Software Design (SD)

Group-12
Govinda Rohith Y (CS21BTECH11062)
R Bhargava Ram (CS21BTECH11052)
P Pranav (CS21BTECH11062)
Teja (CS21BTECH11059)

Contents

1 Overview:

1.1 Goal:

The Hostel Room Management System simplifies and automates hostel operations, providing an intuitive interface for HR and users. It efficiently allocates rooms, manages resident profiles, and facilitates smooth check-in/check-out processes. The system also handles room exchange requests, announcements, and complaint resolutions, contributing to an enhanced hostel experience.

1.2 Overall summary:

This document begins with Goal of our software. Next section "Data Flow Diagrams", which consists of four DFD's (With mai and mao). The first one represents the Main DFD which gives rise to three other DFD's. The other three DFD's are students, HR, HO point of view and their respective functionalities. The next section "Structured Charts" consists of four structured charts, which are Main, Student, HR, HO. The Main structured chart gives rise to three other structured charts. The three other structured charts represents different modules associated with them.

Next sections consists of all final modules(With 1-2 lines of Justifications about cohesion) with type, output, coordinate and type of cohesion. Then analysis about different modules based on complex/error prone in different scenarios. Next the expected size of the software in terms of Lines of code(LoC). Finally ending with Interfaces with classes and attributes.

