Department of Information and Communication Technology
Faculty of Technology
University of Ruhuna

ICT 3273
Advanced Database Management Systems
Assessment 02

Group 2

Submitted to: Mr. P.H.P. Nuwan Laksiri

Submitted by:

TG/2019/491 - P.L.N.N. Dilshan
TG/2019/513 - E.A.P. Lakshitha
TG/2019/516 - R.M.U.H.P. Rathnayake
TG/2019/525 - J.L. Rasanjana

Brief introduction about the problem

The problem at hand revolves around the development of a Hostel Management System tailored for the students residing in the Faculty of Technology at the University of Ruhuna. This system aims to streamline the management of hostel properties and address issues related to property damages or losses.

Key Areas of the Problem Domain:

1. **Property Management:** The primary focus of this system is to manage the hostel's property efficiently. This includes a variety of items such as tables, cupboards, clothes racks, and more. Each property item is labeled with a QR code to aid in its identification.

2. **Damage Reporting:** The system allows students to report damages to hostel property. Students can submit inquiries along with image evidence of the damage, ensuring that the extent of damage is well-documented.

3. **Escalation Mechanism:** The system features an escalation mechanism where reported inquiries are initially reviewed by the sub warden. If not addressed within three days, the inquiry is elevated to the academic warden. In cases where the academic warden also fails to respond within seven days, the issue is escalated to the dean for immediate action.

4. **Communication:** The system facilitates communication between students, sub wardens, academic wardens, and the dean through a web application. This enables efficient interaction and coordination in addressing property-related concerns.

5. **QR Code Scanning:** The system's ability to scan QR codes on property items and capture images of damages simplifies the process of reporting and verifying property issues.

6. **Inquiry Sincerity:** Students can provide details about the sincerity of the damage in their inquiries, helping authorities prioritize and address issues based on severity.

7. **Reporting:** The system generates weekly reports for the academic warden and monthly reports for the dean, offering insights into the status of completed and pending property repairs.

8. **Responsive Design:** The system is designed to be responsive, ensuring it functions seamlessly on various devices, making it accessible and user-friendly.

9. **Technology Stack:** The system is built using Hibernate and incorporates a robust database to manage complaints, property records, user information, and other relevant data.

In summary, the Hostel Management System for the Faculty of Technology at the University of Ruhuna aims to enhance the efficiency of property management, improve communication between stakeholders, and ensure that damaged or missing property is promptly addressed, providing a convenient and effective solution for all users involved.

Brief introduction to the solution

The proposed solution is a Hostel Management System designed to streamline property management and address issues related to property damage or loss in the Faculty of Technology at the University of Ruhuna. This system leverages technology to improve efficiency, communication, and transparency in managing hostel properties. Here, we provide a detailed architecture of the solution to understand its underlying components and functionality.

Solution Architecture:

The Hostel Management System is structured using a modern, web-based architecture to ensure flexibility, scalability, and accessibility. It comprises several key components:

- Web Application Interface: The user interacts with the system through a web-based interface, which is designed to be responsive and accessible from various devices, including desktop computers, laptops, tablets, and smartphones.

- QR Code Scanning Module: This module enables the system to scan QR codes attached to hostel property items. It uses device cameras to capture images of the items and their damages when necessary.

- User Management: The system includes user management functionality to maintain student, sub warden, academic warden, and dean profiles, including their roles and permissions.

- Property Database: A dedicated database stores information about hostel properties, including their type, location, and status. QR codes are associated with specific items, allowing for quick identification.

- Inquiry Submission: Students can submit inquiries about damaged or missing property items via the web application. They can attach images and provide details about the extent of damage.

- Escalation Logic: The system includes an escalation mechanism that monitors the response time of sub wardens and academic wardens. If an inquiry is not addressed within the specified time frames, it automatically escalates to the next level of authority.

- Inquiry Review and Resolution: Sub wardens, academic wardens, and the dean can review submitted inquiries, view images of damages, and take immediate actions to address the issues. They can mark inquiries as resolved, in progress, or pending further action.

- Reporting: The system generates reports to provide insights into the status of property repairs and inquiries. Weekly reports are sent to academic wardens, while monthly reports are sent to the dean.

- Hibernate Framework: The solution is built on the Hibernate framework, which provides a robust and reliable data persistence layer. It ensures that data is securely stored and accessible.

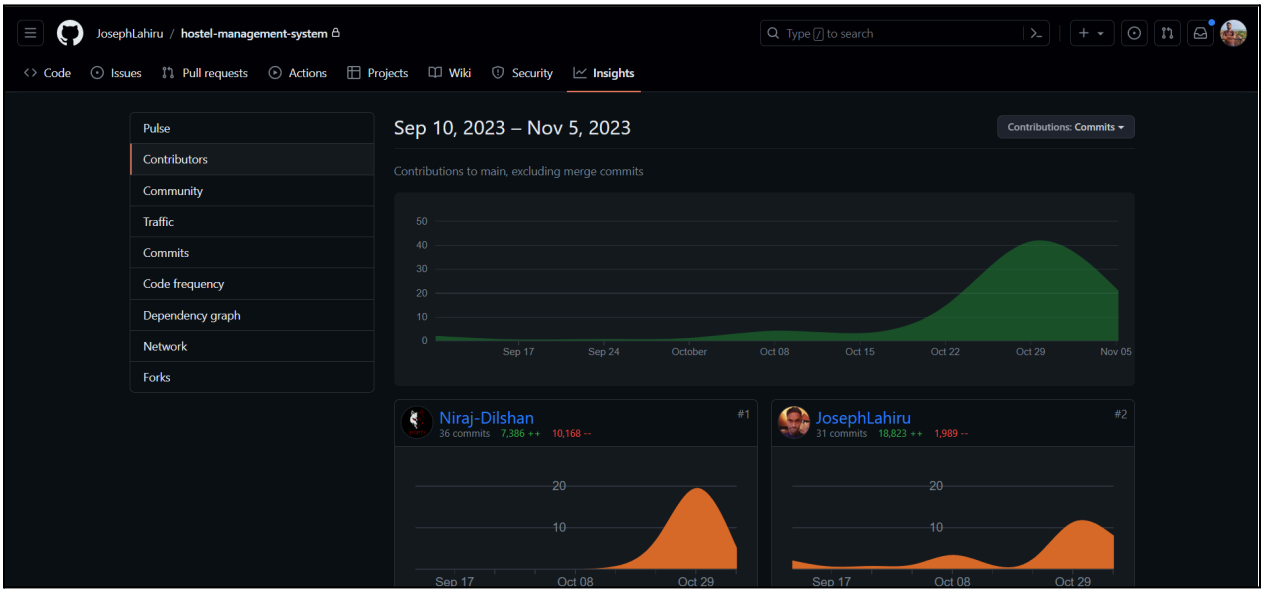
- Image Storage: Images associated with inquiries are stored securely in the system, allowing for easy reference and documentation.

Evidence for usage of versioning control mechanism

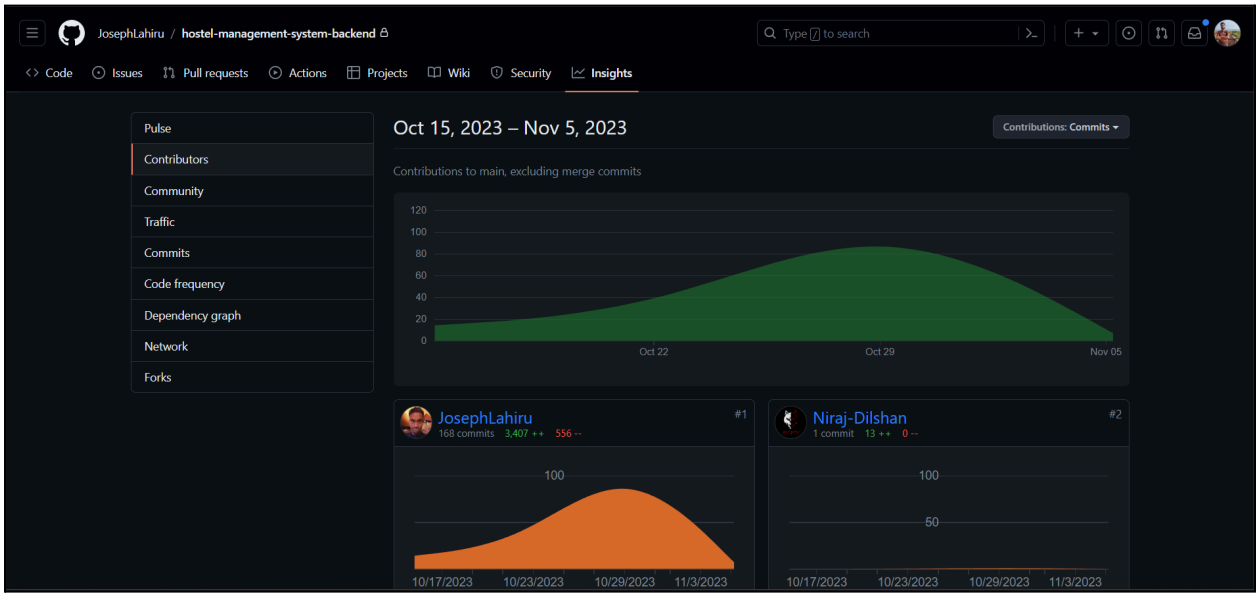
GIT repository link: https://github.com/JosephLahiru/hostel-management-system

Creation date:

Frontend:

