

Project Report

for

Hostel Mess Management System

Prepared by

Prasun Kumar (1801126)
Prateek Jain (1801127)
Shreyansh Singh Tomar (1801169)

Indian Institute of Information Technology, Guwahati

15 April 2021

Contents

1	Introduction	1
1.1	Project Description	1
1.2	Purpose	1
1.3	Product Scope	2
1.4	Current System	2
1.5	Benefits of proposed system	2
2	Overall Description	3
2.1	Product Description	3
2.2	Product Features	3
2.3	User Classes and Characteristics	4
2.4	Constraints	4
2.5	Assumptions	4
3	Requirements	5
3.1	User Interface Requirements	5
3.2	Functional Requirements	5
3.3	Non Functional Requirements	6
3.3.1	Flexibility	6
3.3.2	High Peak Performance	6
3.3.3	Data Integrity	6
3.3.4	Reliability	6
3.3.5	Security	6
3.3.6	Easy to use interface	6

3.4	Software Interface Requirements	6
4	Software Analysis and Design	8
4.1	Use Case Identification and Planning	8
4.2	ER Design	9
4.3	Database Design	10
4.4	Data Flow Diagram	11
5	Implementation	12
5.1	Technologies, Tools and Frameworks	13
5.1.1	Database/Model	13
5.1.2	Backend/Controller	13
5.1.3	Frontend/Views	13
5.2	Process	14
5.2.1	Setting up and Connecting to database	14
5.2.2	Dependencies management	15
5.2.3	Developing Modules	15
5.2.4	Deployment	25
5.2.5	Testing	25
6	References	26

Chapter 1

Introduction

1.1 Project Description

A hostel mess is a vital part of an institute's infrastructure. It provides the people living far from their homes with the basic necessity of life. With such parts of a system, which are vital and essential for daily working, comes the responsibility of making it work as smoothly as possible.

There are quite a few problems in a mess that could be solved or made simpler using a specialised management software. The software could deal with multiple modes of operation such as Food Coupons as well as pay-per-month services to students. Different charges and bills can be issued to different people in such a software using required authentication features which would generally create a problem in implementing without a software on a short-staffed or busy day.

Lodging complaints, daily and anytime accessible bills as well as keeping track of past payments and dues could be simplified significantly using such a software minimising large and repetitive paperwork. Furthermore, the data generated daily could be analysed and inferred to get a prediction of food needed, minimising wastage. This would be helpful to the caterer as well as to our society in general which faces with wastage along with scarcity of food.

1.2 Purpose

The purpose of developing this system is to provide a robust, highly reliable, feature-rich and user-centric centralised management system for hostel messes. The software would be a mobile application which would be available to the residents of a hostel or similar establishment. This software would move much of the workflow of the mess to a digital platform.

1.3 Product Scope

It will automate the work like maintaining records of mess dues and payment history, coupon booking, lodging complaint and providing feedback for both residents and caterers. The system would be flexible enough to easily adapt to any hostel mess and provide services to its users.

1.4 Current System

The way the mess currently operates introduces lots of friction for the caterers to provide the services as well as the residents to avail those services, some of which are listed below:

- Currently, there is no way for the caterers to estimate the number of students who will attend the mess at any point of time. This leads to overestimation or underestimation of the food preparation which in turn leads to food wastage or scarcity.
- Residents tend to skip some meals. However, they still have to pay for the entire month of service.
- There is no standardized procedure for handling complaints and feedback. Students find it difficult to lodge complaints. The mess manager is not easily approachable. Changes take long time to take effect.

1.5 Benefits of proposed system

The students and visitors will be benefited from the app as follows:

- Coupon based system would allow students to pay for only those meals that they eat.
- With the centralized complaints system, it will be easy for them to lodge complaints and offer suggestions. The app will help them track the status of their complaints.

The caterers will be benefited from the system as follows:

- The app will provide the estimate of the students who will attend the mess making it is easier for them to estimate the quantity of raw materials required. This will help overcome situations of food wastage and scarcity.
- Feedback and complaints on the extended services can be easily monitored.
- Do away with manual system of maintaining the paperwork.
- Helps in advance planning.

Chapter 2

Overall Description

2.1 Product Description

End user product is a mobile app through which the user can avail mess services. The app supports three classes of users: **Students**, **Guests** and **Caterers**. Based on the class, a user can perform various actions.

2.2 Product Features

- Sign Up and Login
- *Plans:*
 1. Coupon Based
 2. Pay per Month
- Payment Interface.
- *Tracking:*
 1. Payment History
 2. Coupons Left
 3. Days attended
- Lodge Complaints/ Provide feedback.
- Pause/ Resume Pay Per Month service.
- Weekly Reports and Analytics to Caterers.
- Complaint Handling for Caterers.
- Hassle Free Paperless work for Mess Management.

2.3 User Classes and Characteristics

User Class	Intended uses
Students	Ability to buy coupons and pay bills
Visitors	Buy coupons
Admin	Generate bills, collect payments and complaints

2.4 Constraints

- An employee of the mess would need to be present during food serving to validate coupons using the app by scanning the codes.
- The size of the media uploaded during complaint registration would need to be capped to a limit to minimize server costs.

2.5 Assumptions

- Every consumer in the mess would carry a mobile phone with them.
- The establishment has good internet connectivity which would be required by the mess employee during coupon or pass validation.
- The price of a meal using a coupon is greater than the price by using the pay-per-month service. This will ensure the caterer doesn't suffer any losses by students opting the coupon system.