2 Data Flow Diagrams:

2.1 Main DFD:

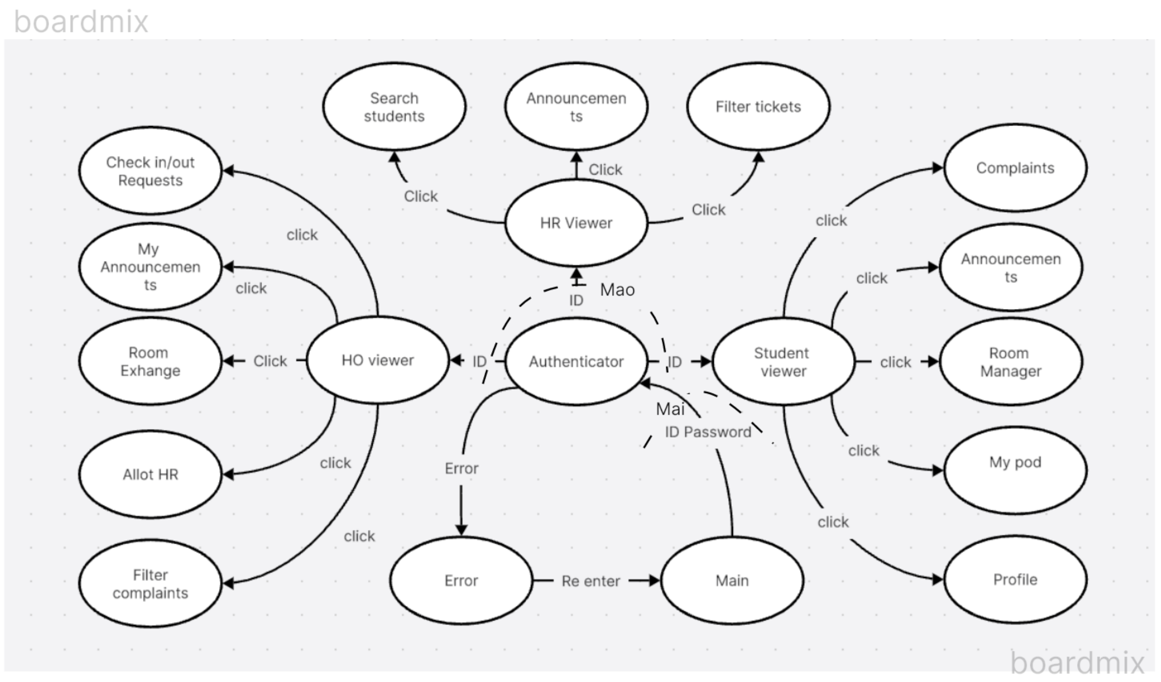

Figure 1: Main DFD

2.2 Student view's DFD:

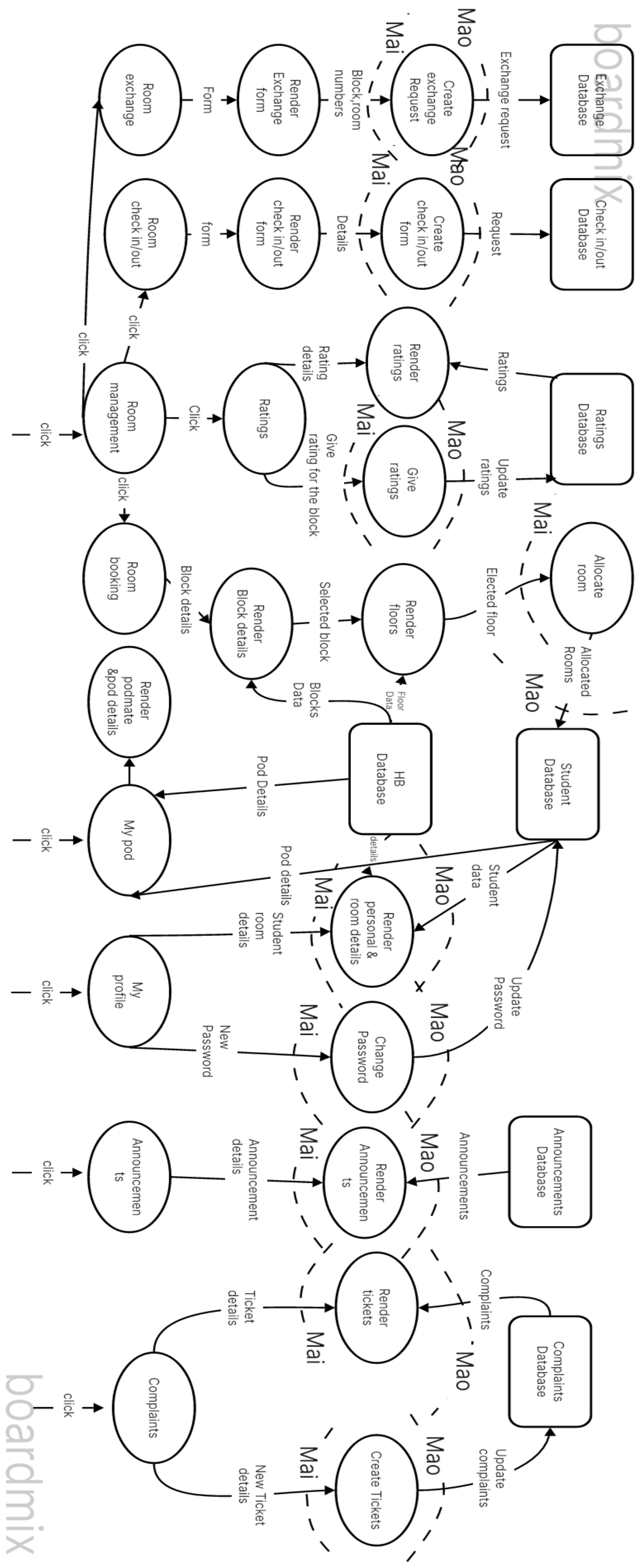

Figure 2: Student DFD

2.3 HR's view DFD:

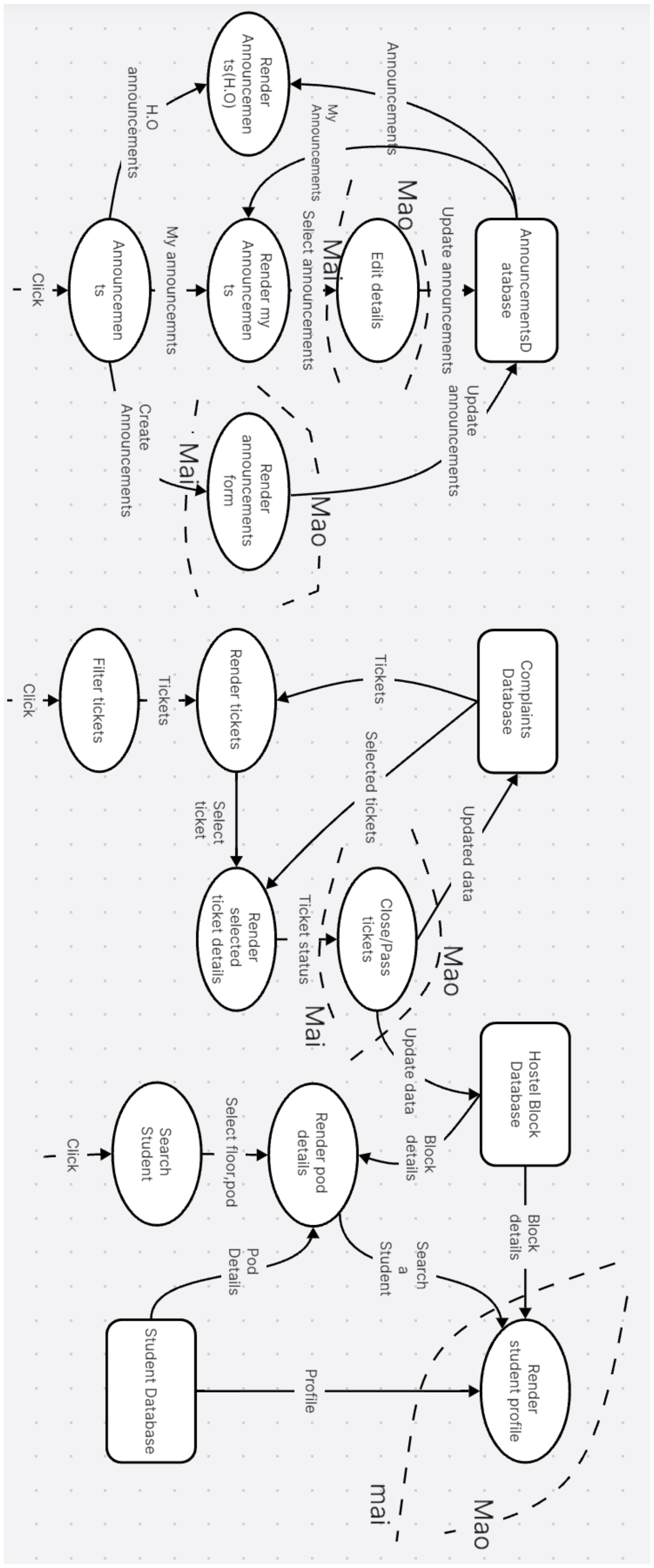

Figure 3: HR's DFD

2.4 HO's view DFD:

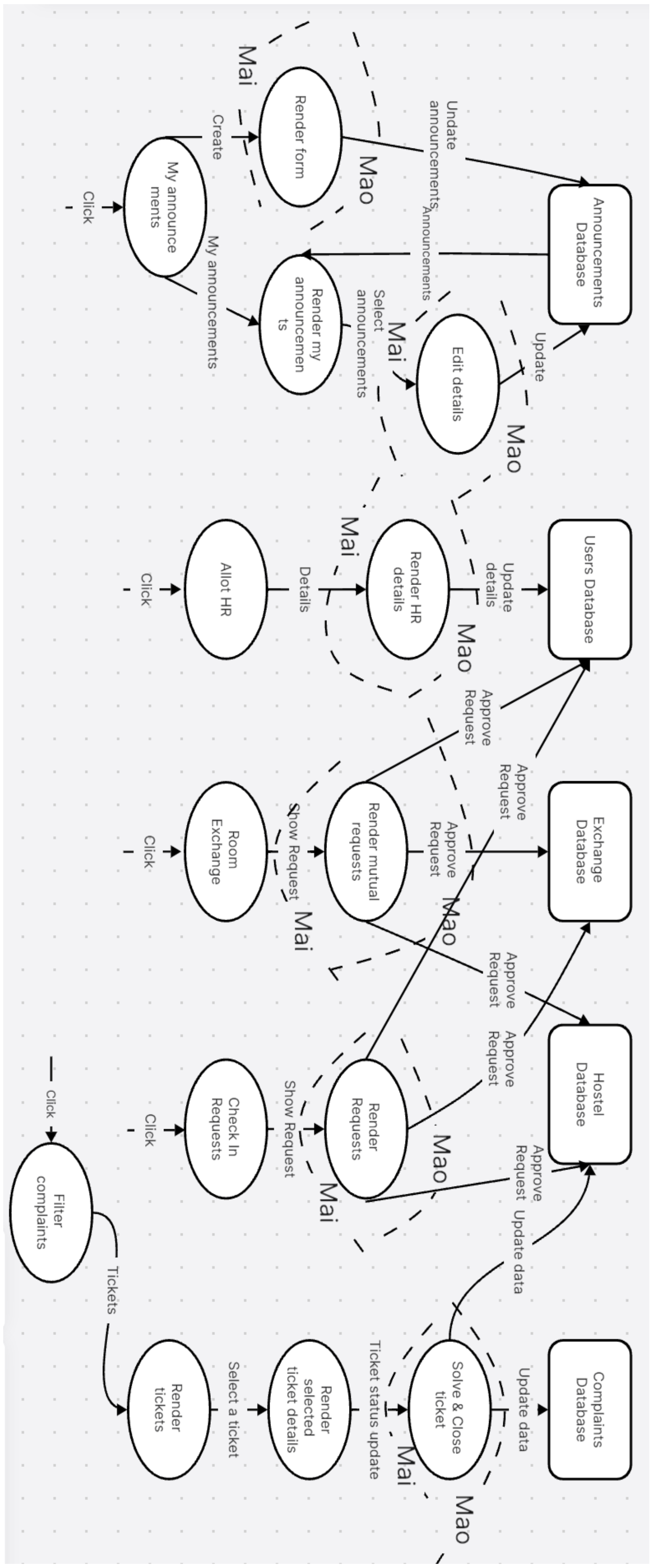

Figure 4: HO's DFD

3 Structured Charts:

3.1 Main:

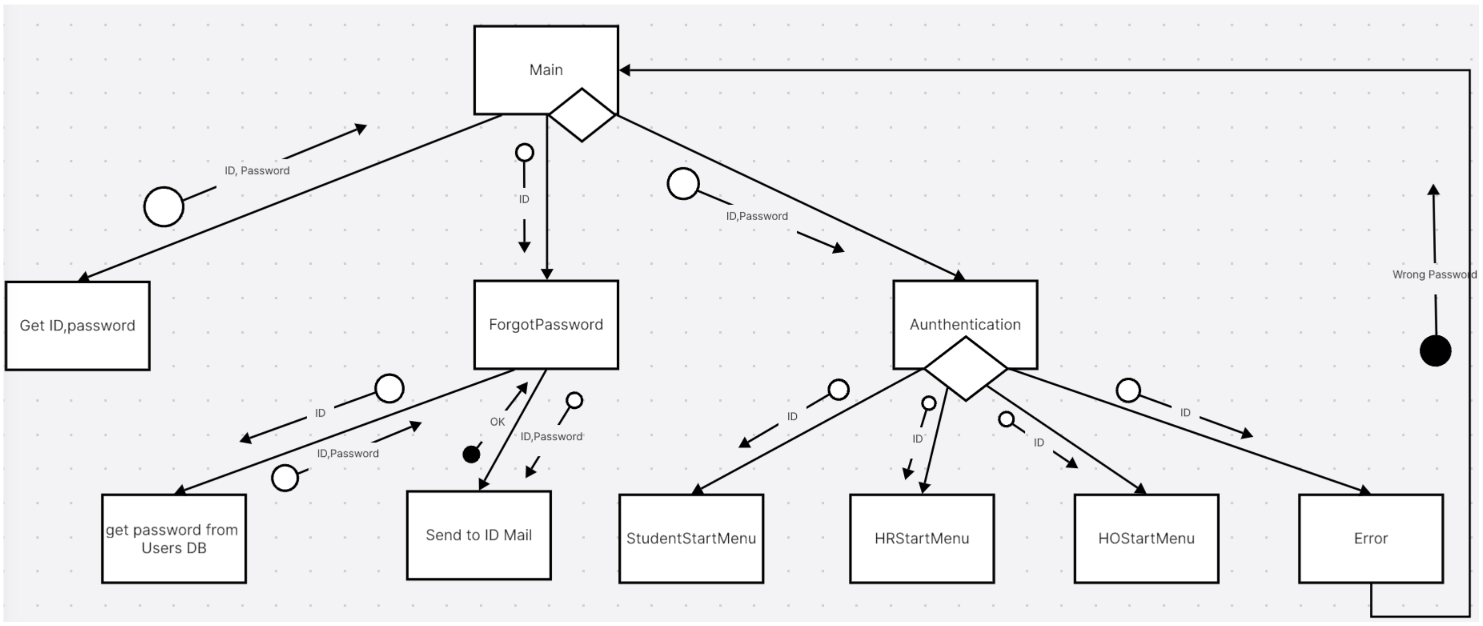

