|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Agile Project Management Portal**  **FSE – Business Aligned Project**  **Case Study Specification**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Khaleelullah Hussaini Syed |  |  | | **Role** | Trainer |  |  | | **Signature** | t-syed8 |  |  | | **Date** | 16 September 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 9](#_Toc116039853)

[**6.1.1** **Module – Project Management** 9](#_Toc116039854)

[**6.1.2** **Module – Sprint management** 12](#_Toc116039855)

[**6.1.3** **Module – Product Backlog Management** 16](#_Toc116039856)

[**6.1.4** **Module – Defects management** 20](#_Toc116039857)

[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 “Agile Project 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 projects for their clients spread across the globe. In order to manage and track the progress of these projects the company has decided that a custom made agile project management tool will be developed by a team of internal developer. The project is aimed at helping the participants of the agile projects in tracking and managing the progress of the project.

## Scope

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

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Project Management** module | * This module will be used by product owner for managing the projects * Product owners can create a new project * Add a new resource * Project owners can also update the current status of the project. |
| REQ\_02 | **Sprint management** module | * The Sprint management module will be used by product owners to manage sprints * This module will allow the product owners to create a new sprint for a project * The module will also allow the product owners to schedule a meeting * Product owners can also re-schedule a meeting |
| REQ\_03 | **Product Backlog Management** module | * The product owner will use this module to create a new EPIC for the sprint along with it’s user stories * The developers will be able to work on their assigned user stories and update them once they are done * The product owner will also be able to see a completion report on various user stories for a given sprint |
| REQ\_04 | **Defects management** module | * All the softwares developed at Cognizant are thoroughly tested for any defects. This module will help the agile teams in tracking and fixing defects in the project * A tester will be able to report any defects detected * A developer will be able to work on fixing the defect and submit a report * The product owner will be able to see a complete report on various defects |