Chapter 3

Requirements

3.1 User Interface Requirements

- A register page for users to create their accounts.
- A login page for users to submit their credentials and log into their accounts.
- A page for buying coupons.
- A billing dashboard where users can see their due and past bills and make payments.
- A complaint lodging page to register complaints.
- A page to track status of complaints.
- A module on the admin side to validate coupons.
- A complaint tracking page on admin side.
- A leave management system for applying and approving leaves.

3.2 Functional Requirements

- Users could register their accounts using a valid college email address or as a visitor.
- Users could opt in for coupon-based system or pay-per-month service for a month.
- Users can pay bills using the inbuilt payment interface.
- Users can book coupons from inside the app.
- Users can keep track of all their past coupons and paid bills.
- Users can lodge complaints or give suggestions regarding the food quality and cleanliness in the establishment.

- If opted for the pay-per-month service, users would be able to apply for a leave if they are leaving for more than five days.

3.3 Non Functional Requirements

3.3.1 Flexibility

The software is flexible enough to adapt to any institution with minimal changes to the code.

3.3.2 High Peak Performance

The system would be designed to handle bursty traffic which would peak at around 3 times a day. The system would be maintained in off hours to ensure smooth functioning during peak traffic.

3.3.3 Data Integrity

The records of transactions kept by the system would be stored reliably and would be backed up regularly to ensure that the data is reliable.

3.3.4 Reliability

The system architecture should be simple to ensure that the software is reliable and bug-free.

3.3.5 Security

The payment module would need to ensure secure payments

3.3.6 Easy to use interface

Since the product is intended for a widespread users, the user interface should be easy to use. This will ensure maximum adoption of the system.

3.4 Software Interface Requirements

For development of the product, the following requirements have been identified

Software Type	Component
Deployment Server OS	Linux
Databases	PostgreSQL
Development tools	VSCode, PyCharm IDE, Android Studio
Design Tool	Figma

For using the product the following components are supported

Software Type	Component
Operating System	Cross Platform
Web Browser	Mozilla Firefox, Google Chrome

