



# CS 492: Senior Design Project

## Final Report

BilMate : Group T2335

Ahmet Salman	21901004
Atasagun Samed Şanap	21902435
Ebrar Bozkurt	21802824
Javid Moradi	21903645
Onuralp Avcı	21902364

19/05/2023

**Jury Members:** Erhan Dolak, Tağmaç Topal, Uğur Doğrusöz

**Innovation Expert:** Ahmet Kocamaz

<b>1. Introduction.....</b>	<b>6</b>
<b>2. Requirement Details.....</b>	<b>6</b>
2.1 Functional Requirements.....	6
2.1.1 Users.....	6
2.1.1.1 General Requirements.....	6
2.1.1.2 House-Seeker Requirements.....	6
2.1.1.3 House-Sharer Requirements.....	7
2.1.2 Admin.....	7
2.2 Nonfunctional Requirements.....	7
2.2.1 Usability.....	7
2.2.2 Security.....	7
2.2.3 Scalability.....	7
2.2.4 Maintainability.....	8
2.3 Pseudorequirements.....	8
<b>3. Final Architecture and Design Details.....</b>	<b>8</b>
3.1 Final Use Case Model.....	8
3.2 Final Scenarios.....	9
3.2.1 Signup Scenario.....	9
3.2.2 Login Scenario.....	9
3.2.3 SeeSharedPosts Scenario.....	9
3.2.4 GetMatched Scenario.....	9
3.2.5 FillExtraMatePreferences Scenario.....	10
3.2.6 Search Scenario.....	10
3.2.7 SeeStarredUser Scenario.....	10
3.2.8 SeeStarredHousePosts Scenario.....	10
3.2.9 Report Post Scenario.....	11
3.2.10 ViewUserReports Scenario.....	11
3.2.11 ViewBannedUsers Scenario.....	11
3.2.12 ViewBannedPosts Scenario.....	12

3.2.13 EditProfile Scenario.....	12
3.2.14 Logout Scenario.....	12
3.2.15 CreatePost Scenario.....	12
3.2.16 EditPost Scenario.....	13
3.2.17 UploadImages Scenario.....	13
3.3 Final Object Model.....	14
3.4 Final Sequence Diagrams.....	16
3.4.1 User Signup.....	16
3.4.2 Update Preferences.....	17
3.4.3 Create House-Sharer Post.....	17
3.4.4 Report Post.....	18
3.4.5 Set Privacy Level.....	18
3.4.6 Upload Images to Post.....	19
3.5 Final Activity Diagrams.....	19
3.5.1 Two-Factor Authentication (2FA).....	19
3.5.2 Report User.....	20
3.5.3 Create Post.....	20
3.5.4 Edit Post.....	21
3.6 Final State Diagrams.....	22
3.6.1 Login/Signup Flow.....	22
3.6.2 Find and Star Post Flow.....	23
3.6.3 Conversation With Housemate.....	24
3.7 Final User Interface.....	25
3.7.1 Register.....	25
3.7.2 Login.....	25
3.7.3 Choose Post Type.....	26
3.7.4 Create Post.....	26
3.7.5 Check House Seeker Posts.....	27
3.7.6 Check Your Own Posts.....	27

3.7.7 Check Your Own Profile.....	28
3.7.8 Edit Your Own Profile.....	28
3.7.9 Check Another User's Profile.....	29
3.7.10 Check Post Details.....	29
3.7.11 Edit Your Preferences.....	30
3.7.12 Report A Post.....	31
3.7.13 Starred Posts.....	32
<b>4. Implementation Details.....</b>	<b>32</b>
<b>5. Test Cases and Results.....</b>	<b>32</b>
5.1 Functional Test Cases.....	32
5.1.1 Unit Testing.....	32
5.1.1.1 User Signs up with a non-Bilkent email.....	32
5.1.1.2 User tries to sign in without verifying their email.....	33
5.1.1.3 User tries to sign in with the wrong email-password combination.....	33
5.1.1.4 User tries to log in using the valid credentials.....	34
5.1.1.5 User tries to post an incomplete post.....	34
5.1.1.6 User tries to post a complete posting.....	34
5.1.1.7 User tries to delete their own post.....	35
5.1.1.8 User tries to delete another user's post.....	35
5.1.1.9 User tries to update their own post.....	36
5.1.1.10 User tries to update another user's post.....	36
5.1.1.11 User tries to update their profile.....	37
5.1.1.12 User tries to update another user's profile.....	37
5.1.2 Sanity and Smoke Testing.....	38
5.1.2.1 Duplicate Users.....	38
5.1.2.2 Missing Fields During Registration.....	38
5.1.2.3 User inputs a wrong email.....	39
5.1.2.4 User tries to star a post that does not exist.....	39
5.1.2.5 User requests a list based on their preference.....	39

5.1.3 Regression Testing.....	40
5.1.3.1 When connection to the database is lost while processing a request.....	40
5.1.3.2 Effect of post creation on the matching algorithm.....	40
5.1.3.3 Effects of starring a post on the matching algorithm.....	41
5.1.3.4 Privacy Settings.....	41
5.1.3.5 User does not indicate preferences.....	42
5.1.4 System Testing.....	42
5.1.4.1 User signs up.....	42
5.1.4.2 User scrolls through posts.....	43
5.1.4.3 User deletes a post.....	43
5.1.4.4 Reporting a user.....	44
5.1.4.5 Account deletion.....	44
5.1.5 Acceptance Testing.....	45
5.1.5.1 Can users register for BilMate.....	45
5.1.5.2 Can users verify their email.....	45
5.1.5.3 Can users create a post.....	46
5.1.5.4 Can the users update their post.....	46
5.1.5.5 Can the user delete their post.....	47
5.1.5.6 Can the user star any post.....	47
5.1.5.7 Can the user update their own profile.....	47
5.1.5.8 Can the user delete a post.....	48
5.1.5.9 Can the user send another verification email.....	48
5.1.5.10 Can the user upload images to their posts.....	49
5.1.5.11 Can the user report a post.....	49
5.1.5.12 Can an admin review and accept a reporting ticket.....	49
5.1.5.13 Can an admin review and reject a reporting ticket.....	50
5.1.6 Input validation.....	50
5.1.6.1 Registration.....	50
5.1.6.2 Updating Fields.....	51

5.2 Non-Functional Test Cases.....	51
5.2.1 Performance Testing.....	51
5.2.2 Usability Testing.....	52
5.2.3 Stress Testing.....	52
5.2.3.1 High Load.....	52
5.2.3.2 High Stress.....	53
5.2.4 Security Testing.....	53
5.2.5 Portability Testing.....	54
5.2.6 Maintenance Testing.....	55
5.2.6.1 Confirmation Testing.....	55
5.2.6.2 RegressionTesting.....	55
5.2.7 Recovery Testing.....	56
5.2.8 Reliability Testing.....	56
<b>6. Maintenance Plan and Details.....</b>	<b>57</b>
<b>7. Other Project Elements.....</b>	<b>58</b>
7.1 Consideration of Various Factors in Engineering Design.....	58
7.1 Public Health.....	58
7.2 Public Safety.....	58
7.3 Public Welfare.....	58
7.4 Global.....	59
7.5 Cultural.....	59
7.6 Social.....	59
7.7 Environmental.....	59
7.8 Economic.....	59
7.2 Ethics and Professional Responsibilities.....	60
7.3 Judgements and Impacts to Various Contexts.....	61
7.4 Teamwork Details.....	61
7.4.1 Contributing and functioning effectively on the Team.....	61
7.4.2 Helping to create a collaborative and inclusive environment.....	62

7.4.3 Taking the lead role and sharing leadership on the team.....	62
7.4.4 Meeting objectives.....	63
7.5 New Knowledge Acquired and Applied.....	63
<b>8. Conclusion and Future Work.....</b>	<b>64</b>

# 1. Introduction

One of the major struggles for university students, particularly in Turkey, is finding acceptable housing. Although there are student housing options, they are sometimes overcrowded, in poor shape, pricey, or located far from the university. Students now have to find suitable housing and roommates on their own as a result of this. While some online tools are available to assist with this, such as "Bilkent İtiraf," they are not designed specifically for the task. They may not be successful in pairing people based on their criteria.

BilMate addresses this issue. Students looking for roommates and those looking for shared housing can connect on this app's social network and search through choices depending on their interests. Users of the app will be able to enter information about their chosen location, number of roommates, rent, smoking policies, pet allowance, and other preferences. The app will match up house-seekers and housemate-seekers based on this information. Users' chances of discovering a compatible match increase with the number of data fields they fill out.

## 2. Requirement Details

### 2.1 Functional Requirements

#### 2.1.1 Users

##### 2.1.1.1 General Requirements

- Unregistered users should be able to register an account using their verified email.
- Unregistered users should be able to view the postings but without seeing any contact information
- Unregistered users should be able to verify their emails
- Registered users should be able to log into their accounts.
- Registered users should be able to log out of their accounts.
- Registered users should be able to change their password.
- Registered users should be able to modify their profile.
- Registered users should be able to change their privacy settings to show/hide their phone number



#### 2.1.1.2 House-Seeker Requirements

- House-seeker should be able to answer questions indicating the preferred traits of their roommate.
- House-seeker should be able to answer any number of questions to refine the matching algorithm.
- House-seeker should be able to stop answering questions at any time.
- House-seeker should be able to filter and sort through post listings manually.
- House-seeker should be able to add specific posts to a Favorites list.
- House-seeker should be able to remove posts from their Favorites list.

#### 2.1.1.3 House-Sharer Requirements

- House-sharer should be able to create their own posts to indicate that they are looking for a roommate.
- House-sharer should be able to add contact information to their posting.
- House-sharer should be able to modify or remove their contact information from their posting.
- House-sharer should be able to delete their own posts.
- House-sharer should be able to indicate their preference on the post before it is published.
- House-sharer should be able to edit their posts once they are published.
- House-sharer should be able to report other users in case the other party breaks the TOS.

#### 2.1.2 Admin

- Admins should be able to ban/unban users from the platform.
- Admins should be able to review report cases.
- Admins should be able to provide a verdict regarding reported users.

## 2.2 Nonfunctional Requirements

### 2.2.1 Usability

It is essential to hide the complexity behind the UI layer in apps with complex algorithms. Regardless of what is going on in the domain layer, users should not be bothered with that part,

and a seamless flow should be offered to them. In BilMate, there will be many questions for users in order to make the matching algorithm as accurate as possible. However, the app will be optimized to make that questioning flow as user-friendly as possible.

### 2.2.2 Security

All information shared by the users can be considered sensitive data. This means that on both the backend and frontend sides, BilMate must be developed with a “data protection by design and default” strategy. All the sensitive content will be encrypted with a well-chosen encryption algorithm, and any data that is not required to be stored will not be stored.

### 2.2.3 Scalability

The app must be scalable first of all for the increasing number of Bilkent students; then for the other universities and their students as well.

