|  |  |
| --- | --- |
|  | **Cognizant Academy**  **One Cognizant Platform**  **FSE – Business Aligned Project**  **Case Study Specification**  **Version 1.0.2** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Khaleelullah Hussaini Syed |  |  | | **Role** | Trainer |  |  | | **Signature** | t-syed8 |  |  | | **Date** | 1 October 2022 |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc115536605)

[2.0 Introduction 5](#_Toc115536606)

[2.1 Purpose of this document 5](#_Toc115536607)

[2.2 Project Overview 5](#_Toc115536608)

[2.3 Scope 5](#_Toc115536609)

[3.0 Use Case Diagram 7](#_Toc115536610)

[4.0 System Architecture Diagram 8](#_Toc115536611)

[5.0 Development Phases 8](#_Toc115536612)

[6.0 System Requirements 9](#_Toc115536613)

[**6.1.1** **Module – Leave management** 9](#_Toc115536614)

[**6.1.2** **Module – Visitor Pass** 12](#_Toc115536615)

[**6.1.3** **Module – Asset management** 16](#_Toc115536616)

[**6.1.4** **Module – Transportation Service** 20](#_Toc115536617)

[7.0 Deployment requirements 24](#_Toc115536618)

[8.0 Design Considerations 24](#_Toc115536619)

[8.1.1 Most Important and Common rules 24](#_Toc115536620)

[9.0 Reference learning 25](#_Toc115536621)

[10.0 Project Templates 27](#_Toc115536622)

[11.0 Change Log 27](#_Toc115536623)

# 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 “One Cognizant Platform” 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 has decide to develop a platform for their internal employees to easy the various day-2-day task. Application will be developed as a Single Page Application (SPA) which will be hosted on an internal server to be accessed by pre-registered employees of the Cognizant. The application will help the employees in managing assets, visitor passes and in availing the Cognizant provided transportation services.

## Scope

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

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Leave management** module | * The Leave management module will allow the Cognizant HRs to feed the details of the on boarded employees into the database. * Employee can record their leave details into the system * HRs will be able to see a leave report based on date range |
| REQ\_02 | **Asset Management** module | * The asset management system in the platform will allow users to raise tickets related to various issues with the Cognizant assets provided to them * In order to use the system a user must be logged in and should register their assigned asset. * Tickets raised by user will be automatically assigned to based on various SLA for different asset types. * User will be able to track the progress on their raised tickets. * Support staff will be able to login and manage tickets |
| REQ\_03 | **Visitor Pass Access** module | * Cognizant employees can plan for the family members and clients visit to their respective campuses on weekend * Visitor pass access management system will aid the employees in raising a pass request which can be approved or rejected by the security head in their respective locations. * Each user can request for a maximum of 2 visitor passes in a month for family members |
| REQ\_04 | **Transport Services** module | * Cognizant provides transportation services to various routes. * Employees with in cognizant will be able to avail the official transportation service * Employees will have to login through the OneCognizant portal and choose one of the available transportation fee by paying the mentioned fee. |