Chapter 4

Software Analysis and Design

4.1 Use Case Identification and Planning

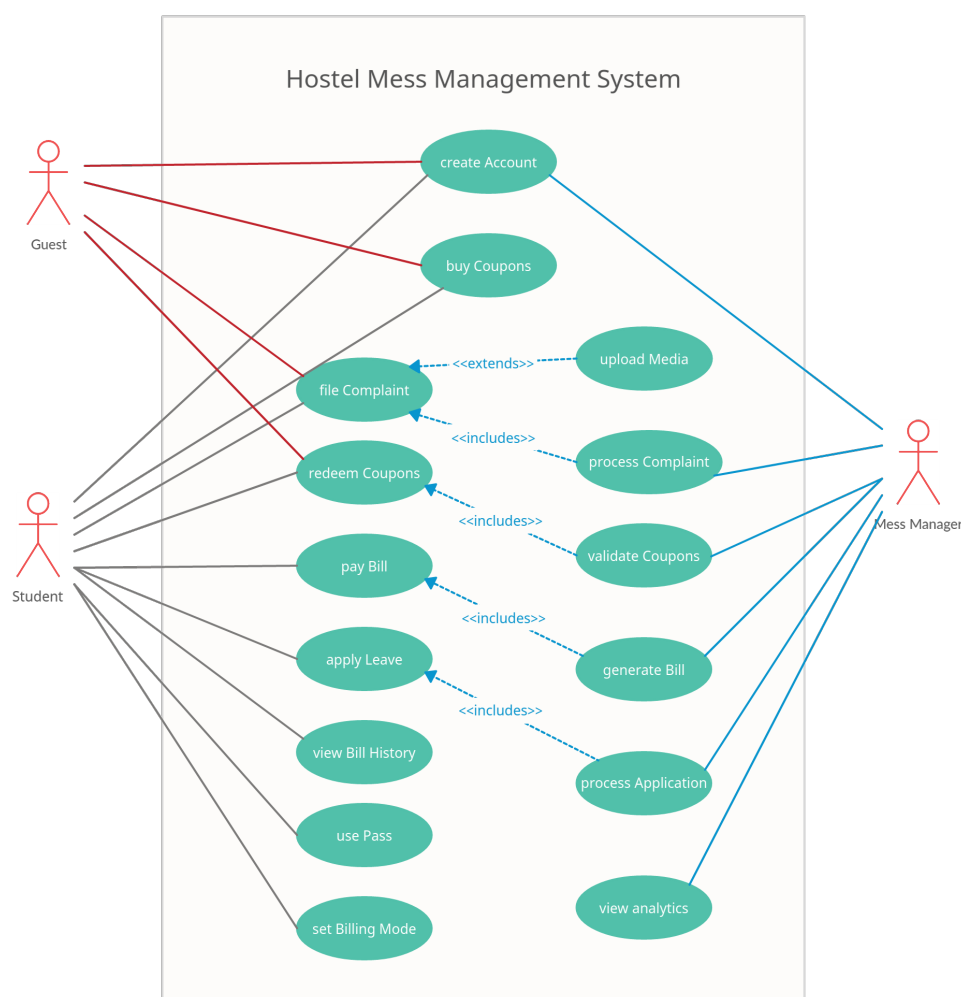


Figure 4.1: Use case diagram

4.2 ER Design

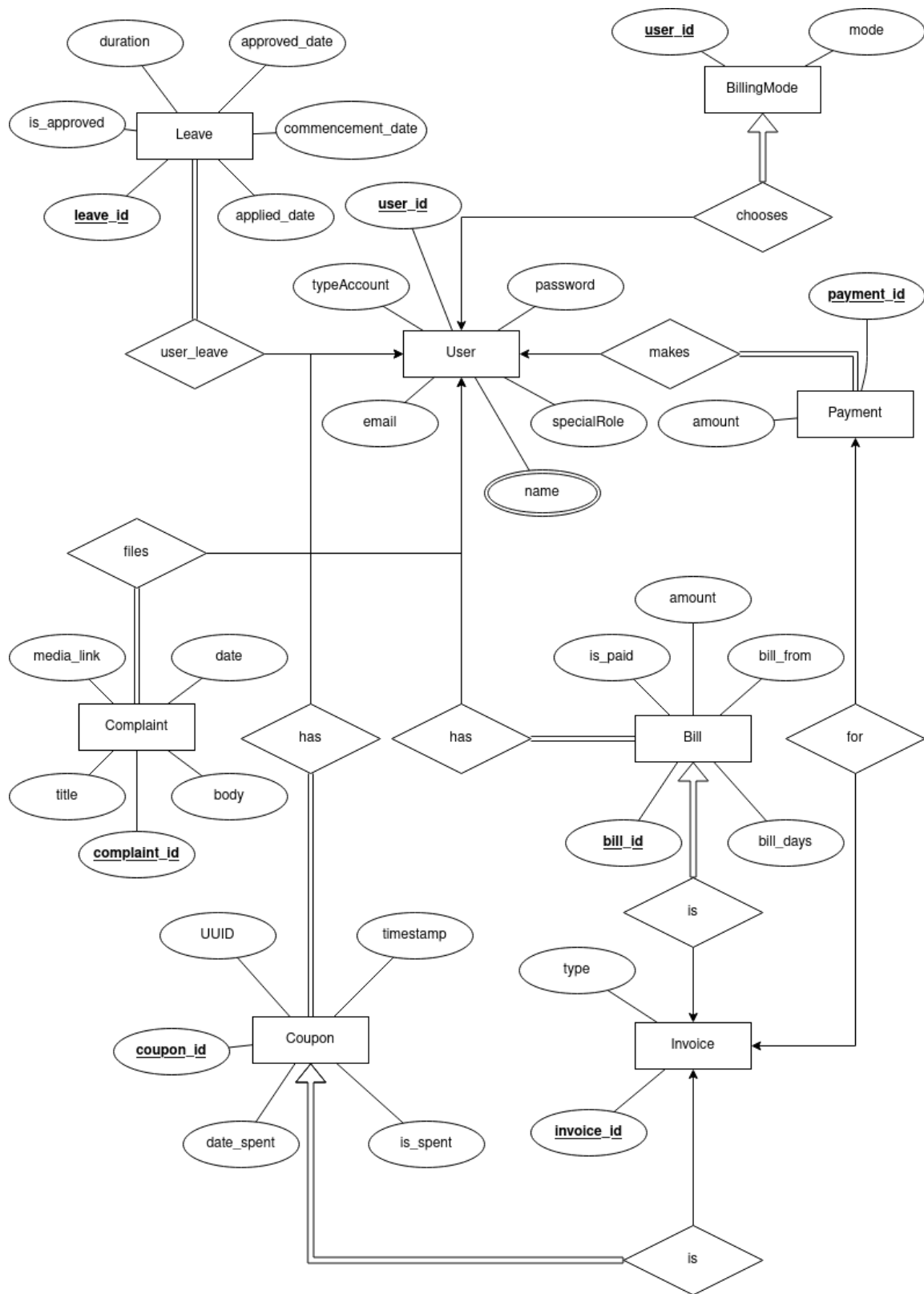


Figure 4.2: ER Diagram

4.3 Database Design

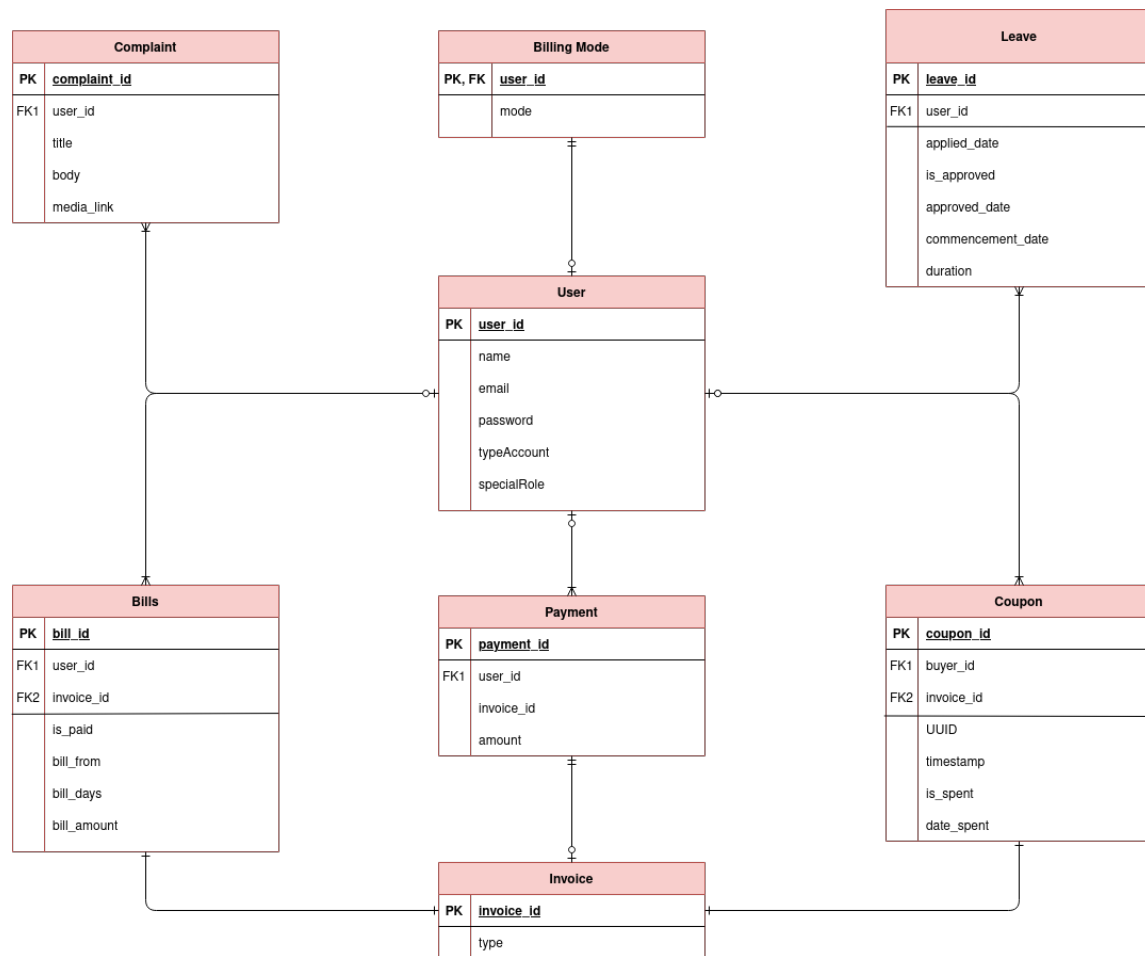


Figure 4.3: Database diagram

4.4 Data Flow Diagram