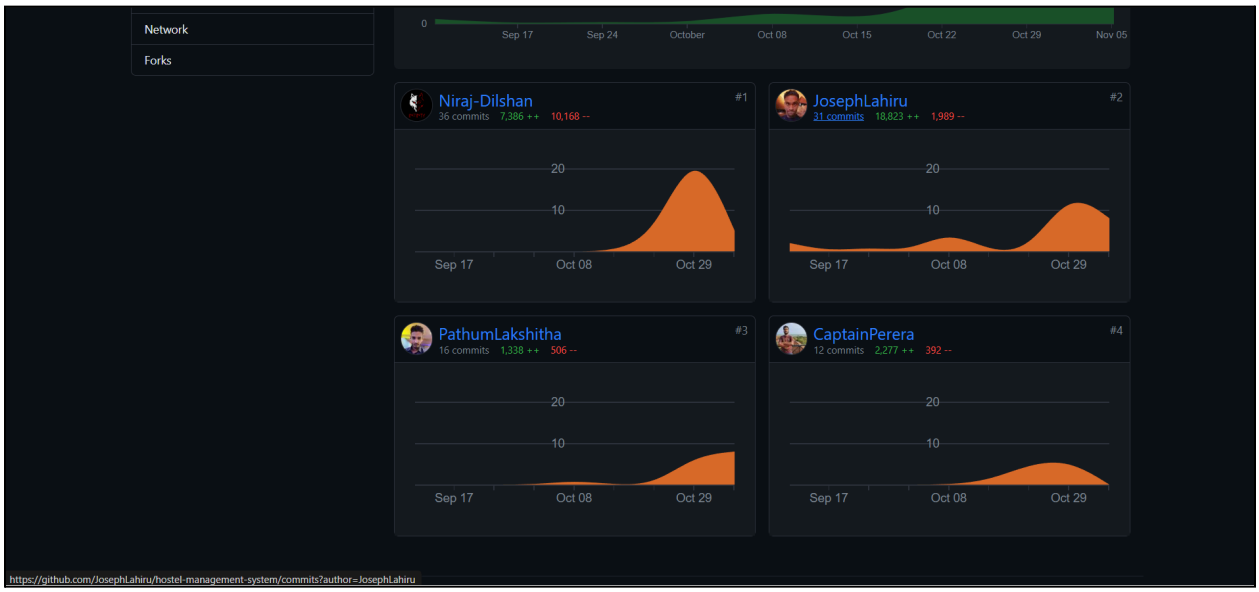


Backend:



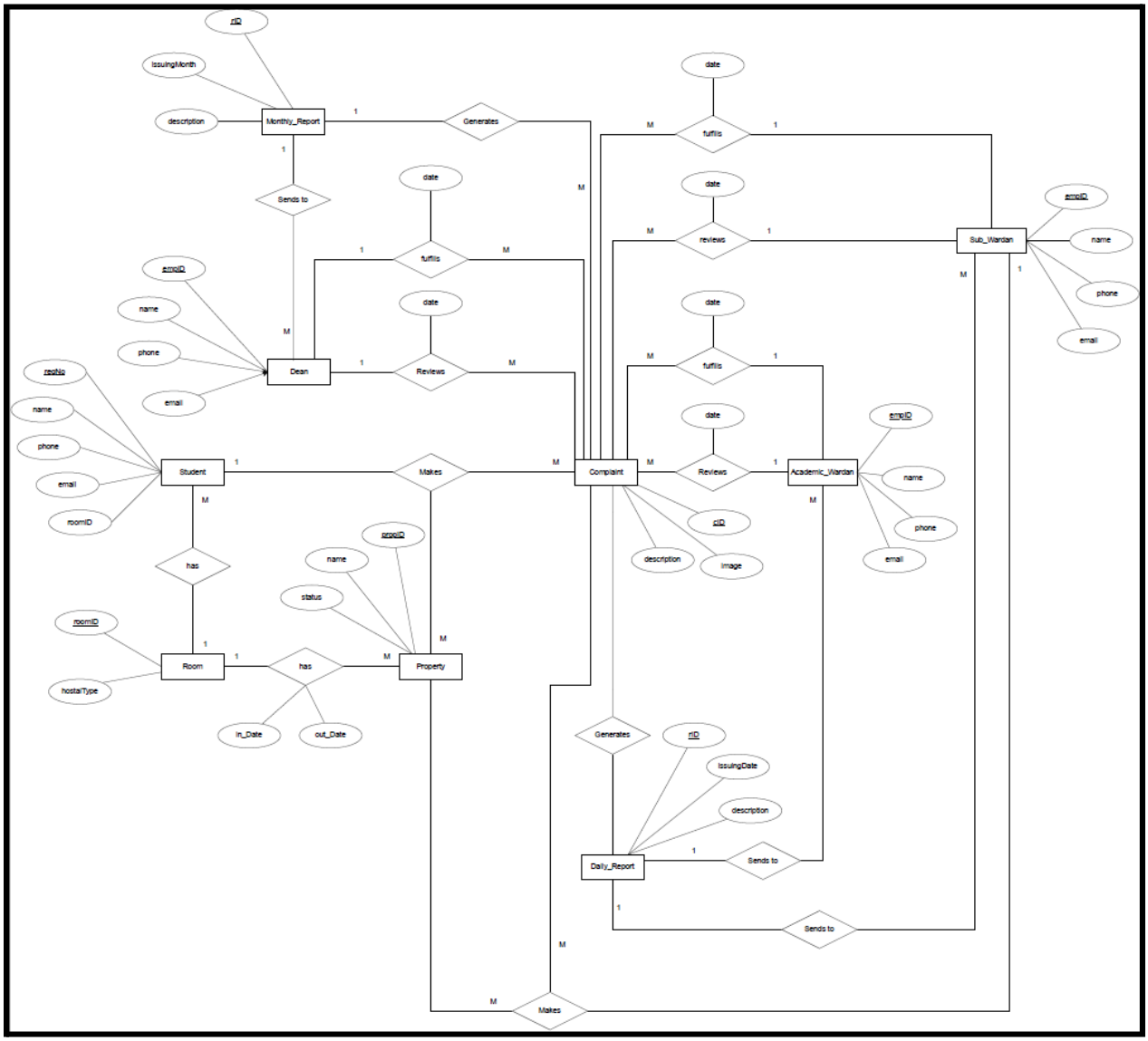
Contributors and individual contribution:

Frontend:



Backend:

Proposed ER/EER diagram



Proposed Relational mapping diagram

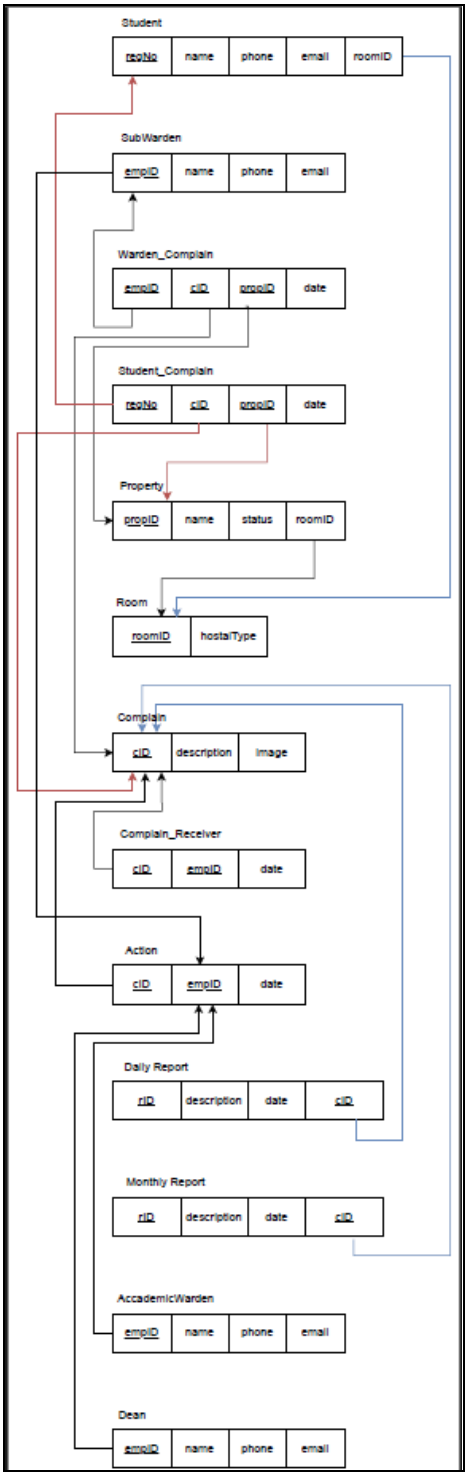
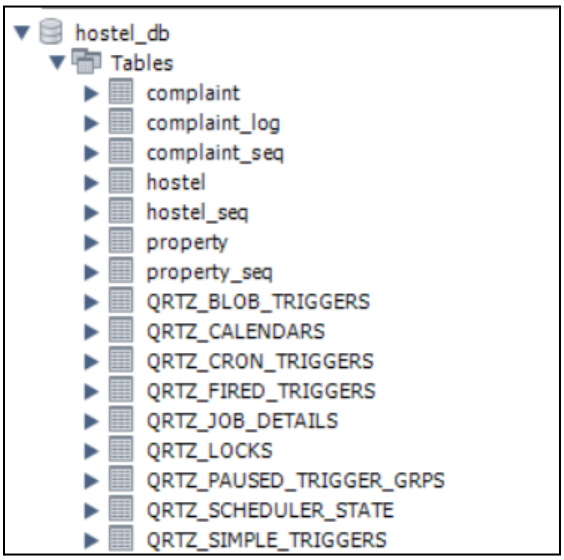
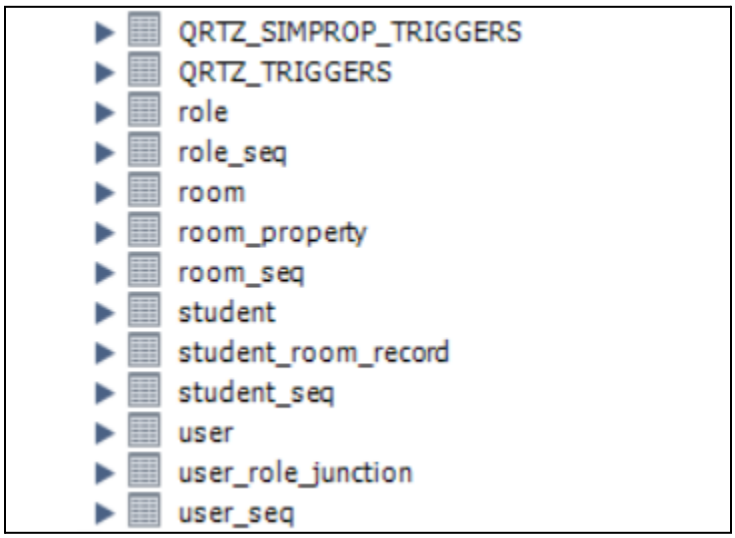


