Emma Code Coverage | <https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php> |
| Lombok Logging | <https://javabydeveloper.com/lombok-slf4j-examples/> |
| Spring Security | <https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world> |
| H2 In-memory Database | <https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot>  <https://www.baeldung.com/spring-boot-h2-database> |
| AppInsights logging | <https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights> |
| Error response in WebApi | <https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api> |
| Read content from CSV | <https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects> |
| Access app settings key from appSettings.json in .Net core application | <https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1> |

# Project Templates

# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on 15-October-2022 by Logeshwaran Mani | | | |
| V1.0.1 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| 2.3 | Logeshwaran Mani | 13 November 2022 | Updated project scope with new modules |
| 3.0 | Updated use case diagram |
| 4.0 | New architecture diagram |
| 6.1.1 | Added new module on project management |
| 6.1.2 | Add new module on sprint management |
| 6.1.3 | Removed API gateway |
| 6.1.4 | Removed API gateway |
| 10 | Added new section on project templates for .net backend, react front end and angular front end |