|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Ride Sharing Platform**  **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** | 23 August 2022 |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc116403328)

[2.0 Introduction 4](#_Toc116403329)

[2.1 Purpose of this document 4](#_Toc116403330)

[2.2 Project Overview 4](#_Toc116403331)

[2.3 Scope 4](#_Toc116403332)

[3.0 Use Case Diagram 6](#_Toc116403333)

[4.0 System Architecture Diagram 7](#_Toc116403334)

[5.0 Development Phases 7](#_Toc116403335)

[6.0 System Requirements 8](#_Toc116403336)

[**6.1.1** **Module – User verification** 8](#_Toc116403337)

[**6.1.2** **Module – Vehicle Management** 11](#_Toc116403338)

[**6.1.3** **Module – Ride management** 15](#_Toc116403339)

[**6.1.4** **Module – Incident management** 19](#_Toc116403340)

[7.0 Deployment requirements 22](#_Toc116403341)

[8.0 Design Considerations 23](#_Toc116403342)

[8.1.1 Most Important and Common rules 23](#_Toc116403343)

[9.0 Reference learning 24](#_Toc116403344)

[10.0 Project Templates 25](#_Toc116403345)

[11.0 Change Log 26](#_Toc116403346)

# 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 “Ride sharing 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

Raheja IT park in Hyderabad has a host of companies in their huge campus. To reduce the vehicular traffic and to make the city more environment friendly, the IT team at Raheja has decided to develop a ride sharing platform. The platform will be accessible to all employees working in various companies with in the IT park. It will allows employees who uses their personal vehicle for communte to provide pooling services at a nominal charge. Platform will help those with a vehicle to make an extra income and riders can safely travel with their fellow colleagues and save money where-as the overall vehicular traffic will be reduced leading to environment friendly eco-system.

## Scope

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

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Application verification** module | * This module will allow the employees working in the IT park to register themselves with the system. * The security team can view the the list of application * The security team can accept/reject a user application |
| REQ\_02 | **Vehicle management** module | * The users of the system will be able to register their vehicle on the platform to be used for sharing the rides to/from from office * All registered vehicle will be inspected by the security admins and after verification the users can avail the ride sharing facility * Uses will also be able to delete vehicle from the system and add a new one which will again be inspected by the security team. |
| REQ\_03 | **Ride management** module | * The ride management module will be used by users who to plan their rides by specifying the starting and endpoint for the rides * Other registered users will be able to search and filter for the available rides * If any rides are matching the users travel requirements they can book the ride by paying the required amount |
| REQ\_04 | **Incident management** module | * The incident management module will allow the users to report any incidents during their ride * The security heads will be able to investigate and submit a report on the incident filed * Users can also track the incidents filed by them to know the status/report |