Table 1 : Application Modules

# Use Case Diagram

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

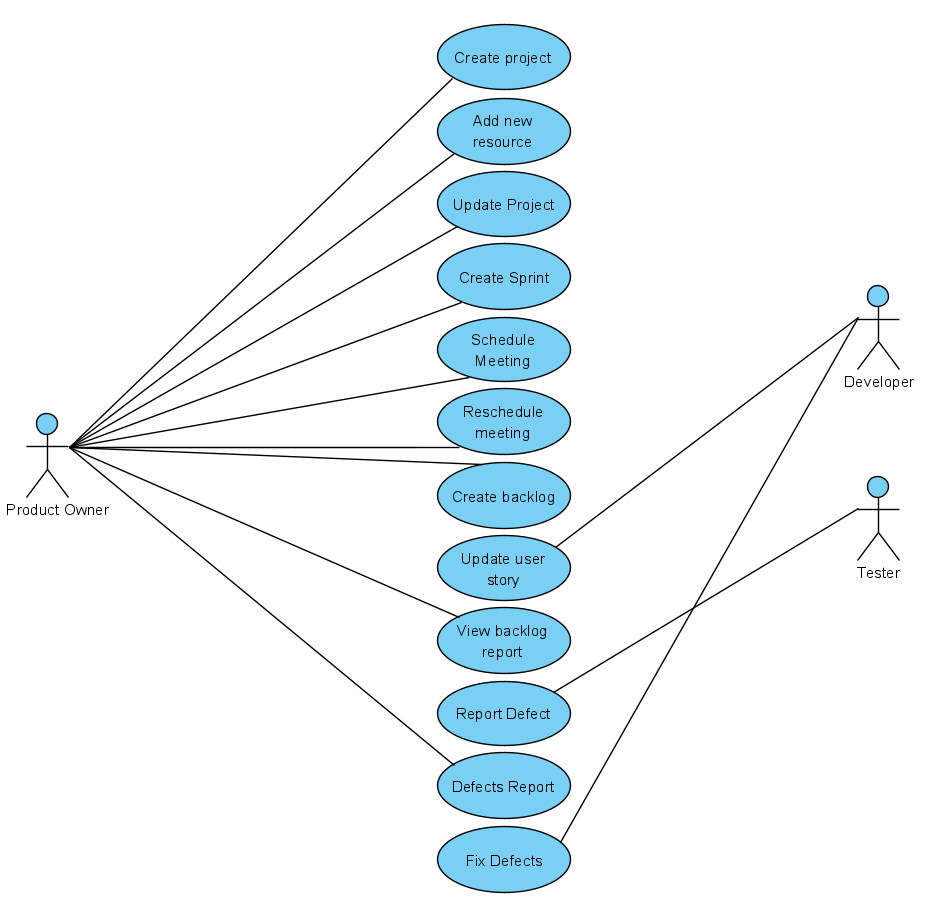


Figure 1 : Use case diagram

# System Architecture Diagram

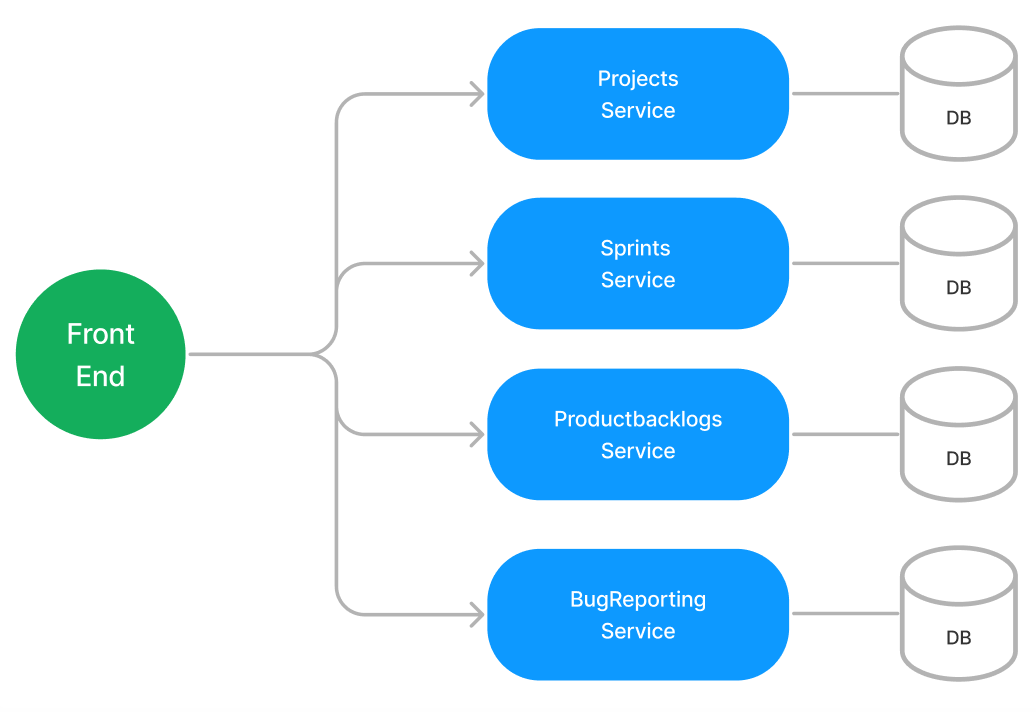


Figure 2 : Application Architecture Diagram

# 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 – Project Management**

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

1. Creation of new project
2. Assigning a resource to a project
3. Update the project status

**Stage: Database Implementation**

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

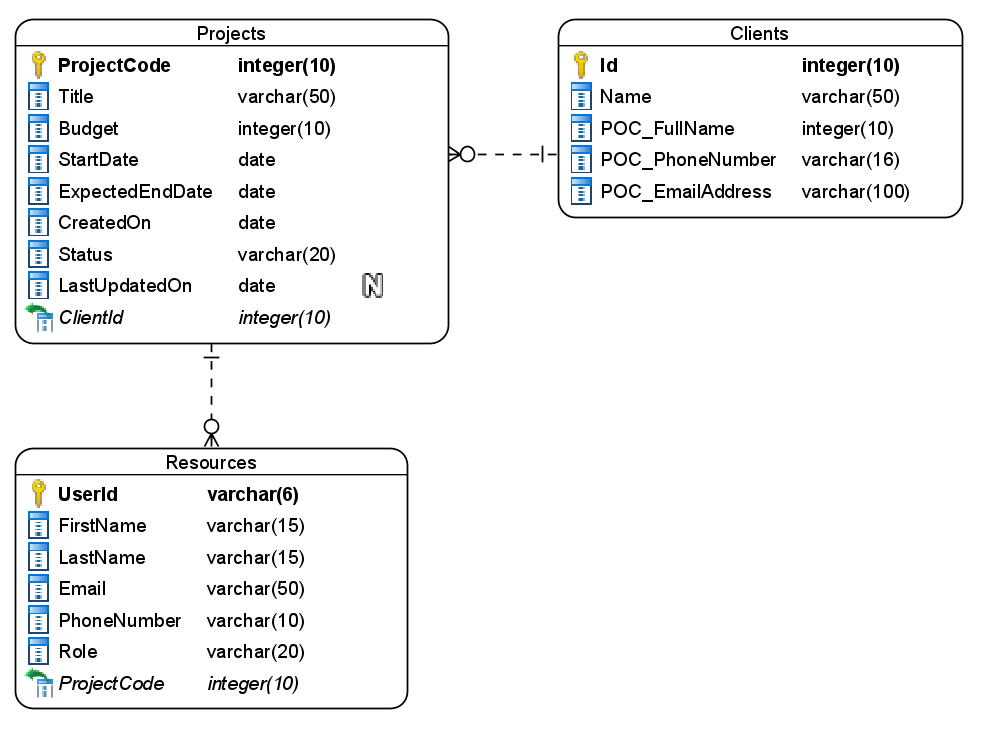


Figure 3 : ER Diagram – Project Management

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. Title, first name and last name must be minimum 3 characters long
   2. ExpectedEndDate must be greater than start date
   3. Allowed values for project status are – New/InProgress/Completed/Delayed/Cancelled
   4. UserID should always be 6 characters long
   5. Allowed values for the role are Developer/Tester

Note: Add few clients on the application startup.

**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 clients
   2. Insert a new project
   3. Update an existing project
   4. Return list of projects
   5. Insert a new resource

**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 clients list
   2. Add new project
   3. Update existing project
   4. Return projects list
   5. Add new resource
5. Following business rules must be implemented as part of the business service class
   1. UserId must be auto-generated. It should be in the format XY0000. First 2 letters of lastname followed by a 4 digit number
   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. Adding a resource to a cancelled projects 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 : Project Management - Endpoint - 1

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

Table 3 : Project Management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/projects/new |
| **Request Type** | POST |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | Using this endpoint the product owner will be able to add a new project details into the system |