### 2.2.4 Maintainability

User traffic on the app will fluctuate based on the time of the year, which means, especially at the start of semesters, the app will face intense user traffic. At those times, there will be some problems on the backend side due to high user requests, and possibly some UI bugs will appear on the frontend side. The architecture must allow developers to fix problems and maintain the app as fast as possible for those scenarios.

## 2.3 Pseudorequirements

- Kotlin will be used to develop the frontend
- FastAPI will be used to develop the backend
- MongoDB will be used as the database
- The application will be an Android OS mobile application

- 
- The diagram is a UML Use Case Diagram for a House-Sharing System. It features three main actors: **House-Share (registered)**, **House-Seeker (unregistered)**, and **Admin**. The use cases are represented by ovals, and their relationships are shown with lines.
- Use Cases and their Extension Points:**
- FillExtraMatePreferences**: No extension points.
  - SignUp**: extension points: alreadyExistingMail, wrongPasswordFormat, unmatchingPassword.
  - FillPreferences**: No extension points.
  - GetMatched**: extension points: editFilter, contactUser, starUser.
  - Search**: extension points: editFilter.
  - ViewUserProfile**: extension points: contactUser, starUser.
  - ViewHousePost**: extension points: contactUser, starPost.
  - Login**: No extension points.
  - EditProfile**: No extension points.
  - SeeSharedPosts**: extension points: editPost, createHouseSharingPost, createHouseSeekingPost, hide/unhidePost.
  - LogOut**: No extension points.
  - ViewUserReports**: extension points: banReportedUser, viewUserProfile, banReportedPost, viewReportedPost.
  - ViewBannedUsers**: extension points: unbanUser, viewUserProfile, unbanPost.
  - ViewBannedPosts**: extension points: unbanPost, viewPost.
  - ReportPost**: extension points: viewHousePost.
  - SeeStarredHousePosts**: extension points: viewHousePost, removeStarFromPost.
  - SeeStarredUsers**: extension points: viewUserProfile, removeStarFromUser.
  - removeStarFromUser**: No extension points.
- Relationships:**
- House-Share (registered)** is associated with: FillExtraMatePreferences, SignUp, FillPreferences, GetMatched, Search, ViewUserProfile, ViewHousePost, Login, EditProfile, SeeSharedPosts, LogOut, ViewUserReports, ViewBannedUsers, ViewBannedPosts, ReportPost, SeeStarredHousePosts, SeeStarredUsers, and removeStarFromUser.
  - House-Seeker (unregistered)** is associated with: SignUp, FillPreferences, GetMatched, Search, ViewUserProfile, ViewHousePost, Login, EditProfile, SeeSharedPosts, LogOut, ViewUserReports, ViewBannedUsers, ViewBannedPosts, ReportPost, SeeStarredHousePosts, SeeStarredUsers, and removeStarFromUser.
  - Admin** is associated with: ViewUserReports, ViewBannedUsers, ViewBannedPosts, and SeeStarredUsers.
- Extension Points (dashed lines):**
- SignUp** extends **FillExtraMatePreferences** at **alreadyExistingMail**.
  - FillPreferences** extends **SignUp** at **wrongPasswordFormat** and **unmatchingPassword**.
  - GetMatched** extends **Search** at **editFilter**.
  - ViewUserProfile** extends **GetMatched** at **contactUser** and **starUser**.
  - ViewHousePost** extends **ViewUserProfile** at **starPost**.
  - SeeSharedPosts** extends **GetMatched** at **editPost**, **createHouseSharingPost**, **createHouseSeekingPost**, and **hide/unhidePost**.
  - ViewUserReports** extends **SeeSharedPosts** at **banReportedUser**, **viewUserProfile**, **banReportedPost**, and **viewReportedPost**.
  - ViewBannedUsers** extends **ViewUserReports** at **unbanUser** and **viewUserProfile**.
  - ViewBannedPosts** extends **ViewBannedUsers** at **unbanPost** and **viewPost**.
  - ReportPost** extends **ViewHousePost** at **viewHousePost**.
  - SeeStarredHousePosts** extends **ReportPost** at **viewHousePost** and **removeStarFromPost**.
  - SeeStarredUsers** extends **SeeStarredHousePosts** at **viewUserProfile** and **removeStarFromUser**.

- Fills up mandatory user preferences and details
- User is registered

### 3.2.2 Login Scenario

Use Case Name: Login

Actors: Admin, House-Sharer and House-Seeker (registered)

Entry Conditions: actor is registered

Exit Conditions: actor is logged in

Flow of Events:

- goes to login page
- Enters credentials
- Logs in

### 3.2.3 SeeSharedPosts Scenario

Use Case Name: SeeSharedPosts

Actors: House-Sharer, House-Seeker

Entry Conditions: Actor is logged in

Exit Conditions: Actor is logged in and has seen his/her shared houses or shared profile posts

Flow of Events:

- Actor clicks on “HouseSharing” posts or “HouseSeeking” posts
- Actor views posts
- Actor can, hide, unhide, edit or create posts
- Actor leaves

### 3.2.4 GetMatched Scenario

Use Case Name: GetMatched

Actors: House-Seeker, House Sharer

Entry Conditions: Actors are logged in

Exit Conditions: Actors have a list of matches

Flow of Events:

- Actor goes to match page
- Actor is shown matches

- Actor can edit filter for better results
- Actor can star user or star post
- Actor can contact user
- Actor can exit

### 3.2.5 FillExtraMatePreferences Scenario

Use Case Name: FillExtraMatePreferences

Actors: House-Sharer, House-Seeker

Entry Conditions: Actors are logged in

Exit Conditions: Actors will have more preferences

Flow of Events:

- Actor goes to profile
- Actor fills up extra preferences
- Actor saves and closes

### 3.2.6 Search Scenario

Use Case Name: Search

Actors: House-Sharer, House-Seeker, Admin

Entry Conditions: Actors are logged in

Exit Conditions: Actor will have a list of related search

Flow of Events:

- Actor goes to search field
- Actor input search parameters
- Actor filters out the search

### 3.2.7 SeeStarredUser Scenario

Use Case Name: SeeStarredUsers

Actors: House-Seeker, House-Sharer

Entry Conditions: Actor is logged in

Exit Conditions: Actor will have seen the starred users

Flow of Events:

- Actor goes to starred users

- Actor sees a list of previously starred users
- Actor can view user profile
- Actor leaves

### 3.2.8 SeeStarredHousePosts Scenario

Use Case Name: SeeStarredHousePosts

Actors: House-Seeker, House-Sharer

Entry Conditions: Actor is logged in

Exit Conditions: Actor will have seen previously starred shared house posts

Flow of Events:

- Actor goes to starred house posts
- Actor sees a list of previously starred houses
- Actor can view house posts
- Actor leaves

### 3.2.9 Report Post Scenario

Use Case Name: Report Post

Actors: House-Seeker, House-Sharer

Entry Conditions: User is logged in, and viewing a post

Exit Conditions: User is reported

Flow of Events:

- User is viewing a house post
- User reports the post

### 3.2.10 ViewUserReports Scenario

Use Case Name: ViewUserReports

Actors: Admin

Entry Conditions: Actor is logged in

Exit Conditions: Actor will view user reports and bans users or posts

Flow of Events:

- Actor views user' reports

- Actor reviews them individually
- Actor can view reported user profile
- Actor can view reported post
- Actor can choose to ban the reported user
- Actor can choose to ban the reported post
- Actor leaves

### 3.2.11 ViewBannedUsers Scenario

Use Case Name: ViewBannedUsers

Actors: Admin

Entry Conditions: Actor is logged in

Exit Conditions: Views banned users and decides to unban them or not

Flow of Events:

- Actor views banned users
- Actor can view the banned user profile
- Actor can choose to unban the user
- Actor leaves

### 3.2.12 ViewBannedPosts Scenario

Use Case Name: ViewBannedPosts

Actors: Admin

Entry Conditions: Actor is logged in

Exit Conditions: Views banned posts and decides to unban them or not

Flow of Events:

- Actor views banned posts
- Actor can view a post
- Actor can unban the post
- Actor leaves

### 3.2.13 EditProfile Scenario

Use Case Name: EditProfile

Actors: Admin, House-Sharer, House-Seeker

Entry Conditions: Actor is logged in

Exit Conditions: Actor edited profile

Flow of Events:

- Actor goes to profile
- Actor edits profile
- Actor saves and exits

### 3.2.14 Logout Scenario

Use Case Name: Logout

Actors: Admin, House-Sharer, House-Seeker

Entry Conditions: Actor is logged in

Exit Conditions: Actor logged out

Flow of Events:

- Actor goes to profile
- Actor logs out

### 3.2.15 CreatePost Scenario

Use Case Name: CreatePost

Actors: House-Sharer, House-Seeker

Entry Conditions: Actor is logged in

Exit Conditions: Actor has a post

Flow of Events:

- Actor goes to main page
- Actor starts filling data for the post
- Actor approves his/her post
- Actor's post is uploaded

### 3.2.16 EditPost Scenario

Use Case Name: EditPost

Actors: House-Sharer, House-Seeker

Entry Conditions: Actor is logged in

Exit Conditions: Actor's post is edited

Flow of Events:

- Actor goes to profile



- Actor goes to “My Posts”
- Actor presses on one of his/her posts
- Actor edits the information on the post
- Actor approves the changes
- Actor’s post is updated

### 3.2.17 UploadImages Scenario

Use Case Name: UploadImages

Actors: House-Sharer, House-Seeker

Entry Conditions: Actor is logged in

Exit Conditions: Actor’s post has images

Flow of Events:

- Actor goes to profile
- Actor goes to “My Posts”
- Actor presses on one of his/her posts
- Actor uploads images to the post
- Actor approves the changes
- Actor’s post is updated

