

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Background**

Ticketing is a common practice in almost all the companies around the world. Tickets are raised for registrations, reporting software or hardware issues, assigning tasks, monitoring sales or production statuses, etc. This system has been around for a long time and it needs to be updated every now and then, according to a company's requirements.

Despite this facile way to raising issues and monitoring progress, there are certain issues that some companies don't include in this system. Some of these issues are centered around an employee's personal struggles or convenience.

Issues like harassment, or a female employee requiring a cab service immediately after a long night of working at the office, or losing possessions that are important to the employee, aren't easily addressable with the current ticketing system, in several organizations. Employees can rather approach the security personnel of their organization for assistance through a simple web portal, that mimics a ticket system.

This can be achieved by making use of a combination of Backend and Frontend technologies like Java, Spring and Hibernate frameworks and Angular 4 to create a useful and user friendly application that both the employees and the security personnel can utilize.

A mobile application for the same purpose can improve the feasibility of the application and improve the comfort and environment of the employees to a great extent.

### **1.2 Problem Statement**

To create a web portal application for the employees and security personnel of my organization to raise certain categories of issues and update issue statuses respectively.

### **1.3 Specific Objective**

The primary aim of the project is to help employees raise issues that are sometimes overlooked despite the seriousness, and for those issues to be resolved as soon as possible. This application will also allow the security personnel to access and resolve issues with ease. Issues such as physical or sexual harassment, late entry permission requests, cab booking requests for female employees and lost and found services are some of the issues that this web portal allows an employee to raise requests for. These are simple yet serious requests that not all ticketing systems in certain organizations, address. An employee of the organization would be able to lodge these complaints or requests and also view the status of their ticket. The portal also allows the security personnel to access the system to view, resolve and update the tickets of the employees. The application was created in the company environment by following a strict agile methodology, and each process was done in an iterative manner. A mobile application prototype was also created to display a working model of the helpline application, accessible via the mobile phones of the employees.

### **1.4 Limitations**

This portal has four different issues that can be raised and also resolved by the employees and security personnel, respectively. There are a lot more issues that could be addressed, but only four are included in this application.

The mobile application is only available for the employees and not for the security personnel. It is also just a prototype and not a legitimate mobile application.

The description fields of the issue categories only take textual information from the employees and they cannot take image or video evidence as a complaint description.

## **CHAPTER 2**

### **REQUIREMENT SPECIFICATION**

The proposed work requires the following system requirements:

- ◆ System with 4GB RAM minimum

These are the required installations:

- ◆ Java installation with version 8 minimum
- ◆ Spring
- ◆ Hibernate
- ◆ NodeJS
- ◆ Angular 4
- ◆ npm Node Modules
- ◆ Postman
- ◆ HTML5
- ◆ CSS
- ◆ Bootstrap

## CHAPTER 3

### ARCHITECTURE

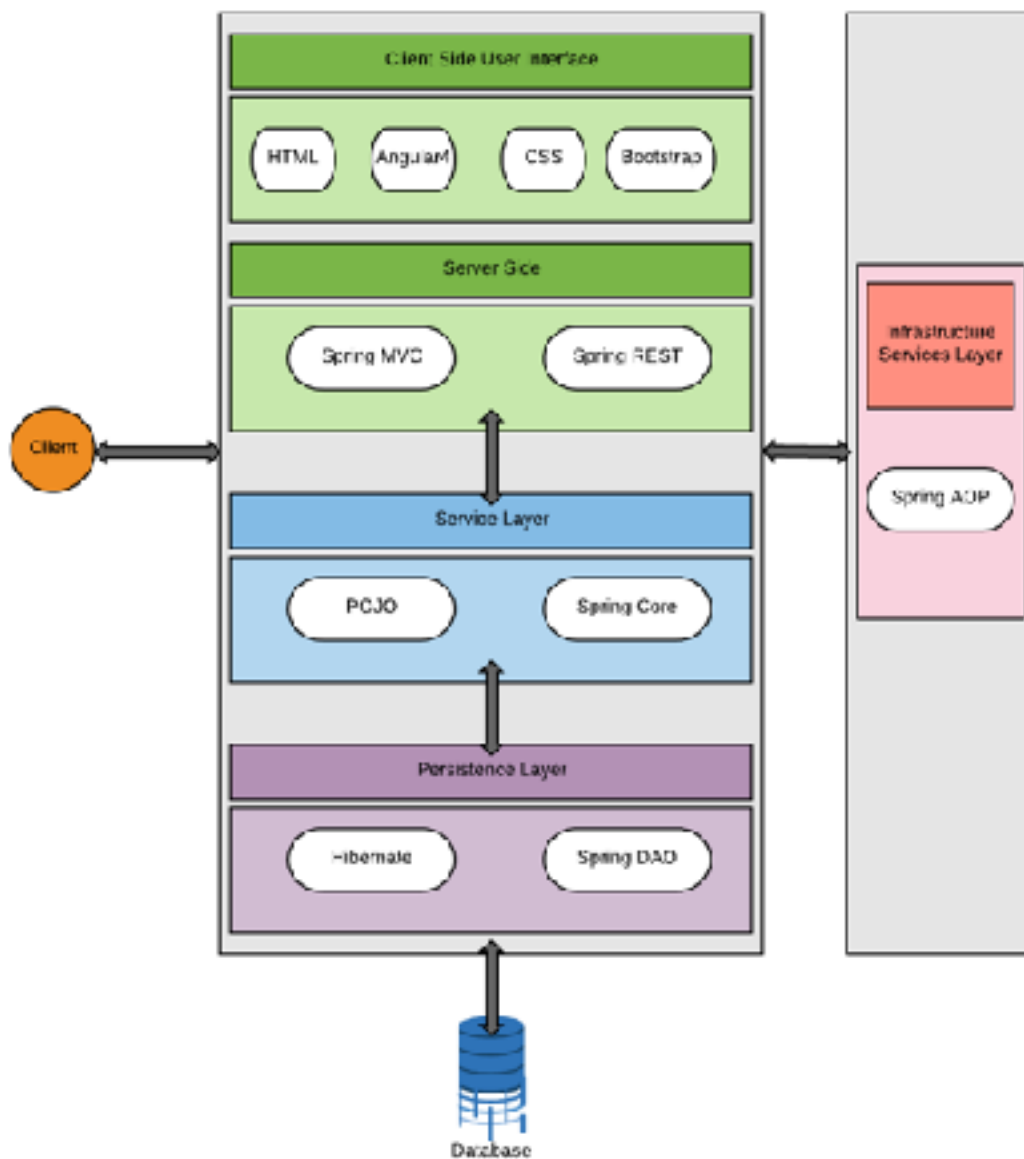


Fig 3.1 Spring Framework Application Architecture

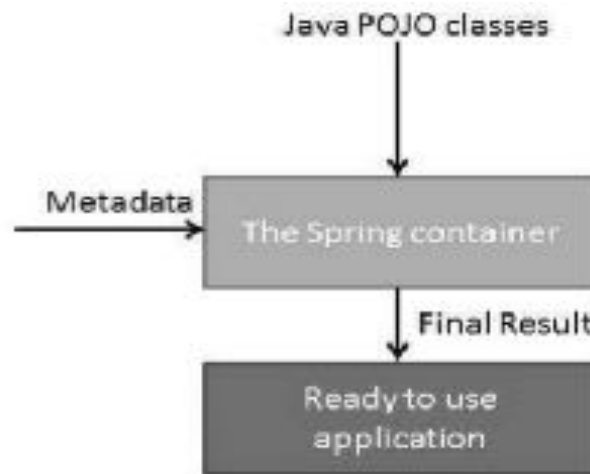
The architecture of this enterprise application is as follows:

- The client or in this case, the employees or the security personnel, will enter or request for certain information by making use of the web application's user interface. This interface is created using Angular4, HTML, CSS and Bootstrap.
- When the requests ping the server with a particular Spring REST (Representational State Transfer) API call, the corresponding request is performed by taking the JSON (JavaScript Object Notation) information from the URL requested by the user, and converting it into an object of a model Java class. The API classes make use of several important Spring annotations.
- With this information, each API method in turn calls a service class method. The service class contains the methods that perform the required business logic of the application. The service class makes use of several Spring annotations.
- These service methods once again call DAO (Data Access Object) methods. These methods work with Entity class objects which are Java classes that interact directly with the database tables. DAO classes mostly make use of Hibernate annotations. The DAO class is also the home of the Criteria Queries, for accessing information from a database table in a more precise and specific manner. Hence, the methods of the DAO class are solely responsible for interacting with the database.
- All of these layers work towards Aspect Oriented Programming which dwells upon cross-cutting concerns, to achieve modularity. All the layers may contain services that need to be provided uniformly across them. These cross-cutting concerns are added at runtime and are kept separately from the rest of the business logic.