Table structure of your solution





Tools and technologies that you have used

Frontend Technologies:

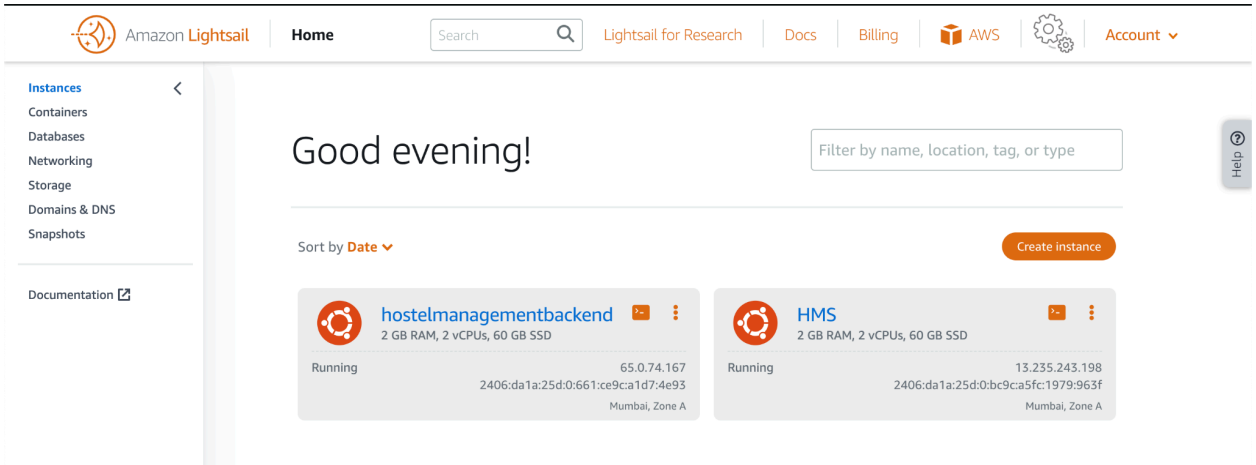
- Next.js: Used for server-side rendering and efficient routing, ensuring fast page loading and SEO optimization.
- React MUI (Material-UI): A React component library that adheres to Material Design guidelines, enhancing the user interface with pre-designed components.

Backend Technologies:

- Spring Boot: A Java-based framework employed for backend development, delivering robust and efficient server-side functionality.
- MySQL: An open-source relational database management system chosen for its performance and reliability, serving as the database solution for secure data storage and retrieval.

Where you have hosted your backend and reasons for the selection

We hosted our backend in AWS Lightsail.
AWS Lightsail enables us to create instances anywhere in the world in any size or any type of resource capacity. It is very flexible. Also we used Docker to containarize and auto deploy our backend whenever we do a change in the source code. Using Github actions we created a workflow to build the docker container and auto deploy it.



Security measures that you have taken to protect your DB

- We created multiple users and gave them access in levels to protect the database.
- We encrypted all the user passwords in the database.
- We used views to create abstract layers and gave access to those views not to the actual base tables.
- We uses JWT token authentication to further protect our API endpoints. By protecting the endpoints we protected the database from unauthorized alterations or data views.

Brief description about DB Accounts/Users and the reasons for creating such Accounts/Users

- root - This is the default user account that the DB contains with all the maximum privileges.
- subwardern - This is the user account that we created to give access to subwardens to view daily reports. And confirm if a complaint is resolved.
- wardern - This is the user account that we created to give access to wardens to view per 3 day reports. And confirm if a complaint is resolved.
- dean - This is the user account that we created to give access to the Dean to view reports that are older than a week. And confirm if a complaint is resolved.

Code of the connection class/es or framework/s of your front end to the database and reason/s for the selection of such class/es or framework/s

We used NextJS as our frontend. And using Typescript we called the API endpoints that we created in the Springboot backend and fetched the data. NextJS and Typescript provide easy access to our backend APIs. And we used JWT tokens to authenticate users then give access to the API endpoints.

```
const fetchComplaints = async () => {
const authRes = await fetch(process.env.NEXT_PUBLIC_API + '/auth/login', {
method: 'POST',
headers: {
'Content-Type': 'application/json',
},
body: JSON.stringify({
```

```
username: process.env.NEXT_PUBLIC_USERNAME,
password: process.env.NEXT_PUBLIC_PASSWORD,
}),
});

const authData = await authRes.json();
const token = authData.jwt;

const res = await fetch(process.env.NEXT_PUBLIC_API + '/api/complaint', {
method: 'GET',
headers: {
'Authorization': 'Bearer ' + token,
},
});

const complaintData: ComplaintData[] = await res.json();
setData(complaintData);
};
```

Code snippets to support your work

*** ORM tools and technologies**

We used hibernate as ORM tools and technologies.

Ex: Complaint.java

```
package me.mtron.hostelmanagementsystembackend.httpentities;

import jakarta.persistence.*;
import java.util.Calendar;
import java.util.Date;

@Entity
public class Complaint {

   @Id
   @GeneratedValue
   private Long id;
   private String item_code;
   private String description;
   private int room_no;
   private String stu_no;
   private String image_url;
   @Column(name = "status", nullable = false, columnDefinition = "varchar(255)
default 'pending'")
   private String status = "pending";
   @Column(name = "created_at")
```

```java
@Temporal(TemporalType.TIMESTAMP)
private Date created_at;

@PrePersist
public void prePersist() {
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(new Date());
    calendar.add(Calendar.HOUR, 5);
    calendar.add(Calendar.MINUTE, 30);
    this.created_at = calendar.getTime();
}

public void setId(Long id) {
    this.id = id;
}

public Long getId() {
    return id;
}

public String getItem_code() {
    return item_code;
}

public void setItem_code(String item_code) {
    this.item_code = item_code;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public int getRoom_no() {
    return room_no;
}

public void setRoom_no(int room_no) {
    this.room_no = room_no;
}

public String getStu_no() {
    return stu_no;
}

public void setStu_no(String stu_no) {
```

```
        this.stu_no = stu_no;
    }

    public String getImage_url() {
        return image_url;
    }

    public void setImage_url(String image_url) {
        this.image_url = image_url;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public Date getCreated_at() {
        return created_at;
    }

    public void setCreated_at(Date created_at) {
        this.created_at = created_at;
    }
}
```

ComplaintRepo.java

```
package me.mtron.hostelmanagementsystembackend.repo;

import me.mtron.hostelmanagementsystembackend.httpentities.Complaint;
import me.mtron.hostelmanagementsystembackend.projection.DailyReportProjection;
import
me.mtron.hostelmanagementsystembackend.projection.TopComplainedItemsProjection;
import
me.mtron.hostelmanagementsystembackend.projection.TopComplainedRoomsProjection;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface ComplaintRepo extends JpaRepository<Complaint, Long> {
    @Query(value = "SELECT * FROM daily_report_view;", nativeQuery = true)
    List<DailyReportProjection> getDailyReport();

    @Query(value = "SELECT * FROM monthly_report_view;", nativeQuery = true)
    List<DailyReportProjection> getMonthlyReport();
```

```
   @Query(value = "SELECT * FROM top_complained_rooms;", nativeQuery = true)
   List<TopComplainedRoomsProjection> getRoomComplaints();

   @Query(value = "SELECT * FROM top_complained_item;", nativeQuery = true)
   List<TopComplainedItemsProjection> getItemComplaints();
}
```