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

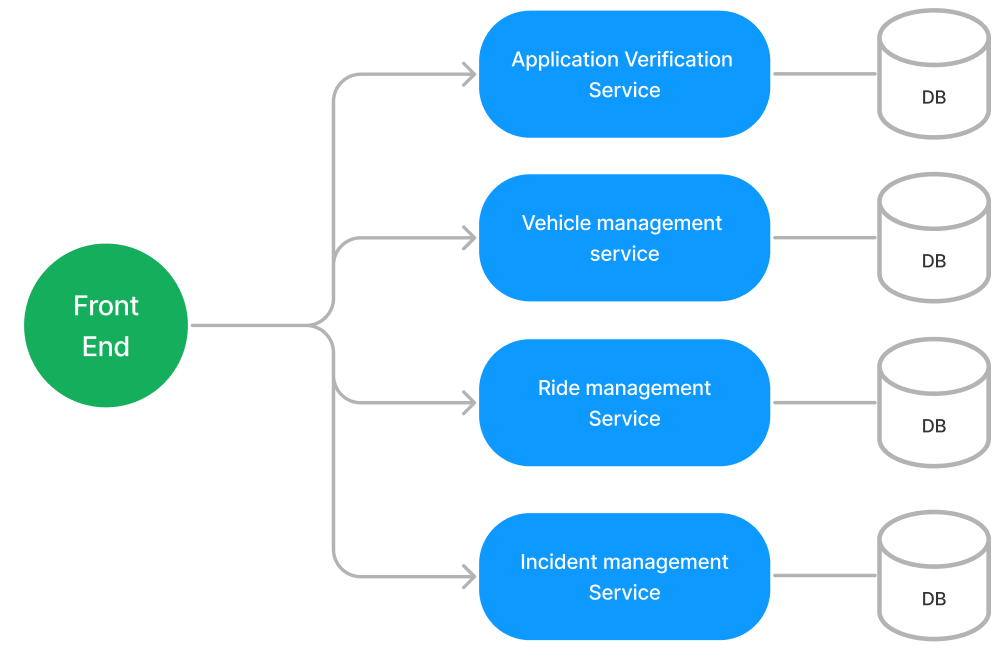


Figure 2 : Application Architecture Diagram

# 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 – User verification**

Employees working in Raheja IT park in various companies can use this module to perform the following operations

1. Apply for using the platform as a motorist/rider
2. The security heads can view the list of application for purpose of verification
3. The security heads will be responsible for accepting and rejecting the application request

**Stage: Database Implementation**

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

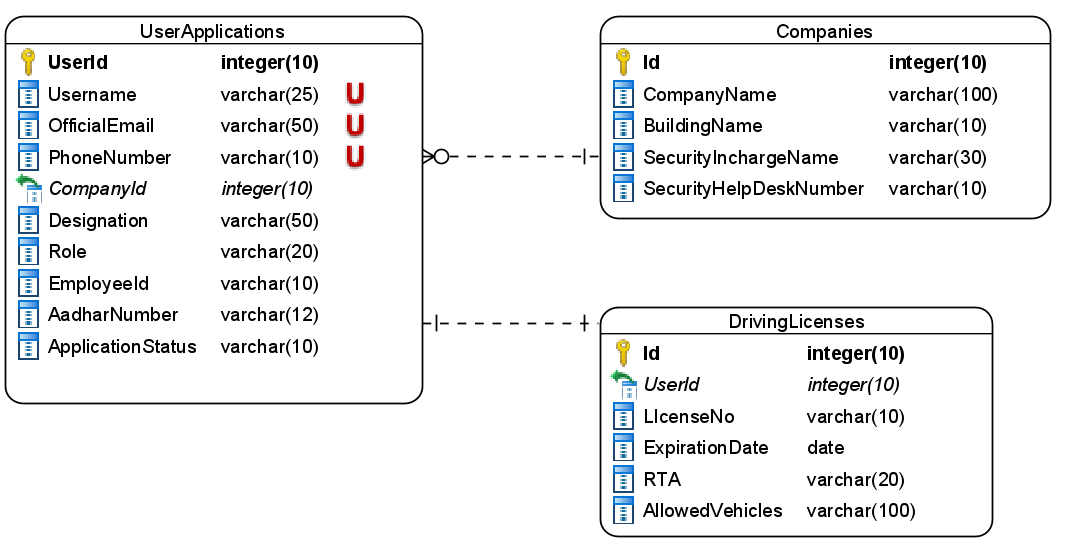


Figure 3 : ER Diagram – User verification

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. Phone number and SecurityHelpDeskNumber must be exactly 10 characters
   2. Aadhar number must be 12 characters long
   3. Registration status should only accept values like “New/Approved/Rejected”
   4. Expiration date must not be a past date
   5. The allowed values for the roles are – Motorist, Rider and SecurityHead

**Note**: Pre-populate the data into the companies table when the application starts. Also pre-populate 1 security head details into the system on 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 the companies list
   2. Insert new application
   3. Return pending application request
   4. Return pending application by UserId
   5. Approve/reject registration request