### **3.1 Detailed Backend Programming Description**

This project is based on making a Java and Angular based web application by making use of RESTful web services and also by making use of the Spring and Hibernate frameworks. Spring is a lightweight application development framework for making Java enterprise applications.

Spring revolves around the concept of DI (Dependency Injection) which is a flavor of IoC (Inversion of Control) of the objects which are created, instantiated, managed and deleted by the IoC container. These objects are called Beans. The concept of IoC and essentially DI, is that instead of an object having to create its own dependencies every time, the dependencies are rather injected into it instead, which forms an inversion to the normal protocol. The IoC container is where the dependencies are configured along with the required configuration meta-data. Whenever each Java class is scanned for beans by making use of the `@ComponentScan` annotation provided by Spring, the beans created for each class is injected with the necessary dependencies from the IoC container.



*Fig 3.2 Spring IoC Container*

In order to help create the Persistence layer of the application, an ORM (Object Relation Mapping) framework is used. This framework consists of a group of predefined classes, interfaces, annotations and methods by which Java classes can be mapped to database tables. In this application, Spring is paired with the Hibernate ORM framework.

### 3.1.1 Model Classes and Entity Classes

Model classes contain functions like getters and setters, instance variables and class objects that would be used to carry information that is entered by the user which is used by the entity class to make updates to the database tables. It is considered as a container of POJO (Plain Old Java Object) where the user's data is placed rather than placing the business logic in it.

In this application, the model classes will pass the information around and persist that information only after the service class has performed some business logic on it, which will later be stored in the entity class. Hence the Model classes are used mainly in regards of the user interface.

Entity classes on the other hand, are programmed like a Model class but carry the annotation `@Entity` before the class definition. This `@Entity` is a JPA (Java Persistence API) entity. The Entity classes are used to create a connection between the databases of the application and Java classes. By making use of an Entity class, one can perform transactions like read, write and update operations on the database by making use of Java classes.

According to the concepts of an ORM framework, each Entity class is represented by a table in the database. Subsequently, each object created in these classes represent a row in the table and all of the instance variables declared, represent the attributes of the respective table.

The `@Table` annotation is used in case the name of the Entity class and the database table don't match. `@Column` is used for the same purpose in case of the table attributes not matching with the instance variable names. `@Id` is an annotation used above the attribute which is to be considered as the primary key. Other annotations can be used as well like `@GeneratedValue`, to provide a means for automatic iteration of the primary key in the database table, after insertions.

Another important set of annotations that are declared in the Entity classes are the Cardinality annotations. These are `@OneToOne`, `@OneToMany`, `@ManyToMany` and `@ManyToOne`. In `@OneToOne` and `@ManyToOne`, types of table connections, the foreign key values are unique, with that particular Entity class being the target class. The foreign keys are added with a `@JoinColumn` annotation. `@OneToMany` requires a third table with unique attributes whereas `@ManyToMany` also requires a third table but with no unique attributes.

```

package com.vault.model;

public class Complaint {
    private int complaintId;
    private String description;
    private String status;
    private String message;

    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public int getComplaintId() {
        return complaintId;
    }
    public void setComplaintId(int complaintId) {
        this.complaintId = complaintId;
    }
}

```

*Fig 3.3 Model Class Code Snippet*

```

package com.vault.entity;

import javax.persistence.Column;

@Entity
@Table(name="COMPLAINT")
public class ComplaintEntity {

    @Id
    @Column(name="complaint id")
    private int complaintId;
    private String description;
    private String status;

    public int getComplaintId() {
        return complaintId;
    }
    public void setComplaintId(int complaintId) {
        this.complaintId = complaintId;
    }
    public String getDescription() {

```



*Fig 3.4 Entity Class Code Snippet*

### **3.1.2 Data Access Object Classes**

DAO classes are used to help perform CRUD (Create, Read, Update and Delete) operations on the objects of the Model and Entity classes, so as to interact with the databases. By making use of the getter and setter methods present in the Model and Entity classes, the values in the database tables can be operated on.

The `@Repository` annotation is used before the class definition. This annotation is similar to the `@Component` annotation which is responsible for enabling Spring Bean creation of that particular Java class. One added advantage it has for DAO classes, is that `@Repository` will translate certain non-user friendly database exceptions, into a more readable format. When `@ComponentScan` happens in the configuration files, `@Repository` will notify it that this particular class has bean objects and that it must be sent to the IoC (Inversion of Control) Container so that the bean object can be injected with the necessary dependencies.

In the configuration file, three beans are pre-written so that they can be used across the application via `@Autowired`, which injects collaborating beans into the dependent bean in a class. One is the `DataSource` bean which will hold constant information about the database, like the admin username and password. The second bean is the `SessionFactory` bean. This bean is used for creating a session object when we interact with the database. For every interaction, it is not possible to create a session object every time. Hence a bean is created and the `SessionFactory` bean is autowired into the DAO classes and the resulting session object can be used in every method that interacts with the database. The third bean created in the configuration file, is the `TransactionManager` bean which is used to manage transactions and connections with a `@EnableTransactionManagement` annotation mentioned on the top of the configuration class file as well.

The notable CRUD functionalities that happen within the DAO class include `get()`, `save()`, `persist()` and `delete()`. The `get()` function fetches values from the database table based on that particular table's primary key, converts it into an object of the Entity class, and finally returns that object. But this entity data object cannot be used across the different layers of the Spring

Framework, hence these objects are yet again converted into their equivalent objects of the Model class and are then returned.

The `save()` and `persist()` functions are used with the session object of that method to save the information updated to the database. The `save()` function will return the primary key row which was inserted or updated. The `persist()` function on the other hand, has a void return type, and doesn't return any extra information other than saving the data.

The `delete()` deletes a particular row and returns an `IllegalArgumentException` if a null value is given to it, as it cannot delete a row that does not exist. The exception handling for the same is done in the service layer.

The last functionality of the DAO classes is the Criteria Queries. These queries can be defined in the necessary methods of the DAO class to manipulate objects and make necessary data available from the database by making use of Hibernate's `createCriteria` methods.

```
package com.vault.dao;

import java.util.ArrayList;

@Repository(value="complaintDao")
public class ComplaintDAOImpl implements ComplaintDAO{

    @Autowired
    SessionFactory sessionFactory;

    public Complaint addNewComplaint(Complaint complaint,Integer employeeid){
        Session session = sessionFactory.getCurrentSession();
        EmployeeEntity empEntity=(EmployeeEntity) session.get(EmployeeEntity.

        List<ComplaintEntity> complaints=empEntity.getComplaints();
        ComplaintEntity complaintEntity=new ComplaintEntity();
        complaintEntity.setComplaintId(complaint.getComplaintId());
        complaintEntity.setDescription(complaint.getDescription());
        complaintEntity.setStatus(complaint.getStatus());

        complaints.add(complaintEntity);
        empEntity.setComplaints(complaints);
        session.save(complaintEntity);
        session.save(empEntity);

        return complaint;
    }
}
```

*Fig 3.5 DAO Class Code Snippet*

### 3.1.3 Service Classes

The Service classes contain their respective business logic for the application (i.e) all the logic regarding the main functionalities of the application are written in the Service classes. It makes use of the DAO methods by autowiring the respective DAO bean into its class.