ComplaintController.java

```
package me.mtron.hostelmanagementsystembackend.controllers;

import me.mtron.hostelmanagementsystembackend.httpentities.Complaint;
import me.mtron.hostelmanagementsystembackend.models.DailyReport;
import me.mtron.hostelmanagementsystembackend.models.TopComplainedItems;
import me.mtron.hostelmanagementsystembackend.models.TopComplainedRooms;
import me.mtron.hostelmanagementsystembackend.projection.DailyReportProjection;
import
me.mtron.hostelmanagementsystembackend.projection.TopComplainedItemsProjection;
import
me.mtron.hostelmanagementsystembackend.projection.TopComplainedRoomsProjection;
import me.mtron.hostelmanagementsystembackend.repo.ComplaintRepo;
import me.mtron.hostelmanagementsystembackend.service.ComplaintService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Controller
@CrossOrigin("*")
@RequestMapping("/api")
public class ComplaintController {
   @Autowired
   private ComplaintRepo complaintRepo;

   @Autowired
   private ComplaintService complaintService;

   @PostMapping("/complaint")
   public ResponseEntity<Complaint> saveComplaint(@RequestBody Complaint c) {
       c = this.complaintService.handleRegistration(c);
       return ResponseEntity.ok().body(c);
   }

   @GetMapping("/complaint")
```

```
    public ResponseEntity<List<Complaint>> getAllComplaints() {
        List<Complaint> all = complaintRepo.findAll();
        return ResponseEntity.ok().body(all);
    }

    @GetMapping("/complaint/{id}")
    public ResponseEntity<Optional<Complaint>> findComplaint(@PathVariable Long
id) {
        Optional<Complaint> c = this.complaintRepo.findById(id);
        return ResponseEntity.ok().body(c);
    }

    @DeleteMapping("/complaint/{id}")
    public ResponseEntity<Optional<Complaint>> deleteComplaint(@PathVariable
Long id) {
        if (complaintRepo.existsById(id)) {
            this.complaintRepo.deleteById(id);
            return ResponseEntity.ok().build();
        }
        return ResponseEntity.status(400).build();
    }

    @PutMapping("/complaint/{id}")
    public ResponseEntity<Complaint> resolveComplaint(@PathVariable Long id) {
        Optional<Complaint> complaintOptional = this.complaintRepo.findById(id);
        if(complaintOptional.isPresent()) {
            Complaint complaint = complaintOptional.get();
            complaint.setStatus("resolved");
            this.complaintRepo.save(complaint);
            return ResponseEntity.ok().body(complaint);
        } else {
            return ResponseEntity.status(404).build();
        }
    }

    @GetMapping("/daily_report")
    public ResponseEntity<List<DailyReport>> getDailyReport() {
        List<DailyReportProjection> projections =
complaintRepo.getDailyReport();
        return getListResponseEntity(projections);
    }

    @GetMapping("/monthly_report")
    public ResponseEntity<List<DailyReport>> getMonthlyReport() {
        List<DailyReportProjection> projections =
complaintRepo.getMonthlyReport();
        return getListResponseEntity(projections);
    }
```

```
   private ResponseEntity<List<DailyReport>>
getListResponseEntity(List<DailyReportProjection> projections) {
       List<DailyReport> reports = new ArrayList<>();
       for(DailyReportProjection projection : projections) {
           DailyReport report = new DailyReport();
           report.setComplaint_id(projection.getComplaint_id());
           report.setDescription(projection.getDescription());
           report.setImage_url(projection.getImage_url());
           report.setItem_code(projection.getItem_code());
           report.setItem_name(projection.getItem_name());
           report.setRoom_no(projection.getRoom_no());
           report.setHostel_type(projection.getHostel_type());
           report.setStu_no(projection.getStu_no());
           report.setStudent_name(projection.getStudent_name());
           report.setStatus(projection.getStatus());

report.setComplaint_created_date(projection.getComplaint_created_date());
           reports.add(report);
       }
       return ResponseEntity.ok(reports);
   }

   @GetMapping("/top_complained_rooms")
   public ResponseEntity<List<TopComplainedRooms>> getTopComplainedRooms() {
       List<TopComplainedRoomsProjection> projections =
complaintRepo.getRoomComplaints();

       List<TopComplainedRooms> complaints = new ArrayList<>();
       for(TopComplainedRoomsProjection projection : projections) {
           TopComplainedRooms complaint = new TopComplainedRooms();
           complaint.setComplaint_count(projection.getComplaint_count());
           complaint.setRoom_no(projection.getRoom_no());
           complaints.add(complaint);
       }
       return ResponseEntity.ok(complaints);
   }

   @GetMapping("/top_complained_item")
   public ResponseEntity<List<TopComplainedItems>> getTopComplainedItems() {
       List<TopComplainedItemsProjection> projections =
complaintRepo.getItemComplaints();

       List<TopComplainedItems> complaints = new ArrayList<>();
       for(TopComplainedItemsProjection projection : projections) {
           TopComplainedItems complaint = new TopComplainedItems();
           complaint.setComplaint_count(projection.getComplaint_count());
           complaint.setProp_name(projection.getProp_name());
           complaint.setItem_code(projection.getItem_code());
           complaints.add(complaint);
```

```
        }
        return ResponseEntity.ok(complaints);
    }
}
```

*** Stored procedures**

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `GetMonthlyReport`()
BEGIN

    DROP VIEW IF EXISTS monthly_report_view;

    CREATE VIEW monthly_report_view AS
    SELECT
        c.id AS complaint_id,
        c.description,
        c.image_url,
        c.item_code,
        p.prop_name AS item_name,
        c.room_no,
        r.hostel_type,
        c.stu_no,
        CONCAT(s.first_name, ' ', s.last_name) AS student_name,
        c.status,
        c.created_at AS complaint_created_date
    FROM
        complaint c
    LEFT JOIN student s ON c.stu_no = s.reg_number
    LEFT JOIN room r ON c.room_no = r.room_no
    LEFT JOIN property p ON c.item_code = p.prop_id
    WHERE
        MONTH(c.created_at) = MONTH(DATE_SUB(CURRENT_DATE(), INTERVAL 1
DAY));

    SELECT * FROM monthly_report_view;
END ;;
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `GetDailyReport`()
```

```sql
BEGIN

    DROP VIEW IF EXISTS daily_report_view;

    CREATE VIEW daily_report_view AS
    SELECT
        c.id AS complaint_id,
        c.description,
        c.image_url,
        c.item_code,
        p.prop_name AS item_name,
        c.room_no,
        r.hostel_type,
        c.stu_no,
        CONCAT(s.first_name, ' ', s.last_name) AS student_name,
        c.status,
        c.created_at AS complaint_created_date
    FROM
        complaint c
    LEFT JOIN student s ON c.stu_no = s.reg_number
    LEFT JOIN room r ON c.room_no = r.room_no
    LEFT JOIN property p ON c.item_code = p.prop_id
    WHERE
    DATE(c.created_at) = DATE(CURDATE() - INTERVAL 1 DAY);

  SELECT * FROM daily_report_view;

END ;;
DELIMITER ;
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `generate_top_complained_rooms_view`()
BEGIN
 DROP VIEW IF EXISTS top_complained_rooms;
 CREATE VIEW top_complained_rooms AS
 SELECT room_no, COUNT(id) AS complaint_count
 FROM complaint
 GROUP BY room_no
 ORDER BY complaint_count DESC
 LIMIT 5;
END ;;
```

```
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `generate_top_complained_item_view`()
BEGIN
 DROP VIEW IF EXISTS top_complained_item;
 CREATE VIEW top_complained_item AS
 SELECT c.item_code, p.prop_name, COUNT(*) as complaint_count
 FROM complaint c
 INNER JOIN property p ON c.item_code = p.prop_id
 GROUP BY c.item_code, p.prop_name
 ORDER BY complaint_count DESC
 LIMIT 5;
END ;;
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `escalated_complaints_level_two`(IN
given_date DATE)
BEGIN
    SET @given_date := given_date;
    SET @sql = CONCAT('CREATE OR REPLACE VIEW
escalated_complaints_level_two AS SELECT cl.id AS complaint_log_id, c.id
AS complaint_id, c.item_code, c.description, c.room_no, c.stu_no,
c.image_url, c.status, c.created_at FROM complaint_log cl JOIN complaint c
ON cl.compaint_id = c.id WHERE c.status = "pending" AND
DATEDIFF("',@given_date,'", cl.created_time) >= 7');
    PREPARE stmt FROM @sql;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END ;;
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `create_escalated_complaints_view`(IN
given_date DATE)
BEGIN
    SET @given_date := given_date;
    SET @sql = CONCAT('CREATE OR REPLACE VIEW
escalated_complaints_level_one AS SELECT cl.id AS complaint_log_id, c.id
```

```
AS complaint_id, c.item_code, c.description, c.room_no, c.stu_no,
c.image_url, c.status, c.created_at FROM complaint_log cl JOIN complaint c
ON cl.compaint_id = c.id WHERE c.status = "pending" AND
DATEDIFF("',@given_date,'", cl.created_time) >= 3');
    PREPARE stmt FROM @sql;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END ;;
DELIMITER ;
```

*** Stored functions**

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` FUNCTION
`get_room_number_from_property_id`(prop_id VARCHAR(255)) RETURNS
varchar(255) CHARSET utf8mb4
    DETERMINISTIC
BEGIN
  DECLARE room_number VARCHAR(255);
  SET room_number = SUBSTRING(prop_id, 3, 3);
  RETURN room_number;
END ;;
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` FUNCTION `get_average_complaints_per_property`()
RETURNS decimal(10,2)
    DETERMINISTIC
BEGIN
    DECLARE avgComplaints DECIMAL(10,2);
    SELECT AVG(complaint_count) INTO avgComplaints
    FROM (
        SELECT COUNT(*) as complaint_count
        FROM complaint
        GROUP BY item_code
    ) AS subquery;
    RETURN avgComplaints;
END ;;
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` FUNCTION `count_complaints_for_property`(propID
VARCHAR(255)) RETURNS int
    DETERMINISTIC
BEGIN
    DECLARE complaintCount INT;
    SELECT COUNT(*) INTO complaintCount
    FROM complaint
    WHERE item_code = propID;
    RETURN complaintCount;
END ;;
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` FUNCTION `check_property_status`(propID
VARCHAR(255)) RETURNS varchar(255) CHARSET utf8mb4
    DETERMINISTIC
BEGIN
    DECLARE propStatus VARCHAR(255);
    SELECT status INTO propStatus
    FROM property
    WHERE prop_id = propID;
    RETURN propStatus;
END ;;
DELIMITER ;
```

```
DELIMITER ;;
CREATE DEFINER=`root`@`%` FUNCTION `check_complaint_status`(complaint_id
INT) RETURNS varchar(255) CHARSET utf8mb4
    DETERMINISTIC
BEGIN
    DECLARE complaint_status VARCHAR(255);
    SELECT status INTO complaint_status
    FROM complaint
    WHERE id = complaint_id;
    RETURN complaint_status;
END ;;
DELIMITER ;
```