**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 the companies list
   2. Create a new application
   3. Return an application by user id
   4. Return new pending application requests
   5. Approve/reject registration request
5. Following business rules must be implemented as part of the business service class
   1. When a new motorist registers he must provide a valid driving license.
   2. Ensure that the driving license is not expired
   3. Driving license number must be 10 characters in format as 3 letters followed by 4 digits and 3 letters.
   4. Ensure that aadhar number is exactly 12 digits and phone number is exactly 10 digits starting with 9
   5. If a user is trying to register as motorist without providing a driving license then raise a user-defined exception as “InvalidMotoristRegistration”
   6. Only users whose request are successfully approved should be allowed to login

**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 : User verification - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/companies |
| **Request Type** | GET |
| **User Role** | Anonymous users |
| **Trigger** | Front end |
| **Description** | Endpoint will be return the data on the companies in Raheja IT park which can be used during registration |
| **Inputs** |  |
| **Outputs** | CompanyDTOs |

Table 3 : User verification - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/applications/new |
| **Request Type** | POST |
| **User Role** | Anonymous users |
| **Trigger** | Front end |
| **Description** | This end point will allow users to create a new application into the system. It can be used by both motorists and riders. |
| **Inputs** | NewApplicationDTO |
| **Outputs** | Status code or validation errors |

Table 4 : User verification - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/applications/ |
| **Request Type** | GET |
| **User Role** | Security Head |
| **Trigger** | Front end |
| **Description** | The security heads will use this endpoint to get a list of pending application in the system |
| **Inputs** |  |
| **Outputs** | NewApplicationDTOs |

Table 5 : User verification - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/applications/approvereject |
| **Request Type** | PUT |
| **User Role** | Security Head |
| **Trigger** | Front end |
| **Description** | Security heads can approve or reject the registration request using this end point |
| **Inputs** | UpdateApplicationDTO |
| **Outputs** | Status code |

Table 6 : User verification - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/applications/<userId> |
| **Request Type** | GET |
| **User Role** | Security Heads |
| **Trigger** | Front end |
| **Description** | Using this endpoint a security head can view get detailed information on an application submitted by user |
| **Inputs** | UserId |
| **Outputs** | NewApplicationDTO |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. NewApplicationComponent
2. Create a component which will be used by motorists and riders to submit a new application for the verification
3. Define a navigation to the component in the application navbar
4. If a motorist is registering with the system then his license details must also be accept.
5. Once all details are validated the form should be allowed to be submitted and an acknowledgement must be displayed.
6. Role of user should be displayed in the form of radio buttons and company must be selected with the help of a dropdown list.
7. ApplicationRequestListComponents
8. Develop a component which is accessible by security heads from the application navigation bar
9. The component should display the application requests which are pending for approval in a bootstrap table
10. Each row should also have a link to navigate to ApplicationRequestComponent by passing the user id.
11. ApplicationRequestComponent
12. Design a component which allows the security head to view a registration request and approve/reject it
13. The component should display one registration request at a time based on the id passed in the route
14. There should also a dropdown to choose the status and update it into the system.

**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 – Vehicle Management**

This module will provide the following features to the motorists and security heads

1. The motorist will be able to register their vehicle into the system by providing the relevant details like RC, Insurance and Pollution under control (PUC)
2. Once the motorists registers their vehicles then a security head will be responsible for inspecting the details and approving/rejecting them
3. A motorist will also be able to remove a vehicle from the system if required.