Figure 5: Authentication Page

3.2 Student:

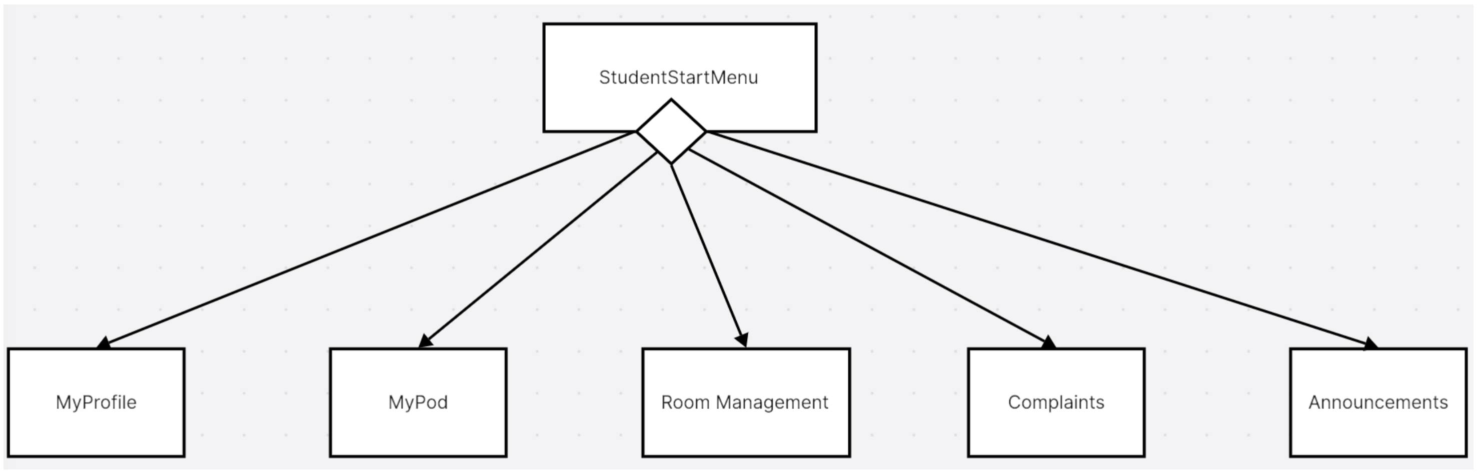

Figure 6: Student Home Page

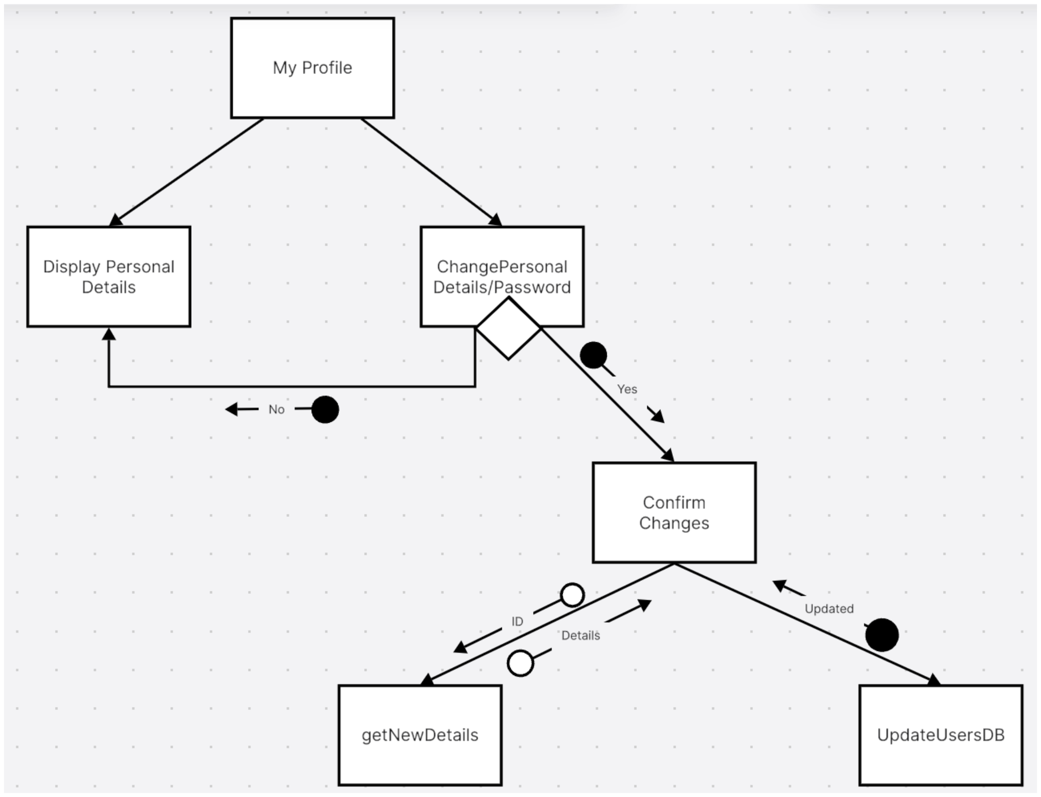

Figure 7: Student Profile

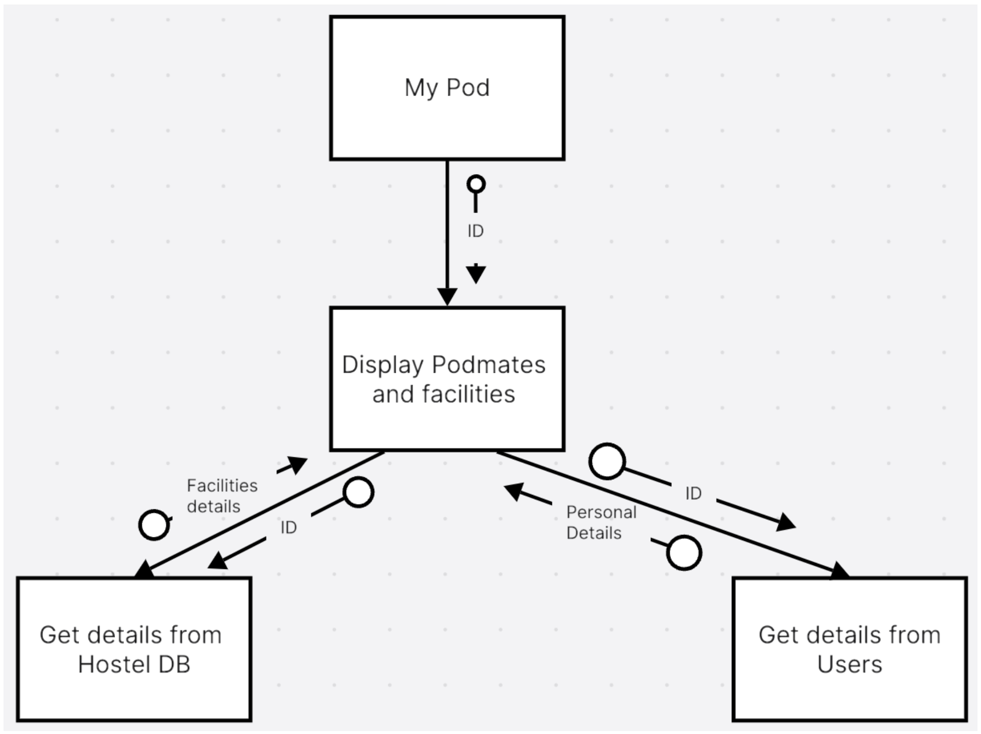

Figure 8: Student MyPod

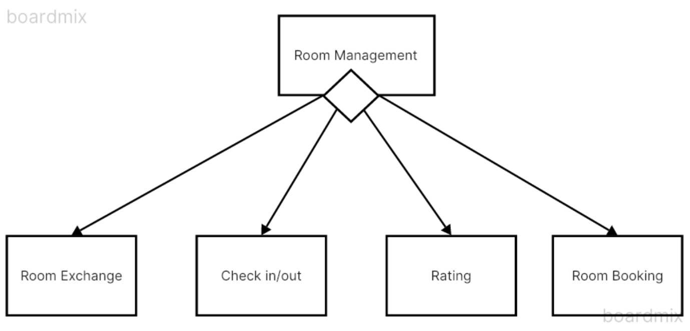

Figure 9: Student Room Management

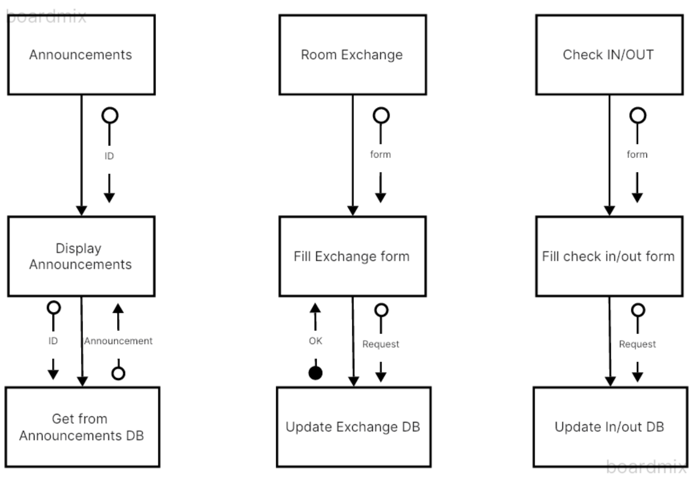

Figure 10: Three Structural charts combined (Announcements,Room Exchange,Check In/Out)