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

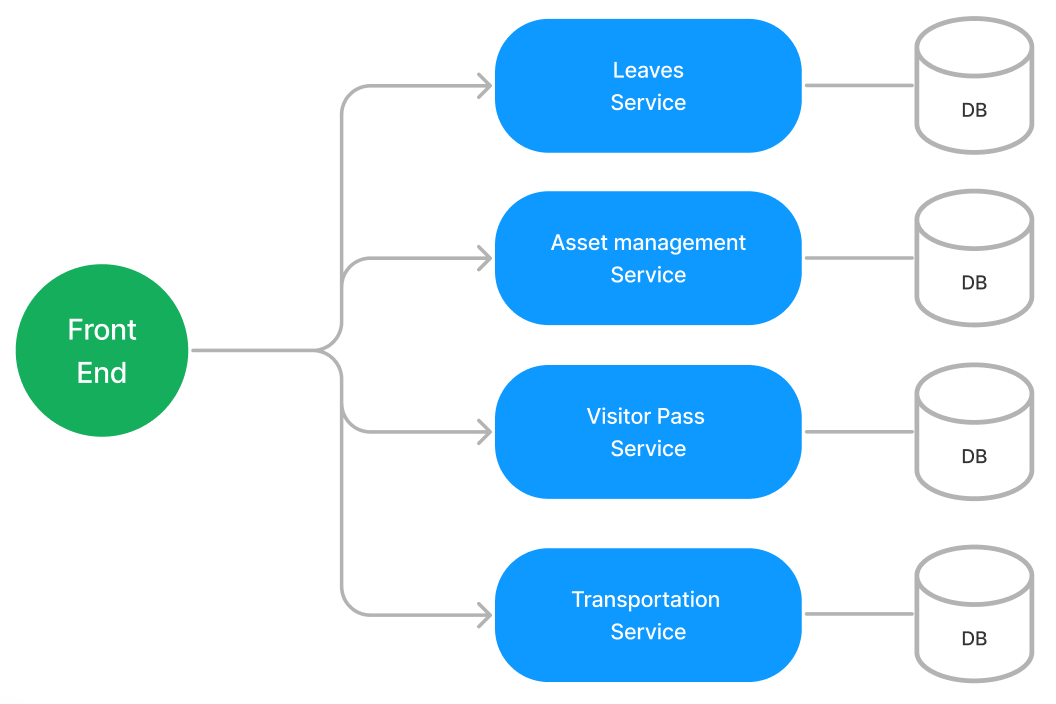


Figure 2 : Application Architecture Diagram

# 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 Front end.

# System Requirements

### **Module – Leave management**

This module will provide the following features

1. HRs of the system will be able to add new employees along with their role information
2. Employees can record their leaves into the system
3. Users who have created their credentials can login to the portal and perform various operations.

**Note**: By default when the application starts an HRs details must be fed into the system so that they can acquire new credentials and add new employees’ details into the system.

**Stage: Database Implementation**

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

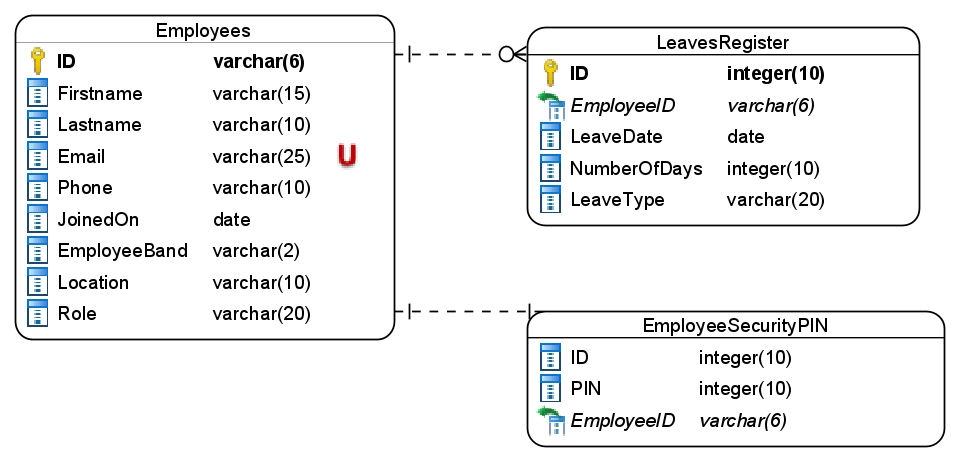


Figure 3 : ER Diagram – Leave management

1. Ensure that phone number is always exactly 10 digits
2. Allowed values for the location is – Hyderabad, Pune, Chennai, Bengaluru
3. Values allowed for the employee band are – A1, A2, B1, B2, C1, C2, L1, L2
4. PIN should be exactly 4 digits
5. Allowed values for leave type are Sick/Annual/Personal

**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. Add a new employee record in the database along with his PIN
   2. Validate PIN
   3. Return an employee by id
   4. Add leave details
   5. Return leave report based on date range

**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. Create a new employee along with PIN
   2. Validate employee PIN
   3. Add leave details
   4. Get leaves report based on date range
   5. Get the employee details based on ID
5. Following business rules must be implemented as part of the business service class
   1. If an employee search is performed based on ID and no such employee exists in the system then a user-defined exception called as “EmployeeNotFound” must be raised
   2. All exceptions which occurs in business logic must be raised to next level
   3. When a new employee is added validate the email address and phone number using a regular expression. Save only if it matches.
   4. Whenever a new employee is added by HR, the employee ID must be auto-generated. It should have 2 letter followed by 4 digits. First letter of first name and last name must be used the first 2 letters of employee ID.
   5. Each employee is allowed 1 sick leave per month, 12 annual leaves and 15 personal leaves annually.