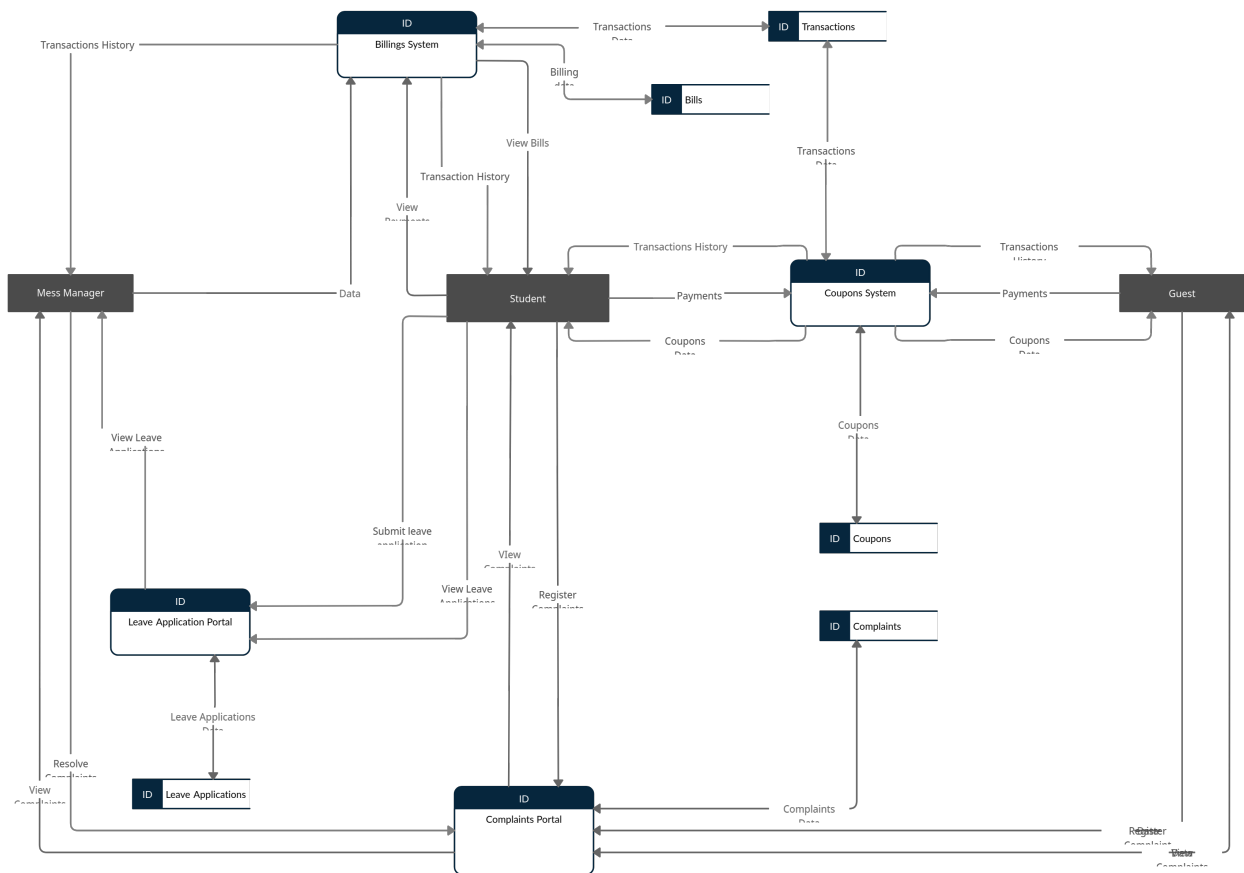


Figure 4.4: Data Flow Diagram

Chapter 5

Implementation

The objective of the project since the beginning has been to build an application which would facilitate the management of a hostel mess. We have used a mobile-first approach for building to allow the software to be instantaneously accessible to the users.