The Service classes exist so that the business logic remains separate from the rest of the database and UI logic, so that if there are any changes to them, the overall business logic will remain unchanged.

The `@Service` annotation is used in the Service classes for the same reason `@Repository` and `@Component` annotations are used. But this annotation specifies the intent better than ambiguously mentioning the `@Component` annotation in a service layer class.

The `@Transactional` annotation by Spring, is used before each Service class method to denote whether a CRUD operation is going to be a database affecting transaction or not and also to deal with transaction propagation. It gives the method the option to join the ongoing transaction of another service method.

```
package com.vault.service;

import java.util.List;

@Service(value="complaintService")
@Transactional(readonly = true)
public class ComplaintServiceImpl implements ComplaintService{

    @Autowired
    ComplaintDAO complaintDAO;

    @Transactional(readonly = false, propagation = Propagation.REQUIRES_NEW)
    public Complaint addNewComplaint(Complaint complaint,Integer employeeId) {
        Complaint complaintAdded=null;
        if(complaintDAO.getComplaintDetails(complaint.getComplaintId())==null)
            complaintAdded=complaintDAO.addNewComplaint(complaint, employeeId)
        }
        else
            throw new Exception("ComplaintService.COMPLAINT_ALREADY_EXISTS");
        return complaintAdded;
    }
}
```

*Fig 3.6 Service Class Code Snippet*

### 3.1.4 API Classes

API (Application Programming Interface) classes mark the beginning of RESTful Web Services, for this application. They serve as the connection between the backend and frontend of the application.

REST API's use a flexible URI which the user will ping to perform an operation. This operation will yield a result in a JSON format. This information is then converted into a Model class object and paired with a HTTP response, and returned. JSON serves as a datatype for transmitting data between HTML and Java strings.

REST makes use of several HTTP client methods, that help with consuming the CRUD methods made by the client. They are GET, POST, PUT and DELETE.

Spring REST is an API specification that provides annotations for making these web services. `@Cross Origin`, `@RestController` and `@RequestMapping` are the three most important annotations to be mentioned above the class definition of the API class. This is because these annotations indicate that the Java class below it forms an API class.

`@CrossOrigin` is used to expose the API and to help connect both the frontend and backend servers as they both use different port numbers. `@RequestMapping` is used above the class definition and also above individual methods, to specify the name of the API when it is exposed and also the names of the individual functions when they are exposed as well. The HTTP method they use is also mentioned using this annotation. `@RestController` annotation will map the HTTP request body to a model class object. The JSON format it arrives as is deserialized into a Java object. The object returned as a result of these API methods is a `ResponseEntity` object which will contain the response body object and also a HTTP status code. This will finally display our data in a JSON format. For the API classes, the respective Service class beans are autowired.

```

package com.vault.api;

import java.util.ArrayList;

@RestController
@CrossOrigin
@RequestMapping(value="complaintApi")
public class ComplaintAPI {

    private ComplaintService service;

    @RequestMapping(method=RequestMethod.POST, value="addComplaint")
    public ResponseEntity<Complaint> addNewComplaint(@RequestBody Complaint complaint){

        Environment environment =ContextFactory.getContext().getEnvironment();
        service=(ComplaintService) ContextFactory.getContext().getBean(ComplaintServiceImpl.class);
        ResponseEntity<Complaint> responseEntity;
        Complaint complaints=new Complaint();

        try{
            Complaint complaints1=service.addNewComplaint(complaint, complaint.getComplaintId());
            complaints.setComplaintId(complaints1.getComplaintId());
            String successMessage=this.getAddSuccessMessage(complaints);
            complaints.setMessage(successMessage);
            responseEntity = new ResponseEntity<Complaint>(complaints,HttpStatus.OK);
        }
        catch(Exception exception){
            String errorMessage = environment.getProperty(exception.getMessage());
            Complaint c = new Complaint();
            c.setMessage(errorMessage);
            responseEntity = new ResponseEntity<Complaint>(c,HttpStatus.BAD_REQUEST);
        }
        return responseEntity;
    }
}

```

*Fig 3.7 API Class Code Snippet*

### 3.1.5 Database Schema

The database of this application was created with Oracle. A total of 7 tables were created, comprising of one-to-many and one-to-one relationships among them. The schema is as follows:

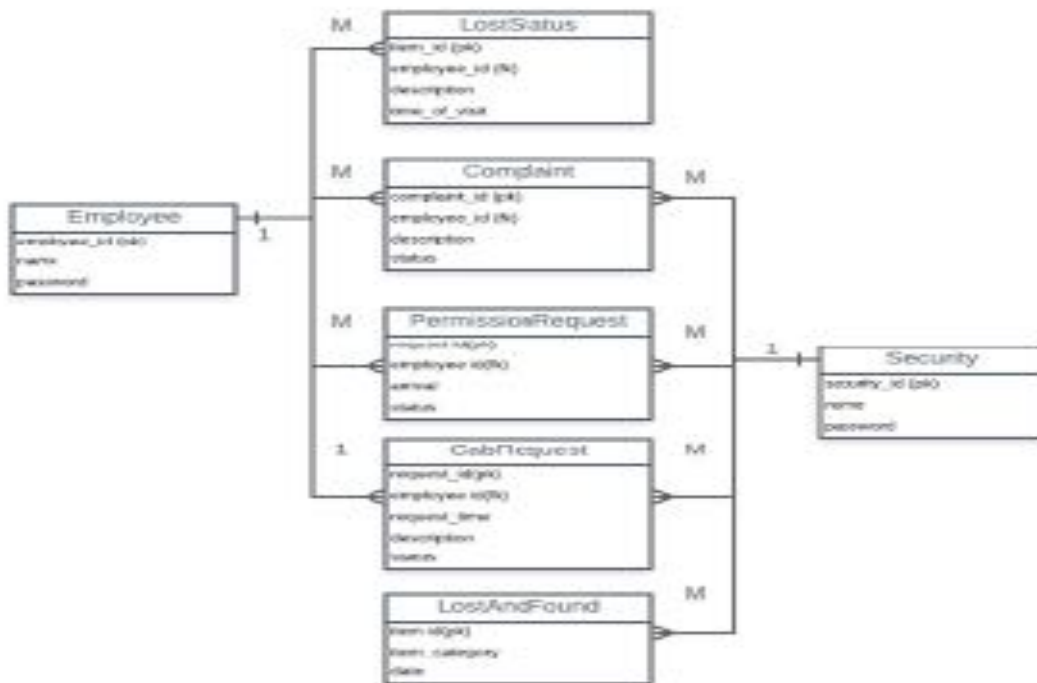


Fig 3.8 Database Schema

```

CREATE TABLE EMPLOYEE(
    employee_id NUMBER(6),
    name VARCHAR(30) NOT NULL,
    password VARCHAR(15) NOT NULL,
    phone_no NUMBER(10) NOT NULL UNIQUE,
    personal_email VARCHAR(30) UNIQUE,
    info_email VARCHAR(30) NOT NULL UNIQUE,
    address VARCHAR(100) NOT NULL,
    ecc_address VARCHAR(100),
    constraint Employee_employee_id_PK primary key (employee_id)
);

INSERT INTO EMPLOYEE (employee_id,name,password,phone_no,personal_email,info_email,address,ecc_address)
VALUES ('100001','Hema','123@INFO','1234567890','hemagmail.com','hemaginfo@sys.com','abc road','ECG 99/100');
INSERT INTO EMPLOYEE (employee_id,name,password,phone_no,personal_email,info_email,address,ecc_address)
VALUES ('100002','VISHWA','1234@INFO','2345678901','vishwa@gmail.com','vishwa@infosys.com','xyz street','ECG 90/400');
INSERT INTO EMPLOYEE (employee_id,name,password,phone_no,personal_email,info_email,address,ecc_address)
VALUES ('100003','MADHURI','12345@INFO','3456789012','mads@gmail.com','mads@infosys.com','lmnop avenue','ECG 95/205');

CREATE TABLE COMPLAINT(
    employee_id NUMBER(6) NOT NULL REFERENCES Employee(employee_id),
    complaint_id NUMBER(5) NOT NULL,
    description VARCHAR(100) NOT NULL,
    status VARCHAR(10) NOT NULL,
    constraint complaint_PK primary key (complaint_id)
);

INSERT INTO COMPLAINT VALUES ('100001','123', 'HARRASSMENT', 'RESOLVED');
INSERT INTO COMPLAINT VALUES ('100002','124', 'THEFT', 'UNRESOLVED');
INSERT INTO COMPLAINT VALUES ('100003','125', 'BULLYING', 'RESOLVED');

```

*Fig 3.9 Database Code Snippet*

## CHAPTER 4

### RESULTS AND DISCUSSIONS

#### User Stories

The first step to creating the application according to the agile development methodology followed at the organization, was creating a set of user stories. User stories are simple descriptions of each feature or module of the application, written in the perspective of the end user. The following user stories were implemented in the application:

### **US01: Employee: Registration**

<b>User Story ID</b>	<b>Priority</b>	<b>Story points</b>
US01	This is very critical to the application	3
As an employee, I want to sign up (register) for the application providing details like name, phone number, employee id, personal email id, Infosys email id, password, confirm password, address, ECC address (if applicable), so that I get the benefit of a registered employee.		
<b><u>Acceptance test criteria</u></b>		



- Validate the format of email id, name, phone number, password and address entered by the seller:

- **Infosys Email Id:** The email id should be of following format:

**<FisrstPart>@<SecondPart>.<ThirdPart>**

- The first part can contain alphabets, numbers, dots and underscore.
- The second part can be only alphabets of at least two characters.
- The third part should start with an alphabet and can contain alphabets and dot.

Example:- [Peter123@infosys.co.in](mailto:Peter123@infosys.co.in), here the first part is 'Peter123', the second part is 'infosys' and the third part is 'co.in'.

- **Personal Email Id:** The email id should be of following format:

**<FisrstPart>@<SecondPart>.<ThirdPart>**

- The first part can contain alphabets, numbers, dots and underscore.
- The second part can be only alphabets of at least two characters.
- The third part should start with an alphabet and can contain alphabets and dot.

Example:- [Peter123@gmail.co.in](mailto:Peter123@gmail.co.in), here the first part is 'Peter123', the second part is 'gmail' and the third part is 'co.in'.

- **Name:** It can only contain alphabets and spaces. It should not start or end with a space. It cannot contain only spaces.
- **Phone number:** It should contain only digits and length should be 10.
- **Password:** It should be 6 to 20 characters in length (both inclusive). It should contain at least one Upper case character, one lower case character, one digit and a special character.
- **Confirm Password:** It should match with the password (entered in the above field).

### US02: Employee: Login

User Story ID	Priority	Story points
US02	This is very critical to the application as no other service is accessible without successful and authentic login of the Employee.	2
As an employee, I want to login (using Employee ID and password), so that I can use the services of Vault which are available to all employees.		
<b><u>Acceptance test criteria</u></b>		
<ul style="list-style-type: none"><li>• Validate the format Employee ID and password entered by the Employee:<ul style="list-style-type: none"><li>○ <b>Employee ID:</b> The Employee ID should be exactly 6 characters in length. All the characters must be digits. Example:- 560682</li><li>○ <b>Password:</b> It should be 6 to 20 characters in length (both inclusive). It should contain at least one Upper case character, one lower case character, one digit and a special character.</li></ul></li><li>• On successful validation of above parameters, verify the credentials from database.</li><li>• On successful verification, redirect to Employee Home Page.</li><li>• Display proper messages in case of any error or exception on the login page.</li></ul>		

User Story ID	Priority	Story points
---------------	----------	--------------

### US03: Employee: Add Lost Item

User Story ID	Priority	Story points
US03	This helps the employee to register their lost items.	2
As a registered employee, I want to add my lost item details, so that I can claim it back.		
<b><u>Acceptance test criteria</u></b>		
<ul style="list-style-type: none"> <li>• The employee can add a description of the item they have lost under the Lost and Found link in the Help Desk.</li> <li>• After submitting it, they will be able to view a list of items found by the security personnel and claim them if it belongs to them.</li> <li>• If their item is not present in the list, they can check back after sometime to see if the list is updated with new items.</li> <li>• After submitting, their item will be added to the database.</li> <li>• After successful addition, a success message should be displayed.</li> <li>• Display proper messages in case of any error or exception.</li> </ul>		

#### **US04: Employee: Add permission request**

User Story ID	Priority	Story points
US04	This helps the employee to add permission request(if he/she is reporting after in time to the campus)	2
As a registered employee, I want to add a permission request, so that I can inform security personnel that I would be arriving late to the campus, and request permission to enter when I arrive.		
<b><u>Acceptance test criteria</u></b>		

- After successfully adding the permission request to the database, a success message should be displayed.
- Display proper messages in case of any error or exception.

### US05: Employee: Add Complaint

User Story ID	Priority	Story points
US05	This helps the employee to lodge a complaint.	2
As a registered employee, I want to add to a complaint against any kind of harassment.		
<b><u>Acceptance test criteria</u></b>		
<ul style="list-style-type: none"> <li>• The employee can add a description of the harassment they faced or witnessed under the Harassment link in the Help Desk.</li> <li>• After submitting it, the complaint will be added to the database and the status variable will be activated as UNRESOLVED. (Changed to RESOLVED after actions taken)</li> <li>• After successful addition, a success message should be displayed.</li> <li>• Display proper messages in case of any error or exception.</li> </ul>		

### US06: Employee: Add Cab Request

User Story ID	Priority	Story points
US06	This helps the employee to request a cab.	2
As a registered employee, I want to request for a cab after working hours.		

**Acceptance test criteria**

- The employee can request for a cab by including a description of their whereabouts and the time at which they want the cab to arrive.
- After submitting it, the request will be added to the database and the status is shown. (Changed to BOOKED or UNBOOKED after actions taken)
- After successful addition, a success message should be displayed.
- Display proper messages in case of any error or exception.

**US07: Employee: Logout**

User Story ID	Priority	S t o r y points
US07	This is very critical to the application	1
As a logged in employee, I want to logout from Vault application, so that I can prevent any unauthorized access to my account.		
<b><u>Acceptance test criteria</u></b>		
<ul style="list-style-type: none"><li>• After logging out the Employee should be redirected to Login page of the application</li></ul>		

**US08: Security: Registration**

User Story ID	Priority	Story points
US08	This is very critical to the application. This allows security personnel to access the application.	3

As security personnel, I want to sign up (register) for the application providing details like Infosys email id, name, phone number and password, so that I can access the information registered by employees.