**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 1 : Leave management – Endpoint – 1

|  |  |
| --- | --- |
| **URL** | api/employees/validatePIN |
| **Request Type** | POST |
| **User Role** | Employees |
| **Trigger** | Front end |
| **Description** | Validate the PIN of an employee to save his/her leave details |
| **Inputs** | PINValidationDTO |
| **Outputs** | Status code on success or validation errors |

Table 2 : Leave management – Endpoint – 2

|  |  |
| --- | --- |
| **URL** | api/leaves/add |
| **Request Type** | POST |
| **User Role** | Employeee |
| **Trigger** | Front end |
| **Description** | This endpoint will save the leave details added by employees |
| **Inputs** | LeaveDTO |
| **Outputs** | Status code |

Table 3 : Leave management – Endpoint – 3

|  |  |
| --- | --- |
| **URL** | api/employees/<id> |
| **Request Type** | GET |
| **User Role** | Authorized users only |
| **Trigger** | Front end and Other micro-services |
| **Description** | Fetch the details of an employee based on ID |
| **Inputs** | Employee ID |
| **Outputs** | Employee details or not found error |

Table 4 : Leave management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | api/employees/add |
| **Request Type** | POST |
| **User Role** | HR |
| **Trigger** | Front end |
| **Description** | An HR should be able to add a new employee into the system along with the employee PIN |
| **Inputs** | EmployeeDTO |
| **Outputs** | Status code and auto-generated employeeid |

Table 5 : Leave management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | api/leaves/report |
| **Request Type** | GET |
| **User Role** | HR |
| **Trigger** | Front end |
| **Description** | An HR should be able to view a leave report containing total leaves taken by each employee under in a given date range |
| **Inputs** | DateRangeDTO |
| **Outputs** | LeaveReportDTO |

**Stage: Front end design**

1. Create the following components as per the specification provided below.
   1. AddLeaveComponent
2. Develop a component which contains a form to add the leave details of an employee into the system
3. The component should accept employee id and PIN and once they are validated the employee should be able to provide his leave details
4. Use a dropdown list to accept the leave type.
5. Once all the details are validated, user must be able to submit the form and get an acknowledgement.
   1. Add EmployeeComponent
6. Create a component which is accessible to HRs which allows creating a new employee and assigning him to a role
7. The employee role must be selected from a set of radio button.
8. Allowed values for roles is Employees, HR, SecurityHead, TechSupportExe
9. Employee location should be accepted using a drop-down list. Use the location which are mentioned in database design
10. For employee band also use a dropdown list and the values provided during the database design
11. All details must be validated before the form is submitted and any validation error must be displayed to the user
    1. LeaveReportComponent
12. Develop a leave report component for HRs which can be accessed from the navigation bar.
13. The component should contain a form to accept the date range and a Get report button.
14. Once the report is fetched it must be displayed in a tabular format using bootstrap table.

**Stage: Integration of Frontend and backend**

1. Create a data service in the Front 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 – Visitor Pass**

This module will provide the following features

1. Employees in the system can raise a visitor pass request for the following types of visitors
   1. Parent
   2. Sibling
   3. Spouse
   4. Child
   5. Clients
2. System should allow employees to raise a new pass request for above mentioned visitor types
3. The security head of the location will be responsible for approving/rejecting the visitor pass requests raised by the employees
4. Users can view their visitor pass request status
5. Use role toggling in the Front-end to show the functionalities of the roles, Login component not needed. For the Employee role, provide a textbox to capture the employee id