The system follows a modified MVC architectural style where a REST API is coupled with a mobile based application for the user interface.

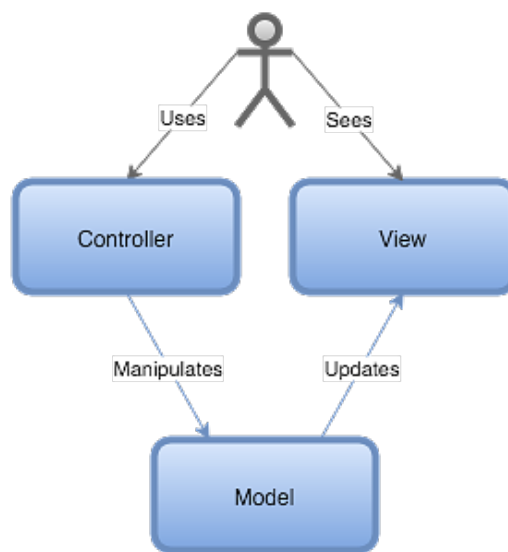


Figure 5.1: Standard MVC pattern

The standard MVC pattern is often used when the views are rendered as templates from the same server which contains the databases and application logic allowing the models, views and controllers to work in conjunction with each other. However, to allow for the software to support multiple different platforms, the views are decoupled from the system and the controllers and models are abstracted away as API endpoints which supports a variety of different 'views'.

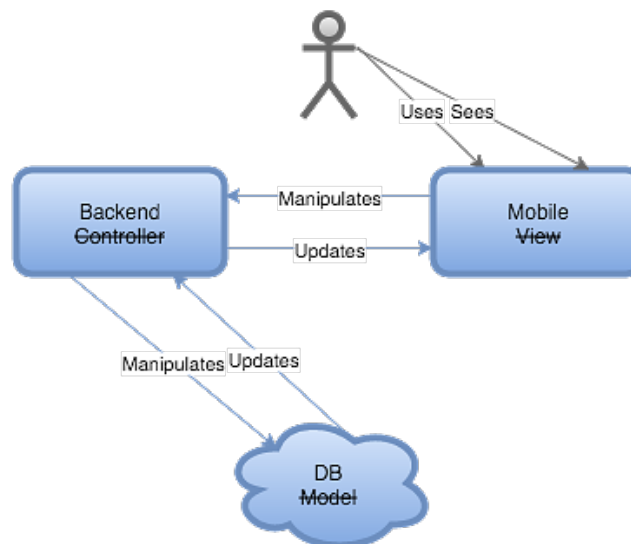


Figure 5.2: Modern MVC pattern

5.1 Technologies, Tools and Frameworks

5.1.1 Database/Model

To handle storage of the essential data of this software, a RDBMS was thought to be a good choice. Thus, PostgreSQL was chosen as the database management system.

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.

5.1.2 Backend/Controller

The application logic or the controlling part of the architecture is splitted between both the frontend and backend. However, a major portion of the logic which handles the databases and important business logic is contained at the backend. For this part of the system, Django was chosen as the framework for development.

Django is an open-source framework for backend web applications based on Python — one of the top web development languages. Its main goals are simplicity, flexibility, reliability, and scalability. Django comes with a "batteries included" philosophy, providing all the essential tools out of the box like ORMs, Http libraries, testing frameworks and others.

5.1.3 Frontend/Views

Although the frontend does share some of the controlling part, it's main function is to provide the 'views' to the users. As this is the part of the software that would be accessible to the users,

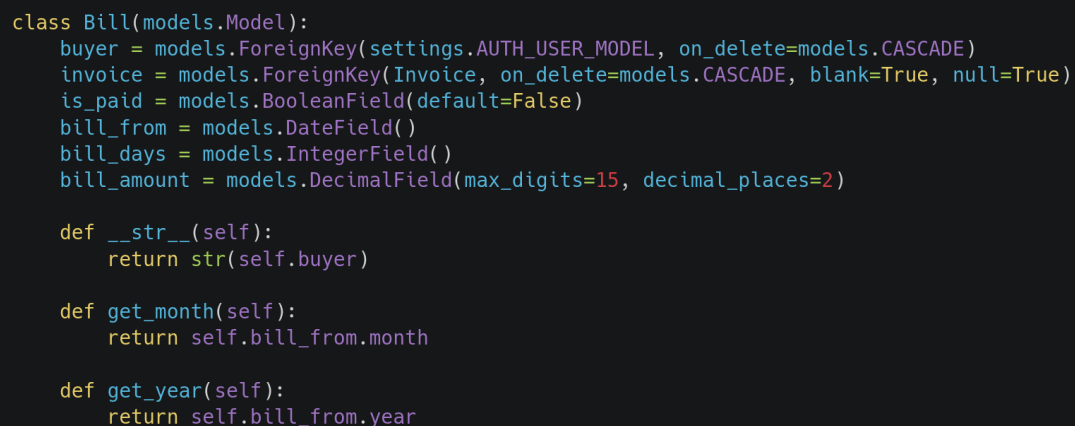
this constitutes an important factor for adoption of the software. To standardise much of the development at this part and to provide cross-platform functionality, React-Native was selected as the framework of choice.

React Native is a framework that allows us to build native mobile apps using JavaScript. Normally, we'd need to program your mobile app using Java (for Android) and Swift/Obj-C (for iOS). React Native removes that requirement, leading to fully functional apps on both platforms in much less time and using just one coding language.