| **Inputs** | ProjectDTO |
| **Outputs** | Status code |

Table 4 : Project Management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/projects |
| **Request Type** | GET |
| **User Role** | Product owners |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the product owners to view all the projects present in the system |
| **Inputs** |  |
| **Outputs** | ProjectDTOs |

Table 5 : Project Management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/projects/<projectcode>/update |
| **Request Type** | PUT |
| **User Role** | Project owner |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the user will be able to change the status of the project in the system |
| **Inputs** | ProjectCode, ProjectDTO |
| **Outputs** | Status code |

Table 6 : Project Management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/projects/addresource |
| **Request Type** | POST |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the product owner can add a new resource into the system |
| **Inputs** | ResourceDTO |
| **Outputs** | Status code and userid of new resource |

**Stage: Font-end design**

Create the following components as per the specification provided below.

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

1. AddResourceComponent
2. Design a add resource component and provide a navigation to it via navbar
3. The component must accept the resource details using the HTML5 form elements.
4. The role of user should be selected using the radio buttons
5. Use a dropdown list to choose a project
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 user id
8. ProjectsComponent
9. Develop a component which will be used for updating the status of the projects
10. Provide a navigation to the component from the menu bar of the application
11. The component should display the existing projects in a bootstrap table.
12. Each row should contain a dropdown list to change the status of the project.
13. If the project is completed or cancelled, then no dropdown for status should be provided
14. Add a button next to dropdown to save the status.
15. Upon saving the status successfully display an acknowledgement.