**Stage: Database Implementation**

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

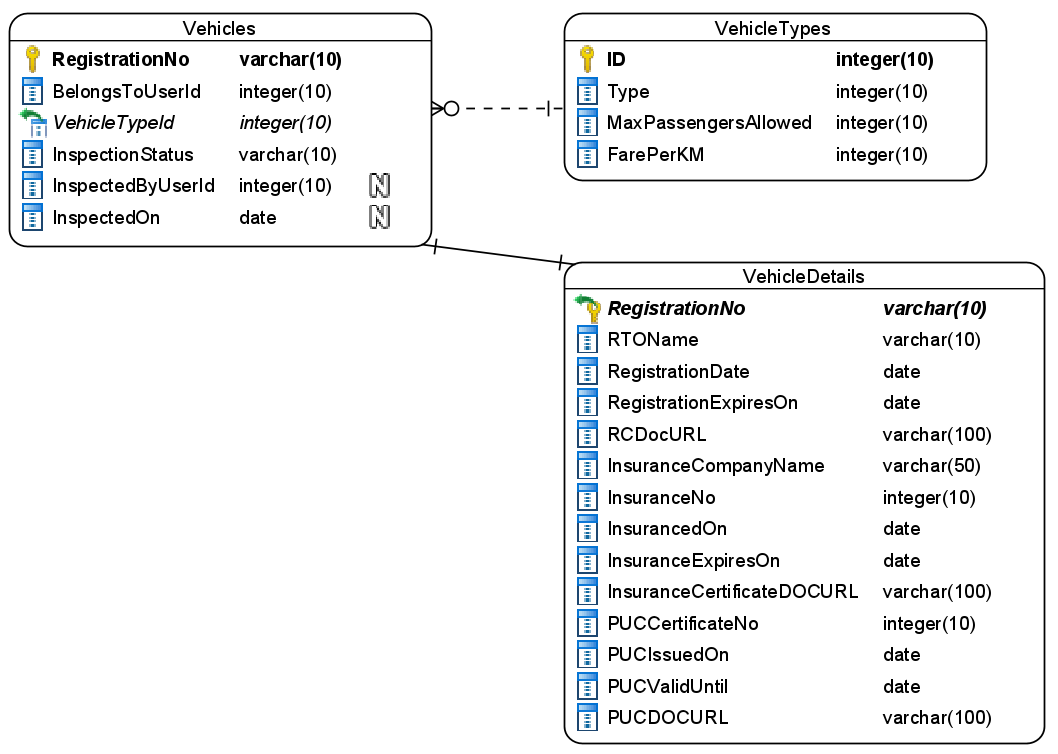


Figure 4 : ER Diagram – Visitor Pass

1. Apply the following constraints apart from primary keys and foreign keys on the database
   1. Inspection status should allows pending/approved/rejected as the values
   2. Vehicle registration number must be exactly 10 characters long
   3. RegistrationDate, InsuranceExpiresOn and PUCValidUntil must not be a past date.

**Note**: The data in the VehicleTypes table must be seeded when the application starts. You can seed few vehicle types like SUV, Sedan, 2Wheeler, etc into the system.

**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 vehicle types
   2. Insert a new vehicle
   3. Delete a vehicle
   4. Fetch a vehicle by user id
   5. Fetch a single vehicle page wise
   6. Approve/reject a vehicle

**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 vehicle types
   2. Add a new vehicle
   3. Delete a vehicle
   4. Fetch vehicle pending for approval
   5. Approve/reject a vehicle
   6. Fetch a vehicle by user id
5. Following business rules must be implemented as part of the business service class
   1. When a user registers a new vehicle ensure that the registration number of vehicle matches the following format – 2 letters followed by 2 digits, followed by 2 letter and 4 digits. It must be exactly 10 characters long
   2. If the vehicle registration doesn’t matches the above given format raise a user-defined exception as “InvalidRegistrationNoException”.
   3. A vehicle which is more than 15 years old should not be allowed to register.
   4. When adding a new vehicle ensure that RC is valid for atleast 2 years, Insurance is valid for 1 year and PUC is valid for a minimum of 6 months.
   5. Ensure that the document being uploaded in the form of RC, Insurance certificate and PUC is not more than 1024 Kilo byte each.