### 3.3 Final Object Model

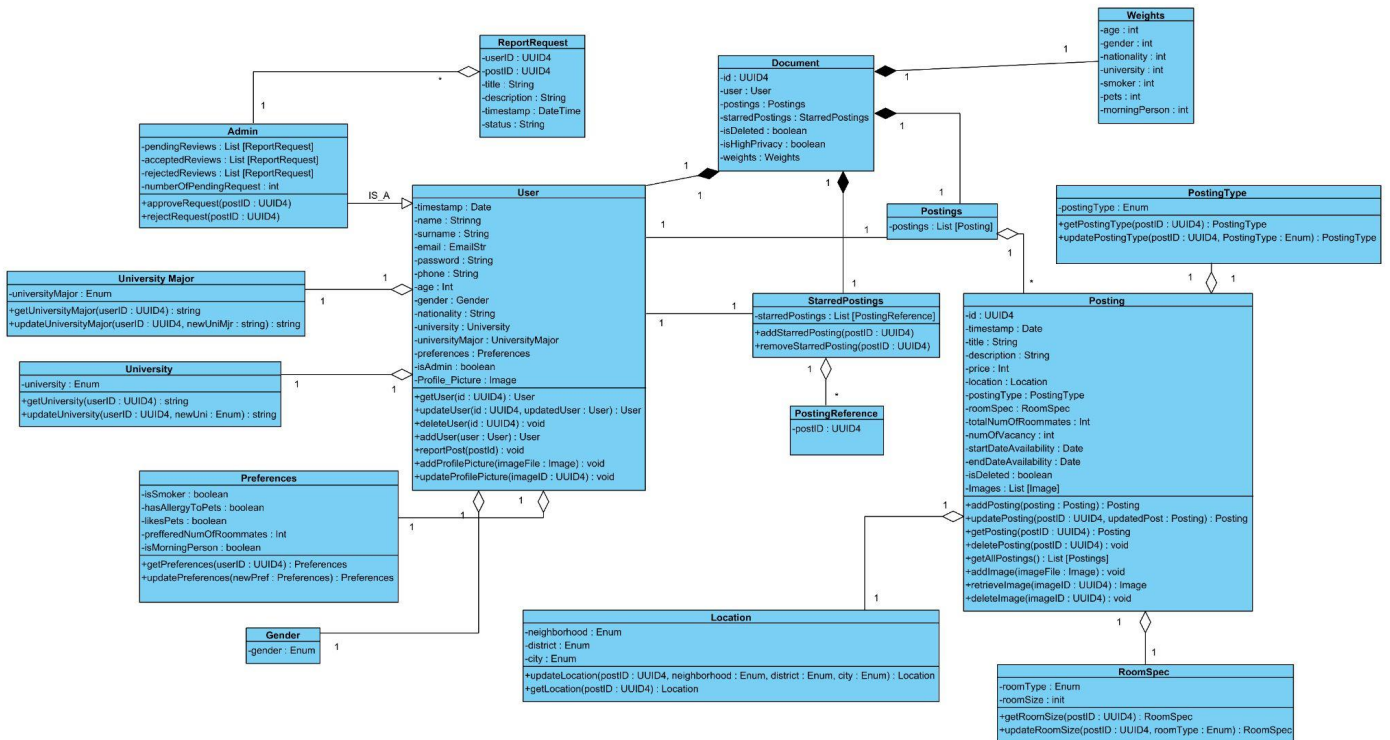


Figure 2: Object / Class Diagram

Class	Explanation
Document	Document is the class responsible for “gluing” together all the functions needed, it provides a high-level abstraction of all the other classes
User	The User class for all the functions and properties belonging to a user including the creation and modification of the user
Postings	This class hold a list of Posting objects which in turn is held by the Document class
Posting	This class contains all the functions and

	properties for the Posting object, which a User can have many of
StarredPostings	This class holds a list of PostingReference which can be used to refer to any Posting object in the system
postingReference	This class is responsible for holding an ID referring to a Posting object
Admin	This class is responsible for managing all the functions of admins which include the accepting of report requests, which results in banning
Location	This class holds an enumeration of the location for each Posting object
RoomSpec	This class holds the specifications of each room in Posting object
PostingType	This class holds an Enumeration of the posting type, whether it is a house-sharer post or a house-seeker
UniversityMajor	This class holds an enumeration of general university majors (e.g. Natural sciences, engineering, etc.)
University	This class holds an enumeration of all universities in Ankara
Preferences	This class holds the properties that will be used to create a better match for the user
Gender	This class holds an enumeration of genders

Admin	This class is responsible for all the functionalities of an admin
ReportRequest	This class holds the model for the request sent when a user reports a post
Weights	This class holds the data needed for applying weights used in the matching algorithm

## 3.4 Final Sequence Diagrams

### 3.4.1 User Signup

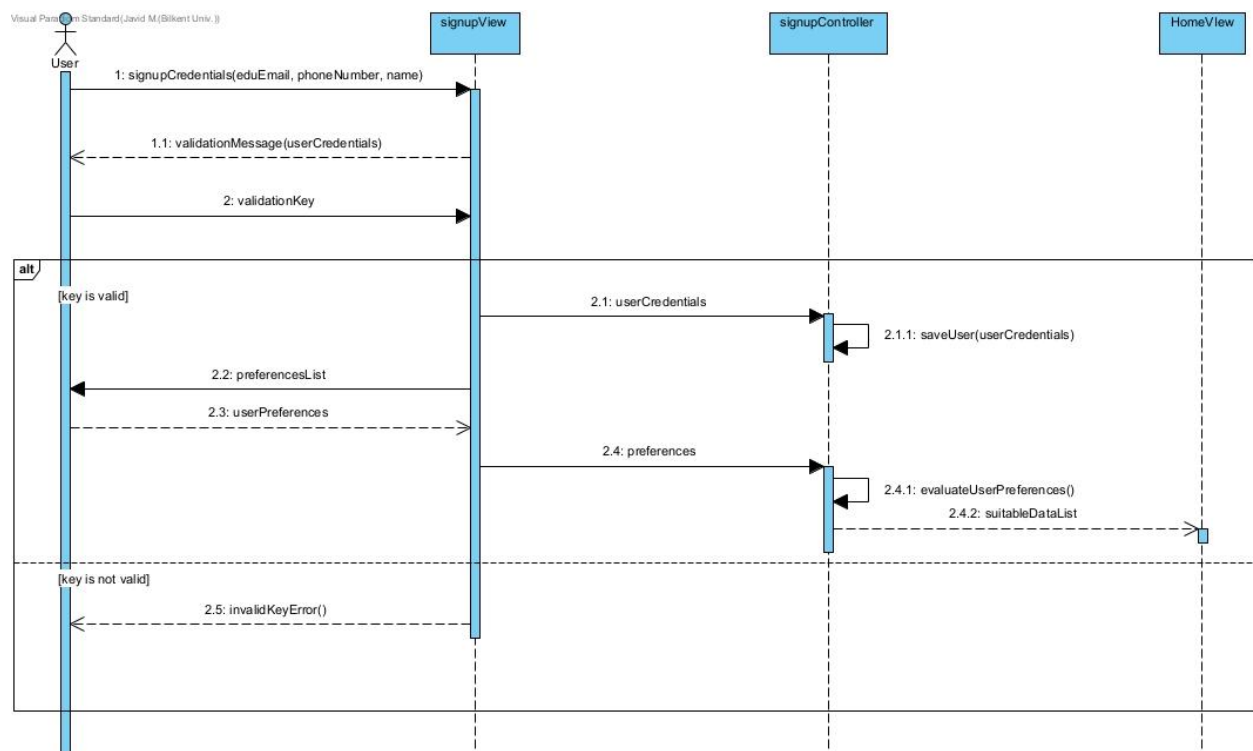
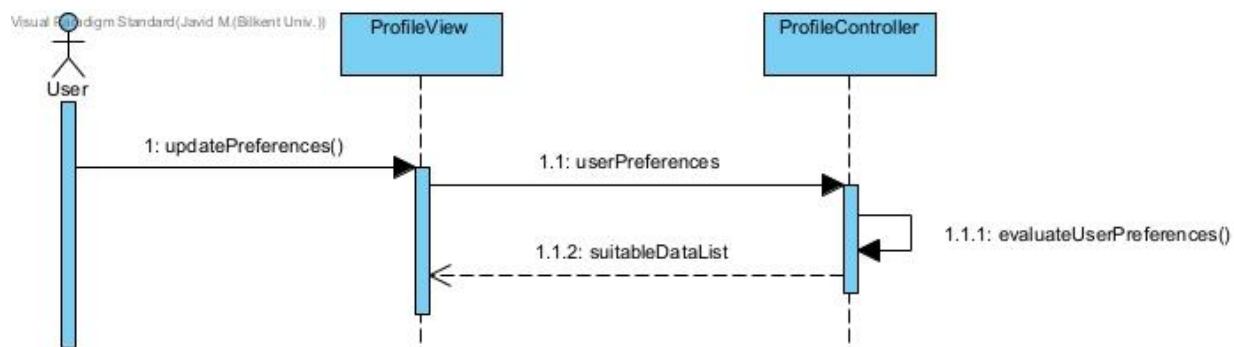


Figure 3: Sequence diagram for “signup” flow

First scenario of a sequence diagram is a new user's sign-up/profile creation. First, a user must provide some credentials to the view, which are the user's educational email address, phone number, and name; please note that these are noteworthy credentials, as the application might require further user credentials. Later, the application will send a verification key via email to the user and expects the user to provide it for security purposes. If the key is invalid, the user will be notified with an error message. If the key is correct, the user will be saved to the system. Afterward, the user will be given the option to answer several preference-related questions. These initial preferences will be evaluated, and suitable data will appear on the user's homepage view.

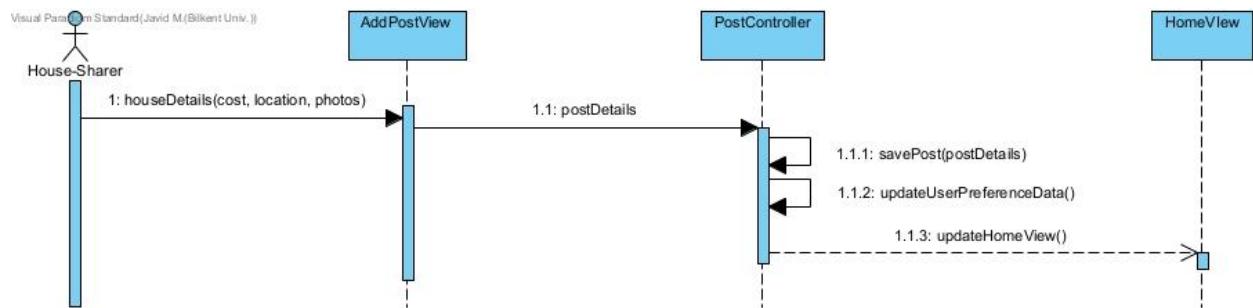
### 3.4.2 Update Preferences



**Figure 4: Sequence diagram for “update preference” flow**

The second diagram is about a user updating their preferences list. In their profile section of the application, users can change their preferences, such as if they smoke, like pets, etc. The updated list of preferences will be sent to the controller, where suitable recommended data of the user will be updated, as the recommendation algorithm will update itself based on the provided new preferences list.

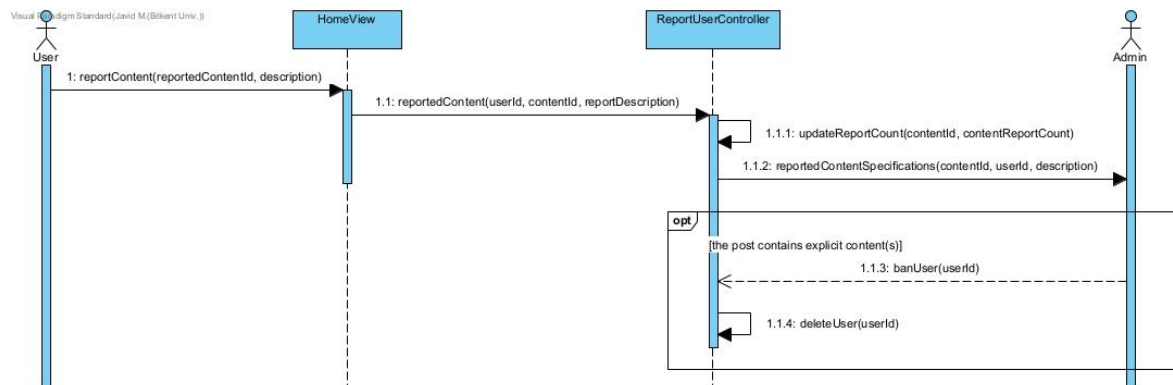
### 3.4.3 Create House-Sharer Post



**Figure 5: Sequence diagram for “share house post” flow**

The diagram above is the event of a house-sharer sharing a post. First, a house-sharer will provide necessary information about their post, such as cost, location, etc. Later, these details will be sent to the system and saved. Users’ preference list will be updated according to this new post’s content, which consequently updates users’ home views. Please note that a house-sharer is also a user.