5.2 Process

5.2.1 Setting up and Connecting to database

An instance of PostgreSQL was started on a Ubuntu Linux server. A database was created with appropriate user role. Usually, one would populate the database with all the tables as decided in the database design. However, using raw queries to access and update databases is not scalable as well as very unsafe. Django provides an Object Relational Mapping or ORM which removes this awkward connection between the backend and the database. An ORM allows us to treat the relations as classes and simplifies database communication.



```
class Bill(models.Model):
    buyer = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    invoice = models.ForeignKey(Invoice, on_delete=models.CASCADE, blank=True, null=True)
    is_paid = models.BooleanField(default=False)
    bill_from = models.DateField()
    bill_days = models.IntegerField()
    bill_amount = models.DecimalField(max_digits=15, decimal_places=2)

    def __str__(self):
        return str(self.buyer)

    def get_month(self):
        return self.bill_from.month

    def get_year(self):
        return self.bill_from.year
```

Figure 5.3: A sample Bill model

Since the database is hosted on the same server as the backend, rather than using a TCP/IP connection, the connection to database was done using a UNIX domain socket. This eliminates the need for supplying username and password of the database server to the backend as well as offers performance improvements. One less password to store in the config file.

5.2.2 Dependencies management

For external dependency management, a virtual environment is being used which provides proper versioning of all the required 3rd party libraries. The tool used for managing is *pipenv*. Pipenv is a tool that aims to bring the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world. It solves the issues of setting up the project at different machines with different installed versions of libraries.

5.2.3 Developing Modules

Django provides a module based development process which allows for proper and scalable separation of application logic. The project uses this functionality to break the project into multiple modules.

Users

The users module handles the logic related to management of users. It provides the requirements of user registration, user login, password reset, user profile and choosing billing mode.

auth

POST

/auth/login/

auth_login_create

GET

/auth/logout/

Calls Django logout method and delete the Token object assigned to the current User object.

auth_logout_list

POST

/auth/logout/

Calls Django logout method and delete the Token object assigned to the current User object.

auth_logout_create

POST

/auth/password/change/

Calls Django Auth SetPasswordForm save method.

auth_password_change_create

POST

/auth/password/reset/

Calls Django Auth PasswordResetForm save method.

auth_password_reset_create

POST

/auth/password/reset/confirm/

Password reset e-mail link is confirmed, therefore this resets the user's password.

auth_password_reset_confirm_create

POST

/auth/register/

auth_register_create

POST

/auth/register/verify-email/

auth_register_verify-email_create

GET

/auth/user/

Reads and updates UserModel fields Accepts GET, PUT, PATCH methods.

auth_user_read

PUT

/auth/user/

Reads and updates UserModel fields Accepts GET, PUT, PATCH methods.

auth_user_update

PATCH

/auth/user/

Reads and updates UserModel fields Accepts GET, PUT, PATCH methods.

auth_user_partial_update

PUT

/auth/user/update/mode/{id}/

auth_user_update_mode_update

PATCH

/auth/user/update/mode/{id}/

auth_user_update_mode_partial_update

Figure 5.4: Endpoints provided by Users module

The endpoints are consumed by the frontend to generate views which allow users to create account, login and view their profile.

STUDENT Registration

Name

Email

Password

Confirm Password

REGISTER

Already have an account? [Login here.](#)

Mess Login

Email

Password


LOGIN

(a) Registration view

(b) Login view

←

Your Profile



Prasun Kumar

Student No Special Role

Contact Details

✉ prasun.code@email.com

☎ +91-999-999-1000

🍴 MONTHLY

Switch Billing Mode

COUPON MONTHLY

(c) Profile view

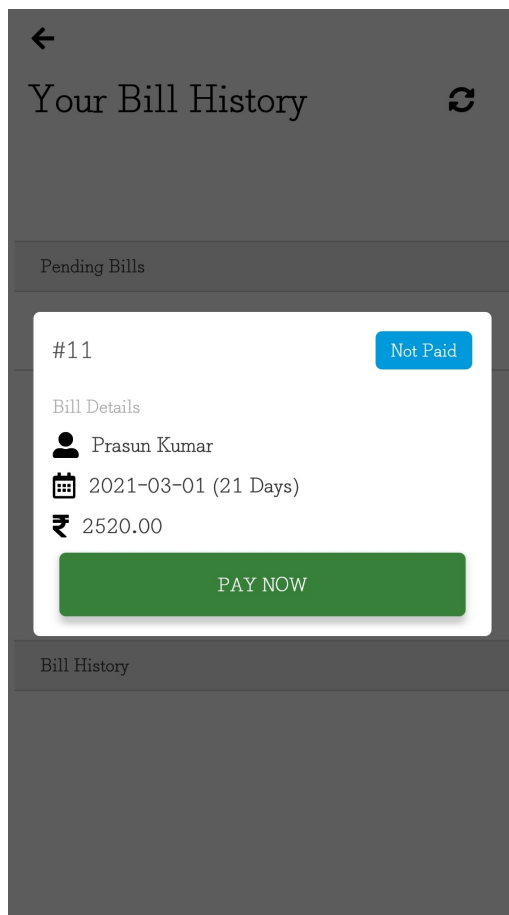
Bills

The bills module handles the generation, displaying and payment of bills. For payment, libraries by Razorpay has been integrated at frontend and backend.