**Acceptance test criteria**

- Validate the format of email id, name, phone number, password entered by the security personnel:
  - **Infosys Email Id:** The email id should be of following format:  
**<FisrstPart>@<SecondPart>.<ThirdPart>**
    - The first part can contain alphabets, numbers, dots and underscores.
    - The second part can be only alphabets of at least two characters.
    - The third part should start with an alphabet and can contain alphabets and dot.

Example:- [Peter123@infosys.co.in](mailto:Peter123@infosys.co.in), here the first part is 'Peter123', the second part is 'infosys' and the third part is 'co.in' .
  - **Name:** It can only contain alphabets and spaces. It should not start or end with a space. It cannot contain only spaces.
  - **Phone number:** It should contain only digits and length should be 10.
  - **Password:** It should be 6 to 20 characters in length (both inclusive). It should contain at least one Upper case character, one lower case character, one digit and one special character.
- On successful validation of above parameters, verify from database, the availability of the email id and phone number. Neither the email id nor the phone number should be used by other security personnel.
- Add the new security person to the database.
- After successfully adding the new security person, display the success message with the registered email id.
- Display proper messages in case of any error or exception.

### US09: Security: Login

User Story ID	Priority	Story points
US09	This is very critical to the application	4
As a security personnel, I should be able to login (using security id and password), so that I can access the services of the application and facilitate the employees.		
<u>Acceptance test criteria</u>		
<ul style="list-style-type: none"><li>• Validate the format of password entered by the customer:<ul style="list-style-type: none"><li>○ Password: It should be 6 to 20 characters in length (both inclusive). It should contain at least one Upper case character, one lower case character, one digit and a special character.</li><li>○ Security email: It should be 8 digits exact. All the characters must be digits.</li></ul></li><li>• On successful validation of password, verify the credentials from database.</li><li>• On successful verification, redirect to Security Personnel's Home Page.</li><li>• Display proper messages in case of any error or exception.</li></ul>		

### US10: Security: View Complaint

User Story ID	Priority	Story points
US10	This helps the security personnel to view the complaints added by the employees.	2
As a registered security personnel, I want to view the existing list of complaints to resolve them.		
<u>Acceptance test criteria</u>		



- The complaint details that are to be displayed are - Employee id, Employee name, Complaint description, Category, Contact number and Status.
- All the details must be displayed in tabular format along with and UPDATE STATUS button next to every complaint displayed.
- If the number of products exceeds 10, display the details in paginated format.
- Display proper messages in case of any error or exception.

### **US11: Security: Update Complaint Status**

User Story ID	Priority	Story points
US11	This helps the security personnel to update the status of an employee's complaint.	2

As a registered security personnel, I want to update the status of complaint.

#### **Acceptance test criteria**

- The status will be updated after the complaint is resolved.
- Clicking on Update Status button, the security should be able to select a status among [YET TO RESOLVE, RESOLVED] and update the status of that employee's complaint.
- When the security personnel update the status, the new details should be saved in the DB.
- In case of any error, appropriate error message should be displayed.

### US12: Security: Add Found Item

User Story ID	Priority	Story points
US12	This helps the security personnel to add an item, if found, to the database.	2
As a registered security personnel, I want to add a found item, so that the employees would be able to view it and claim it, if it belongs to them.		
<b><u>Acceptance test criteria</u></b>		
<ul style="list-style-type: none"><li>• The item found is added to the database by writing the category the item comes under in a description box, and then clicking on Add button.</li><li>• After successfully adding the item to the database, a success message should be displayed.</li><li>• It is displayed as a link the employees can click on, under the Lost and Found link in the Help Desk.</li><li>• Display proper messages in case of any error or exception.</li></ul>		

### US13: Security: Remove Found Item

User Story ID	Priority	Story points
US13	This is very critical to the application	3
As a registered security personnel, I should be able to remove the lost items that have been returned back to the rightful owner.		
<b><u>Acceptance test criteria</u></b>		

- Once the lost item has been returned to the owner, the security personnel can remove it from the database by clicking on remove button.
- When the button is clicked, the item must be deleted from the database, and a success message should be displayed.
- In case of any error, appropriate error message should be displayed.

#### **US14: Security: Update Permission Status**

User Story ID	Priority	Story points
US14	This is very critical to the application	3
As a registered security personnel, I should be able to update permission status of the employee.		
<b><u>Acceptance test criteria</u></b>		
<ul style="list-style-type: none"> <li>• The status will be updated after the employee reports to the campus.</li> <li>• Clicking on Update Status button, the security should be able to select a status among [ACCEPT, DENY] and update the status of that employee's request.</li> <li>• When the security personnel update the status, the new details should be saved in the DB.</li> <li>• In case of any error, appropriate error message should be displayed.</li> </ul>		

#### **US15: Security: Update Cab status**

User Story ID	Priority	Story points
US15	This is very critical to the application	3

As a registered security personnel, I should be able to update the status of the employee's cab requests.

**Acceptance test criteria**

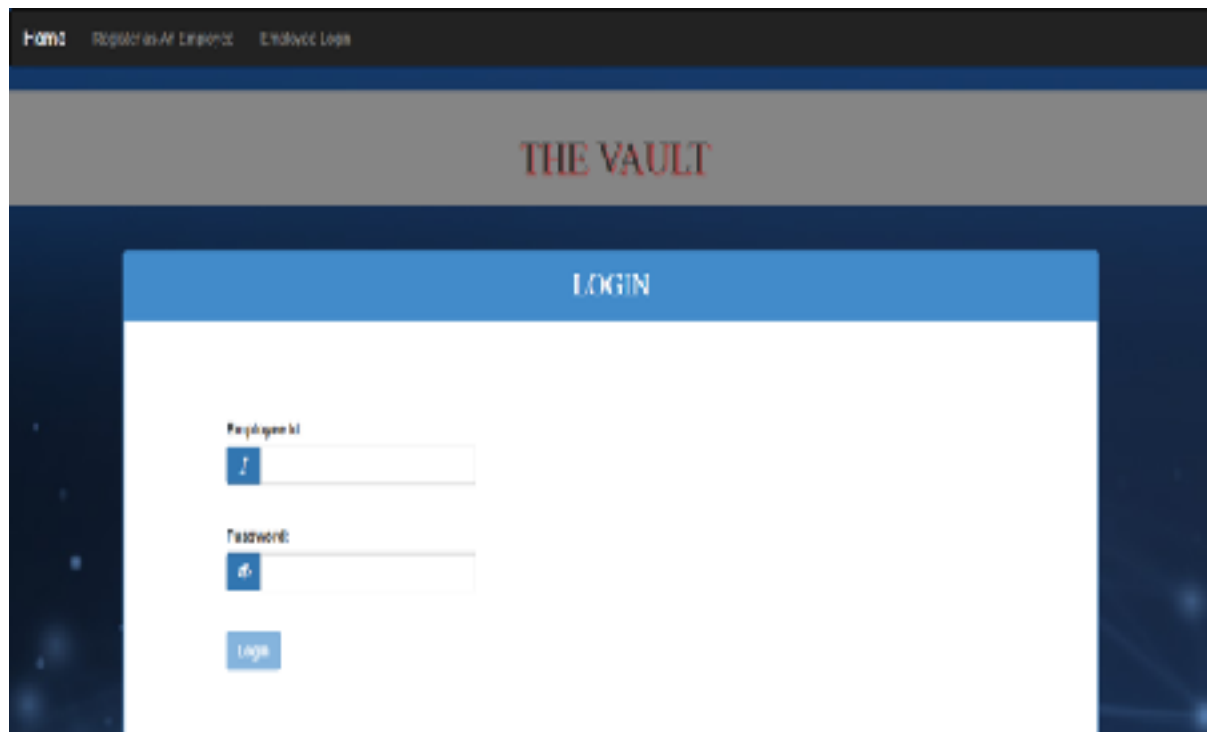
- The status will be updated after the cab has been booked and sent to the location successfully.
- Clicking on Update Status button, the security should be able to select a status among [BOOKED, UNBOOKED] and update the status of that employee's cab request.
- When the security personnel update the status, the new details should be saved in the DB.
- In case of any error, appropriate error message should be displayed.