* Events

```
DELIMITER ;;
CREATE EVENT `call_create_escalated_complaints_view`
  ON SCHEDULE EVERY 1 DAY
  STARTS '2023-11-03 00:00:00'
  ON COMPLETION NOT PRESERVE ENABLE DO
  CALL create_escalated_complaints_view(CURDATE());;

CREATE EVENT `call_create_escalated_complaints_view_dean`
  ON SCHEDULE EVERY 1 DAY
  STARTS '2023-11-03 00:00:00'
  ON COMPLETION NOT PRESERVE ENABLE DO
  CALL create_escalated_complaints_view_dean(CURDATE());;

CREATE EVENT `update_daily_report`
  ON SCHEDULE EVERY 1 DAY
  STARTS '2023-11-03 00:00:00'
  ON COMPLETION NOT PRESERVE ENABLE
  DO BEGIN
    CALL GetDailyReport();
  END;;

CREATE EVENT `update_monthly_report`
  ON SCHEDULE EVERY 1 MONTH
  STARTS '2023-11-01 00:00:00'
  ON COMPLETION NOT PRESERVE ENABLE
  DO BEGIN
    CALL GetMonthlyReport();
  END;;

CREATE EVENT `update_top_complained_views`
  ON SCHEDULE EVERY 5 HOUR
  STARTS '2023-11-05 21:40:34'
  ON COMPLETION NOT PRESERVE ENABLE
  DO CALL generate_top_complained_item_view();
```

* Triggers

```
-- Trigger to update property status and log complaint on insert

DELIMITER ;;
CREATE TRIGGER `complaint_AFTER_INSERT` AFTER INSERT ON `complaint` FOR EACH
ROW
```

```
BEGIN
  UPDATE property
  SET status = NEW.description
  WHERE prop_id = NEW.item_code;

  INSERT INTO complaint_log (compaint_id, created_time)
  VALUES (NEW.id, NOW());
END;;
DELIMITER ;

-- Trigger to resolve complaint and property on update

DELIMITER ;;
CREATE TRIGGER `complaint_AFTER_UPDATE` AFTER UPDATE ON `complaint` FOR
EACH ROW
BEGIN
  IF NEW.status = 'resolved' THEN
    UPDATE property
    SET status = 'no damage'
    WHERE prop_id = NEW.item_code;

    UPDATE complaint_log
    SET resolved_time = NOW()
    WHERE compaint_id = NEW.id;
  END IF;
END;;
DELIMITER ;

-- Trigger to map property to room on insert

DELIMITER ;;
CREATE TRIGGER `after_property_insert` AFTER INSERT ON `property` FOR EACH ROW
BEGIN
  DECLARE room_number VARCHAR(255);
  DECLARE property_id BIGINT;

  SET room_number = get_room_number_from_property_id(NEW.prop_id);
  SET property_id = NEW.id;

  INSERT INTO room_property(room_id, property_id)
  VALUES ((SELECT id FROM room WHERE room_no = room_number), property_id);
END;;
DELIMITER ;
```

* Views

```
CREATE VIEW daily_report_view AS
  SELECT
    c.*, p.*, s.*, r.*
  FROM
    complaint c
    JOIN property p
    JOIN student s
    JOIN room r
  WHERE
    CAST(c.created_at AS DATE) = CURRENT_DATE() - INTERVAL 1 DAY;

CREATE VIEW monthly_report_view AS
  SELECT
    c.*, p.*, s.*, r.*
  FROM
    complaint c
    JOIN property p
    JOIN student s
    JOIN room r
  WHERE
    MONTH(c.created_at) = MONTH(CURRENT_DATE() - INTERVAL 1 DAY);

CREATE VIEW escalated_complaints_level_one AS
  SELECT
    cl.*, c.*
  FROM
    complaint_log cl
    JOIN complaint c
  WHERE
    c.status = 'pending'
    AND DATEDIFF(DATE, cl.created_time) >= 3;

CREATE VIEW top_complained_rooms AS
  SELECT
    room_no, COUNT(*) AS complaint_count
  FROM complaint
  GROUP BY room_no
  ORDER BY complaint_count DESC
  LIMIT 5;
```

Problems that you faced during the development of the backend

- Very little documentation about the springboot latest library versions so I had to do a lot of research in order to get it working.
- Learning new technologies such as nextJS in the short amount of time for the development

Solutions/how you have overcome the above identified problems

- We have researched and developed according to the tutorials and documentation and we implemented our own code to the project.
- Watching tutorials and implementing them into our project was a better practice for us to develop our skills in the language.

New database technologies/trends that you have used to develop the backend

We have used events, stored procedures, stored functions, events, views, and triggers we have learned in the module. The springboot API handling was also a new technology we used for our development.

If you are going to host your backend in a cloud environment what are the things/changes that you must do in your backend

Both of our frontend and backend are already hosted.

Frontend: https://hms.mtron.biz
Backend: https://hmsapi.mtron.biz/api/

Hypothetically, If we are planning to host our backend in a cloud environment, several important considerations and changes must be made to ensure a smooth and efficient deployment. Firstly, we need to adapt our backend architecture to the cloud infrastructure, which may involve refactoring our application to be cloud-native. This includes designing for scalability and high

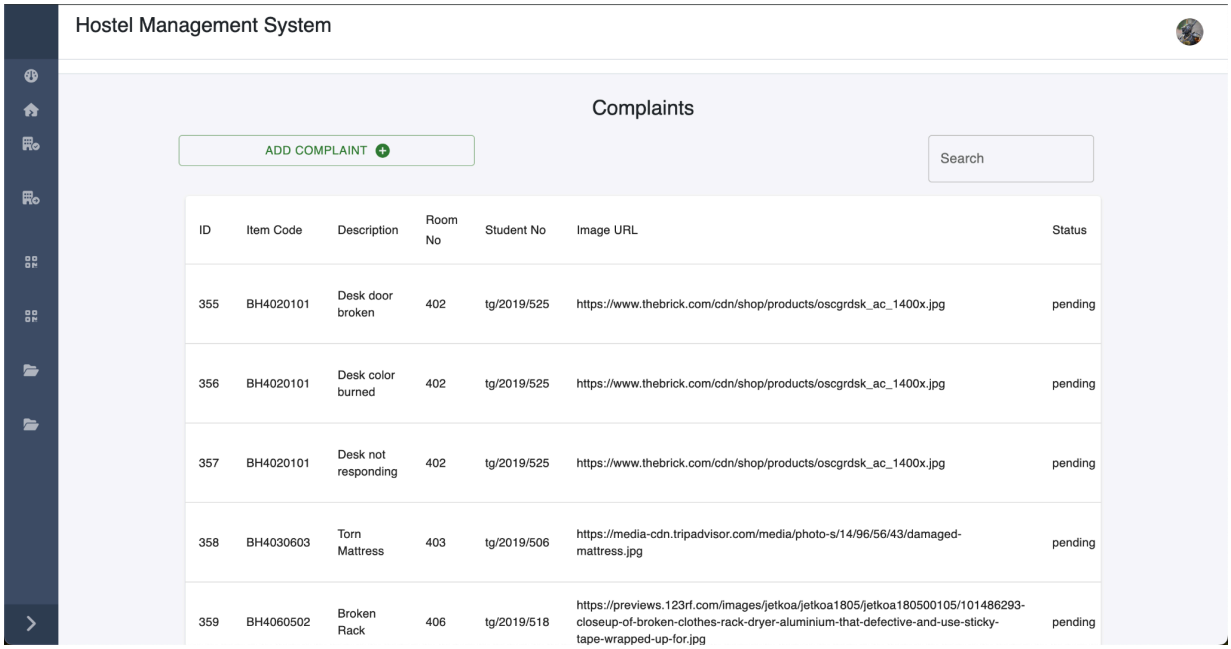
availability, as cloud environments offer dynamic resource allocation and redundancy. We should also reconfigure our database to align with cloud database services or utilize managed database solutions provided by the cloud provider. Additionally, security measures, such as access control, encryption, and identity management, should be reinforced to protect our data and resources in the cloud. Monitoring and logging tools should be integrated to track the performance and health of our backend, as cloud environments offer extensive monitoring capabilities. Finally, we must optimize our backend for cost-efficiency, as cloud hosting typically involves pay-as-you-go pricing, so optimizing resource usage and selecting the appropriate instance types are essential to control costs. Transitioning to a cloud environment offers benefits in terms of scalability, reliability, and flexibility, but it also requires careful planning and adaptation of our backend infrastructure to harness these advantages effectively.