### 3.4.4 Report Post

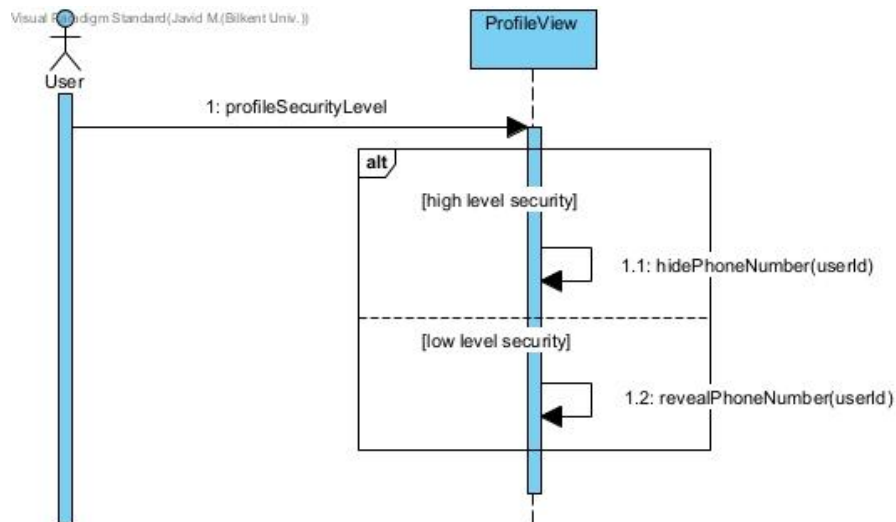


**Figure 6: Sequence diagram for “report post” flow**

Here, an event of a user reporting a post is displayed. First, a user reports a post by providing the reasoning behind their action. Later, the post and its other details, such as its owner, are sent to the system. Note that on each report, the post’s report counter will increase since a post can be reported by multiple users. Afterward, the reported content will be sent to an admin.

Admin will examine the post with given comments, and if the decision is positive, that is, the post contains explicit material(s), the post is deleted, and the post's owner is expelled from the application.

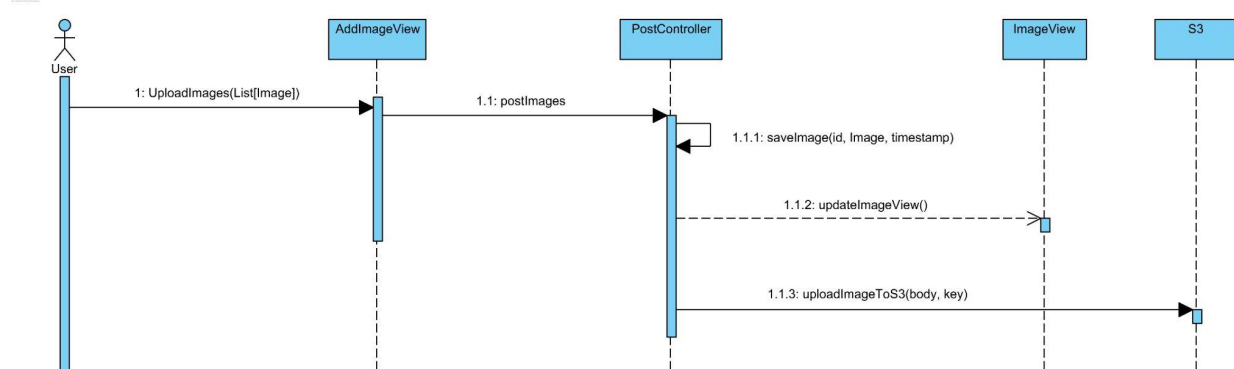
### 3.4.5 Set Privacy Level



**Figure 7: Sequence diagram for “set privacy level” flow**

This diagram is about users' security regarding their phone number's public availability. Users can either hide their phone numbers due to confidentiality-related reasons. If they do, the application will show their email address as the main communication method. If they decide to reveal their phone number, other users can contact them via email or phone call.

### 3.4.6 Upload Images to Post

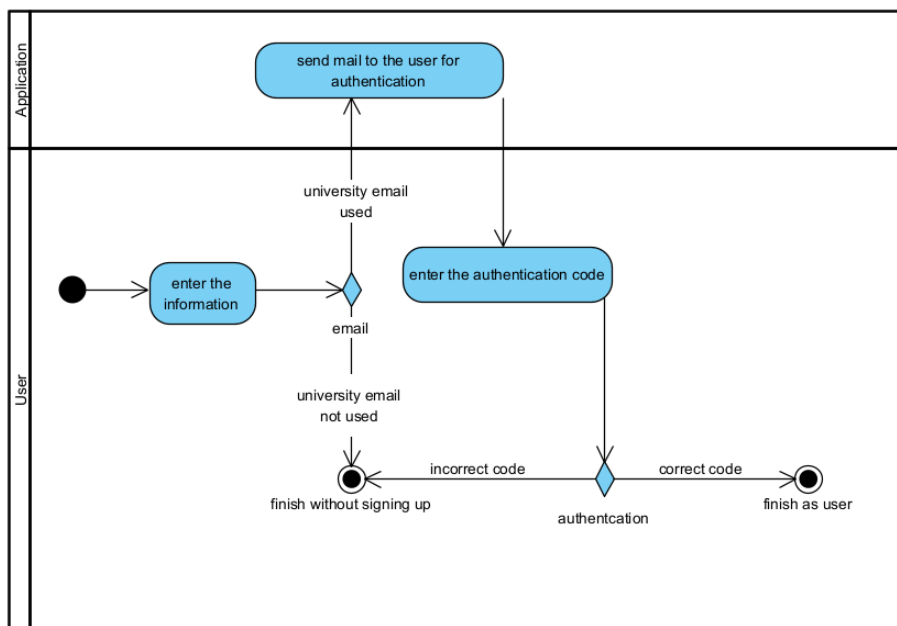


**Figure 8: Sequence diagram for “Upload Image” flow**

The diagram here showcases the procedure of adding images to a user’s post. They start by uploading it to the UI, which then passes a request to the API. Furthermore, it saves the metadata of the image locally and then saves the actual picture in S3.

## 3.5 Final Activity Diagrams

### 3.5.1 Two-Factor Authentication (2FA)

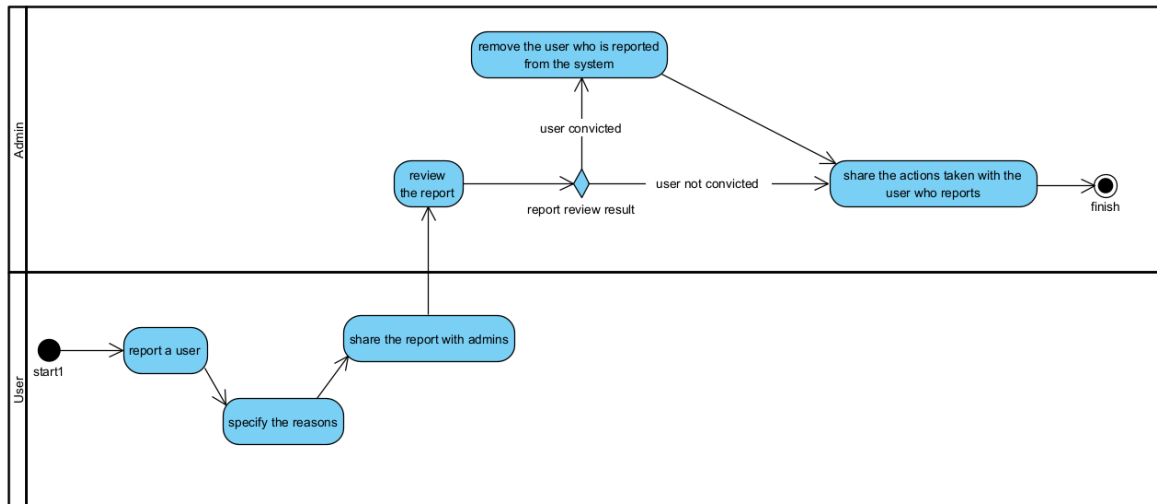


**Figure 9: Activity diagram for “2FA” flow**

First activity diagram shows two way authentication of the system. User first enters their information. Only university emails are accepted, otherwise the signup becomes unsuccessful. If the user uses university email, an authentication mail is sent. If the user enters the code correctly, signup becomes successful.



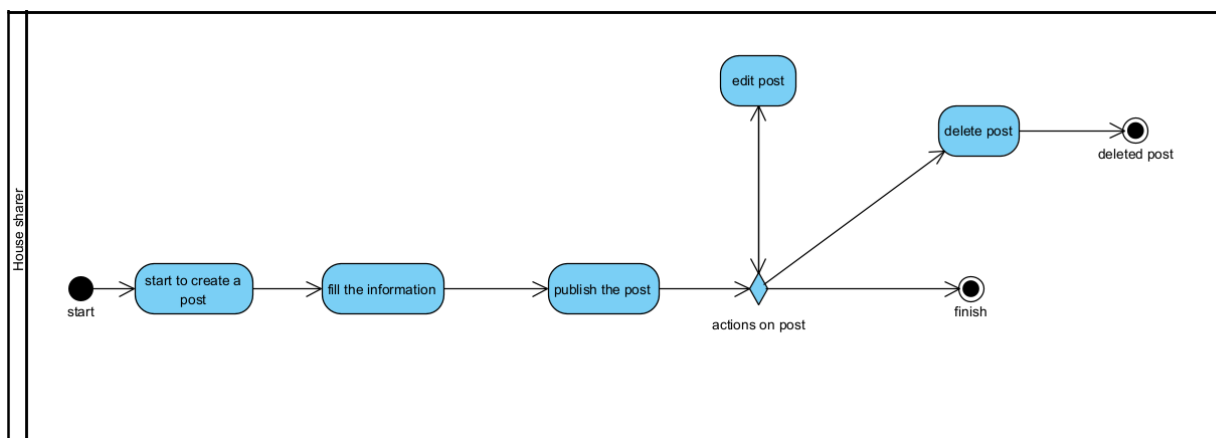
### 3.5.2 Report User



**Figure 10: Activity diagram for “report user” flow**

Second activity diagram shows the flow of events in reporting a user. A user can report a user by specifying the reasons. Then s/he shares the reports with admins. Admins review the report. They may either remove the user or not find the user not guilty. In both cases, the user reporting is informed about the decision.