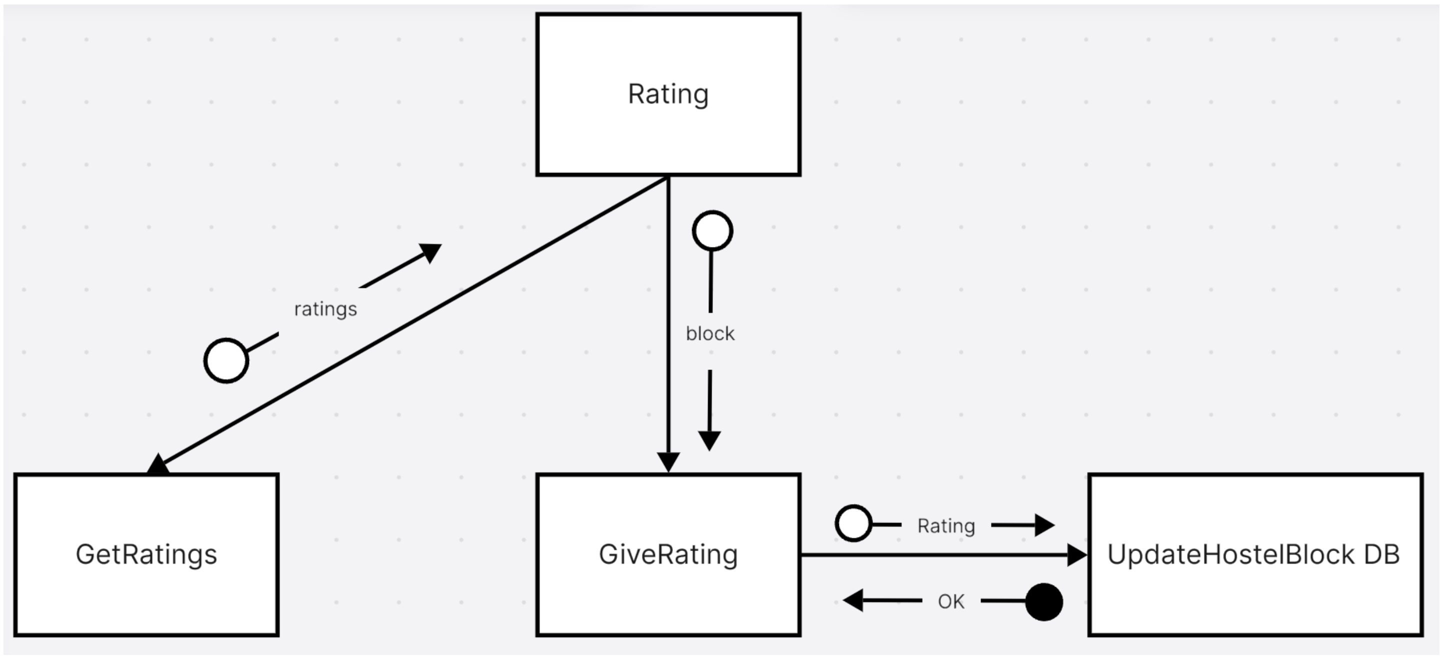

Figure 11: Student Rating

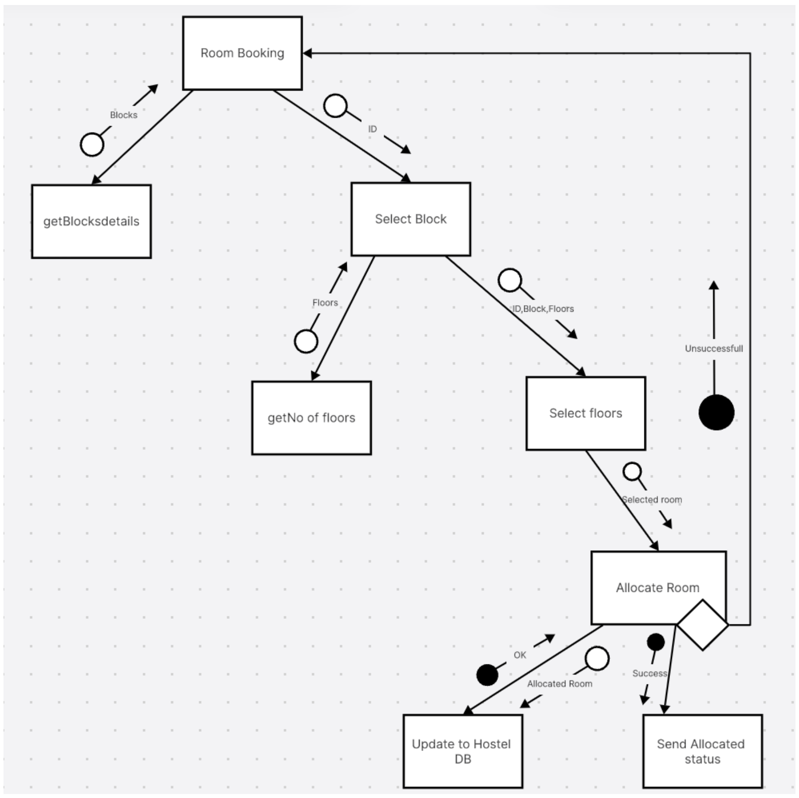

Figure 12: Student Booking

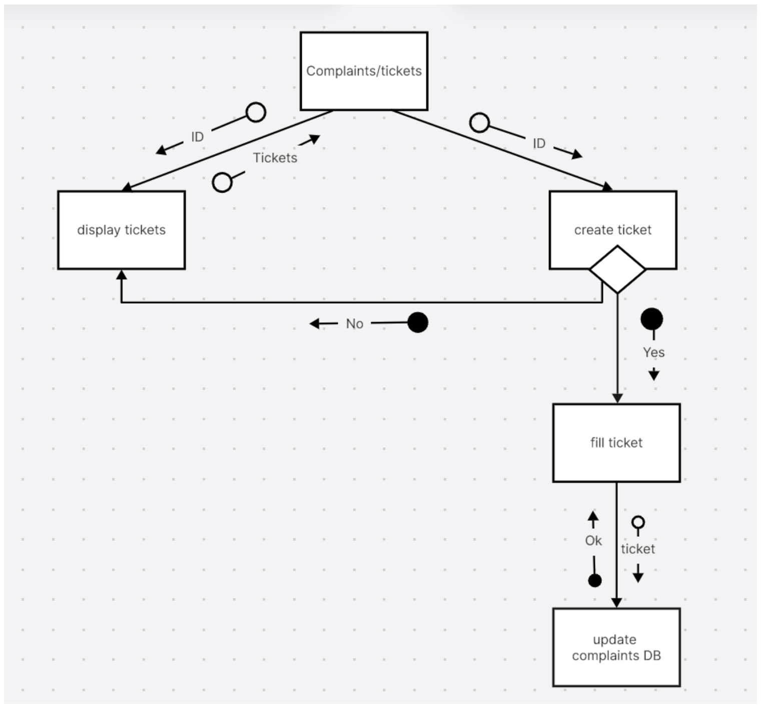

Figure 13: Student Tickets

3.3 HR:

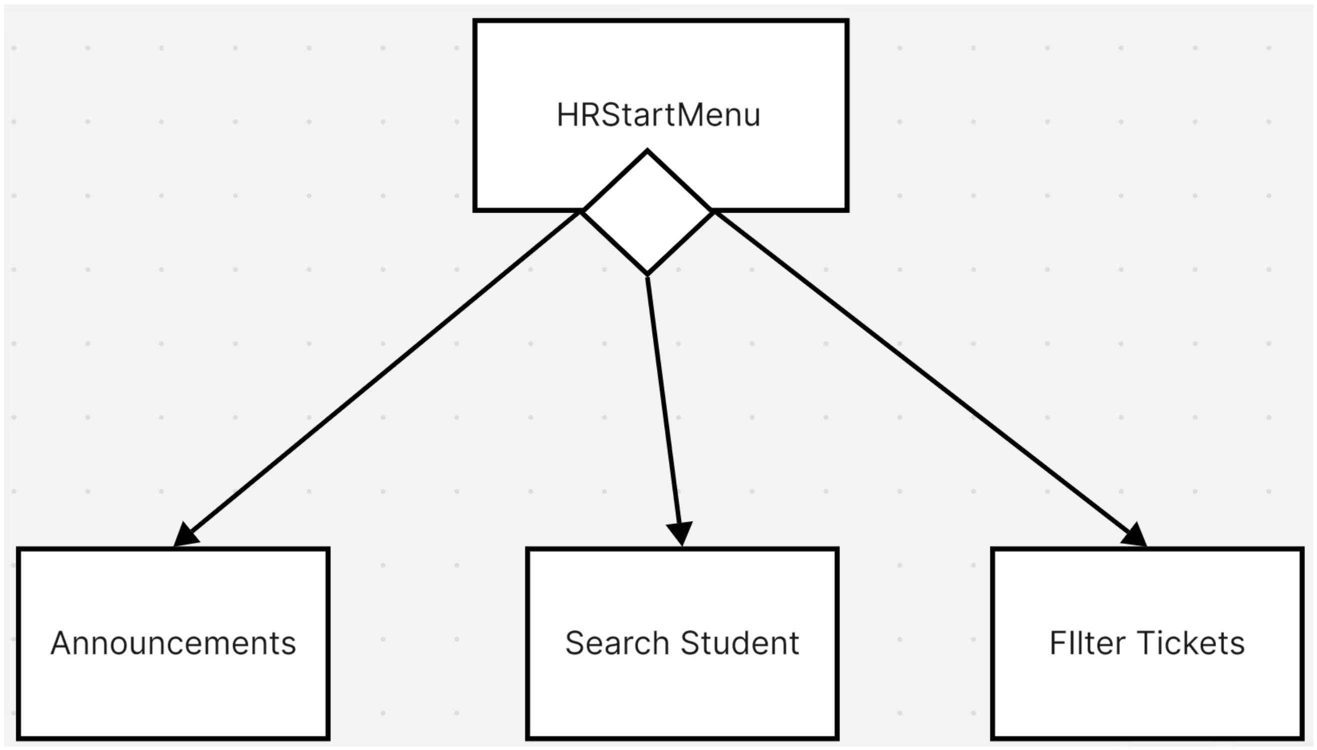

Figure 14: HR Home Page

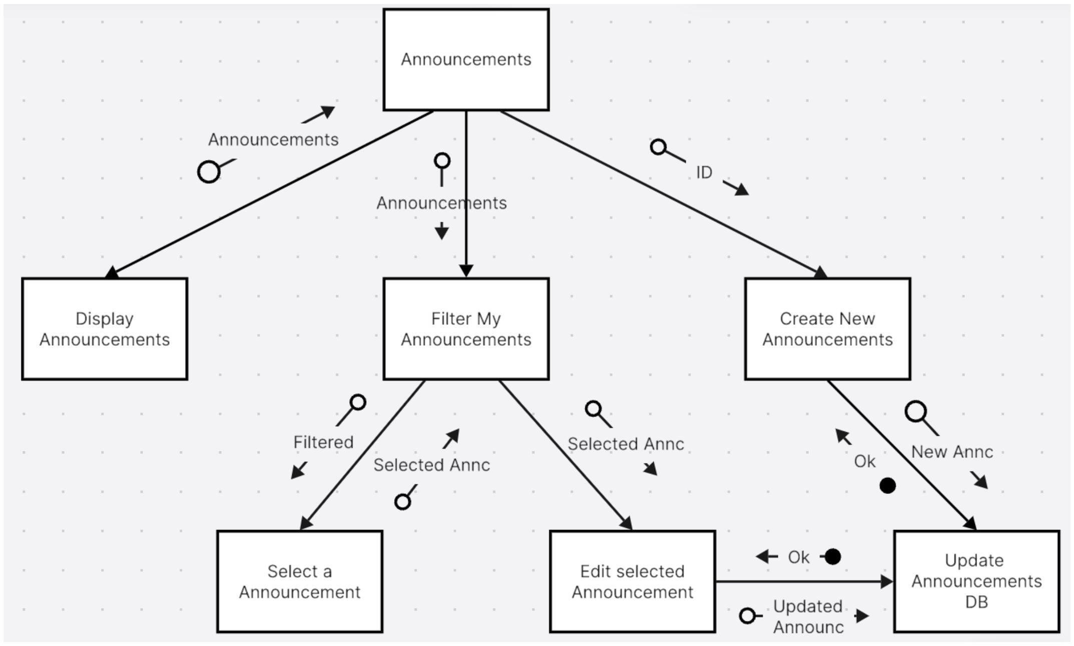

Figure 15: HR Announcements

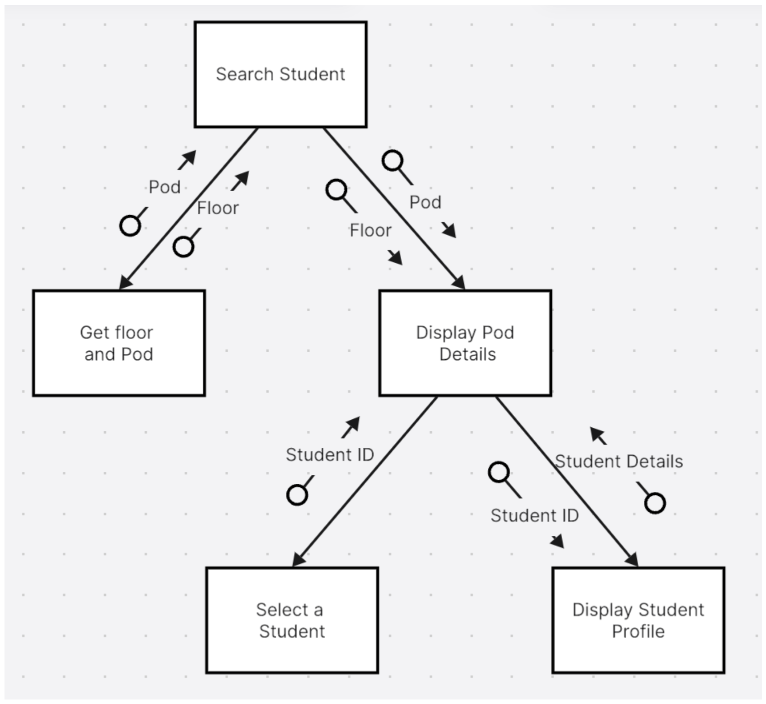

Figure 16: HR's Student Search

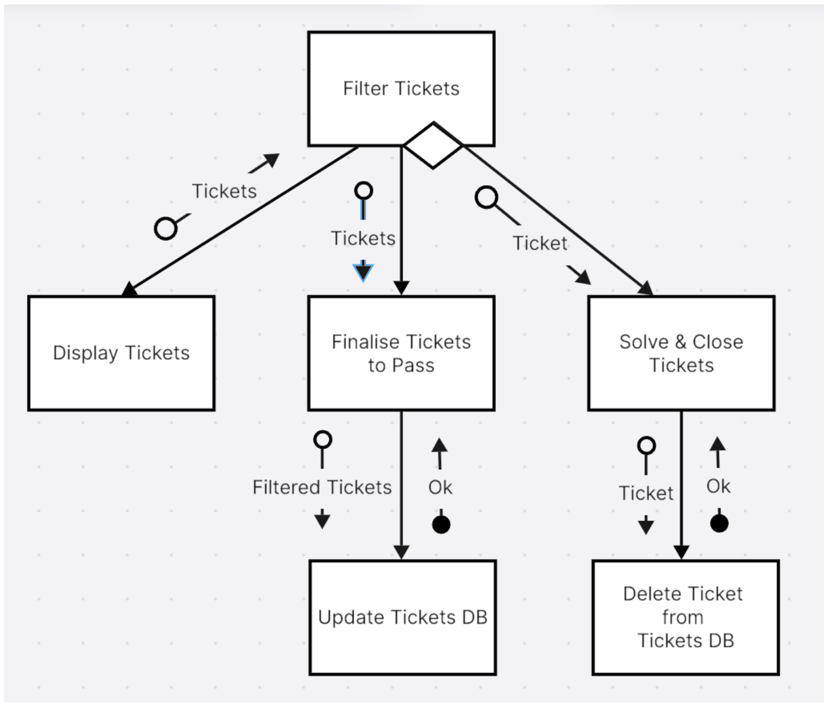

Figure 17: HR Filter tickets

3.4 HO:

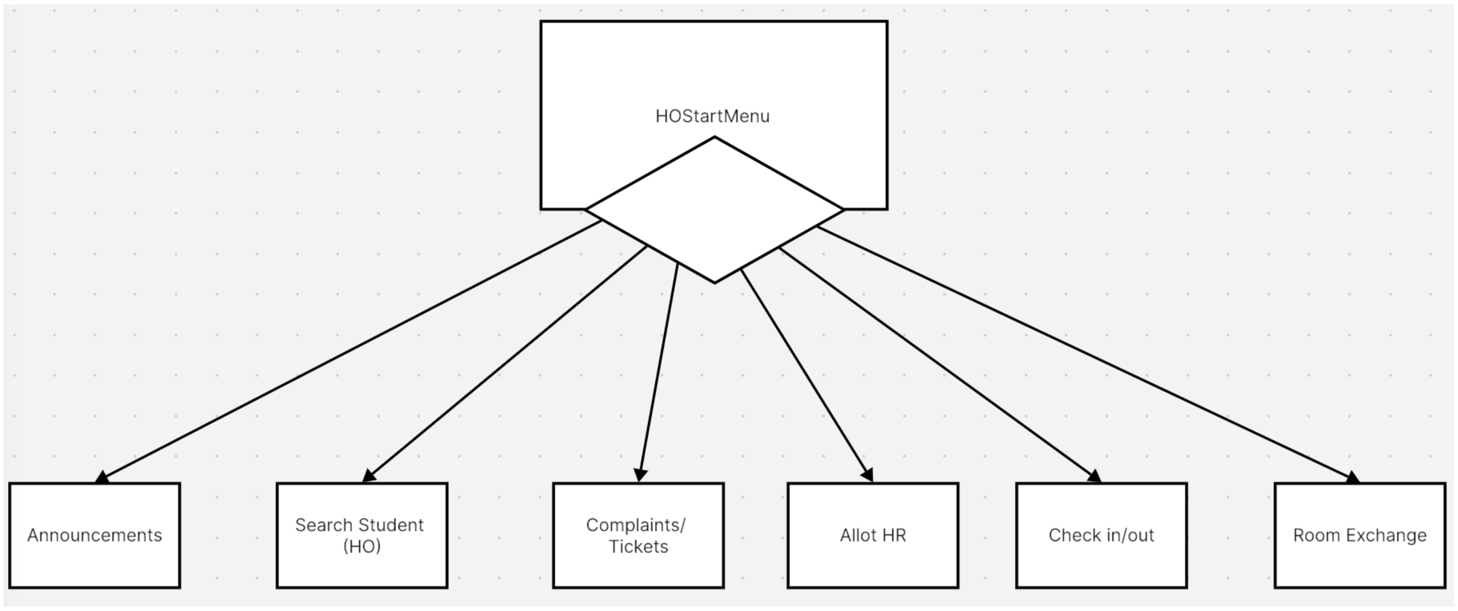

Figure 18: HO Start Page

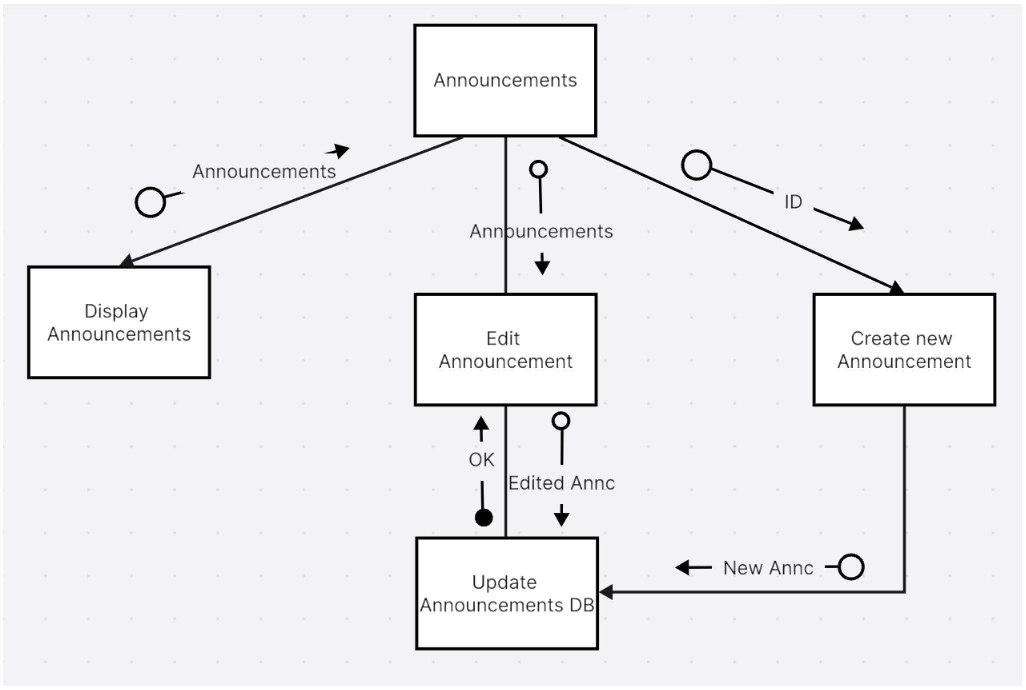

Figure 19: HO Announcements

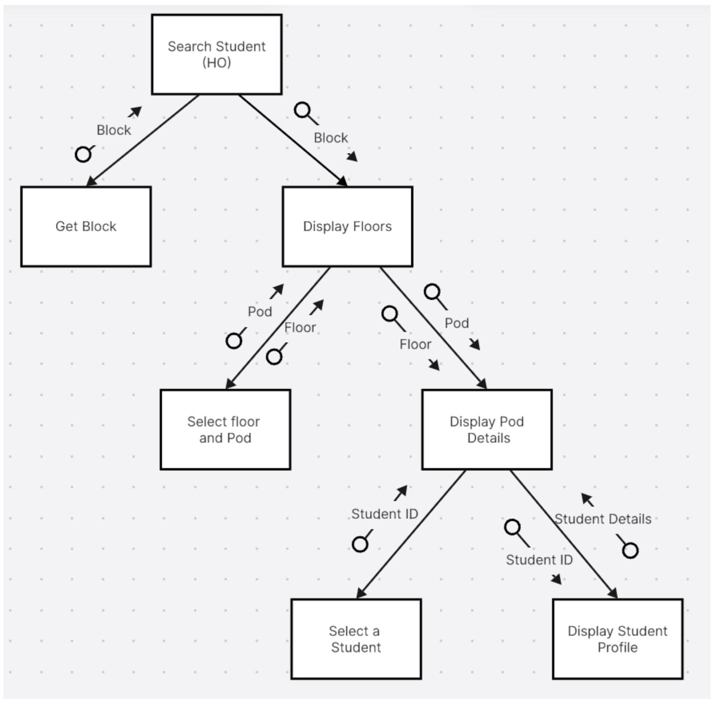

Figure 20: Search for a student by HO

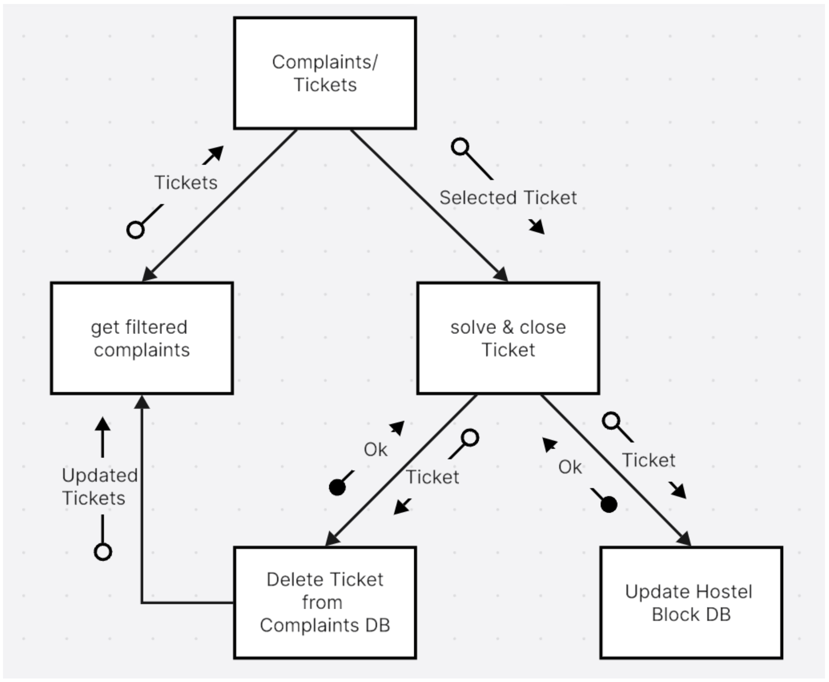

Figure 21: Solve tickets

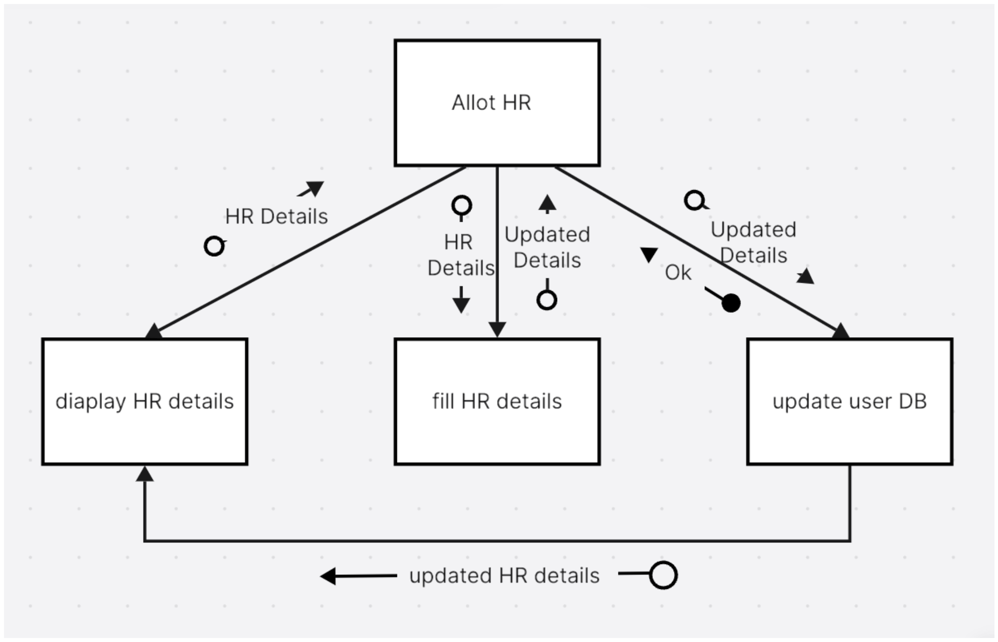

Figure 22: Allot a new HR

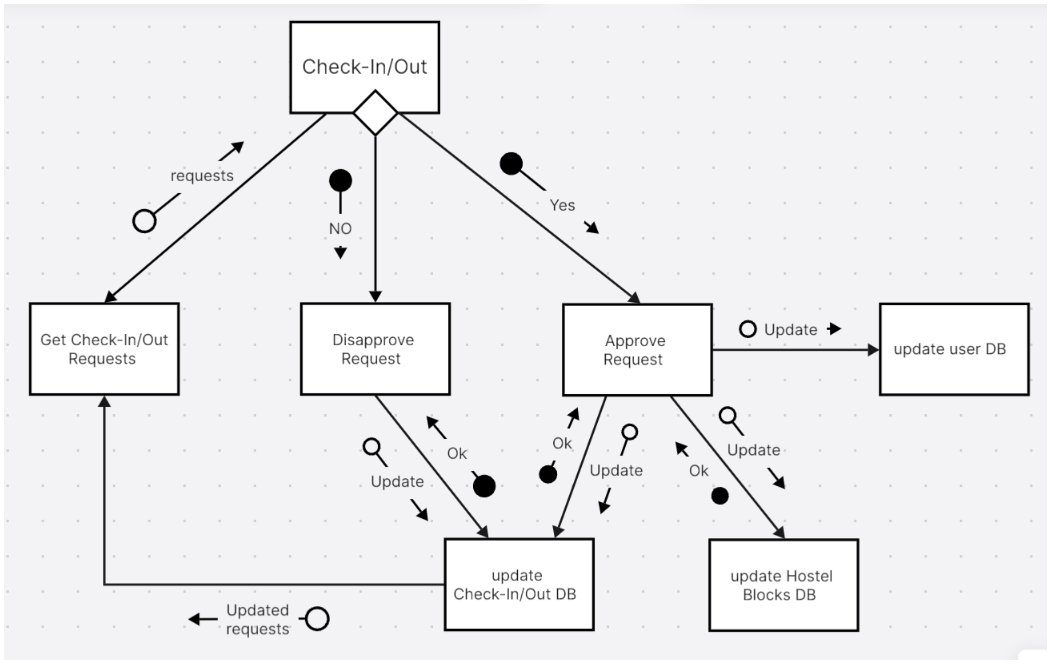

Figure 23: Approve check in/outs

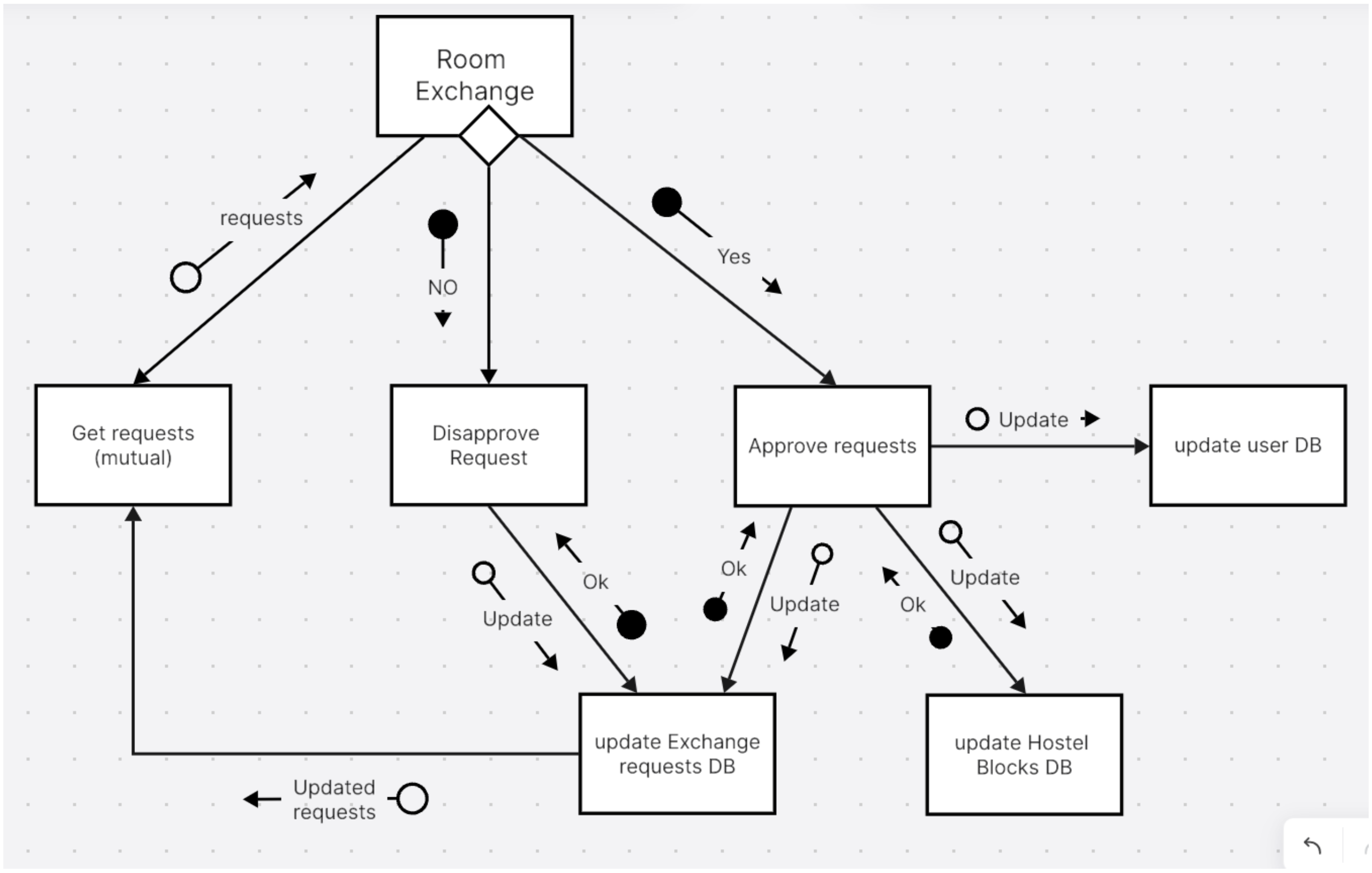

Figure 24: Approve Exchanges

4 Design Analysis:

4.1 Main:

Module Name	Type	Cohesion Type	Estimated Size	Coupling Degree
Main	Coordinate	Sequential	Small	High
Get ID, password	Input	Functional	Small	Low
ForgotPassword	Composite	Functional	Small	Medium
Send to ID Mail	Output	Functional	Medium	Low
StudentStartMenu	Coordinate	Logical	Small	Low
HRStartMenu	Coordinate	Logical	Small	Low
HOStartMenu	Coordinate	Logical	Small	Low
Error	Output	Functional	Medium	Low to Medium
Authentication	Transform	Functional	Large	Medium to High
Get password from Users DB	Input	Functional	Small	Low

4.2 Student:

Module Name	Module Type	Type of Cohesion	Estimated Size	Degree of Coupling
StudentStartMenu	Coordinate	Logical	Small	Medium
MyProfile	Composite	Logical	Small	High
Display Personal Details	Output	Communicational	Medium	Low
ChangePersonalDetails /Password	Coordinate	Logical	Small	Medium
Confirm Changes	Coordinate	Temporal	Small	Low
getNewDetails	Input	Functional	Small	Low
UpdateUsersDB	Transform	Functional	Small	Low
MyPod	Composite	Sequential	Small	Medium
Get details from Hostel DB	Input	Functional	Small	Low
Get details from Users	Input	Functional	Small	Low
Display Podmates and facilities	Output	Functional	Medium	Low to Medium
Room Management	Coordinate	Logical	Small	High