**Stage: Database Implementation**

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

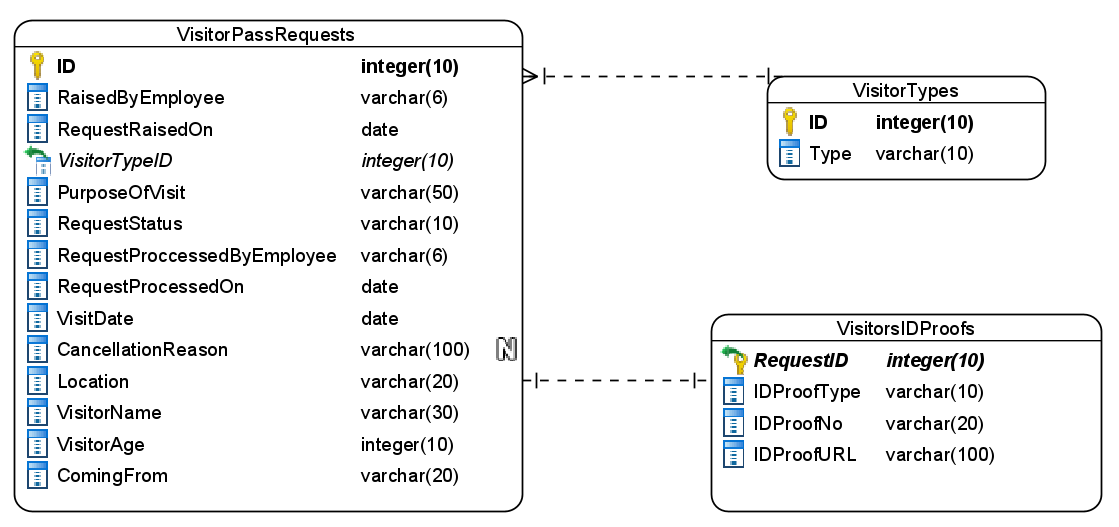


Figure 4 : ER Diagram – Visitor Pass

1. Enforce the following constraints apart from primary and foreign keys
   1. RequestStatus must only accept values as – pending, approved, rejected
   2. IDProofTypes must be one of the these – Aadhar, VoterID, Passport
   3. RequestRaisedOn must automatically take the today’s date

**Note**: Visitor types table can be auto-populated with the following values as – Parents, Child, Sibling, Spouse, and Client.

**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. Fetch the visitor types
   2. Add a new visit request
   3. Fetch pending visit request based on location
   4. Approve/reject request
   5. Fetch requests created by a user

**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. Return a collection of visitor types
   2. Add a visitor request
   3. Approve or reject a visitor request
   4. Fetch request from a particular user
   5. Return all pending request based on a location
5. Following business rules must be implemented as part of the business service class
   1. A new visitor request must be raised 1 week in advance
   2. Each employee can raise only 2 visitor request for their family members in a given calendar month
   3. Visit request for family member can only be raised for weekend
   4. For client visit there is no restriction on how many visit requests are placed.
   5. When a request is rejected by the security head a valid reason must be provided
   6. All visitor ID proofs must be uploaded into a local folder. The filename must be based on VisitorIDProof number
   7. If visitor pass request has reached the maximum limit for a user in given month then throw a user-defined exception as “MaximumPassRequestLimitReached”

**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 6 : Visitor Pass – Endpoint – 1

|  |  |
| --- | --- |
| **URL** | api/passrequests/<location> |
| **Request Type** | GET |
| **User Role** | SecurityHead |
| **Trigger** | Front end |
| **Description** | A security head must be able to fetch all the pending visitor pass requests |
| **Inputs** | Location |
| **Outputs** | VisitorPassRequestDTOs containing all visitor pass request details along with ID proof |

Table 7 : Visitor Pass – Endpoint – 2

|  |  |
| --- | --- |
| **URL** | api/passrequests/new |
| **Request Type** | POST |
| **User Role** | Employee |
| **Trigger** | Front end |
| **Description** | A logged in employee should be able to successfully raise a visitor pass request |
| **Inputs** | VisitorPassRequestDTO containing all visitor pass request details |
| **Outputs** | Status code on success or validation errors |

Table 8 : Visitor Pass – Endpoint – 3

|  |  |
| --- | --- |
| **URL** | api/vistortypes |
| **Request Type** | GET |
| **User Role** | Employee |
| **Trigger** | Front end |
| **Description** | An employees should be able to view the types of visitor available |
| **Inputs** |  |
| **Outputs** | VisitorTypeDTOs |