What are the possibilities for you to replace your relational DB backend with a non-relational DB technology

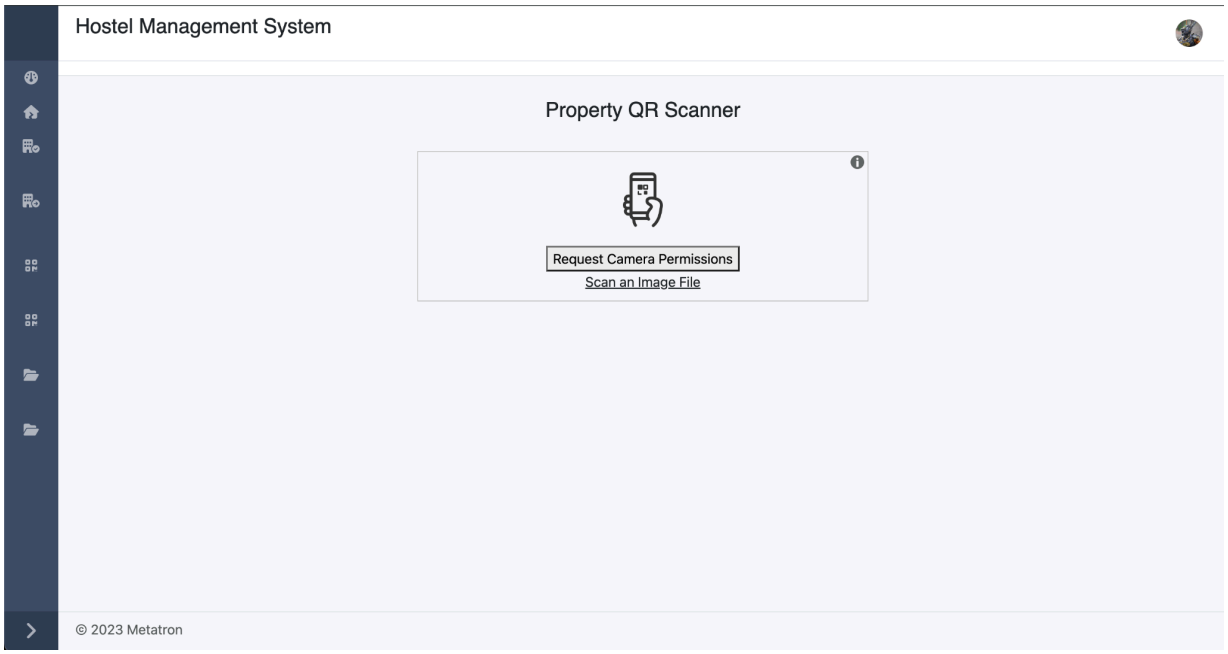
To replace our MySQL relational database with MongoDB, we would need to follow a systematic approach. Begin by defining the data model in MongoDB, as it uses a document-based, NoSQL data structure. Migrate our existing schema to a schema-less document format, considering how data will be organized within collections. Next, develop data migration scripts or tools to transfer data from MySQL tables to MongoDB collections. We should also rewrite the data access layer of our application to use MongoDB's query language and API. Update the code that interacts with the database to work with MongoDB's document-based approach, rather than SQL. It's essential to ensure data consistency during the migration process and account for any differences in data modeling. Moreover, consider how MongoDB's horizontal scalability and flexibility can benefit our application and its future growth. Thorough testing and validation are vital to guarantee a smooth transition and ensure that our application operates effectively with the new database technology.

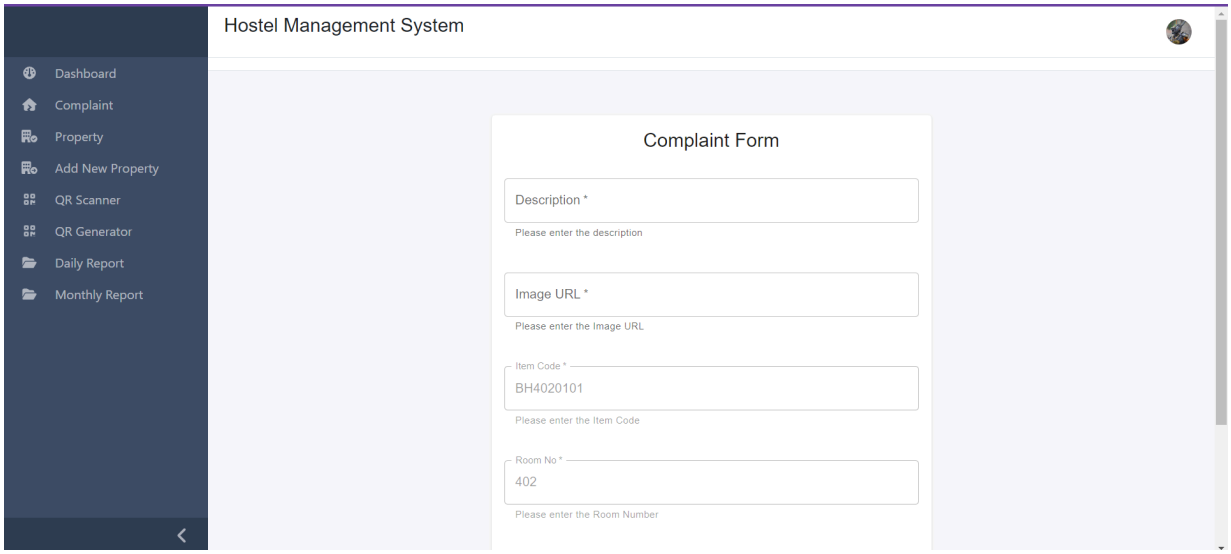
Individual contribution to the backend development/Detailed description of your contribution

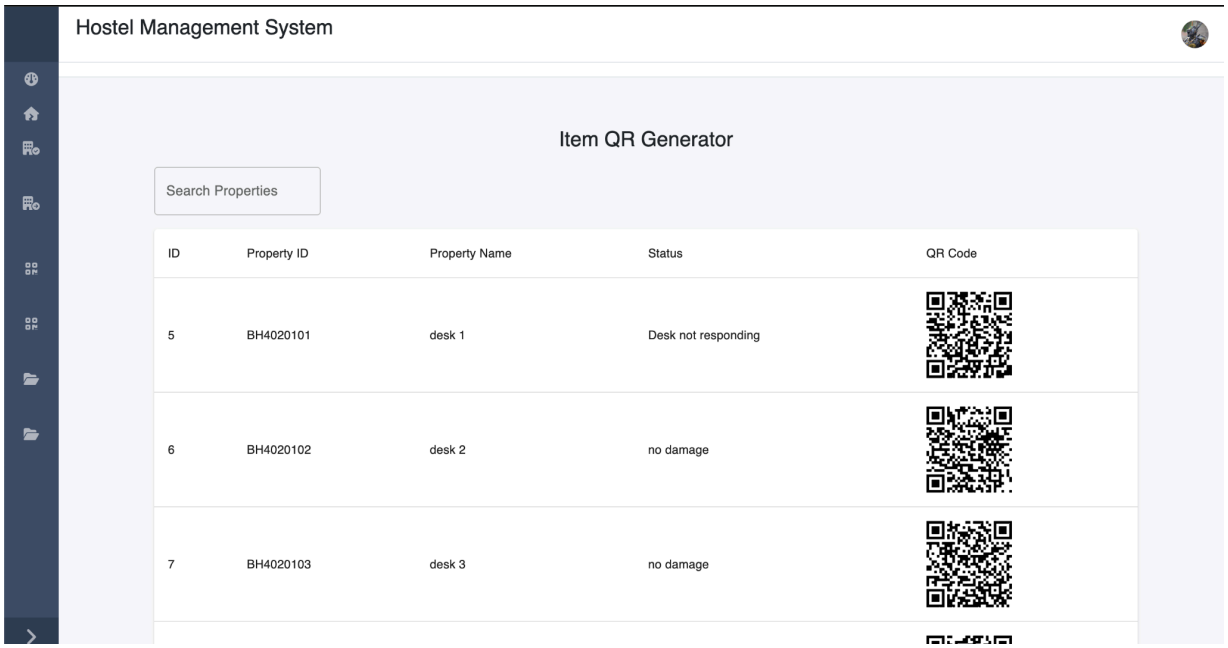
TG491 - I have implemented complaint views, complaint adding functionality, QR code generation and scanning, and the basic authentication flow for the front-end. I've also established the basic structure and admin layout for the front-end. In the database, I've linked tables with relationships and created triggers for handling related complaints and other tables. Additionally, I've developed stored procedures for managing complaints and created views for escalating issues after 3 and 7 days, which are automated with events.



ID	Item Code	Description	Room No	Student No	Image URL	Status
355	BH4020101	Desk door broken	402	tg/2019/525	https://www.thebrick.com/cdn/shop/products/oscgrdsk_ac_1400x.jpg	pending
356	BH4020101	Desk color burned	402	tg/2019/525	https://www.thebrick.com/cdn/shop/products/oscgrdsk_ac_1400x.jpg	pending
357	BH4020101	Desk not responding	402	tg/2019/525	https://www.thebrick.com/cdn/shop/products/oscgrdsk_ac_1400x.jpg	pending
358	BH4030603	Torn Mattress	403	tg/2019/506	https://media-cdn.tripadvisor.com/media/photo-s/14/96/56/43/damaged-mattress.jpg	pending
359	BH4060502	Broken Rack	406	tg/2019/518	https://previews.123rf.com/images/jetkoa/jetkoa1805/jetkoa180500105/101486293-closeup-of-broken-clothes-rack-dryer-aluminium-that-defective-and-use-sticky-tape-wrapped-up-for.jpg	pending