Room Exchange	Composite	Sequential	Small	Medium
Fill Exchange Form	Input	Functional	Medium	Low to Medium
Update Exchange DB	Transform	Functional	Small	Low
Check in/out	Composite	Sequential	Small	High
Fill check in/out form	Input	Functional	Medium	Low to Medium
Update In/out DB	Transform	Functional	Small	Low
Rating	Composite	Logical	Small	Medium
GetRatings	Output	Communicational	Medium	Low
GiveRating	Input	Functional	Medium	Low to Medium
UpdateHostelBlock DB	Transform	Functional	Small	Low
Room Booking	Composite	Sequential	Small	High
getBlocksDetails	Output	Communicational	Medium	Medium
Select Block	Input	Functional	Small	Low
getNoOfFloors	Output	Communicational	Small	Low
Select floors	Input	Functional	Small	Medium
Allocate Room	Transform	Functional	Large	Medium
Update to Hostel DB	Transform	Functional	Small	Low
Send Allocated status	Output	Functional	Small	Low
Complaints	Composite	Logical	Small	High
display tickets	Output	Communicational	Large	Low
create ticket	Coordinate	Logical	Small	Medium
fill ticket	Input	Functional	Medium	Medium
update complaints DB	Transform	Functional	Small	Low
Announcements	Composite	Sequential	Small	Medium
Display Announcements	Output	Communicational	Medium	Low to Medium
Get from Announcements DB	Input	Functional	Small	Low

4.3 HR:

Module Name	**Module Type**	**Type of Cohesion**	**Estimated Size**	**Degree of Coupling**
HRStartMenu	Coordinate	Logical	Small	High
Announcements	Composite	Logical	Small	High
Display Announcements	Output	Communicational	Large	Low
Filter My Announcements	Output	Communicational	Large	Medium
Select a Announcement	Transform	Functional	Small	Low
Create New Announcements	Input	Functional	Medium	Low
Edit selected Announcement	Input	Functional	Medium	Medium
Update Announcements DB	Transform	Functional	Small	Low
Search Student	Composite	Sequential	Small	Medium
Get floor and Pod	Input	Functional	Small	Low
Select a Student	Input	Functional	Small	Low
Display Pod Details	Output	Communicational	Large	Medium
Display Student Profile	Output	Communicational	Medium	Low
Filter Tickets	Composite	Logical	Small	High
Display Tickets	Output	Communicational	Large	Low
Finalise Tickets to Pass	Input	Functional	Large	Medium
Solve & Close Tickets	Input	Functional	Medium	Medium
Update Tickets DB	Transform	Functional	Small	Low