Table 9 : Visitor Pass – Endpoint – 4

|  |  |
| --- | --- |
| **URL** | api/passrequests/approvereject |
| **Request Type** | POST |
| **User Role** | Security Head |
| **Trigger** | Front end |
| **Description** | The location security head is responsible for updating the status of visitor pass request |
| **Inputs** | UpdatePassRequestDTO containing requestid, status and reason |
| **Outputs** | Status code |

Table 10 : Visitor Pass – Endpoint – 5

|  |  |
| --- | --- |
| **URL** | api/passrequests/<requestid> |
| **Request Type** | GET |
| **User Role** | SecurityHead |
| **Trigger** | Front end |
| **Description** | Security head should be able to examine all details of a given visitor pass request |
| **Inputs** | RequestID |
| **Outputs** | VisitorPassRequestDTOs containing all visitor pass request details |

**Stage: Front end design**

1. Create the following components as per the specification provided below.
   1. PassRequestListComponent
2. Create a component which can display all the visitor pass request into a tabular format
3. Against each table row a view details buttons must be available
4. Define a route for the component in the nav bar of the application
5. If an employee is accessing the component then only his pass request will be displayed
6. If a security head is accessing the component then pass request raised under his location will be displayed.
   1. PassRequestDetailsComponent
7. An employee or the security head will be able to view details of a single pass request by clicking view details button in the PassRequestList component
8. All details related to a pass request must be displayed in this component in a definition list
9. If the user accessing the component is a security head then ApproveRejectRequest component must be rendered as a child of this component
   1. ApproveRejectRequestComponent
10. Design a component which will allow security heads to approve or reject a pass request
11. The component should provide with 2 buttons one for approve and other for reject request.
12. Component should also contain a textbox for rejection reason in case user click on reject then reason must be provided
13. Once the operation is completed successfully then an appropriate message must be displayed
    1. NewPassRequestComponent
14. Create a component which allows employees to raise a new pass request
15. The component should accept all the required details for the pass request
16. Visitor types must be selected from a dropdown list whose data must be populated based on VisitorType table.
17. Once the user fills all the data, proper validation must be in place to ensure it’s free of errors before submitting.

**Stage: Integration of Frontend and backend**

1. Create a data service in the Front 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 – Asset management**

This module will provide the following features

1. An employee of the Cognizant will be able to register the assets assigned to them on the portal
2. Employees will also be able to raise support tickets for their assets
3. The support executives will be responsible for resolving the tickets
4. Employees can track their support tickets