**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 7 : Vehicle Management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/vehicles/vehicletypes |
| **Request Type** | GET |
| **User Role** | Motorists |
| **Trigger** | Front End |
| **Description** | A motorist should be able to register his vehicle by choosing the vehicle types available in the system |
| **Inputs** |  |
| **Outputs** | VehicleTypeDTOs |

Table 8 : Vehicle Management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/vehicles/addvehicle |
| **Request Type** | POST |
| **User Role** | Motorist |
| **Trigger** | Front End |
| **Description** | A registered motorist should be able to register their vehicle into the system by providing details of the vehicle, RC, Insurance and PUC |
| **Inputs** | RegisterVehicleDTO |
| **Outputs** | Status code or validation errors |

Table 9 : Vehicle Management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/vehicles/delete/<vehicleid>/<userid> |
| **Request Type** | DELETE |
| **User Role** | Motorist |
| **Trigger** | Front end |
| **Description** | A motorist should be able to remove a vehicle along with it’s details from the system. A motorists can only delete his/her own vehicle |
| **Inputs** | VehicleID and UserId |
| **Outputs** | Status code or error message |

Table 10 : Vehicle Management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/vehicle/<userid> |
| **Request Type** | GET |
| **User Role** | Motorists |
| **Trigger** | Front end, Ride Micro service |
| **Description** | This endpoint will allow users to get all details of a vehicle along with it’s type and other details |
| **Inputs** | UserId |
| **Outputs** | VehicleDTO containing vehicle, it’s details and it’s type |

Table 11 : Vehicle Management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/vehicles/pendingapprovals/<pageno> |
| **Request Type** | GET |
| **User Role** | Security Head |
| **Trigger** | Front End |
| **Description** | A security head will use this end point to view details of new vehicle registration for the purpose of inspection so that it can be approved or rejected |
| **Inputs** | PageNo – data will be divided into pages where in each page will display a single record |
| **Outputs** | VehicleDTO containing vehicle, it’s details and it’s type |

Table 12 : Vehicle Management - Endpoint - 6

|  |  |
| --- | --- |
| **URL** | /api/vehicles/approveorreject |
| **Request Type** | PUT |
| **User Role** | Security Head |
| **Trigger** | Font end |
| **Description** | This end point will allow the security head update the vehicle registration status |
| **Inputs** | UpdateVehicleRegistrationDTO |
| **Outputs** | Status code or error messages |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. AddVehicleComponent
2. Create a new AddVehicleComponent which provides a form to the user for registering their vehicle into the system
3. Form should contain a dropdown list to allow user for choosing the vehicle types based on which the fare per km and maximum number of passengers allowed must be displayed
4. Once all details are validated then form must be allowed to be submitted and acknowledgment must be displayed
5. Define a navigation link to the component available only for Motorists
6. MyVehicleComponent
   1. Create a MyVehicle component which will be used by motorists to see their registered vehicle details.
   2. A motorist should also have an option to delete the vehicle from the system
   3. A delete should only take place after a confirmation from the user
   4. After the operation is completed an appropriate message in the form of acknowledgement must be displayed to the user
   5. Create a navigate to the component such that it is accessible to the motorists.
7. ApproveRejectVehicleComponent
   1. Design a component for security heads which will display details of vehicles which are newly registered into the system
   2. The component should display details of one vehicle at a time which an option to navigate to the next and previous vehicle details
   3. It should also provide the user with an option to approve or reject the vehicle registration
   4. Provide a navigation to the component in the application menu.