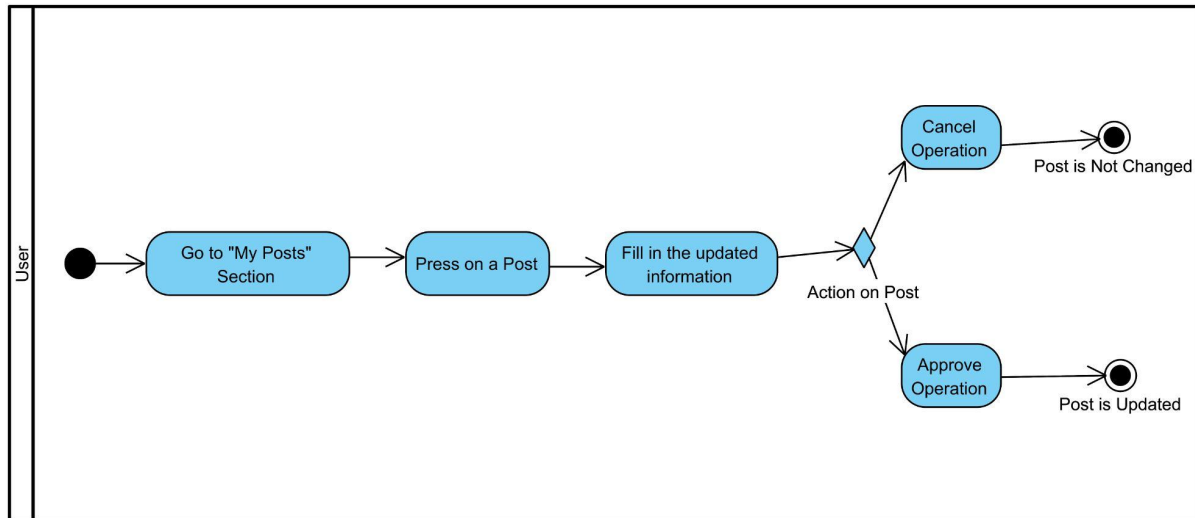
### 3.5.3 Create Post



**Figure 11: Activity diagram for “create post” flow**

Third activity diagram shows the flow of events in creating a post. First user starts creating a post action by pressing the necessary buttons. Then, the user fills in the necessary information such as title, description, location. Then, the user can publish the post or delete the post before publishing. If s/he publishes the post, s/he can do nothing, edit the post later or delete it.

### 3.5.4 Edit Post



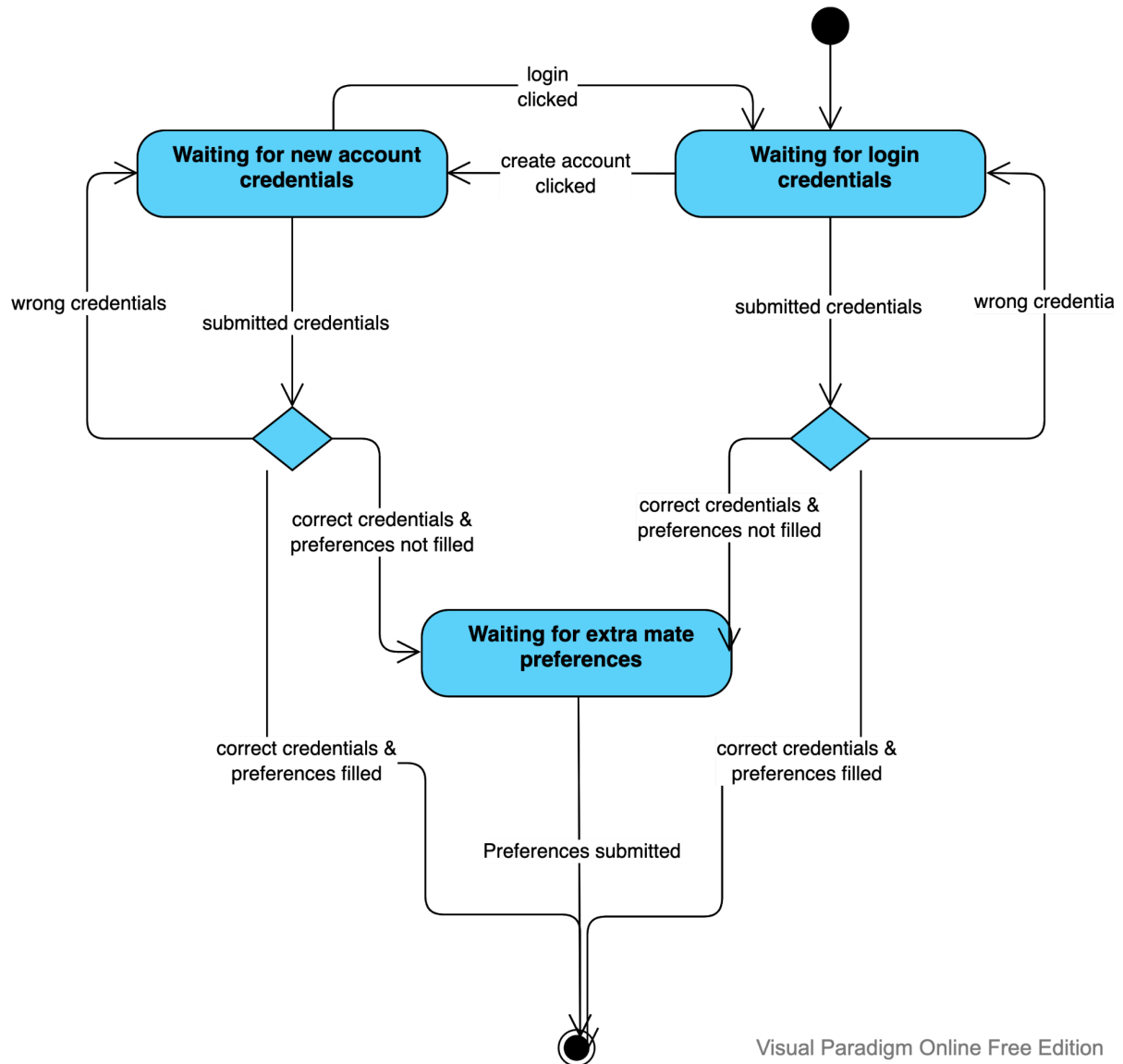
**Figure 12: Activity diagram for “update post” flow**

This shows the flow of updating the post. The user must have created a post earlier. The user then navigates to the “My Posts” section and press on one of their posts. Then they have the option to edit all the fields in their post, from dates to price to images. Finally, they have the option to approve their changes or to cancel them and nothing will change for the post.

## 3.6 Final State Diagrams

### 3.6.1 Login/Signup Flow

Visual Paradigm Online Free Edition



Visual Paradigm Online Free Edition

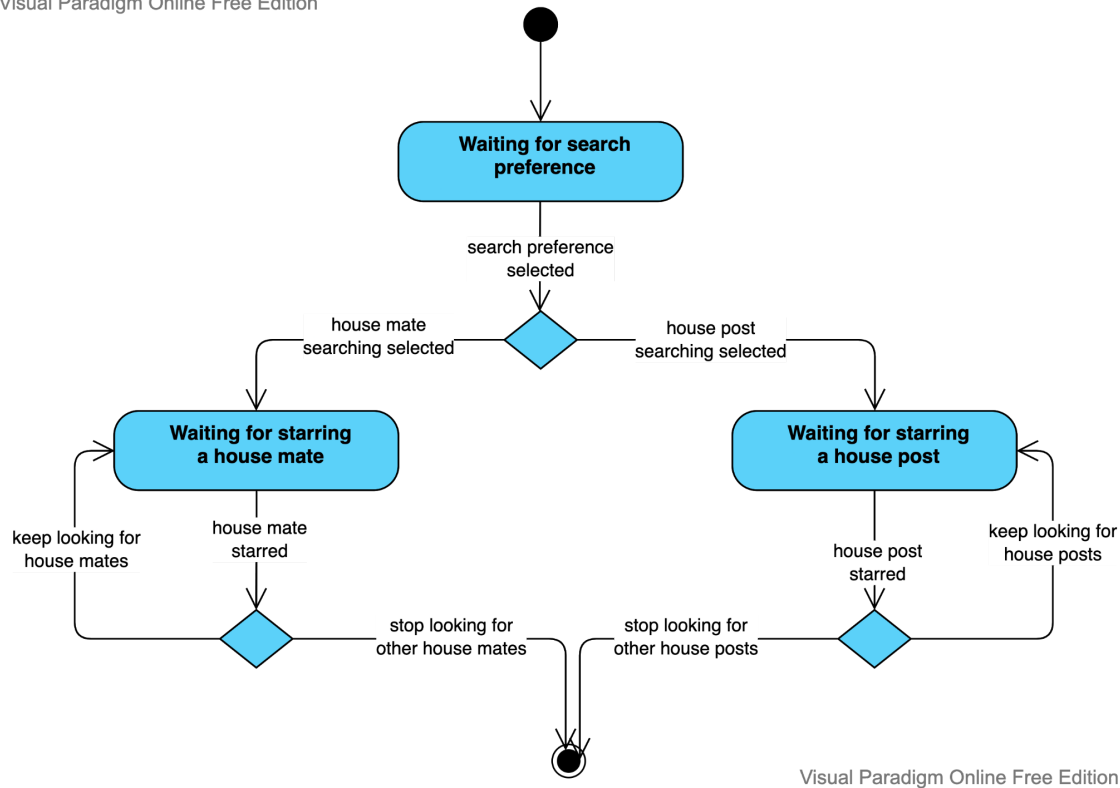
**Figure 13: State diagram for “login/signup” flow**

The first state diagram shows the states during the login/signup flow. The initial state for this flow is waiting for login credentials. If the user is not signed up yet, he can be navigated to the signup flow. After successful login, depending on whether user preferences are filled or not,

they will be navigated to either the home screen (exit state) or they will be first asked to fill in their preferences.