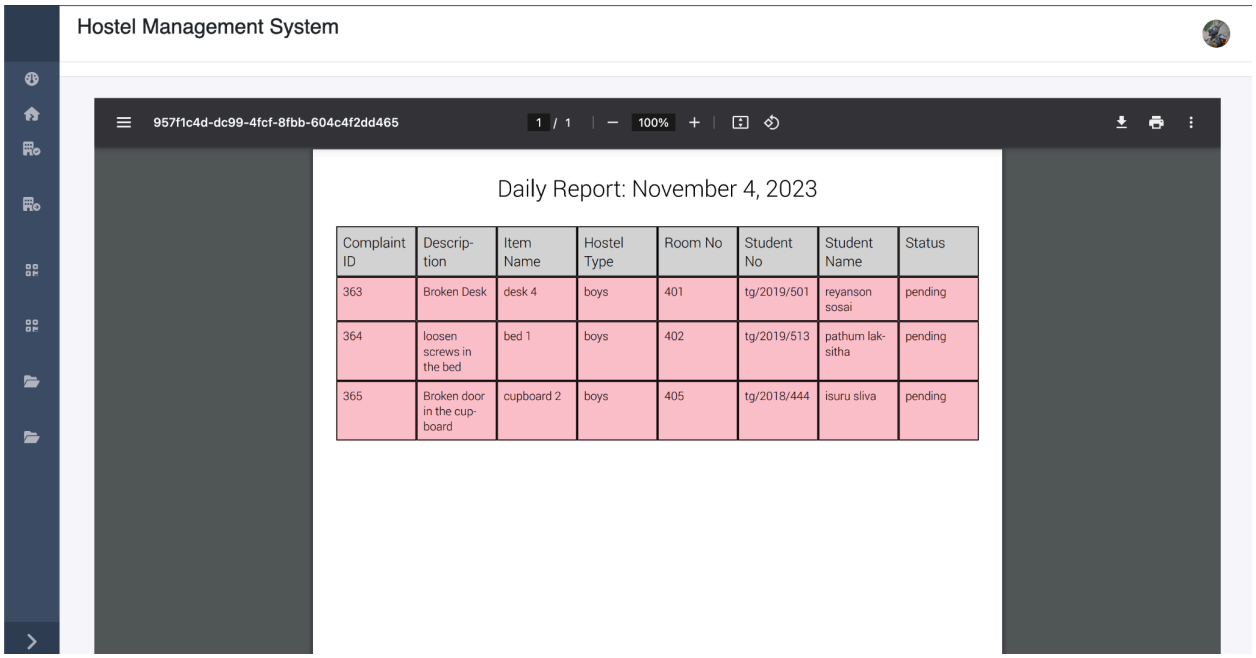



```
CREATE DEFINER=`root`@`%` PROCEDURE `create_escalated_complaints_view`(IN
given_date DATE)
BEGIN
    SET @given_date := given_date;
    SET @sql = CONCAT('CREATE OR REPLACE VIEW escalated_complaints_level_one AS
SELECT cl.id AS complaint_log_id, c.id AS complaint_id, c.item_code, c.description,
c.room_no, c.stu_no, c.image_url, c.status, c.created_at FROM complaint_log cl JOIN complaint
c ON cl.compaint_id = c.id WHERE c.status = "pending" AND DATEDIFF('",@given_date,'",
cl.created_time) >= 3');
    PREPARE stmt FROM @sql;
    EXECUTE stmt;
```

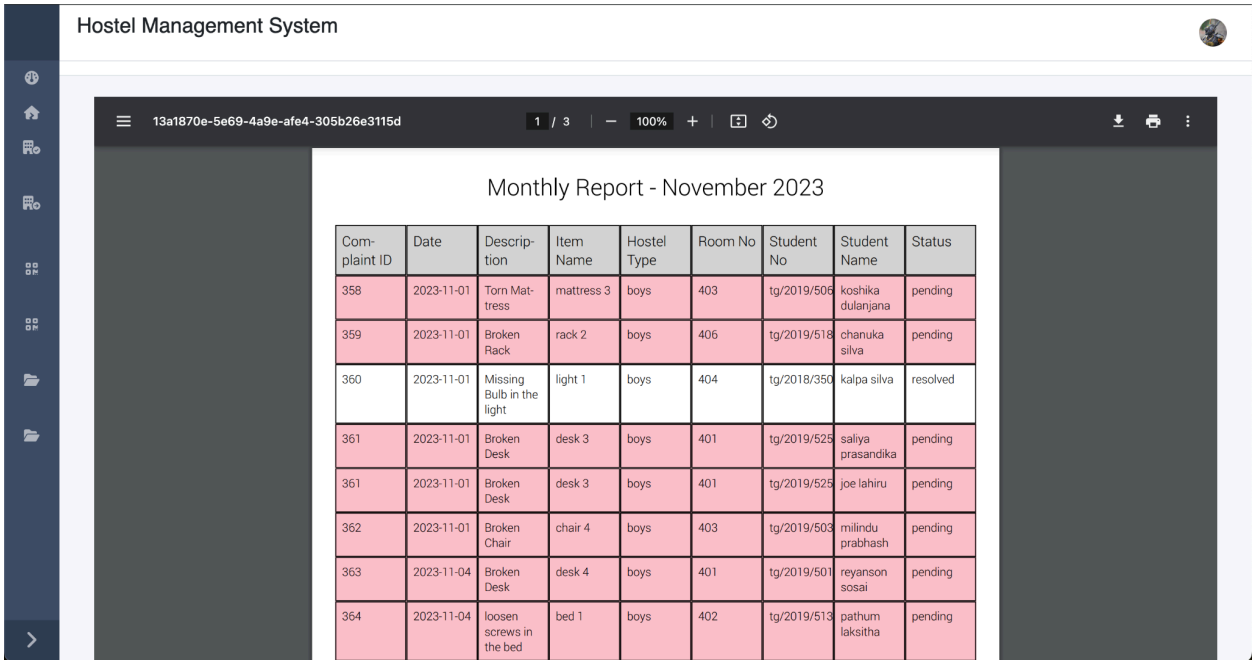
```
    DEALLOCATE PREPARE stmt;
END

CREATE DEFINER=`root`@`%` PROCEDURE `create_escalated_complaints_view_dean`(IN
given_date DATE)
BEGIN
    SET @given_date := given_date;
    SET @sql = CONCAT('CREATE OR REPLACE VIEW escalated_complaints_level_two AS
SELECT cl.id AS complaint_log_id, c.id AS complaint_id, c.item_code, c.description,
c.room_no, c.stu_no, c.image_url, c.status, c.created_at FROM complaint_log cl JOIN complaint
c ON cl.compaint_id = c.id WHERE c.status = "pending" AND DATEDIFF('',@given_date,'',
cl.created_time) >= 7');
    PREPARE stmt FROM @sql;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END
```

TG513 - I created the report generation of the system so the admins of the system can check daily or monthly. They can be viewed in the system also as a pdf format as the format given below. I have also done the stored procedures and views related to report generation.

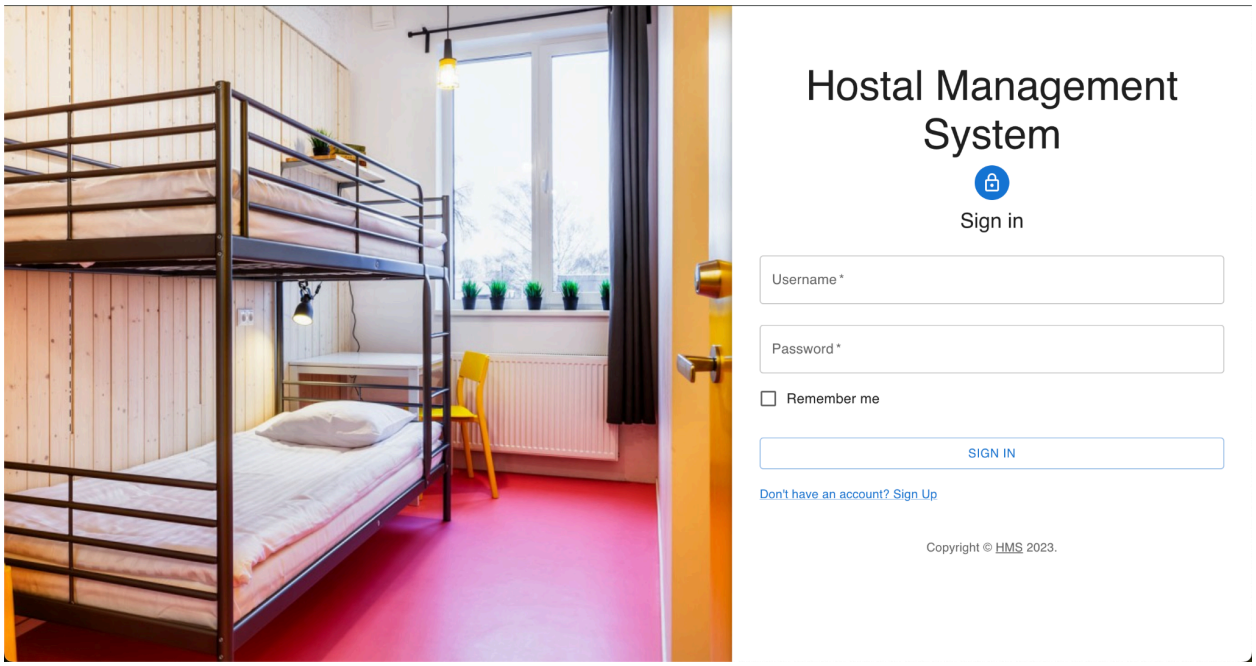


Complaint ID	Descrip-tion	Item Name	Hostel Type	Room No	Student No	Student Name	Status
363	Broken Desk	desk 4	boys	401	tg/2019/501	reyanson sosai	pending
364	loosen screws in the bed	bed 1	boys	402	tg/2019/513	pathum lak-sitha	pending
365	Broken door in the cup-board	cupboard 2	boys	405	tg/2018/444	isuru sliva	pending



Com-plaint ID	Date	Descrip-tion	Item Name	Hostel Type	Room No	Student No	Student Name	Status
358	2023-11-01	Torn Mat-tress	mattress 3	boys	403	tg/2019/506	koshika dulanjana	pending
359	2023-11-01	Broken Rack	rack 2	boys	406	tg/2019/518	chanuka silva	pending
360	2023-11-01	Missing Bulb in the light	light 1	boys	404	tg/2018/350	kalpa silva	resolved
361	2023-11-01	Broken Desk	desk 3	boys	401	tg/2019/525	saliya prasandika	pending
361	2023-11-01	Broken Desk	desk 3	boys	401	tg/2019/525	joe lahiru	pending
362	2023-11-01	Broken Chair	chair 4	boys	403	tg/2019/503	milindu prabhash	pending
363	2023-11-04	Broken Desk	desk 4	boys	401	tg/2019/501	reyanson sosai	pending
364	2023-11-04	loosen screws in the bed	bed 1	boys	402	tg/2019/513	pathum laksitha	pending