**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 – Ride management**

The ride management module will be used by motorists and riders to achieve the following objectives

1. A motorist should be able to schedule a new ride
2. A rider should be able to search for a ride based on their preference
3. Once a rider finds a ride of their choice they should be able to make a booking for it

**Stage: Database Implementation**

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

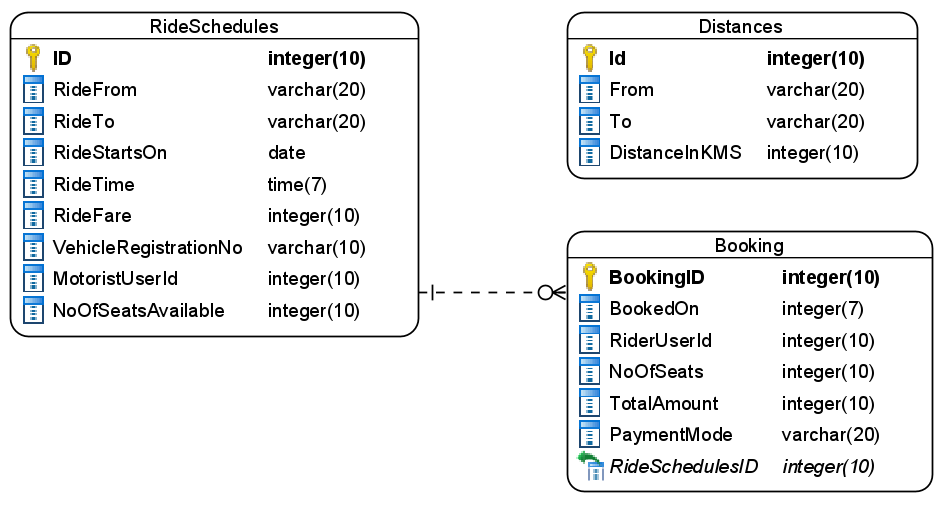


Figure 5 : ER Diagram - Asset Management

1. Apart from primary and foreing keys implement the following additional constraints
   1. RideStartsOn and RideTime must not be a past date time
   2. Vehicle registration number me exactly 10 characters long
   3. From and To in the distance table should not be same for a given route
   4. Each users cannot book more than 2 seats

Note: Pre-poplate the Distance table with data

**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 distance details
   2. Insert a new ride schedule
   3. Search a ride schedule
   4. Insert a new booking
   5. Search existing bookings for 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 following the single responsibility principle which perform the given operations as follows
   1. Fetch the distance details
   2. Create a new ride
   3. Search for a ride
   4. Book a ride
   5. Calculate fare for a ride
5. Following business rules must be implemented as part of the business service class
   1. A motorist should be allowed to create a new ride only if their vehicle is approved by the security head
   2. The number of seats in a ride should not exceed the limit mentioned in vehicle type in vehicle management module
   3. When a new ride is created, fare should be calculated automatically based on distance and price per KM for the given vehicle type
   4. A rider should not be allowed to reserve more than 2 seats in a ride

**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.
5. This micro-service will also communicate with Vehicle Management Service to get details of the vehicle for calculation of fares and max passenger limit

Table 14 : Ride Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/distances |
| **Request Type** | GET |
| **User Role** | Motorists, Riders |
| **Trigger** | Front end |
| **Description** | A motorist should be able to create a new ride for the distance present in the database and same can be used by riders for searching rides |
| **Inputs** |  |
| **Outputs** | DistanceDTOs |

Table 15 : Ride Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/rides/calculatefare |
| **Request Type** | GET |
| **User Role** | Motorists |
| **Trigger** | Front end |
| **Description** | A motorist will use the end point to calculate the fare for a given ride |
| **Inputs** | FareParametersDTO containing distance, vehicle regno |
| **Outputs** | Fare amount |

Table 16 : Ride Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/rides/schedule |
| **Request Type** | POST |
| **User Role** | Motorists |
| **Trigger** | Front end |
| **Description** | Using this end-point a motorist will be able to create a new ride |
| **Inputs** | RideScheduleDTO |
| **Outputs** | Status code |

Table 17 : Ride Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/rides/search |
| **Request Type** | GET |
| **User Role** | Riders |
| **Trigger** | Front end |
| **Description** | Using this end point a rider should be able to search for an available ride using the search parameters like from, to, min and max price and available seats |
| **Inputs** | SearchCriteriaDTO |
| **Outputs** | RideScheduleDTOs |