Delete Ticket from Tickets DB	Transform	Functional	Small	Low

4.4 HO:

Module Name	Module Type	Type of Cohesion	Estimated Size	Degree of Coupling
HOStartMenu	Coordinate	Logical	Small	High
Announcements	Composite	Logical	Small	High
Create new Announcement	Input	Functional	Small	Low
Display Announcements	Output	Communicational	Medium	Medium
Edit Announcement	Input	Functional	Medium	Medium
Update Announcements DB	Transform	Functional	Small	Low
Search Student	Composite	Sequential	Small	High
Get Block	Input	Functional	Medium	Low
Select floor and Pod	Input	Functional	Small	Low
Select a Student	Input	Functional	Small	Low
Display Floors	Output	Communicational	Small	Medium
Display Pod Details	Output	Communicational	Large	Medium
Display Student Profile	Output	Communicational	Medium	Low
Complaints/Tickets	Composite	Sequential	Small	Medium to High
get filtered complaints	Output	Communicational	Medium	Low
Delete Ticket from Complaints DB	Transform	Functional	Small	Low
Update Hostel Block DB	Transform	Functional	Large	Low
solve & close Ticket	Input	Functional	Medium	Medium
Allot HR	Composite	Sequential	Small	High
display HR details	Output	Communicational	Medium	Low
fill HR details	Input	Functional	Medium	Low
update user DB	Transform	Functional	Small	Low to Medium
Check in/out	Composite	Logical	Small	High
Get Check-In/Out Requests	Output	Communicational	Medium	Low
Update user DB	Transform	Functional	Small	Low
Update Check-In/Out DB	Transform	Functional	Small	Low to Medium
Update Hostel Blocks DB	Transform	Functional	Small	Low
Approve Request	Input	Functional	Small	Medium
Disapprove Request	Input	Functional	Small	Medium
Room Exchange	Composite	Logical	Small	High
Get requests (mutual)	Output	Communicational	Large	Low
Update Exchange requests DB	Transform	Functional	Small	Low to Medium

4.5 Justification for Cohesion Type:

4.5.1 Logical Cohesion:

- **Definition:** When elements are grouped logically that perform similar tasks or functions, but are not related to each other in any other way. For example, in "MyProfile" module we can view, edit, etc..
- **Example:** Also decision point in the "Check-In/Out" Module that leads to either "Disapprove Request" or "Approve Request" could imply logical cohesion at a higher process level.