### 3.6.2 Find and Star Post Flow

Visual Paradigm Online Free Edition

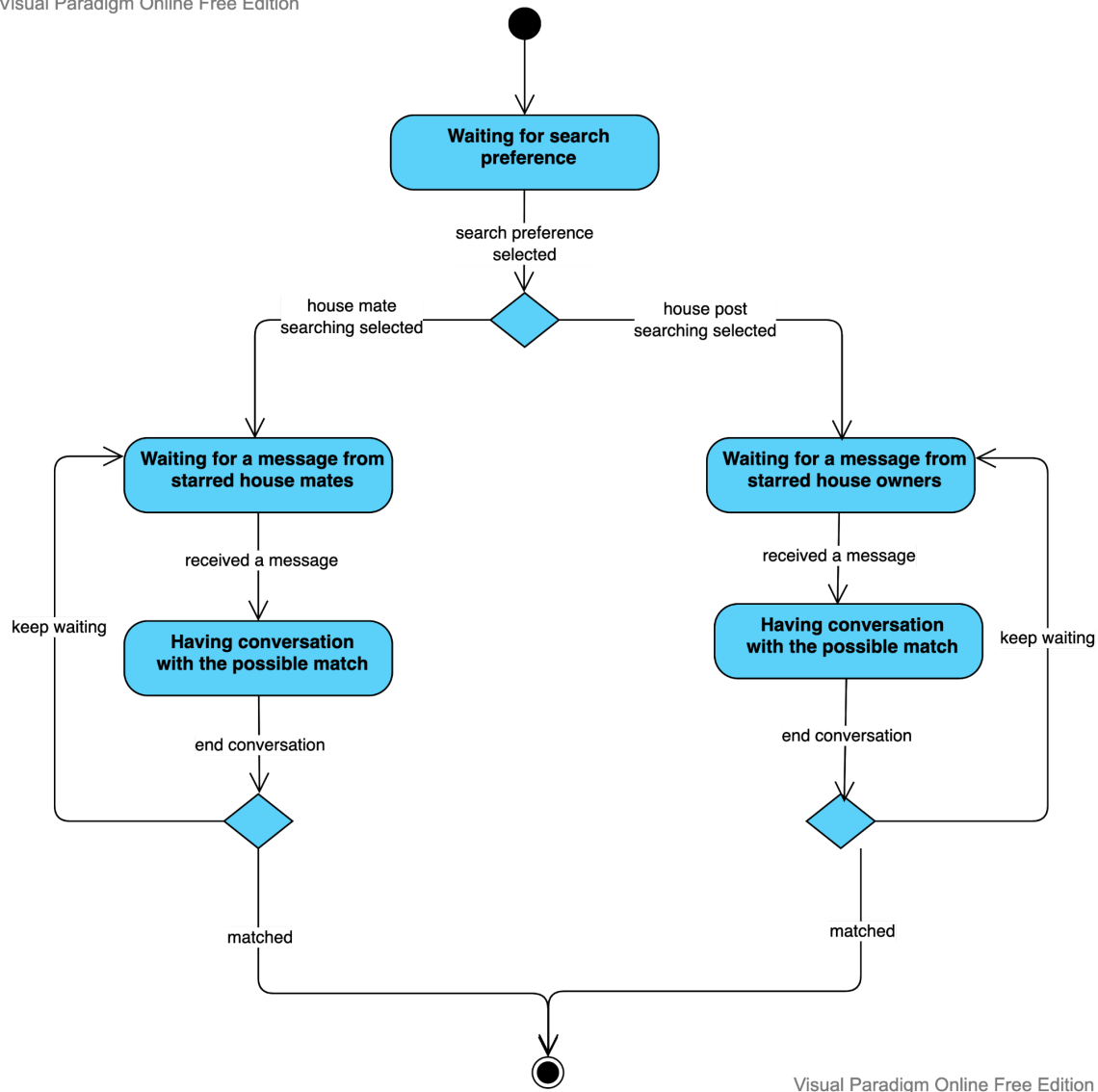


**Figure 14: State diagram for “find & star post” flow**

The second diagram shows the states during the starring posts flow. This flow starts on the already logged-in state, where the user selects their preference of searching. It ends when they like one or more posts.

### 3.6.3 Conversation With Housemate

Visual Paradigm Online Free Edition



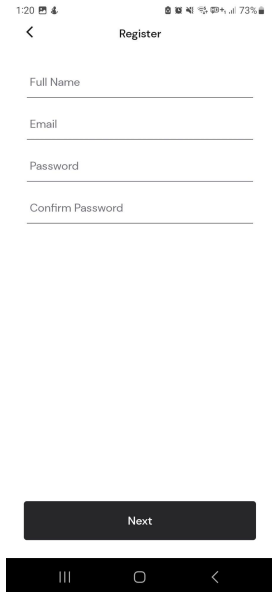
Visual Paradigm Online Free Edition

**Figure 15: State diagram for “conversation with housemate/owner” flow**

The third diagram shows the states for having a conversation with the housemate/owner via a third-party platform. It again starts in the already logged-in state, where the user selects their preference for searching. Then they wait for a message from other users. Either they agree and exit the flow, or they keep waiting for others. It is important to note that “Having a conversation with the possible match” will occur in another app’s task (either email or a chatting app) that will be navigated through our app.

## 3.7 Final User Interface

### 3.7.1 Register



1:20 4 73%

< Register

Full Name

Email

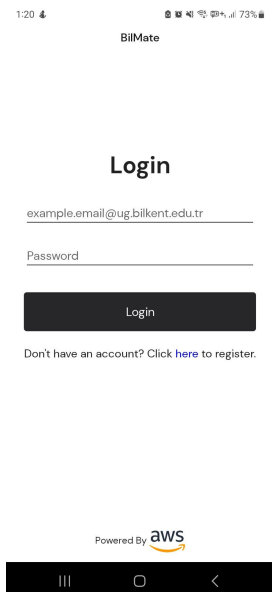
Password

Confirm Password

Next

||| □ <

### 3.7.2 Login



1:20 4 73%

BillMate


Login

example.email@ug.bilkent.edu.tr

Password

Login

Don't have an account? Click [here](#) to register.

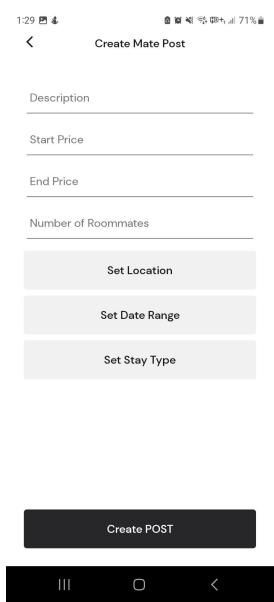
Powered By 

||| □ <

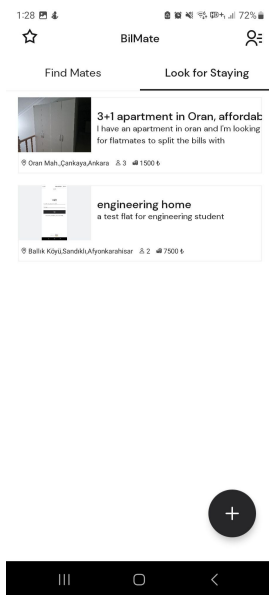
### 3.7.3 Choose Post Type



### 3.7.4 Create Post



## 3.7.5 Check House Seeker Posts

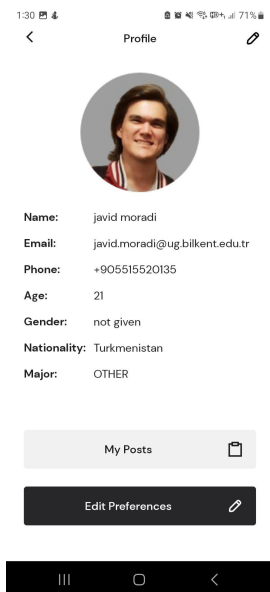


## 3.7.6 Check Your Own Posts

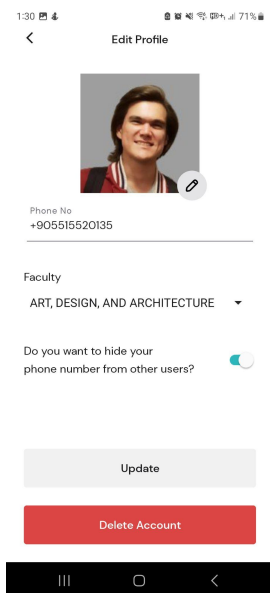




### 3.7.7 Check Your Own Profile



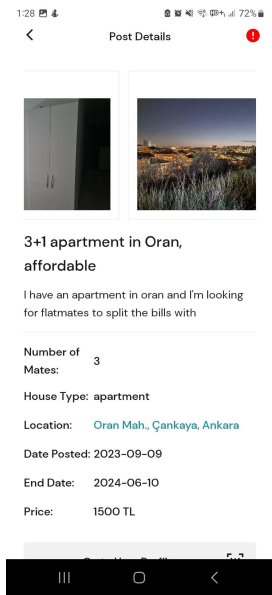
### 3.7.8 Edit Your Own Profile

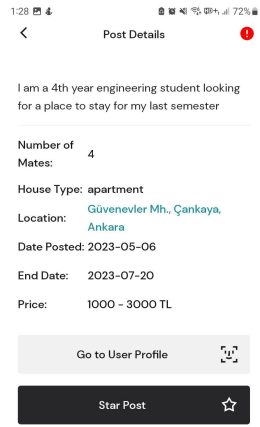


### 3.7.9 Check Another User's Profile

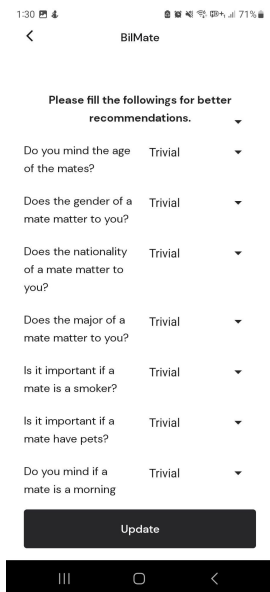


### 3.7.10 Check Post Details

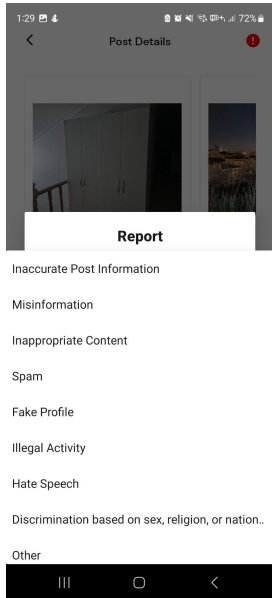
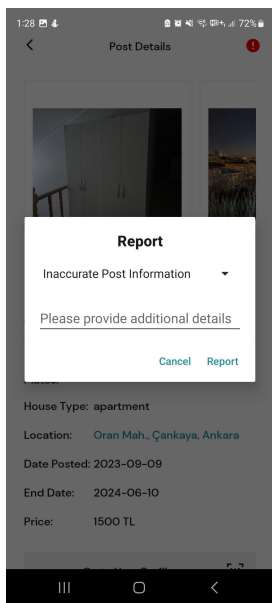




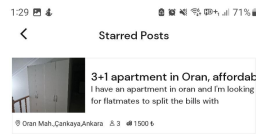
## 3.7.11 Edit Your Preferences



### 3.7.12 Report A Post



### 3.7.13 Starred Posts



## 4. Implementation Details

The BilMate application harnesses the power of modern technologies, creating a multi-faceted solution that addresses the complex problem of finding suitable shared accommodation for university students.

### 4.1 System Architecture

The BilMate application is underpinned by a robust, tri-component system architecture, comprising a backend, a database, and a frontend.