Table 18 : Ride Management - End point - 5

|  |  |
| --- | --- |
| **URL** | /api/rides/book |
| **Request Type** | POST |
| **User Role** | Riders |
| **Trigger** | Front end |
| **Description** | This end point will allow the riders to make a new booking and obtain a booking ID |
| **Inputs** | BookingDTO |
| **Outputs** | Status code and booking id |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. ScheduleRideComponent
2. Create a component which will allow the motorists to create a new ride
3. The component should allow motorist to select the from and to distances from a dropdown list
4. Once they choose the distance then the fare amount should be auto-calculated and displayed
5. They should also be allowed to select the ride date and time and the number of passengers they intend to onboard during the ride
6. Once all details are validated then the form must be submitted and an acknowledgement should be displayed
7. Provide a navigation to component for motorists via navbar
8. SearchRidesComponent
9. Design a search ride component which should be accessed by riders using the navigation bar
10. The component should allow the users to select the from and to distances from a dropdown
11. All available rides for the given distance should be displayed as bootstrap cards with a book button which navigates to bookride component
12. User should also be allowed to filter the rides based on minimum and maximum fare amount using a range slider
13. BookRideComponent
    1. Develop a book ride component which can be navigated to from search rides component
    2. It should allow users to choose a payment option and the number of seats they intent to reserve.
    3. Once the booking is confirmed display an acknowledgement along with booking 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 – Incident management**

During a ride, if there is any unfortunate incident like accident or quarrel with a fellow passenger, etc, then the services provided by this module can be availed.

1. It will allow the users to create a new incident
2. Track an incident
3. Update the investigation details into the system

**Stage: Database Implementation**

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

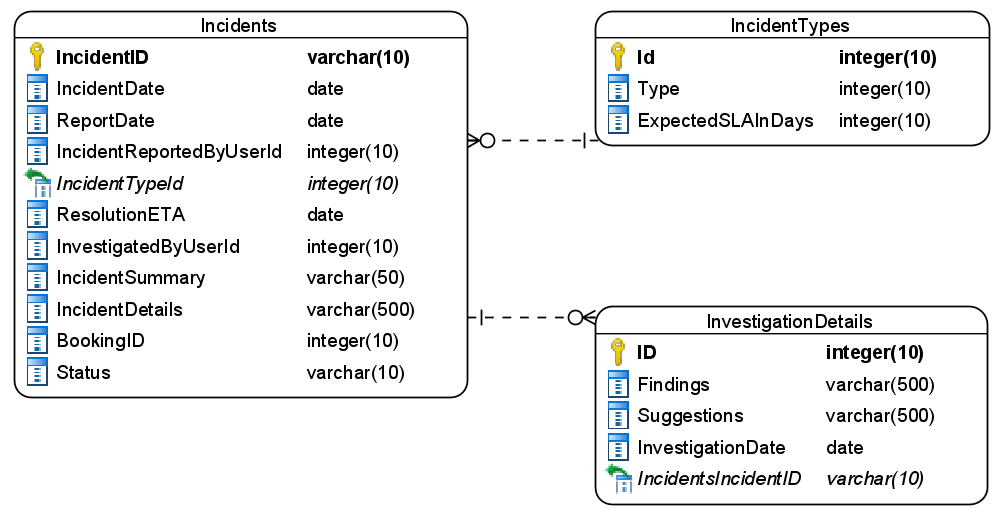


Figure 6 : ER Diagram - Transportation Service

1. Enfore the following constraints along with primary and foreign keys
   1. Report date should not be a future date
   2. Incident date can be today or a past date only
   3. Status for incident should be pending/closed.
   4. Pending should be the default value for status.