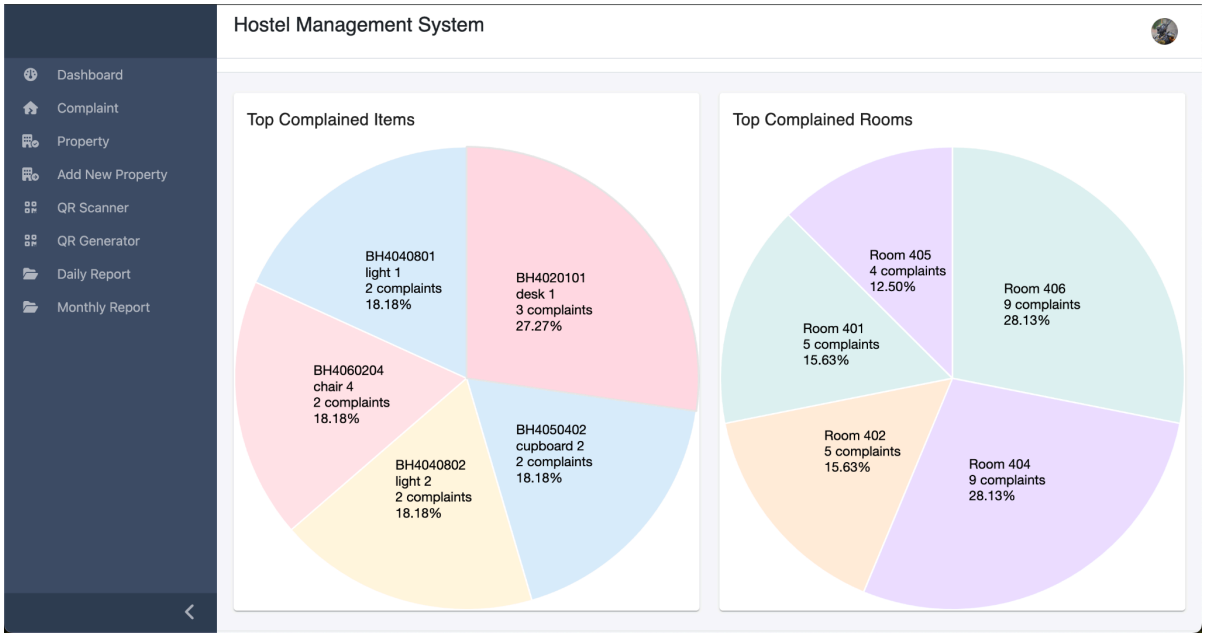
TG516 - I created the login, sign up and the dashboard for the frontend of the project. I created the UI with the help of material ui and next.js I made all the diagrams and relational mapping that needs to development. Also I added the data to the database that needs for the functionality of the system.

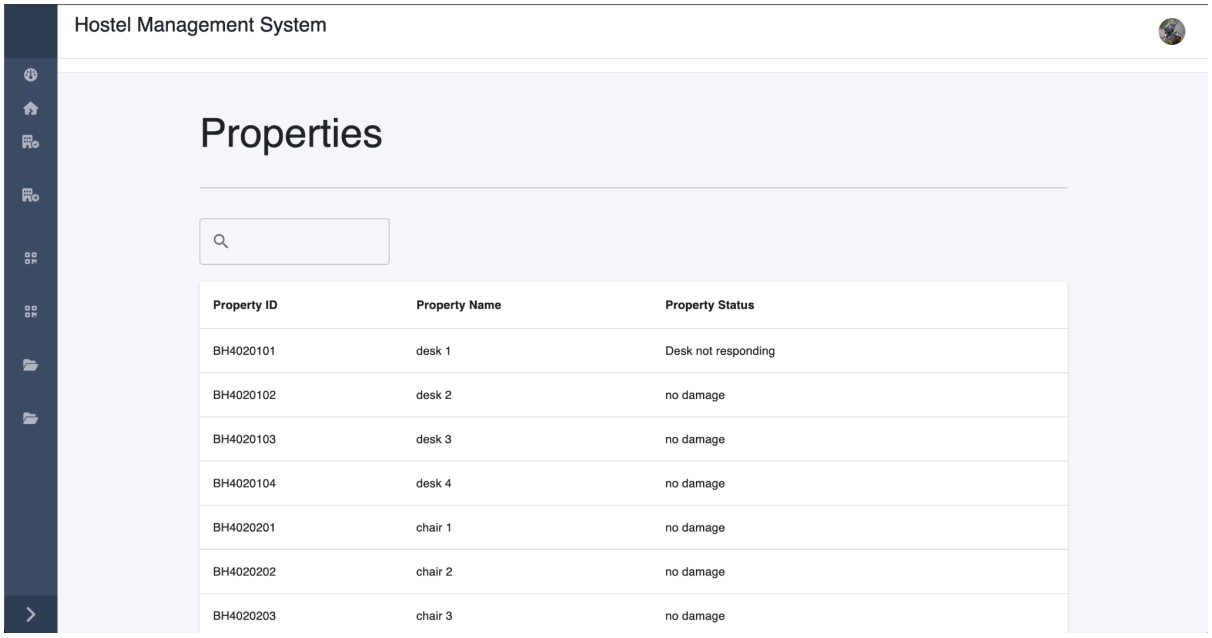


TG525 - I created and managed the Springboot backend and API functionalities with the CI-CD pipline in Github Actions. Also I did the property viewing, property adding pages and dashboard pie charts in the frontend. Also I hosted both backend and database in AWS Lightsail. I created

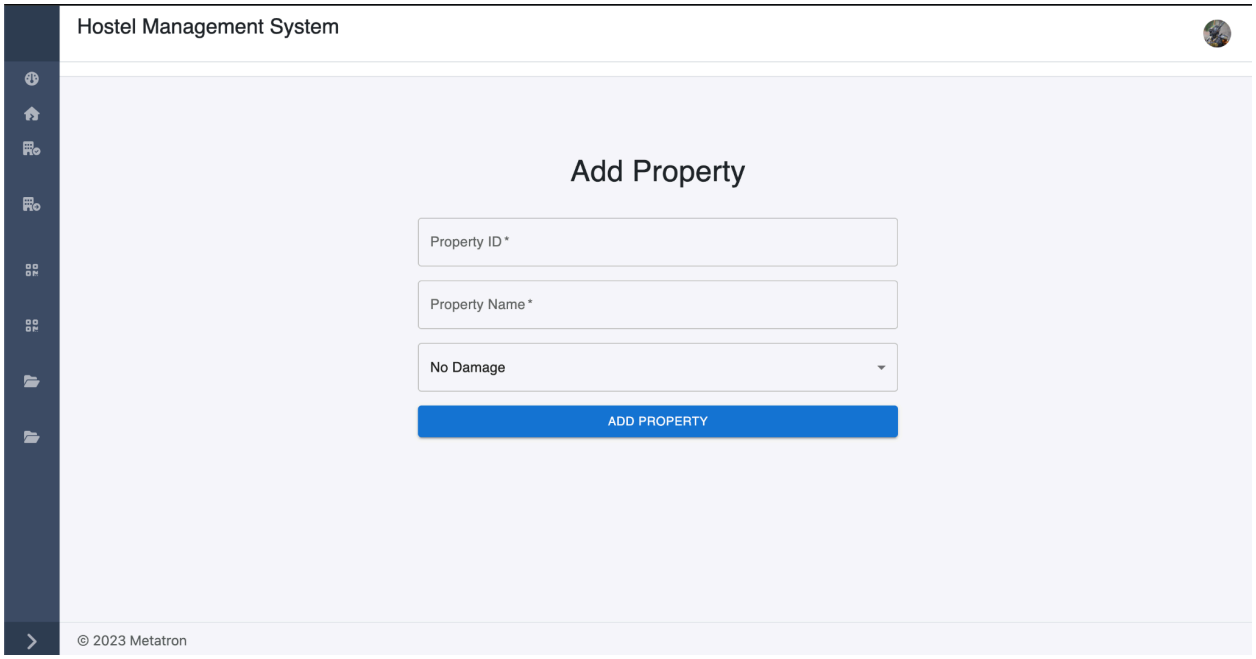
the functions, procedures, event and views for the dashboard graphs. Using the springboot backend and hibernate I designed the main tables and main database structure in the database.

Screenshots:





Property ID	Property Name	Property Status
BH4020101	desk 1	Desk not responding
BH4020102	desk 2	no damage
BH4020103	desk 3	no damage
BH4020104	desk 4	no damage
BH4020201	chair 1	no damage
BH4020202	chair 2	no damage
BH4020203	chair 3	no damage



Code Snippits:

```
CREATE DEFINER=`root`@`%` PROCEDURE `generate_top_complained_rooms_view`()
BEGIN
 DROP VIEW IF EXISTS top_complained_rooms;
 CREATE VIEW top_complained_rooms AS
 SELECT room_no, COUNT(id) AS complaint_count
 FROM complaint
 GROUP BY room_no
 ORDER BY complaint_count DESC
 LIMIT 5;
END

CREATE DEFINER=`root`@`%` PROCEDURE `generate_top_complained_item_view`()
BEGIN
 DROP VIEW IF EXISTS top_complained_item;
 CREATE VIEW top_complained_item AS
 SELECT c.item_code, p.prop_name, COUNT(*) as complaint_count
 FROM complaint c
 INNER JOIN property p ON c.item_code = p.prop_id
 GROUP BY c.item_code, p.prop_name
 ORDER BY complaint_count DESC
 LIMIT 5;
END
```

```
CREATE DEFINER=`root`@`%` FUNCTION `get_room_number_from_property_id`(prop_id
VARCHAR(255)) RETURNS varchar(255) CHARSET utf8mb4
    DETERMINISTIC
BEGIN
  DECLARE room_number VARCHAR(255);
  SET room_number = SUBSTRING(prop_id, 3, 3);
  RETURN room_number;
END
```