**Stage: Database Implementation**

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

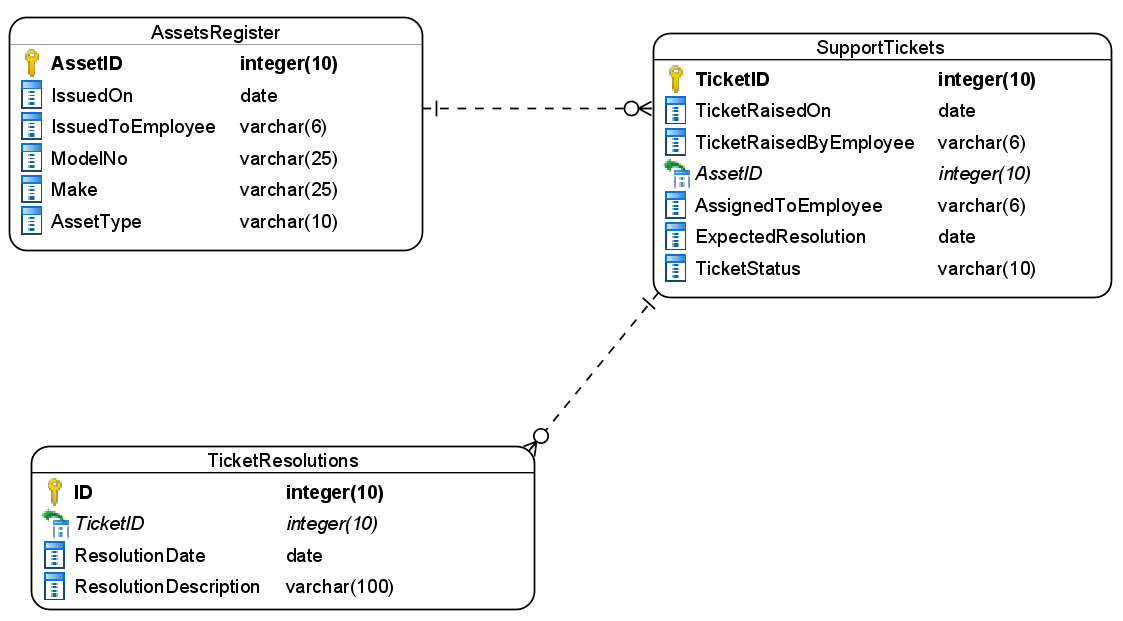


Figure 5 : ER Diagram - Asset Management

1. Apart from primary and foreign keys mentioned in the ER diagram enforce the following additional constraints
   1. IssuedOn should not be a future date
   2. ResolutionDate must be automatically taken as current date time
   3. Allowed values for ticket status is – Open, Resolved

**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. Add new record in asset table
   2. Add new record in support tickets
   3. Add new record in resolutions table
   4. Update the ticket status
   5. Return all support tickets assigned to a user
   6. Search a single support ticket by ticket id

**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. Register a new asset
   2. Create a new support ticket
   3. Add a resolution and close the ticket
   4. Get all tickets assigned to support executive
   5. Get a ticket by ID
5. Following business rules must be implemented as part of the business service class
   1. Each user in cognizant is assigned 5 devices and they can register them on the portal. More than 5 devices cannot be registered. The types of assets user can register are Laptop, Mobile, DataCard, Headphone and Storage device. If a user tries to add more than 5 devices then a user-defined exception must be raised as “MaximumDeviceLimitReached”
   2. When a new support ticket is created it must be automatically assigned to a support staff. Each support staff can have a maximum of 10 open support tickets per day. Expected resolution date must be auto-generated based on the following SLAs
      1. Laptop – 2 days
      2. Mobile – 5 days
      3. DataCard, HeadPhone and Storage – 3 days
   3. A ticket ID must be auto-generated and displayed to the user whenever a ticket is created
   4. If an employee is trying to raise a support ticket for non-registered asset then “AssetNotRegistered” exception must be thrown by the system

**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.
3. 100% of the code must be covered by unit tests

**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 11 : Asset Management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | api/asset/register |
| **Request Type** | POST |
| **User Role** | Employee |
| **Trigger** | Front end |
| **Description** | A logged in employee should be able to register their asset into the system |
| **Inputs** | AssetDetailsDTO |
| **Outputs** | Status code on success or validation errors |

Table 12 : Asset Management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | api/supportrequests/new |
| **Request Type** | POST |
| **User Role** | Employee |
| **Trigger** | Front end |
| **Description** | An employee who has registered an asset into the system should be able to create a new support ticket |
| **Inputs** | SupportTicketDTO |
| **Outputs** | Status code and ticketId on success or validation errors |

Table 13 : Asset Management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | api/supportrequests/<executiveid> |
| **Request Type** | POST |
| **User Role** | TechSupportExe |
| **Trigger** | Front end |
| **Description** | A tech support execute should be able to fetch all open tickets which are assigned to him/her |
| **Inputs** | Support executive employee id |
| **Outputs** | SupportTicketDTO |

Table 14 : Asset Management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | api/supportrequests/<ticketid> |
| **Request Type** | GET |
| **User Role** | Employee, TechSupportExe |
| **Trigger** | Front end |
| **Description** | An employee or a tech support executive must be able to view all detail based on a support request id/ticket id |
| **Inputs** | TicketId |
| **Outputs** | SupportTicketDetailsDTO containing all ticket details along with details of asset and the employee who raise the ticket |

Table 15 : Asset Management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | api/supportrequests/<ticketid>/resolve |
| **Request Type** | PUT |
| **User Role** | TechSupportExe |
| **Trigger** | Front end |
| **Description** | A tech support executive should be able to write the resolution provided for a ticket and close it |
| **Inputs** | TicketId, Resolution |
| **Outputs** | Status code on success or validation errors |