**US16: Security: Logout**

User Story ID	Priority	S t o r y points
US16	This is very critical to the application	1
As a logged in security personnel, I want to logout from Vault application, so that I can prevent any unauthorized access to my account.		
<b><u>Acceptance test criteria</u></b>		
<ul style="list-style-type: none"><li>• After logging out the security personnel should be redirected to Login page of the application</li></ul>		

The above mentioned user stories were designed specifically for this application by conducting several scrum meetings among the team to discuss the different requests an end user may have. For this particular product, the employees and security personnel were taken into account as the end users and research was done on the kind of services they would require. The features that were finally deduced were selected based on the services that needed to be properly looked upon, that weren't already well covered under the organization's existing ticketing system.

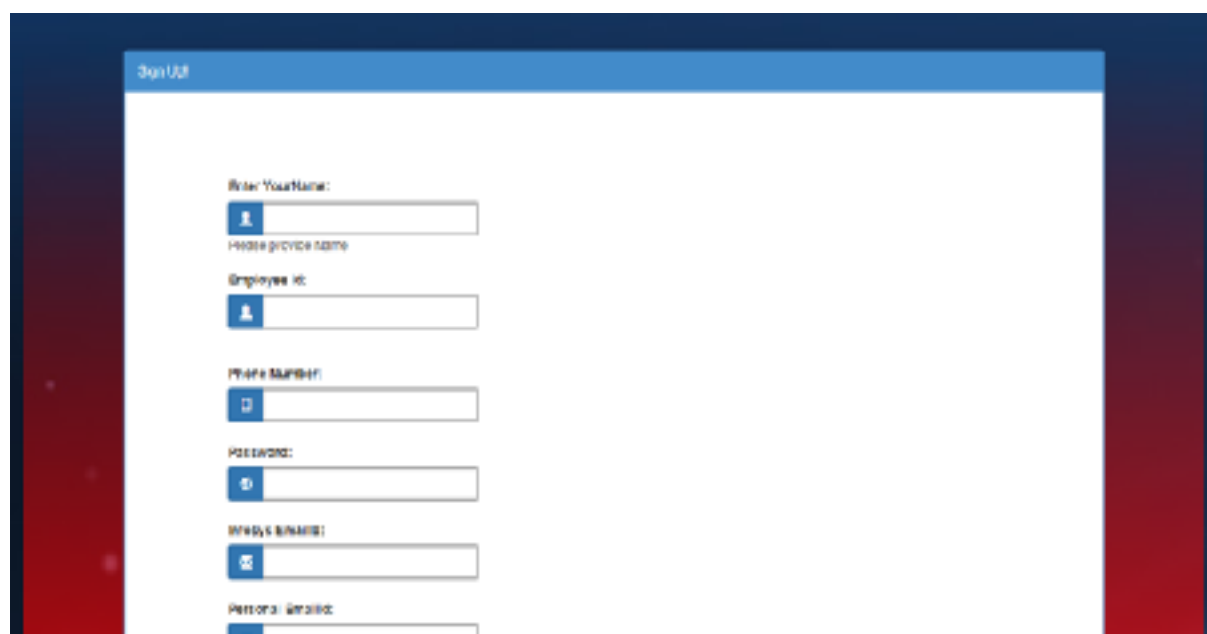
## EMPLOYEE LOGIN PAGE



The screenshot shows the Employee Login Page of a web application. At the top, a dark blue navigation bar contains the links "Home", "Registration", "Employee", and "Employee Login". Below this is a grey header with the text "THE VAULT" in red. The main content area has a blue header with the word "LOGIN" in white. The login form is centered and includes a "Employee Id" label, a text input field with a blue icon, a "Password:" label, another text input field with a blue icon, and a blue "Login" button.

*Fig 4.1 Employee Login Page*

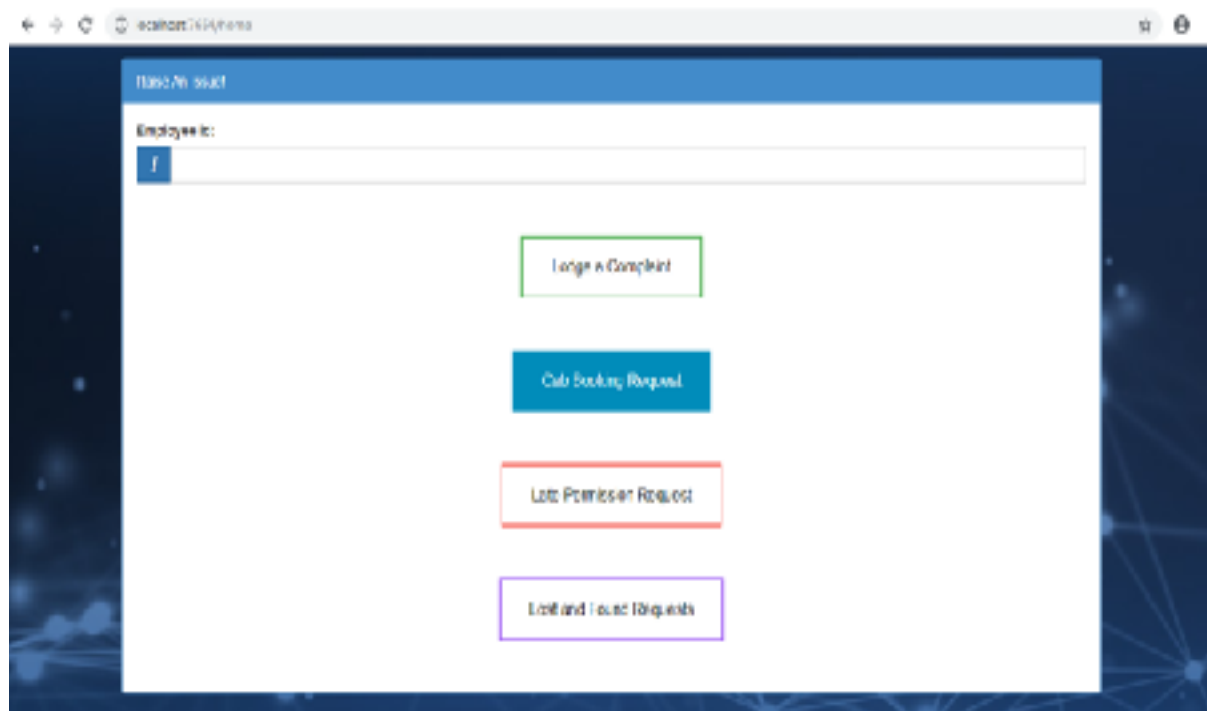
## EMPLOYEE REGISTRATION PAGE



The screenshot shows the Employee Registration Page. It features a blue header with the text "Sign Up!". The registration form is centered and includes the following fields: "Enter Your Name:" with a text input field and a blue icon, "Please provide name" (a label), "Employee Id:" with a text input field and a blue icon, "PHONE NUMBER:" with a text input field and a blue icon, "PASSWORD:" with a text input field and a blue icon, "WEBSITE URL:" with a text input field and a blue icon, and "Personal Email:" with a text input field and a blue icon.