**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 – Sprint management**

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

1. Create a new sprint for the project
2. Schedule a new meeting for a sprint
3. Update the meeting status

**Stage: Database Implementation**

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

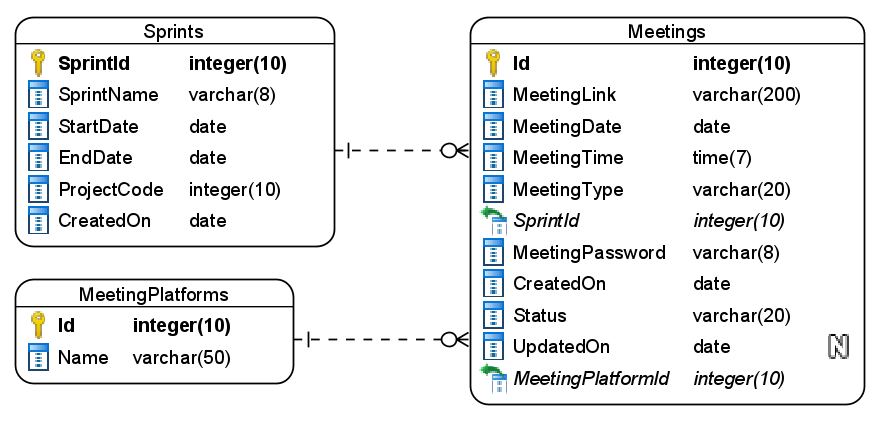


Figure 4 : ER Diagram – Sprint management

1. Apply the following constraints apart from primary keys and foreign keys on the database
   1. Sprint created on date must be taken as today by default.
   2. Allowed values for meeting types are – DailyScrum/SprintPlanning/SprintReview/SprintRetrospective
   3. Values allowed for the meeting status are – scheduled, completed, cancelled, rescheduled
   4. Sprint end date must be after the sprint start date

**Note**: During the application startup, seed the data into the MeetingsPlatforms table as Zoom, Teams, WebEx, etc.,

**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 meeting platforms
   2. Insert new sprint
   3. Insert new meeting
   4. Return a meeting by ID
   5. Update a meeting

**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 meeting platforms
   2. Add a new sprint
   3. Schedule a new meeting
   4. Get a meeting by id
   5. Reschedule a meeting
5. Following business rules must be implemented as part of the business service class
   1. Duration for a sprint must be in the range of 1 week to 4 weeks maximum.
   2. Sprint name must be autogenerated as – Sprint-xx, where x is the sprint number.
   3. If meeting type is DailyScrum then multiple meetings must be created with 15 minutes duration for each day of sprint automatically when user create it for the first time.
   4. If meeting type is SprintPlanning it must be on date of sprint start, where are for SprintReview and SprintRetrospective it must be on sprint end date.
   5. If a user tries to update a meeting which is already completed then generate a user-defined exception as “MeetingStatusUpdateFailedException”.

**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 : Sprint management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/meetings/platform |
| **Request Type** | GET |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | This endpoint will be used to provide a list of available meeting platform present in the database |
| **Inputs** |  |
| **Outputs** | MeetingPlatformDTOs |

Table 10 : Sprint management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/sprints/new |
| **Request Type** | POST |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | Using this endpoint the product owner will create a new sprint |
| **Inputs** | SprintDTO |
| **Outputs** | Status code along with sprint name |

Table 11 : Sprint management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/sprints |
| **Request Type** | GET |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | Product owner will use the endpoint to get a list of sprints available in the database |
| **Inputs** |  |
| **Outputs** | SprintDTOs |

Table 12 : Sprint management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/meetins/new |
| **Request Type** | POST |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the product owners to create a new meeting |
| **Inputs** | MeetingDTO |
| **Outputs** | Status code |

Table 13 : Sprint management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/meetings/<meetingid> |
| **Request Type** | GET |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | Using this endpoint the user will be able to see the details of a meeting using the meeting id |
| **Inputs** | MeetingID |
| **Outputs** | MeetingDTO |

Table 14 : Sprint management - Endpoint - 6

|  |  |
| --- | --- |
| **URL** | /api/meetings/<meetingid>/reschedule |
| **Request Type** | PUT |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | The product owners will use this endpoint to update the details of a meeting |
| **Inputs** | MeetingID, MeetingDTO |
| **Outputs** | Status code |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. AddSprintComponent
   1. Create a sprints component 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. AddMeetingsComponent
   1. Create a add meetings component which is accessible to product owner by navigation from sprints component
   2. The component should contain a form to accept the new meeting details and save it.
   3. The sprint 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. ReScheduleMeetingComponent
   1. Develop a component which is accessible from the menu bar for product owners.
   2. The component should accept a meeting ID in a textbox and a search button should fetch and display the meeting details in a form
   3. Once the user updates the meeting start and end date time 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 – Product Backlog Management**

The developers and product owner in the agile teams will be able to work on product backlog which consists of Epic and a set of user stories it. This module will provide various features for developer and product owners to manage them.