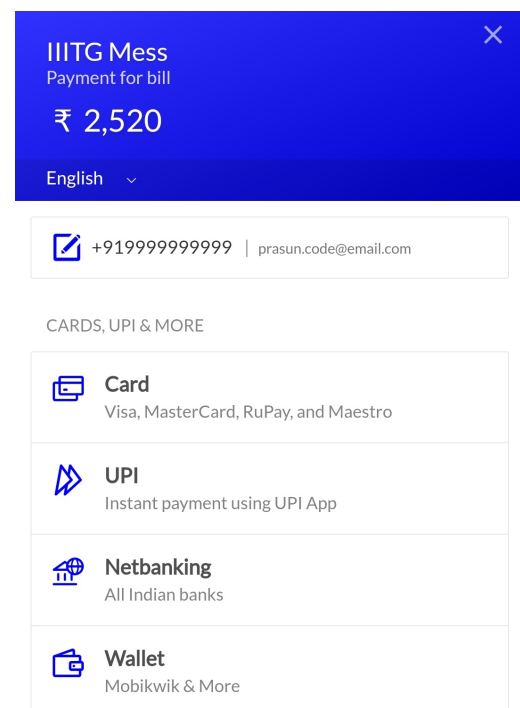
bill			▼
GET	/bill/	bill_list	🔒
POST	/bill/generate/	bill_generate_create	🔒
POST	/bill/pay/confirm/	bill_pay_confirm_create	🔒
POST	/bill/pay/request/	bill_pay_request_create	🔒
GET	/bill/report/	bill_report_list	🔒

Figure 5.6: Endpoints provided by Bill module

The endpoints are consumed by the frontend to generate views which allow users to view bill, pay bill and for caterer to generate bills.



(a) View bills view



(b) Pay bills view

Leaves

The leaves module provides a centralised leave management system. It handles the registering and approving of leaves.

leave			▼
GET	/leave/	leave_list	🔒
POST	/leave/	leave_create	🔒
PUT	/leave/approve/{id}/	leave_approve_update	🔒
PATCH	/leave/approve/{id}/	leave_approve_partial_update	🔒

Figure 5.8: Endpoints provided by Leave module

The endpoints are consumed by the frontend to generate views which allow users to apply for leave and check its status.



Your Leave Requests



Prasun Kumar
#1

Leave Application

Thu Apr 01

Approved

Prasun Kumar
#10

Leave Application

Wed Apr 07

Approved

Prasun Kumar
#15

Leave Application

Thu Apr 15

Approved

(a) Leave status view

What is the reason?

04/15/21 --- 04/15/21

APPLY FOR LEAVE

(b) Apply leave view



Leaves Filed



Prasun Kumar
#10

Leave Request

Wed Apr 07

Approved

Prateek Jain
#11

Leave Request

Sun Apr 11

Approved

Prasun Kumar
#1

Leave Request

Thu Apr 01

Approved

Prateek Jain
#12

Leave Request

(c) Approve leave(caterer side) view

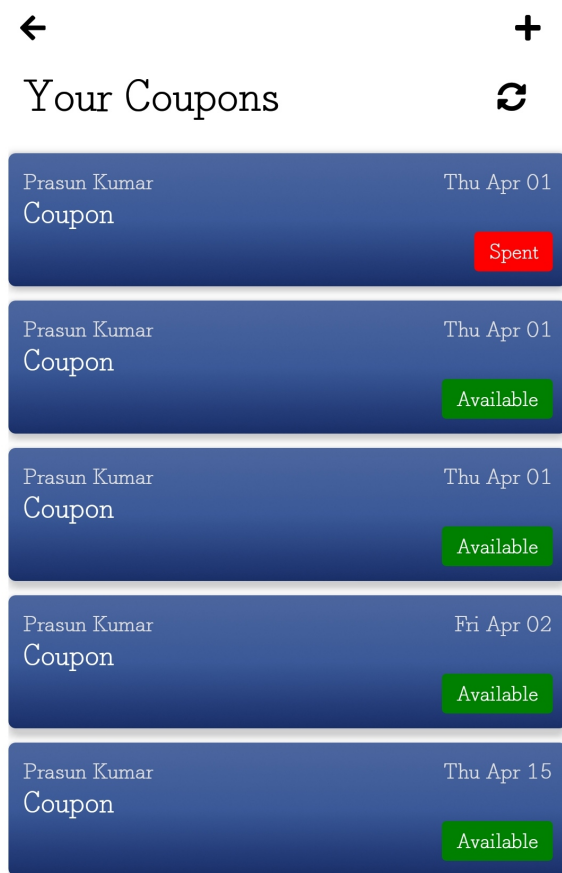
Coupons

The coupons module handles the buying, displaying and redeeming of coupons. For payment, libraries by Razorpay has been integrated at frontend and backend.

complaint			⌵
GET	/complaint/	complaint_list	🔒
POST	/complaint/	complaint_create	🔒
PUT	/complaint/resolve/{id}/	complaint_resolve_update	🔒
PATCH	/complaint/resolve/{id}/	complaint_resolve_partial_update	🔒

Figure 5.10: Endpoints provided by Coupon module

The endpoints are consumed by the frontend to generate views which allow users to view coupon, buy coupon and for caterer to verify coupons.