*Fig 4.2 Employee Registration Page*

## EMPLOYEE HOME PAGE



*Fig 4.3 Employee Home Page*

## EMPLOYEE COMPLAINT PAGE

**LODGE A COMPLAINT!**

Employee ID:

Description:

Lodge Complaint

Show History of Complaints

Complaint ID	Employee ID	Description	Status
563001	12001	Outlying	RESOLVED
563002	12002	Anonymous Probs	UNRESOLVED
563003	12003	Witnessed Harassment	RESOLVED

Fig 4.4 Employee Complaint Page

## EMPLOYEE CAB REQUEST PAGE

**BOOK A CAB!**

Employee ID:

Request Cab at Time:

Pick Up Description:

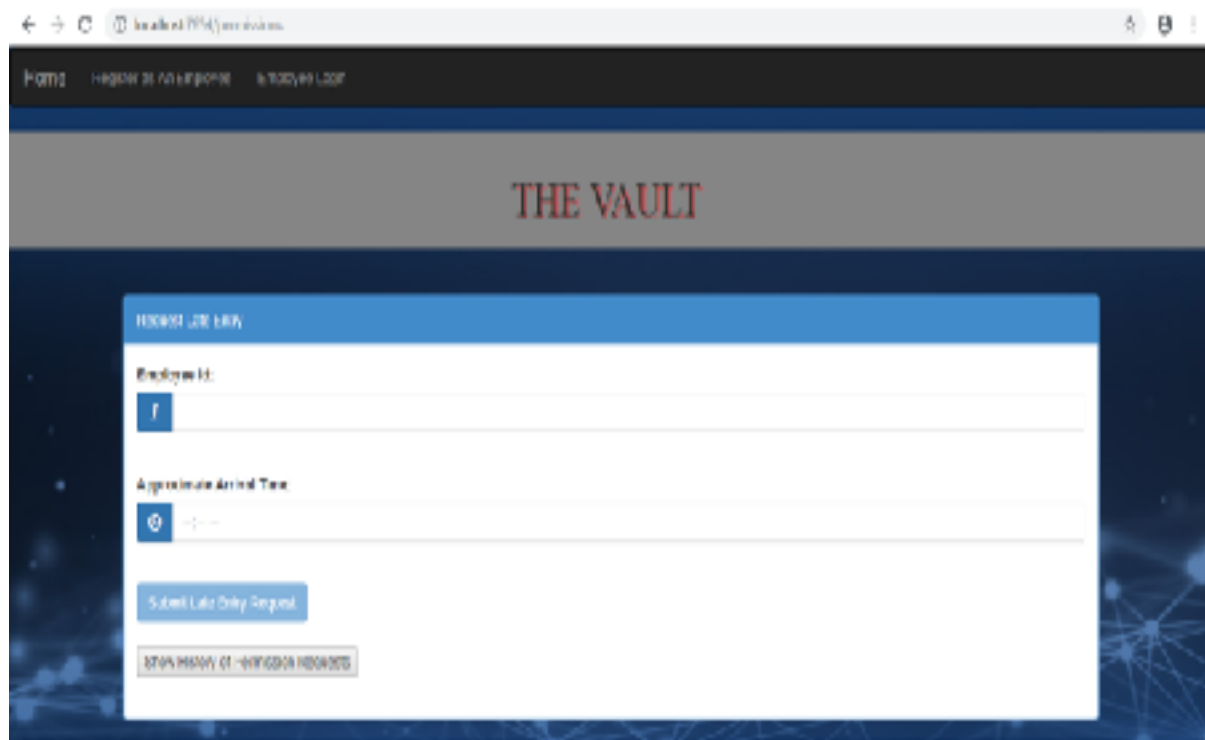
Book

Show History of Cab Requests

CabRequest ID	Employee ID	Pick Up	Status
100001	100001	INTOPS LOBBY A	BOOKED
100002	100002	Myself Field	UNBOOKED
100003	100003	ASRTC Bus Deck	BOOKED

*Fig 4.5 Employee Cab Request Page*

## EMPLOYEE LATE ARRIVAL PERMISSION PAGE



The screenshot shows a web browser window with the URL "localhost:7054/jsp/index.jsp". The page has a dark blue header with navigation links: "Home", "Request an Employee", and "Employee List". Below the header is a grey banner with the text "THE VAULT" in red. The main content area features a white form titled "REQUEST LATE PERMISSION" in blue. The form contains two input fields: "Employee ID:" with a blue icon and "Approximate Arrival Time:" with a blue clock icon. Below these fields is a blue button labeled "Submit Late Only Request" and a grey button labeled "RETURN MESSAGE OF PERMISSION REQUEST".