1. A product owner can create product backlog which consists of EPIC and it’s associated user stories. Consider EPIC as a module and user stories are requirements under it.
2. Developers can work on the user stories and update their status accordingly
3. The product owners can also see a progress report on product backlog completion which will show the progress of each user story along with an EPIC completion percentage.

**Stage: Database Implementation**

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

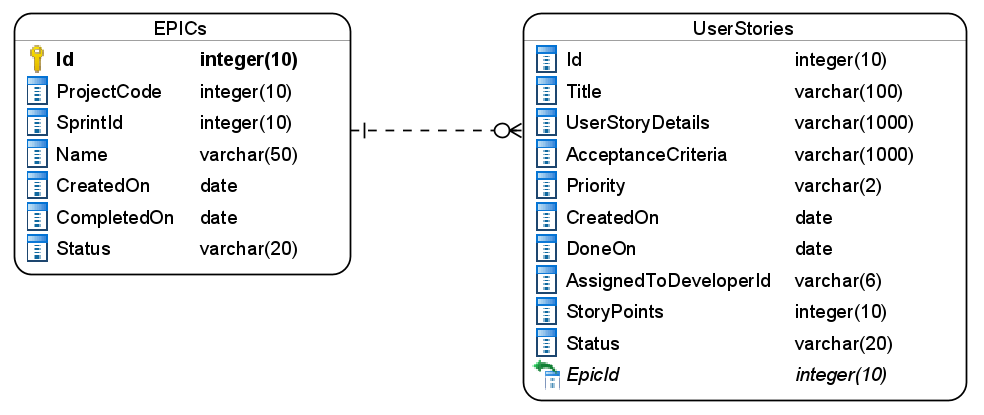


Figure 5 : ER Diagram - Product Backlog Management

1. Apart from primary and foreing keys implement the following additional constraints
   1. Created on date must be taken by default as today
   2. EPIC status should be InProgress/Done. The default value must be InProgress when new epic is created
   3. The values allowed for user story status are – New/Planning/Coding/Testing/Done. The default value must be New.
   4. User story points must be in the range of 1-20

**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 EPIC along with it’s user stories
   2. Return user stories
   3. Update user story status
   4. Return product backlog 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 a new epic with user stories
   2. Get all user stories
   3. Update user story status
   4. Get product backlog report by project id
5. Following business rules must be implemented as part of the business service class
   1. A user story title must be atleast 10 words where as user story details and acceptance criteria should be atleast 50 words each
   2. Each developer should not be assigned more than 5 active user stories.
   3. When all user stories under a single EPIC are done then EPIC status should also be changed to done
   4. When report is generated then all user stories under different stages must be grouped along with their EPIC names and the total progress of the epic. Example:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Authorization - 50%   |  |  | | --- | --- | | **Stage** | **Count** | | New | 5 | | Planning | 5 | | Coding | 5 | | Testing | 10 | | Done | 10 | |
| Payments – 30%   |  |  | | --- | --- | | **Stage** | **Count** | | New | 5 | | Planning | 5 | | Coding | 5 | | Testing | 10 | | Done | 3 | |

**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 : Product Backlog Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/productbacklog/create |
| **Request Type** | POST |
| **User Role** | Product owners |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the product owners to create the new product backlog which includes an epic item and 1 or more user stories in the epic |
| **Inputs** | ProductBackLogDTO |
| **Outputs** | Status code |

Table 17 : Product Backlog Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/productbacklog/userstories/<developerid> |
| **Request Type** | GET |
| **User Role** | Developers |
| **Trigger** | Front end |
| **Description** | Using this endpoint developer can view the set of user stories assigned to them |
| **Inputs** | DeveloperId |
| **Outputs** | UserStoryDTO |