4.5.2 Temporal Cohesion:

- **Definition:** Elements are related by the fact that they are processed at the same time.
- **Example:** "Confirm Changes" Module represents temporal cohesion, because after the user makes changes, this module confirms those changes. The activities within this module are related to the point in time when the changes are made.

4.5.3 Communicational Cohesion:

- **Definition:** When elements are grouped because they operate on the same data (not necessarily performing the same operation).
- **Example:** All the display Modules like "Display Announcements" represents Communicational cohesion, because it fetches and displays announcements from same database.

4.5.4 Sequential Cohesion:

- **Definition:** This is a stronger form of cohesion where the output from one part of the module is the input to another part.
- **Example:** "Main" Module represents Sequential Cohesion, because there is a clear sequential flow from collecting ID and password, authenticating them, and then navigating to the appropriate start menu or error handling.

4.5.5 Functional Cohesion:

- **Definition:** The strongest form of cohesion, functional cohesion occurs when every element of the module is essential to the performance of a single function.
- **Example:** "Delete Ticket from Tickets DB" Module represents Functional Cohesion, because it is solely responsible for deleting ticket query. It is focused on a single task.

4.6 Justification for degree of coupling:

1. The ones which are of Low degree of coupling are the modules which are not dependent on other modules.
2. The ones which are of Low to Medium degree of coupling are the modules which are dependent on one module.

3. The modules which are dependent on greater than two modules are classified as medium or large degree of coupling, but the first level factored modules which are in bold text are classified as High or Medium degree of coupling.

4. For some of the first level factored modules like for example complaints/tickets in HO section; search for student in HR section; Announcements, Rating sections in Students section where it depends on very few modules like 2 modules, then it is classified as Medium degree of coupling.

4.7 Total Number of different types of Modules:

Type	Input	Output	Coordinate	Transform	Composite
Main	2	2	4	1	1
Student	10	8	7	7	8
HR	6	5	1	4	3
HO	9	8	1	8	6
Total	27	23	13	20	19

Table 1: Count of different types

4.8 Complex or Error prone:

1. **In Input:** From Input perspective, the process to take input (Block and floor no) is more complex and error prone(Room Booking Module). This is because if there are x number of rooms in a floor of given block. If more than x users are trying to book the room, then the system should not allow the user to book the room. This is a complex process and error prone due to syncronization issues.

2. **In transformation:** From transformation perspective, the complaint raise and close ticket modules are more complex and error prone. This is because the system should be able to handle the complaints raised by the users and it should modify different facilitiy status accordingly.

3. **In Output:** From Output perspective, the process to display the Room Exchange requests (Render mutual requests) is more complex and error prone. This is because the system should be able to display the mutual requests of the users and we should handle the timeout issues. If mutual requests timeout, then the system should not show the requests.

4.9 Top-3 Modules:

1. **In terms of Fan-IN:** Display Personal Details , display tickets, update announcements. All these modules have highest fan-in number that is '2'.

2. **In terms of Fan-OUT:** HO-startmenu-6, StudentStartMenu-5, Room Management-4.

3. **In terms of Fan-IN+ Fan -OUT:** HO-startmenu-7(1+6), StudentStartMenu - 6(1+5), Room Management-5(1+4).

4.10 Expected size in terms of LoC:

Note: For small sized module would be around 25 LoC, Medium sized module would be around 50 LoC, Large sized module would be around 100 LoC.

User	Small Size	Medium Size	Large Size	Total
Main	7	2	1	**375**
Student	27	9	2	**1325**
HR	10	4	5	**950**
HO	20	9	3	**1250**
Total	**66**	**24**	**11**	**3900**

Table 2: Total LoC count=375+1325+950+1250=3900

Type of Module	Small Size	Medium Size	Large Size	Total LoC
Input	15	11	1	1025
Output	3	13	7	1425
Coordinate	11	0	0	275
Transform	17	0	3	725
Composite	18	0	0	450
Total	**64**	**24**	**11**	**3900**

Total expected size of Our Hostel Room Management System Software is around 3900 LoC.

5 Detailed Design Specification:

5.1 Main:

```
class GetIDPassword {
  String userID;
  String password;

  public void collectLoginCredentials() {
      // collect the login credentials
   }
  private boolean validateInputFormat() {
      // validate the input format
  }
}
```

```
class ForgotPassword {
  String userID;

  public void processForgotPassword() {
      // process the forgot password request
  }

  private void sendPasswordReset() {
      // send password reset link to the user
  }
}
class SendToIDMail {
  String email;
  String message;

  public void sendEmail() {
      // send email to the user
  }
}
class UserStartMenu {
  public void displayMenu() {
      // display menu and route to the selected option
  }
}
class Authentication {
  String userID;
  String password;

  public boolean authenticateUser() {
      // authenticate the user
  }
  private boolean checkCredentials() {
      // check if the credentials are valid
  }
}
class Error {
  public void displayErrorMessage() {
      // display error message
  }
}
```

5.2 Student:

Student StartMenu:

```
public class startmenu {
  // This is the main menu for the student
}
```

```
public class Myprofile {
  // we can view Myprofile here
}
public class Mypod {
  // we can view Mypod here
}
public class RoomManagement {
  // we can do RoomManagement here
}
public class Complaints {
  // we can fill Complaints here
}
public class Announcements {
  // we can see the announcements here
}
```

Student View:

```
public class MyPod {
    private HostelDB hostelDB;
    private UserDetails userDetails;

    public MyPod(HostelDB hostelDB, UserDetails userDetails) {
        this.hostelDB = hostelDB;
        this.userDetails = userDetails;
    }

    public void displayPodInformation(int userId) {
        PodDetails podDetails = hostelDB.getFacilitiesDetails(userId);
        User user = userDetails.getUserDetails(userId);
        new PodDisplay().display(podDetails, user);
    }
}
public class HostelDB {
    public PodDetails getFacilitiesDetails(int userId) {
        // return facilities details
        return new PodDetails();
    }
}
public class UserDetails {
    public User getUserDetails(int userId) {
        // return user details
        return new User();
    }
}
```

```
public class PodDisplay {
    public void display(PodDetails podDetails, User user) {
        // display pod details
    }
}
```

Announcements:

```
public class IAnnouncements {
    List<String> listAnnouncements();
}

public class DisplayAnnouncements {
    // display announcements
    void displayAnnouncements(List<String> announcementIds(int student_ID));
}

public class getAnnouncementsDB {
    // Method to get an announcement from the DB by ID.
    String getAnnouncementById(int student_ID);
}
```

Check in/out:

```
public class CheckInOut {
    public void initiateCheckInOut() {
        // Method implementation
    }
}

public class CheckInOutForm {
    private File file;

    // Method to fill the form with data.
    public void fillForm(String data) throws IOException {
        file.write(data.getBytes());
    }

    // Retrieves the filled form data.
    public String getFormData() throws IOException {
        byte[] data = new byte[(int) file.length()];
        file.read(data);
        return new String(data);
    }
}

public class InOutDatabase {
    public void updateDatabase(String requestData) {
    }
}
```

MyProfile:

```
public class MyProfile {
    // Method to initiate the profile management process.
    public void manageProfile() {
        // Logic to manage the profile.
    }
}

public class User {
    // Attributes for the user.
    private String name;
    private String email;
    private String password;
    private String hostelblock;
    private String room;
    private String floor;
}

public class DisplayPersonalDetails {
    // Method to display user's personal details.
    public void displayDetails(User user) {
        // Logic to display user details.
    }
}
```

```
public class ChangePersonalDetails {
    // Attributes for the details to be changed.
    private String newPassword;
    private Map<String, String> newDetails;

    // Method to update user's personal details or password.
    public void changeDetails(String password, Map<String, String> details) {
        if(confirmChanges()) {
            this.newPassword = password;
            this.newDetails = details;
        }
        else{
            DisplayPersonalDetails();
        }
        // Logic to change details.
    }

    // Confirm changes before they are sent to the database.
    public boolean confirmChanges() {
        // Logic to confirm changes.
        return true;
    }

    // Retrieves new details for the update.
    public Map<String, String> getNewDetails(User user) {
        return this.newDetails;
    }
}
public class UpdateUsersDB {
    // Method to update the user database with new details.
    public void updateDatabase(User user, Map<String, String> details) {
        // Logic to update the database.
    }
}
User user = new User();
```

Rating:

```
public class Rating {
    // This class acts as the controller for the rating process.
    private GetRatings getRatings;
    private GiveRating giveRating;

    public void handleRatingProcess() {
        // Logic to handle the overall process.
    }
}
```

```
public class hostelblock {
    // Attributes for the hostel block.
    private String blockName;
    private double rating;
    private int num_of_rooms;
    private int num_of_floors[num_of_rooms];
}
```

```
public class GetRatings {
    // Method to retrieve ratings from the database.
    public List<Rating> retrieveRatings() {
        // Logic to get ratings from the database.
        return new ArrayList<>();
    }
}
```

```
public class GiveRating {
    // Method for a user to give a rating.
    public void submitRating(int blockId, double ratingValue) {
        // submit rating
    }
}
```

```
public class UpdateHostelBlockDB {
    // Method to update the database with a new rating.
    public boolean updateDatabase(int blockId, double ratingValue) {
        // update into database
        return true;
    }
}
```