The backend, the heart of our system, is built using Python and FastAPI. FastAPI, known for its high-speed performance and flexibility, serves as our primary tool for constructing an efficient API. This API forms the conduit between our Python backend—where we craft our database models and perform associated operations—and our MongoDB database. MongoDB, a non-relational database, is particularly adept at handling large volumes of data and enabling swift transactions, thus fitting our requirements for storing diverse user data and post data perfectly.

In addition to the MongoDB database, we incorporate Amazon S3 into our system architecture to manage our image data storage. To streamline our operations and separate large file handling from MongoDB, our Python backend communicates directly with Amazon S3, which provides a flexible, scalable, and cost-effective solution for storing large image files.

Our frontend is developed in Java, using the Android Studio environment. This part of the architecture communicates with our FastAPI backend to fetch and deliver data to the users in a seamless, efficient manner, ensuring a top-tier user experience on our mobile application.

## 4.2 Code Structure and Design

The codebase of BilMate demonstrates strategic organization and systematic design principles, promoting operational efficiency and maintainability. Our API calls are handled through classical routing, triggering dedicated Python functions that perform the necessary operations as requested by the frontend.

We have meticulously structured the models for our diverse classes in Python, creating a well-ordered and intuitive codebase. This solid framework not only facilitates easy navigation through the code, but also simplifies the process of implementing future updates and enhances the maintainability of the code.

## 4.3 API Design

The API of BilMate is robust and comprehensive, providing various endpoints to facilitate an extensive range of functionalities. Key endpoints include those for creating, deleting, and editing house-sharer and house-seeker posts, as well as user creation and deletion. Additionally, our API design incorporates endpoints for post filtering and querying, demonstrating our commitment to providing a user-centric design that caters to a wide array of user requirements.

## 4.4 Database Design

Our MongoDB database is organized into several purpose-driven collections: admin, feedbacks, reports, location, statistics, and users. Within the 'users' collection, we have strategically chosen to store postings as arrays within user documents. This design choice eliminates the need for a separate collection, thereby optimizing database performance.

To handle image data, each posting contains an array of unique image IDs, which correspond to the images stored in Amazon S3. Our 'reports' collection handles the task of post reporting, while the 'statistics' collection stores important app metrics. This detailed database design forms the bedrock of BilMate's administrative capabilities, contributing to the robustness and scalability of our application.

## 4.5 Image Storage

Our application makes use of Amazon S3 for image data storage, a choice driven by the need for a platform capable of handling large files efficiently. Each image uploaded to S3 receives a unique ID. This ID is subsequently stored in the corresponding user posting in our MongoDB database, which aids in the efficient retrieval of images when required.

## 4.6 User Interface

BilMate's user interface, developed using Java within the Android Studio environment, is designed to be user-friendly and intuitive. Our frontend architecture follows a modular design approach. This not only ensures smooth navigation for users but also fosters maintainability and the integration of future functionalities in a systematic manner.

## 4.7 Testing

Throughout the development process, a rigorous regimen of testing was conducted to ensure the app's stability and reliability. Every module underwent comprehensive unit testing to confirm its individual performance. Further, integration tests were performed to validate the interoperability of these modules, effectively ensuring seamless operation of the overall system.

## 4.8 Challenges & Solutions

The journey of developing BilMate was marked by several challenges which we overcame through strategic decision-making and efficient problem-solving.

Our choice of MongoDB as our primary database was driven by the need for swift database transactions, a requirement that this non-relational database handles remarkably well due to its high-speed performance. We faced the challenge of large image storage, for which we turned to Amazon S3. This platform offered a scalable, efficient, and cost-effective solution for storing large image files.

In addressing the issue of inappropriate posts and user regulation, we designed a robust reporting system. This allowed us to effectively moderate content and ensure a safe and respectful platform for our users. To track the growth and user engagement of our app over time, we introduced a dynamic statistics model, providing us valuable insights into user trends and app usage.

## 5. Test Cases and Results

### 5.1 Functional Test Cases

#### 5.1.1 Unit Testing

##### 5.1.1.1 User Signs up with a non-Bilkent email

ID: 1

Test Type: Security

Procedure of Testing Steps:

- Enter a Bilkent email address during signup and complete the signup process to make sure everything works ideally
- Go back to the signup page and now enter a non-Bilkent email address for signup
- Make sure that the signup process halts with a non-Bilkent email



- Make sure there is a user prompt for the reason it halts which is about non-Bilkent email.

Expected Result: The sign-up flow will fail, and the user will not gain access to BilMate

Priority: Critical

#### 5.1.1.2 User tries to sign in without verifying their email

ID: 2

Test Type: Security

Testing Steps:

- Sign up with a valid Bilkent email address.
- Log out and try to sign in before verifying the email.
- Verify that an error message appears informing the user that they need to verify their email before logging in.

Expected Result: The login flow will fail, and the system will instruct the user to verify their email.

Priority: Critical

#### 5.1.1.3 User tries to sign in with the wrong email-password combination

ID: 3

Test Type: Security

- Sign up with a valid Bilkent email address and password.
- Attempt to log in using the correct email but an incorrect password.
- Verify that an error message appears indicating the user has entered an incorrect password.
- Attempt to log in using an email that is not associated with the account.
- Verify that an error message appears indicating that the user has entered an incorrect email or password.

Expected Result: The login flow will fail, and the user will be asked to write the correct email-password combination or use the "Forgot Password" option.

Priority: Critical

#### 5.1.1.4 User tries to log in using the valid credentials

ID: 4

Test Type: Security

- Sign up with a valid Bilkent email address and password.
- Log out and attempt to log in using the correct email and password.
- Verify that the user is able to log in and access the application's main features.

Expected Result: The login follow will succeed granting said user access to BilMate.

Priority: Critical

#### 5.1.1.5 User tries to post an incomplete post

ID: 5

Test Type: Functionality

- Log in to the application.
- Navigate to the post creation page and attempt to create a post without filling in all the required fields.
- Verify that an error message appears indicating that the user must fill in all required fields before submitting the post.
- Fill in all required fields and submit the post.
- Verify that the post appears correctly on the main feed.

Expected Result: The post verification system will not allow the post to go live, and the system will instruct the user to fill in the missing fields.

Priority: Major

#### 5.1.1.6 User tries to post a complete posting

ID: 6

Test Type: Functionality

Procedure of Testing Steps:

- Log in to the application.
- Navigate to the post creation page and fill in all required fields.

- Submit the post.
- Verify that the post appears correctly on the main feed.

Expected Result: The post verification system will allow the post to go live

Priority: Minor

5.1.1.7 User tries to delete their own post

ID: 7

Test Type: Functionality

Procedure of Testing Steps:

- Log in to the application.
- Create a post.
- Click the delete button for the post created by the user.
- Verify that the post is successfully deleted and does not appear on the main feed anymore.

Expected Result: The identity management system will grant the user the privileges of deleting their own post

Priority: Major

5.1.1.8 User tries to delete another user's post

ID: 8

Test Type: Security

Procedure of Testing Steps:

- Log in to the application.
- Create a post.
- Log out and create another user account.
- Attempt to delete the post created by the first user.
- Verify that an error message appears indicating that the user is not authorized to delete another user's post.

Expected Result: The identity management system will reject access to deletion flow for users that do not own the post.

Priority: Critical

5.1.1.9 User tries to update their own post

ID: 9

Test Type: Functionality

Procedure of Testing Steps:

- Log in to the application.
- Create a post.
- Click the edit button for the post created by the user.
- Change the content of the post.
- Submit the updated post.
- Verify that the post is updated and appears correctly on the main feed.

Expected Result: The identity management system will grant the user the privileges of updating his/her post.

Priority: Minor

5.1.1.10 User tries to update another user's post

ID: 10

Test Type: Security

Procedure of Testing Steps:

- Log in to the application.
- Create a post.
- Log out and create another user account.
- Attempt to update the post created by the first user.
- Verify that an error message appears indicating that the user is not authorized to update another user's post.

Expected Result: The identity management system will reject access to (post) update flow for users that do not own the post

Priority: Critical

5.1.1.11 User tries to update their profile

ID: 11

Test Type: Functionality

Procedure of Testing Steps:

- Log in to the application.
- Navigate to the user profile page.
- Click the edit button for the user profile.
- Change the user profile information.
- Submit the updated profile.
- Verify that the user profile is updated correctly.

Expected Result: The identity management system will grant the user the privilege of modifying their account.

Priority: Minor

5.1.1.12 User tries to update another user's profile

ID: 12

Test Type: Security

Procedure of Testing Steps:

- Log in to the application.
- Create a post.
- Log out and create another user account.
- Attempt to update the profile of the first user.
- Verify that an error message appears indicating that the user is not authorized to update another user's profile.

Expected Result: The identity management system will reject access to the (account) updating flow of another user

Priority: Critical

## 5.1.2 Sanity and Smoke Testing

### 5.1.2.1 Duplicate Users

ID: 13

Test Type: Security

Procedure of Testing Steps:

- Attempt to create two user accounts with the same email address.
- Verify that the system does not allow the creation of two user accounts with the same email address.

Expected Result: If a user already registered an account and they attempt to create another account using the same email. The registration system will reject the attempt.

Priority: Critical

### 5.1.2.2 Missing Fields During Registration

ID: 14

Test Type: Functional

Procedure of Testing Steps:

- User fills in incomplete information during registration
- User clicks the register button
- System verifies the information provided

System rejects the request made by the user

Expected Result: If, during registration, the user does not provide full information such as the email, password, major, etc. The registration system will reject the request made by that user

Priority: Critical

#### 5.1.2.3 User inputs a wrong email

ID: 15

Test Type: Functional

Procedure of Testing Steps:

- User provides a wrong email address during registration
- User clicks the register button
- System verifies the email address
- System sends a verification code to the email address provided by the user
- User tries to verify their account using the wrong email address

Expected Result: In this case, the verification code will be sent to the wrong email, and the user will not be able to verify their account

Priority: Major

#### 5.1.2.4 User tries to star a post that does not exist

ID: 16

Test Type: Functional

Procedure of Testing Steps:

- User attempts to star a post that has been soft deleted
- User clicks the star button on the post
- System verifies the isDeleted flag of the post
- Post should not appear in the starred list of any user

Expected Result: Due to the nature of soft removal, a post will remain in the database for a certain amount until it is actually deleted. In order to apply soft deletion, an isDeleted flag is set to true, and the post no longer appears in the results. So in the case that a post has been deleted, it should not appear in the starred list of any user

Priority: Minor

#### 5.1.2.5 User requests a list based on their preference

ID: 17

Test Type: Functional

Procedure of Testing Steps:

- User logs in with valid credentials
- User selects the list preference option
- System retrieves list of postings personalized to the specific user
- System displays the personalized list of postings to the user

Expected Result: Assuming the user provides the valid credentials, a list of postings will be returned that is personalized to the specific user, rather than for other users

Priority: Critical

### 5.1.3 Regression Testing

5.1.3.1 When connection to the database is lost while processing a request

ID: 18

Test Type: Non-functional

Procedure of Testing Steps:

- Disconnect the database while a request is being processed
- Observe the system's behavior
- Check if the system attempts to recover
- Check if the system shuts down gracefully with suitable error messages if the recovery fails
- Verify that the system restarts gracefully
- Ensure that the system works without any side-effects

Expected Result: Due to the nature of Promises and asynchronous code, there are safety mechanisms to ensure in case of unexpected behavior (lost connection, large requests, long loading times, etc.). The system will attempt to recover. If that fails, the system will shut down and restart gracefully with suitable error messages.