**Stage: Front end design**

1. Create the following components as per the specification provided below.
   1. RegisterAssetComponent
2. Create a component which allows users to register a new asset into the system
3. Component should accept asset type from a radio button. Type of allowed assets is mentioned earlier.
4. User should be able to register an asset only if all the input fields are valid.
5. Provide a navigation to the component in application navbar.
   1. SupportTicketsListComponent
6. Create a component which can display the open support tickets in as bootstrap cards.
7. Each support ticket should also have a navigation link to Resolve ticket
8. On successful creation of a ticket the ticketid must be displayed to the user.
   1. ResolveTicketComponent
9. Create a ResolveTicket component such that a tech support executive should be able to navigate to it by clicking the resolve ticket button in support tickets list
10. All the ticket details along with asset details and details of employee must be displayed on the page.
11. The component should provide a form to enter the resolution details and a resolve button
12. Resolution details are mandatory to resolve a ticket
    1. TrackTicketComponent
13. Create a component using which a user can track the status of their ticket with the help of ticket id.
14. Once the ticket id is entered the user should be able to view the ticket details along with the resolution provided for the ticket.

**Stage: Integration of Frontend and backend**

1. Create a data service in the Front 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 – Transportation Service**

The registered employees of One Cognizant portal will be able to avail the transportation services. This module will provide the following services to the employees.

1. Search for the available transportation services
2. Subscribe to the transportation service
3. Unsubscribe from a transportation service.

**Note**: Pre-populate few records in the Transporation services tables

**Stage: Database Implementation**

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

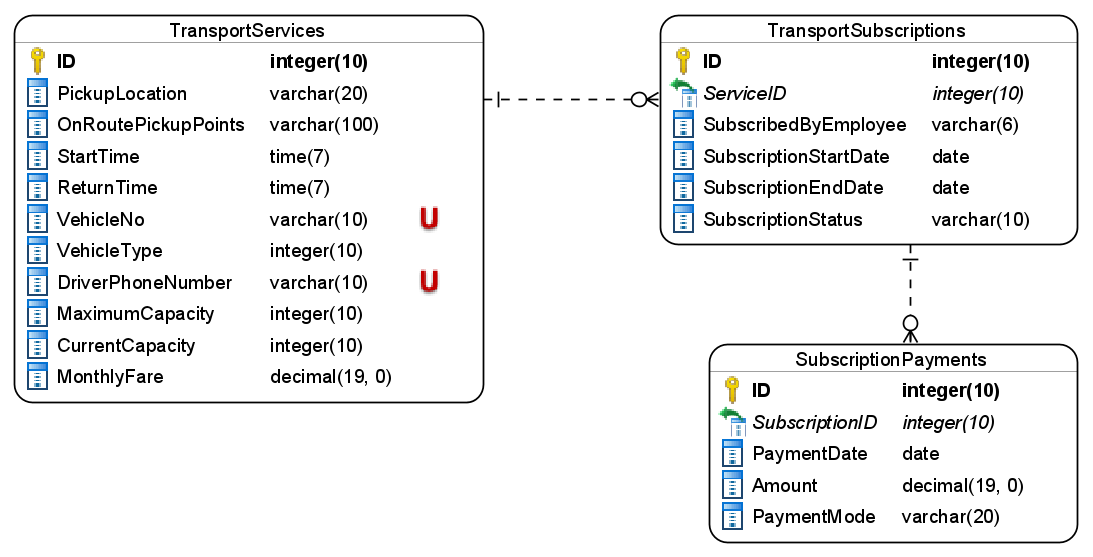


Figure 6 : ER Diagram - Transportation Service

1. Enfore the following additional constraints apart from primary key, foreign key and unique keys
   1. Return time must also be greater than start time
   2. Driver phone must be exactly 10 digits
   3. Current capacity should not exceed maximum capacity
   4. Payment date should be automatically taken
   5. Subscription end date should not be less then subscription start date
   6. Allowed values for subscription status are – Active, Expired, Cancelled