*Fig 4.6 Employee Late Arrival Permission Page*



## EMPLOYEE LOST AND FOUND PAGES

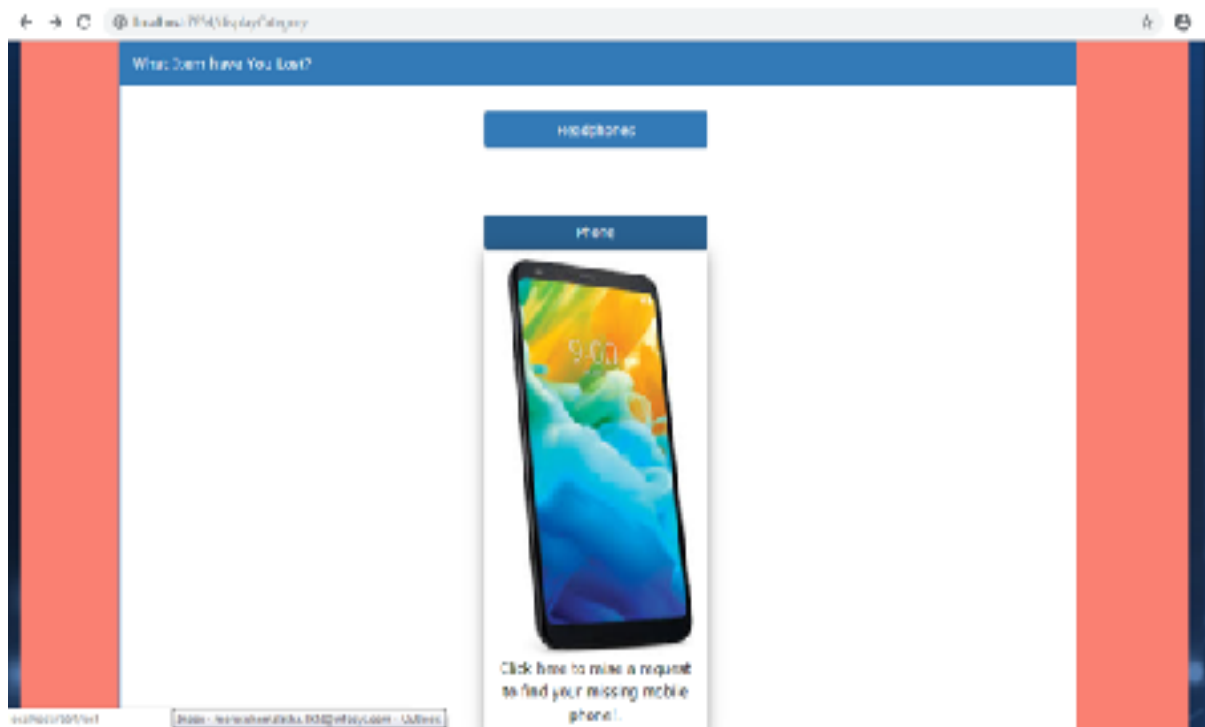


Fig 4.7 Employee Lost Item Category Page

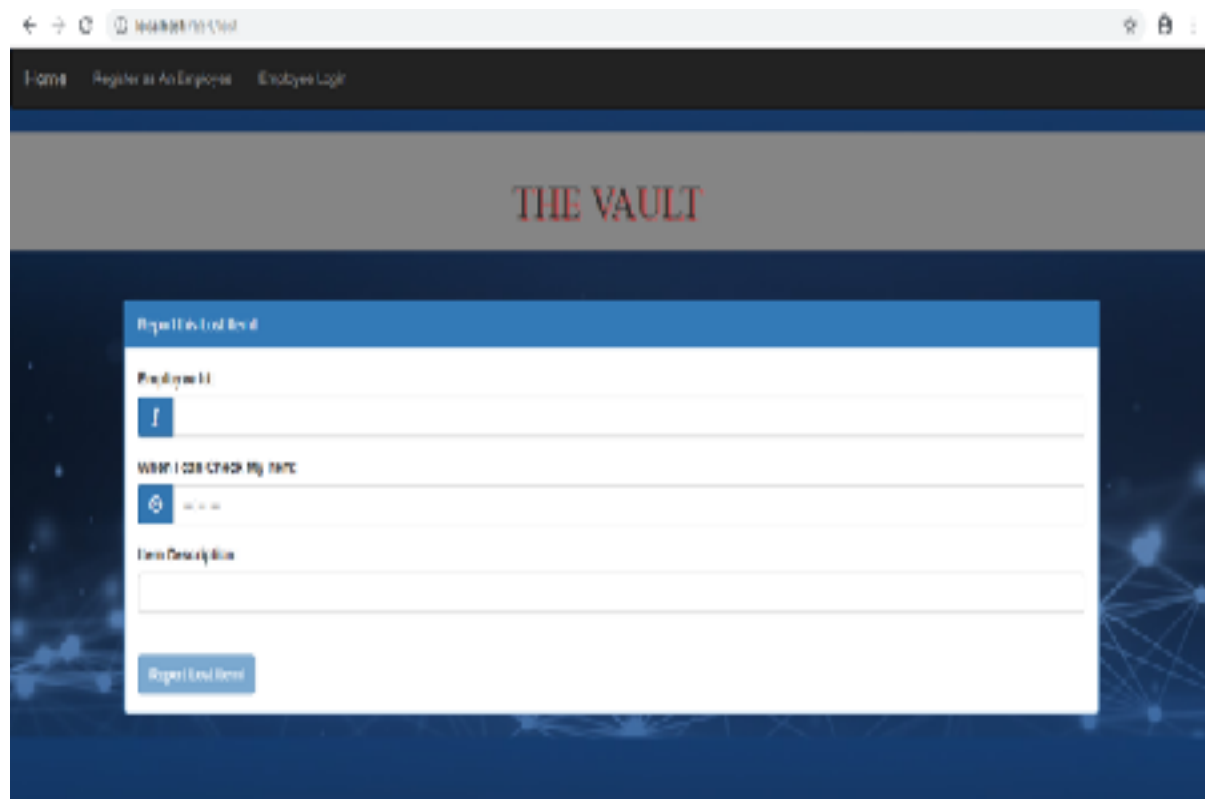


Fig 4.8 Employee Lost Item Reporting Page

## SECURITY CAB BOOKING PAGE

Home Register as An Security Personnel Security Login

THE VAULT

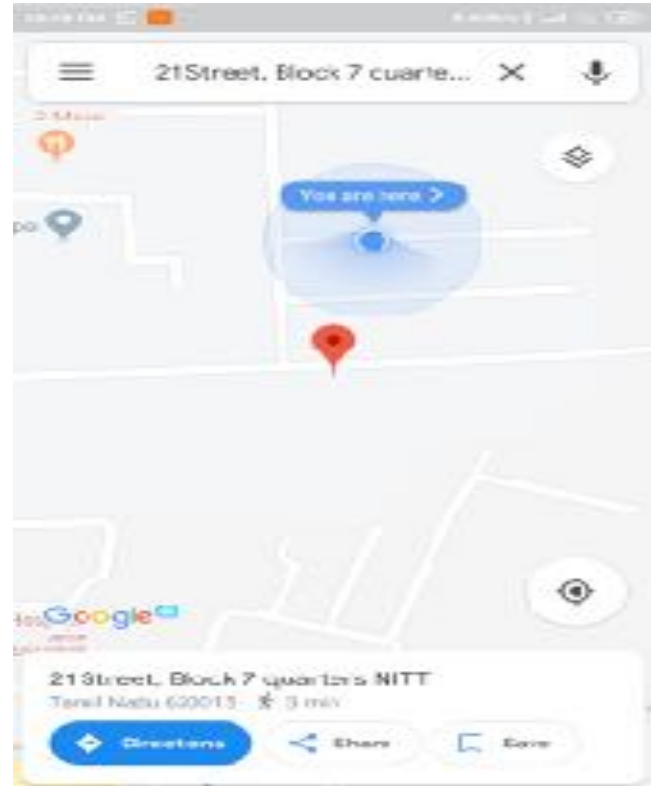
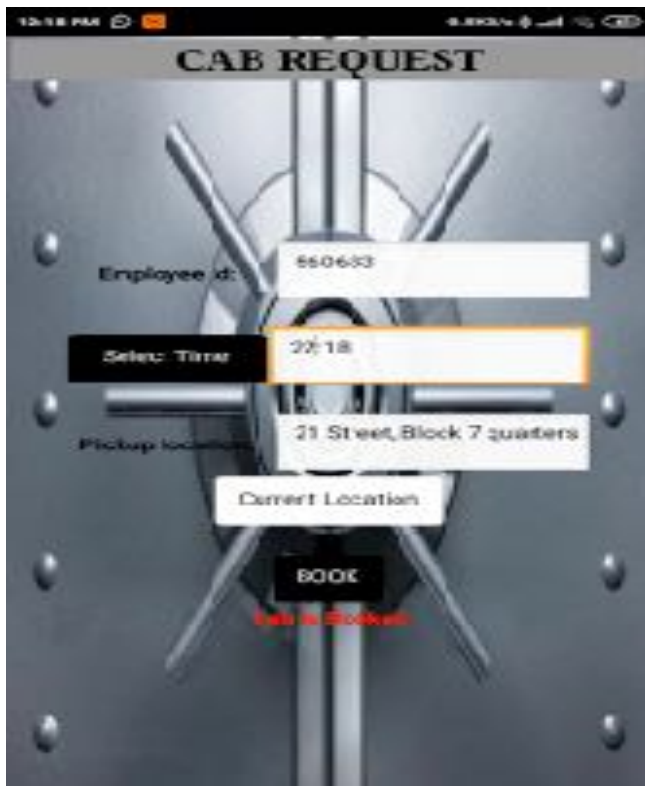
UPDATE CAB STATUS!

Book A Cab For Employee

CabRequestID	EmployeeID	PickUp	Status
56675	13671	Indraya Gate 4	<input type="button" value="Update Status as Booked"/> SUCCESS
56680	13670	Mysore Road	<input type="button" value="Update Status as Booked"/>
56686	13673	KSRTC Bus Depot	<input type="button" value="Update Status as Booked"/>

Fig 4.9 Security Cab Booking Page

A mobile application prototype was created using the MIT App Inventor software.



*Fig 4.10 Vault App Cab Request Screen with Google Map Location*



*Fig 4.11 Vault App Permission Request Screen*

## CHAPTER 5

## CONCLUSION

The web application was successfully completed with no incomplete sections under the Product Backlog Grooming sheet, as part of the Agile methodology procedures conducted by Infosys.

The backend and frontend of the application were properly integrated and tested with the database values.

The mobile application prototype of the helpline application was created to give insight to a future enhancement to the project. Further, the app can be enhanced by making use of Android Studio software and more functionalities.

The entire project was created following a strict agile development and the team involved successfully completed and submitted a two sprinted agile sheet to the organization, by properly

accounting for each change and discussion made for each of the user stories. Scrum meetings were held and proper assignment of Scrum masters were done.