Room Management:

```
public class RoomManagement {
    // Attributes to manage sub-modules
    private RoomExchange roomExchange;
    private CheckInOut checkInOut;
    private Review review;
    private RoomBooking roomBooking;

    public void manageRoomProcesses() {
        // manage room processes
    }
}
```

```
public class RoomExchange {
    // Method to initiate room exchange
    public void initiateRoomExchange() {
        // room exchange process
    }
}
```

```
public class CheckInOut {
    // Method for checking in or out
    public void checkInOrOut() {
        // check-in or check-out process
    }
}

public class Review {
    // Method to display or submit reviews
    public void handleReview() {
        // handle review process
    }
}

public class RoomBooking {
    // Method to book a room
    public void bookRoom() {
        // room booking process
    }
}
```

Room Exchange:

```
public class RoomExchange {
    public void initiateExchangeProcess() {
        // Logic to start the exchange process.
    }
}

public class student {
    private String name;
    private String password;
    private String hostelblock;
    private String room;
    private String floor;
}
```

```
public class ExchangeForm {
    private ExchangeRequest exchangeRequest;

    public void fillForm(student a, student b, String reason) {
        this.exchangeRequest = new ExchangeRequest(a, b, reason);
    }

    public ExchangeRequest getExchangeRequest(student a, student b, String reason)
        if(confirmExchange()) {
            return this.exchangeRequest;
        }
        else{
            return null;
        }
    }
}
```

```
public class ExchangeDBUpdater {
    public boolean updateExchangeDB(ExchangeRequest request) {
        // update the exchange database.
        return true;
    }
}
```

5.3 HR:

Announcements:

```
// Data Store (Not a module)
class Announcements {
    private List<Announcement> announcementList;
}
```

```
// Display Announcements Module
class DisplayAnnouncements {
    public void displayAllAnnouncements(userInput) {
        // display announcements
    }
}
```

```
// Filter My Announcements Module
class FilterMyAnnouncements {
    public List<Announcement> filterAnnouncements(User user, Announcements announc
        // filter the announcements to get our(user) announcements
    }
}
```

```
// Select a Announcement Module
class SelectAnnouncement {
    public Announcement selectOne(List<Announcement> announcements) {
        // select the announcement
    }
}
```

```
// Create New Announcements Module
class CreateNewAnnouncement {
    public Announcement createAnnouncement(userInput) {
        // create announcements
    }
}
// Edit Selected Announcement Module
class EditSelectedAnnouncement {
    public Announcement editAnnouncement(Announcement announcement) {
        // edit the announcement
    }
}
// Update Announcements DB Module
class UpdateAnnouncementsDB {
    public void updateAnnouncement(Announcement announcement) {
        // update the announcement
    }
}
```

HR_Search:

```
class FloorPodInfo {
    int studentID;
    String floor;
    String pod;

    // Constructors, getters, and setters would be present here
}
class Student {
    String studentID;
    String studentDetails;
    // Constructors, getters, and setters would be present here
}
class GetFloorAndPod {
    public FloorPodInfo inputFloorAndPod(int studentID) {
        return new FloorPodInfo(studentID);
        // user inputting the floor and pod of the studentID.
    }
}
class SelectStudent {
    public Student selectStudentFromID(String studentID) {
        // select the student with studentID
    }
}
```

```
class DisplayPodDetails {
    public void displayDetails(FloorPodInfo info) {
        // display pod details
    }
}
class DisplayStudentProfile {
    public void displayProfile(Student student) {
        // display student profile details
    }
}
```

HR_StartMenu:

```
class HRStartMenu {
    AnnouncementsModule announcementsModule;
    SearchStudentModule searchStudentModule;
    FilterTicketsModule filterTicketsModule;

    // Methods to invoke the respective modules
    public void accessAnnouncements() {
        announcementsModule.manageAnnouncements();
    }

    public void performStudentSearch() {
        searchStudentModule.searchForStudent();
    }

    public void applyTicketFilters() {
        filterTicketsModule.filterTickets();
    }
}
class AnnouncementsModule {
    public void manageAnnouncements() {
        // manage announcements here
    }
}
class SearchStudentModule {
    public void searchForStudent() {
        // search for student
    }
}
class FilterTicketsModule {
    public void filterTickets() {
        // filter tickets for the user separately
    }
}
```

HR_Tickets:

```
class Ticket {
    String ticketID;
    String ticketDetails;
    // Constructors, getters, and setters would be present here
}
class TicketDatabase {
    private List<Ticket> tickets;

    public boolean updateTicket(Ticket ticket) {
        // update the ticket
    }

    public boolean deleteTicket(String ticketID) {
        // delete the ticket
    }
}
class DisplayTickets {
    public void displayTickets(List<Ticket> tickets) {
        // display tickets
    }
}
class FinaliseTicketsToPass {
    public void finaliseTicket(Ticket ticket) {
        // choose the finalized tickets
    }
}
class SolveAndCloseTickets {
    public void solveAndCloseTicket(Ticket ticket) {
        // solve and close the tickets after seeing the ticket details
    }
}
```

5.4 HO:

AllotHR:

```
public class AllotHR {
    public void startAllotment() {
        // Start the HR allotment process
    }
}
public class DisplayHRDetails {
    // Methods to display HR details
    public void showHRDetails() {
        // display HR details
    }
}
```

```
public class FillHRDetails {
    private String hrName;
    private String hrDepartment;

    // Method to fill HR details
    public void enterHRDetails(String name, String department) {
        // enter the HR details here for further processing
    }
}
```

```
public class UpdateUserDB {
    // Methods to update the user database
    public bool updateDatabase(String hrDetails) {
        // Update the database
        return true;
    }
}
```

HO_StartMenu:

```
public class HOStartMenu {
    public void displayMenu() {
        // display menu and route to the selected option
    }
}

public class Announcements {
    public void displayAnnouncements() {
        // display announcements of HO and HRs
    }
}

public class SearchStudentHO {
    public void searchStudent(String query) {
        // search for a student
    }
}

public class ComplaintsTickets {
    public void manageComplaints() {
        // display and handle complaints/tickets
    }
}

public class AllotHR {
    public void allotHumanResources() {
        // manage the allotment of HR
    }
}
```

```
public class CheckInOut {
    public void handleCheckInOut() {
        // handle check-in or check-out requests
    }
}

public class RoomExchange {
    public void manageRoomExchange() {
        // facilitate room exchanges
    }
}
```

Announcements:

```
public class Announcements {
    // Announcements module
}

public class CreateNewAnnouncement {
    // Method to create a new announcement
    public void createAnnouncement(String content) {
        // create a new announcement
    }
}

public class DisplayAnnouncements {
    // Method to display announcements
    public void display() {
        // display current announcements
    }
}

public class EditAnnouncement {
    // Attributes for an announcement that might be edited
    private String announcementContent;

    // Method to edit an existing announcement
    public void editAnnouncement(String content) {
        // edit an announcement
    }
}

public class UpdateAnnouncementsDB {
    // Method to update the announcements database
    public boolean updateDB(String updatedContent) {
        // update the database with new/edited announcements
        return true;
    }
}
```

Check in/out:

```
public class CheckInOut {
    public boolean processRequest(boolean isApproved) {
        if (isApproved) {
            // approve request and update DBs
            return true;
        } else {
            // disapprove request
            return false;
        }
    }
}

public class GetCheckInOutRequests {
    // Method to get check-in/out requests
    public String[] retrieveRequests() {
        // Return a list of requests
        return new String[t]; // Placeholder for actual implementation
    }
}

public class UpdateUserDB {
    // Method to update user database
    public boolean updateUserDB(String details) {
        // update to user database with details
        return true;
    }
}

public class UpdateCheckInOutDB {
    public boolean updateCheckInOutDB(String details) {
        // update the check-in/out database with details
        return true;
    }
}

public class UpdateHostelBlocksDB {
    public boolean updateHostelBlocksDB(String details) {
        // update the hostel blocks database with details
        return true;
    }
}

public class ApproveRequest {
    public void approve(String requestId) {
        // approve the request
    }
}
```

```
public class DisapproveRequest {
    public void disapprove(String requestId) {
        // disapprove the request
    }
}
```

Room Exchanges:

```
public class RoomExchange {
    public boolean processExchangeRequest(boolean isApproved) {
        // approve or disapprove the exchange request
        if (isApproved) {
            approveExchangeRequest(isApproved);
            return true;
        } else {
            disapproveExchangeRequest(isApproved);
            return false;
        }
    }
}

public class GetMutualRequests {
    public MutualRequest[] retrieveMutualRequests() {
        // Return array of mutual requests
        return new MutualRequest[0]; // Placeholder for actual implementation
    }
}

public class UpdateUserDB {
    public boolean updateUserDatabase(UserDetails userDetails) {
        // update user details in the database
        return true;
    }
}

public class UpdateExchangeRequestsDB {
    public boolean updateExchangeRequestsDatabase(ExchangeRequestDetails
    exchangeRequestDetails) {
        // update exchange requests in the database
        return true;
    }
}

public class UpdateHostelBlocksDB {
    public boolean updateHostelBlocksDatabase(BlockDetails blockDetails) {
        // update hostel block details in the database
        return true;
    }
}
```

```
public class ApproveRequests {
    public void approveExchangeRequest(String requestId) {
        // approve the request and trigger updates
    }
}

public class DisapproveRequest {
    public void disapproveExchangeRequest(String requestId) {
        // disapprove the request
    }
}
```

Search Student:

```
public class SearchStudentHO {
    // search student module to search for student details
}

public class GetBlock {
    public Block getBlockInformation() {
        // Return the block information
        return new Block(); // Placeholder
    }
}

public class SelectFloorAndPod {
    public FloorAndPod selectFloorAndPod() {
        // Return the selected floor and pod
        return new FloorAndPod(); // Placeholder
    }
}

public class SelectStudent {
    // Method to input or retrieve selected student
    public Student selectStudent() {
        // Return the selected student
        return new Student(); // Placeholder
    }
}

public class DisplayFloors {
    public void displayFloors(Block block) {
        // display floors
    }
}

public class DisplayPodDetails {
    public void displayPodDetails(FloorAndPod floorAndPod) {
        // display pod details
    }
}
```

```
public class DisplayStudentProfile {
    public Student displayStudentProfile(Student student) {
        // display student profile
        return student;
    }
}
```

Complaint Solver:

```
public class ComplaintsTickets {
    // Methods to fill the complaints/tickets
}

public class GetFilteredComplaints {
    public Complaint[] getFilteredComplaints(FilterCriteria criteria) {
        // Return filtered complaints
        return new Complaint[t]; // Placeholder
    }
}

public class DeleteTicketFromComplaintsDB {
    public void deleteTicket(String ticketId) {
        // delete the ticket from the database
    }
}

public class UpdateHostelBlockDB {
    public void updateHostelBlockDatabase(HostelBlockUpdateInfo updateInfo) {
        // update the hostel block database
    }
}

public class SolveCloseTicket {
    public void solveAndCloseTicket(String ticketId, ResolutionInfo resolution) {
        // solve and close the ticket
    }
}
```