**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 the available transports based on pickup point
   2. Add a new subscription
   3. Update an existing subscription
   4. Fetch fare for a transportation service
   5. Update the transportation capacity

**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. Fetch available transportation for a given pickup point
   2. Create a new subscription and pay for it
   3. Cancel a subscription and return refundable amount
   4. View a subscription by id
   5. Calculate the total fare for a given subscription
   6. Update the current capacity
5. Following business rules must be implemented as part of the business service class
   1. When user searches for transportation display only those which are available for booking
   2. Employees can subscribe to monthly, quarterly, half yearly and yearly services. Subscription end date must be accordingly generate and amount payable must be displayed accordingly.
   3. Whenever a user successfully subscribes/unsubscribes to a service then the current capacity must be updated accordingly
   4. Whenever a subscription is successfully created an auto-generated subscriptionid must be displayed backed

**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 16 : Transportation Service - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | api/transports/<pickup-point> |
| **Request Type** | GET |
| **User Role** | All registered users |
| **Trigger** | Front end |
| **Description** | Users should be able to view the available transporation services for their pickup locations |
| **Inputs** | Pickup point |
| **Outputs** | TransportationServicesDTOs |

Table 17 : Transportation Service - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | api/subscriptions/new |
| **Request Type** | POST |
| **User Role** | All registered users |
| **Trigger** | Front end |
| **Description** | If a transportation is available user should be able to subscribe to it by making a payment |
| **Inputs** | SubscribeDTO with subscription details and payment details |
| **Outputs** | Status code or error messages |

Table 18 : Transportation Service - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | api/transports/calculatefare |
| **Request Type** | POST |
| **User Role** | All registered users |
| **Trigger** | Front end |
| **Description** | User will be able to view the applicable fare for the plan they have selected |
| **Inputs** | FareDTO |
| **Outputs** | Fare amount and status code |

Table 19 : Transportation Service - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | api/subscription/<subscription-id> |
| **Request Type** | GET |
| **User Role** | All registered users |
| **Trigger** | Front end |
| **Description** | A user having a subscription should be able to view the subscription details |
| **Inputs** | Subscription id |
| **Outputs** | SubscriptionDetailsDTO including subscription details and selected transportation details |

Table 20 : Transportation Service - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | api/subscriptions/<subscriptionid>/unsubscribe |
| **Request Type** | DELETE |
| **User Role** | All registered users |
| **Trigger** | Front end |
| **Description** | A subscribed user with an active subscription must be able to cancel the subscription |
| **Inputs** | Subscriptionid |
| **Outputs** | Status code |

**Stage: Front end design**

1. Create the following components as per the specification provided below.
   1. SearchTransportationComponent
2. Create a component which allows users to view the currently available transportation services in a tabular format
3. User should provide a pickup location and click the search button to view the available transport services
4. If a service is available it should have a button to subscribe
5. Provide a navigation link in the navbar of the application to access the component
   1. SubscribeTransportationComponent
6. Create a component which will show the transporation detail to the user for a particular transportation selected by user in search transporation component.
7. It should display all the transportation details and accept the subscription details and payment options
8. Subscription duration should be selected from a set of radio button and payment mode must be selected via a drop down list
9. Once a user selects these details calculate and display the fare to the user
10. Once all the details are validated then user should be able to subcribe by clicking the subscribe button.
    1. CancelSubscriptionComponent
11. A logged in user should be able to navigate to cancel subscription component
12. Create a cancel subscription component which will accept the subscription id and displays the subscription details to the user
13. If the current status of subscription is active user should be able to cancel

**Stage: Integration of Frontend and backend**

1. Create a data service in the Front 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

|  |
| --- |
| [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 21-August-2022 by Khaleelullah Hussaini Syed | | | |
| V1.0.1 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| 8 | Khaleelullah Hussaini Syed | 16 September 2022 | Added link to design consideration for .NET and Java |
| V1.0.2 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| 2.3 | Khaleelullah Hussaini Syed | 1 October 2022 | Updated project scope with new module on leave management |
| 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 | New leave management module inplace of authentication |
| 6.1.2 | Updated visitor pass module to remove filters api endpoint and API gateway |
| 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 |