Table 18 : Product Backlog Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/productbacklog/updatestory |
| **Request Type** | PUT |
| **User Role** | Developers |
| **Trigger** | Front end |
| **Description** | Developers will change the status of their user stories using this endpoint |
| **Inputs** | UpdateStoryDTO |
| **Outputs** | Status code |

Table 19 : Product Backlog Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/productbacklog/report/<projectid> |
| **Request Type** | GET |
| **User Role** | Product owners |
| **Trigger** | Front end |
| **Description** | Using this endpoint product owners will be able to view a report of product backlogs |
| **Inputs** | ProjectId |
| **Outputs** | ProductBackLogReportDTO |

Table 20 : Product Backlog Management - End point - 5

|  |  |
| --- | --- |
| **URL** | /api/productbacklog/userstories/<userstoryid> |
| **Request Type** | GET |
| **User Role** | Developers |
| **Trigger** | Front end |
| **Description** | This endpoint will display details of a specific user story |
| **Inputs** | UserStoryId |
| **Outputs** | UserStoryDTO |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. CreateProductBacklogComponent
2. Develop a component to be used by product owner which contains a form to create a new epic and a set of user stories.
3. Use the master details layout for creating the form.
4. Form should display the each user story added by user in the table form
5. There should be a submit button which should save the epic and all the user stories together after all the validation are passed.
6. Developers details must be selected from a dropdown whose data can be fetched from Project Management module.
7. Provide a navigation to component for motorists via navbar
8. UserStoriesListComponent
9. Design a component which can be used by developers to view the list of user stories assigned to them in the form of table.
10. Each row in table must contain a button to view details which should redirect to UserStoryDetails component by passing in the user story id as parameter.
11. UserStoryDetailsComponent
    1. Create a component which will display the details of a single user story.
    2. Component should also provide a button to update the status of the user story.
    3. The status must be updated in the sequence as New -> Planning -> Coding -> Testing -> Done
12. UseStoryReportComponent
    1. Develop a user story report component which will be used by the product owner to see a report of project progress
    2. Provide a navigation to the component in the application menu
    3. Component should contain a textbox to accept the project code and a “Show Report” button. Once the users provides a project code 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. Product owners 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.

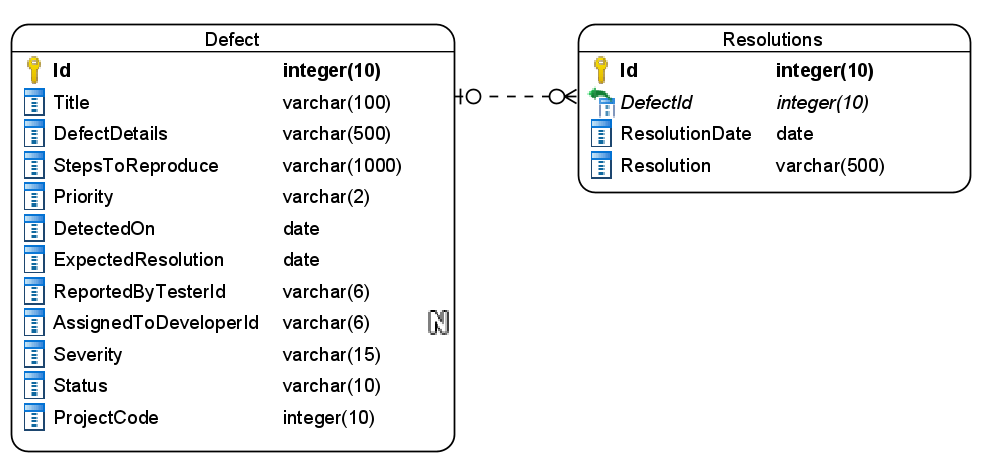


Figure 6 : ER Diagram - Defects Management

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** | Product owners |
| **Trigger** | Front end |
| **Description** | The product owners 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 product owner 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 16-September-2022 by Khaleelullah Hussaini Syed | | | |
| V1.0.1 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| 2.3 | Khaleelullah Hussaini Syed | 6 October 2022 | Updated project scope with new modules |
| 3.0 | Updated use case diagram |
| 4.0 | New architecture diagram |
| 5.0 | Removed API Gateway phase and replaced it with Deployment |
| 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 |