(a) View coupons view



ID: 1c7e4878-c3e1-4474-a3b2-a2fd28df2d69

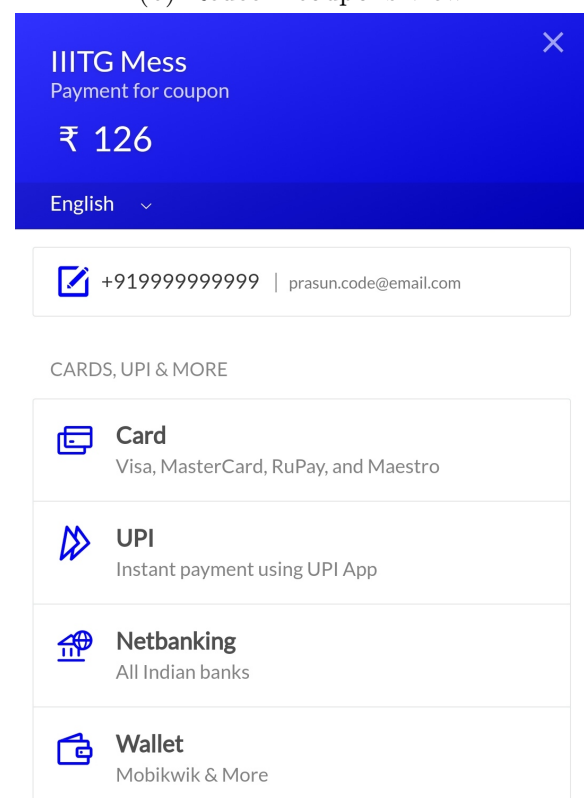
(b) Redeem coupons view

←

Your Coupon History

Total Coupons Purchased	5
Total Coupons Used	1
Total Coupons Unused	4
Total Amount Spent	Rs 630

(c) View coupons history view



(d) Buy coupons view


Complaints

The complaints module provides a centralised complaint management system. It handles the registering and resolving of complaints.


complaint			⌵
GET	/complaint/	complaint_list	🔒
POST	/complaint/	complaint_create	🔒
PUT	/complaint/resolve/{id}/	complaint_resolve_update	🔒
PATCH	/complaint/resolve/{id}/	complaint_resolve_partial_update	🔒

Figure 5.12: Endpoints provided by Complaint module

The endpoints are consumed by the frontend to generate views which allow users to file complaint and check its status.




Your Complaints


Prasun Kumar

#2

Very bad food


Fri Apr 02

Resolved

Prasun Kumar

#3

Bugs in food

Fri Apr 02

Resolved

+

↺

What is your complaint?


Provide a title

Describe your issue


FILE COMPLAINT

(a) Complaint status view

(b) Register complaint view


Fri Apr 02

Resolved


Prasun Kumar

#1

Bad food

Manish

#5


Wed Apr 14

Pending


Mess does not start on time

In the morning, I had to wait for half an hour to get my food

MARK AS RESOLVED

Wed Apr 14

Pending

Manish

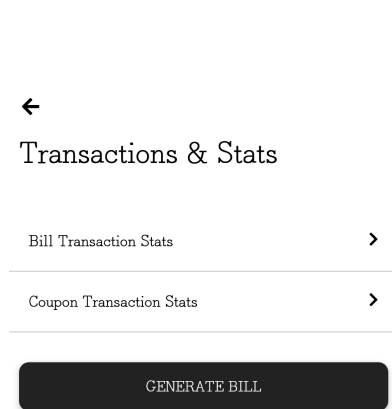
#6

Timing not appropriate

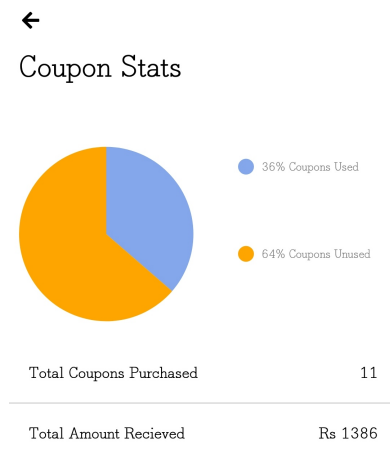
(c) Resolve complaint(caterer side) view

Statistics

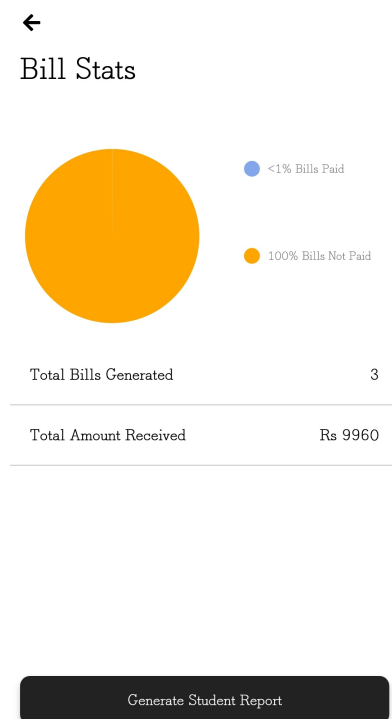
The caterer is also offered statistics through the frontend. These statistics include a detailed billing report, statistics for bills and statistics for coupon transactions.



(a) Transactions statistics view



(b) Coupon statistics view



(c) Bill statistics view

report_0 - Read-only

Read Only - To make changes, save a c...

	A	B	C	D
1	Email	Name	Due Bill	
2	prasun.code@email.com	Prasun Kumar	2520	
3	test@email.com	Developer	3720	
4	prasun.code@gmail.com	Prasun Kumar	3720	
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				

(d) Student Report

5.2.4 Deployment

The backend Django server and PostgreSQL database server were deployed on an AWS EC2 instance. The instance runs a headless Ubuntu Linux operating system and offers easy accessibility over SSH.

The mobile app is distributed via signed APK which can be installed on both Android and iOS devices.

5.2.5 Testing

The testing is carried out by following tools:

- The models are tested through the Django inbuilt testing framework
- For testing of REST APIs, Postman is used

Chapter 6

References

The following references were used during preparation of this document

- Pressman, Roger S. Software Engineering: A Practitioner's Approach, page-136. Eighth edition, McGraw-Hill Education, 2015.
- <https://carbon.now.sh/>
- <https://www.postman.com/>