**Note**: Feed the data into the IncidentTypes table when the application starts

**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 incident types
   2. Insert a new incident
   3. Return all pending incidents
   4. Return an incident by id
   5. Update an incident 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. Get the incident types
   2. Calculate the ResolutionETA
   3. Add new incident
   4. Get pending incidents
   5. Get incident by Id
   6. Update incident report
5. Following business rules must be implemented as part of the business service class
   1. When a new incident is created an Incident ID should be auto-generated in the format as – xxxx-xxxx. The first 4 letters should be year of incident followed by 4 digit unique number
   2. ResolutionETA should be auto-calculated
   3. A user should only be allowed to report an incident within 2 days of the actual incident. If a user tries to report an incident after 2 days, raise a user-defined exception as “CannotReportIncidentException”
   4. A user cannot file more than 1 incident for a given booking id

**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 20 : Incident Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/incidents/types |
| **Request Type** | GET |
| **User Role** | Registered users |
| **Trigger** | Front end |
| **Description** | A user should be able to choose an incident type while filing a new incident report |
| **Inputs** |  |
| **Outputs** | IncidentTypeDTOs |

Table 21 : Incident Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/incidents/report |
| **Request Type** | POST |
| **User Role** | Registered users |
| **Trigger** | Front end |
| **Description** | Any registered user of the system should be able to report an incident happened during their ride |
| **Inputs** | NewIncidentDTO |
| **Outputs** | Status code and incident id |

Table 22 : Incident Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/incidents |
| **Request Type** | GET |
| **User Role** | Security Head |
| **Trigger** | Front end |
| **Description** | A security head will use this endpoint to fetch all the pending incident from the system sorted in order of incident report date |
| **Inputs** |  |
| **Outputs** | IncidentsDTOs |

Table 23 : Incident Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/incidents/<id> |
| **Request Type** | GET |
| **User Role** | Registered users |
| **Trigger** | Font end |
| **Description** | A registered user who has an incident id should be able to track the incident details. At the same time a security head may also access the incident details to file the investigation report |
| **Inputs** | IncidentId |
| **Outputs** | IncidentDTO |

Table 24 : Incident Management - End point - 5

|  |  |
| --- | --- |
| **URL** | /api/incidents/<id>/investigationreport |
| **Request Type** | PUT |
| **User Role** | Security head |
| **Trigger** | Font end |
| **Description** | The security head in the system will use this end point to provide the investigation report and close a given incident |
| **Inputs** | IncidentId and InvestigationReportDTO |
| **Outputs** | Status code |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. NewIncidentComponent
2. Create a component and allow the navigation to it for motorists and riders
3. The component should provide a form for the users to report an incident
4. Use a dropdown to list the types of incidents
5. User a multi-line textbox for the incident details
6. Validate all the data before it’s submitted
7. On successful submission of incident display the auto-generated incident id
8. IncidentsListComponent
9. Create a component which is accessible to security heads
10. The component should display the pending incidents in a tabular formats
11. The security heads should be able to sort the incidents based on incident type and incident dates
12. Each incident should have view details button which should navigate to track incident.
13. TrackIncidentComponent
14. Design a new component for application users to track an incident filed by them
15. The component should display all the incident details along with the investigation details
16. Provide a navigation in the application navbar to access the component
17. If a security head is navigating to track incident component by clicking the view details button in IncidentList component than that particular incident details must be displayed
18. Along with incident details a security head should be provided with a form to file the investigation findings

**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 25-August-2022 by Khaleelullah Hussaini Syed | | | |
| V1.0.1 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| 2.3 | Khaleelullah Hussaini Syed | 11 October 2022 | Updated project scope with new modules |
| 3.0 | Updated use case diagram |
| 4.0 | New architecture diagram |
| 5.0 | Removed Micro services phase and replaced it with Deployment |
| 6.1.1 | Added new module on project management |
| 6.1.2 | Remove the api gateway |
| 6.1.3 | Remove the api gateway |
| 6.1.4 | Remove the api gateway |
| 10 | Added new section on project templates for .net backend, react front end and angular front end |