Priority: Critical



#### 5.1.3.2 Effect of post creation on the matching algorithm

ID: 19

Test Type: Functional

Procedure of Testing Steps:

- Create a post as user A
- Verify that the post does not appear on user A's recommendation page or anywhere else
- Check if the post appears only on user A's posts page
- Verify that the post does not appear on any other user's page
- Ensure that the matching algorithm works correctly after the post creation

Expected Result: When a user creates a post, let's call him user A. That post should not appear on user A's recommendation page or anywhere else, even if they actively search for it. The only location it should appear in is A's posts page.

Priority: Major

#### 5.1.3.3 Effects of starring a post on the matching algorithm

ID: 20

Test Type: Functional

Procedure of Testing Steps:

- Star a specific post as a user
- Verify that the algorithm re-calibrates the parameters
- Check if the system serves the user better postings in the future
- Ensure that the system works without any side-effects

Expected Result: When a user stars a specific post, the algorithm will re-calibrate the parameters to serve the user better postings in the future.

Priority: Major

#### 5.1.3.4 Privacy Settings

ID: 21

Test Type: Non-functional

#### Procedure of Testing Steps:

- Change the account's privacy settings to high privacy
- Verify that all posts that were made (active and inactive) have the phone number hidden from all the other users of the system
- Check if the system works without any side-effects

Expected Result: When a user decides to change their account's privacy settings to high privacy, all posts that were made (active and inactive) will have the phone number hidden from all the other users of the system.

Priority: Minor

#### 5.1.3.5 User does not indicate preferences

ID: 22

Test Type: Functional

#### Procedure of Testing Steps:

- Create a user without indicating any preferences
- Verify that the system assigns all parameters to be equally important
- Ensure that the system matches users accordingly
- Check if the system works without any side-effects

Expected Result: In this case, the system will assign all parameters to be equally important and match the users accordingly; this will change when either the user manually changes the preferences or through extended use of BilMate's starring system, which includes a feedback loop to improve the algorithm.

Priority: Minor

### 5.1.4 System Testing

#### 5.1.4.1 User signs up

ID: 23

Test Type: Component

#### Procedure of Testing Steps:

- User fills out registration form with valid information and submits it.
- System should verify the email address belongs to a Bilkent University student.
- Verification code should be sent to the user's email.
- User inputs verification code and is directed to their profile.
- User should have no preferences or posts initially.

Expected result: The system should register the user, then after verifying that the email belongs to a Bilkent University student, a verification code will be sent. After verification, the user should be able to access the system with initially no preferences and no posts, until they create a post.

Priority: Critical

#### 5.1.4.2 User scrolls through posts

ID: 24

Test Type: Functional

#### Procedure of Testing Steps:

- User logs in to the system with valid credentials.
- User scrolls through the posts.
- User selects a post and views further details.
- User contacts the owner of the post.

Expected Result: After a user has been authenticated, they can scroll through BilMate's selection of posts; the user can then press on any post and will be redirected to the respective post, which he/she will be able to view further details and contact the owner of the post.

Priority: Major

#### 5.1.4.3 User deletes a post

ID: 25

Test Type: Component

#### Procedure of Testing Steps:

- User logs in to the system with valid credentials.
- User heads to the manage posts section.
- User selects a post to delete.
- User confirms the deletion of the post.
- The post is initially removed from the system with soft removal, followed by hard removal.

Expected Result: A user needs to be authenticated to carry out this action. The post will be initially deleted from the system with soft removal, followed by hard removal. At the point of soft removal, the post will stop appearing for any other users of BilMate.

Priority: Minor

#### 5.1.4.4 Reporting a user

ID: 26

Test Type: Functional

Procedure of Testing Steps:

- User logs in to the system with valid credentials.
- User navigates to the report section.
- User selects a post or user to report.
- User submits the report.
- Admins review the report ticket.
- Admin approves or rejects the report ticket.

Expected Result: If a user of BilMate believes that a post or another user is breaking the ToS of BilMate by uploading content deemed inappropriate, they can report the post or the user to the admins. After a review process, the admin can approve the ticket, thus deleting the post. Or it could be rejected, which results in the post remaining on the platform.

Priority: Major

#### 5.1.4.5 Account deletion

ID: 27

Test Type: Functional

#### Procedure of Testing Steps:

- User logs in to the system with valid credentials.
- User navigates to the account deletion section.
- User inputs a valid password for account verification.
- User confirms the account deletion.
- All posts and account information should be deleted from the system.

Expected Result: In case a user no longer wants to use BilMate, they have the right to be forgotten. After confirmation, all user posts, in addition to their account, will be removed from the system.

Priority: Critical

### 5.1.5 Acceptance Testing

#### 5.1.5.1 Can users register for BilMate

ID: 28

Test Type: Functional

#### Procedure of Testing steps:

- Navigate to BilMate's registration page
- Fill in all the required fields with valid information
- Submit the registration form
- Check if a confirmation message is displayed on the screen
- Verify that a verification email is sent to the provided email address

Expected Result: The users who test BilMate should be able to seamlessly create an account, assuming they have an active Bilkent email

Priority: Major

#### 5.1.5.2 Can users verify their email

ID: 29

Test Type: Functional

Procedure of Testing steps:

- Navigate to the BilMate email verification page
- Click on the verification link in the email sent to the user
- Enter the verification code provided in the email
- Click on the verification button
- Verify that the email is verified successfully

Expected Result: The users should be able to access their email provider and verify their email

Priority: Major

5.1.5.3 Can users create a post

ID: 30

Test Type: Functional

Procedure of Testing steps:

- Navigate to the BilMate post creation page
- Fill in all the required fields with valid information
- Submit the post creation form
- Check if a confirmation message is displayed on the screen
- Verify that the created post is visible on the BilMate home page

Expected Result: The users should be able to fill in a form and create a post, assuming all the fields are valid

Priority: Major

5.1.5.4 Can the users update their post

ID: 31

Test Type: Functional

Procedure of Testing steps:

- Navigate to the BilMate post update page

- Select a post to update
- Modify any of the fields in the post
- Submit the post update form
- Check if a confirmation message is displayed on the screen
- Verify that the updated post is visible on the BilMate home page

Expected Result: After creating a post, the user should be able to update fields such as date, location, price, etc. on their post

Priority: Minor

#### 5.1.5.5 Can the user delete their post

ID: 32

Test Type: Functional

Procedure of Testing steps:

- Navigate to the BilMate post deletion page
- Select a post to delete
- Confirm the deletion
- Check if a confirmation message is displayed on the screen
- Verify that the deleted post is no longer visible on the BilMate home page

Expected Result: After creating a post, the user should be able to delete that post from the system

Priority: Major

#### 5.1.5.6 Can the user star any post

ID: 33

Test Type: Functional

Procedure of Testing steps:

- Navigate to the BilMate post page
- Select a post to star
- Click on the star button

- Verify that the post is starred successfully

Expected Result: The user should be able to star any post he/she finds interesting, however, the only limitation is that the user should not be able to star their own posts.

Priority: Minor

#### 5.1.5.7 Can the user update their own profile

ID: 34

Test Type: Functional

Procedure of Testing steps:

- Login to BilMate
- Navigate to the profile management section
- Update several fields such as password, number, preferences, etc.
- Save the changes

Expected Result: The user should be able to update several fields, such as password, number, preferences, etc., in the profile management section of BilMate.

Priority: Major

#### 5.1.5.8 Can the user delete a post

ID: 35

Test Type: Functional

Procedure of Testing steps:

- Login to BilMate
- Navigate to the post that the user wants to delete
- Click on the delete button
- Confirm the deletion

Expected Result: The user, assuming they are authenticated, should be able to delete their account from the system.

Priority: Major



#### 5.1.5.9 Can the user send another verification email

ID: 36

Test Type: Functional

Procedure of Testing steps:

- Login to BilMate
- Navigate to the verification email section
- Request another verification code to be sent to their email
- Check the email for the new verification code

Expected Result: If the verification email expired, the user should be able to request another verification code to be sent to their email.

Priority: Minor

#### 5.1.5.10 Can the user upload images to their posts

ID: 37

Test Type: Component

Procedure of Testing steps:

- Login to BilMate
- Navigate to the create post section
- Fill in the post details
- Upload picture(s)

Expected Result: The newly created post should have the images displayed.

Priority: Critical

#### 5.1.5.11 Can the user report a post

ID: 38

Test Type: Functional

Procedure of Testing steps:

- Login to BilMate

- Navigate to the view postings section
- Report a user by including a reason and description of the offense

Expected Result: The reporting request will be sent and allocated to an admin for review.

Priority: Major

#### 5.1.5.12 Can an admin review and accept a reporting ticket

ID: 39

Test Type: Functional

Procedure of Testing steps:

- Login to BilMate's admin page
- Navigate to the review reports section
- Press on a report request and view the details
- Accept the report request, thus approving that it is a valid offense

Expected Result: The offending post will be deleted, and the owner of that post will be warned.

Priority: Major

#### 5.1.5.13 Can an admin review and reject a reporting ticket

ID: 40

Test Type: Functional

Procedure of Testing steps:

- Login to BilMate's admin page
- Navigate to the review reports section
- Press on a report request and view the details
- Reject the report request, thus approving that the post does not break the ToS

Expected Result: The offending post will remain active on BilMate.

Priority: Major

### 5.1.6 Input validation

#### 5.1.6.1 Registration

ID: 41

Test Type: Functional

Procedure of Testing steps:

- Navigate to the registration page
- Attempt to submit a registration form with invalid inputs for each of the required fields (email, phone number, nationality, university major)
- Verify that an appropriate error message is displayed for each invalid field
- Submit the form with valid inputs for all required fields
- Verify that the user is successfully registered and redirected to the login page

Expected Result: Several fields during registration need to be validated. As previously mentioned, all the emails must end with `bilkent.edu.tr`. Additionally, a phone number must start with the international code (+90 in the case of Turkey) and be of a specific length. Nationality and university majors are already limited to a specific list of enumerables; this ensures consistency among all users.

Priority: Major

#### 5.1.6.2 Updating Fields

ID: 42

Test Type: Functional

Procedure of Testing steps:

- Log in to BilMate with valid credentials
- Navigate to the profile management section
- Attempt to update each of the optional fields (password, preferences, importance)
- Verify that an appropriate error message is displayed for any invalid input
- Submit the form with valid inputs for all optional fields
- Verify that the updated fields have been successfully saved to the user's profile

Expected Result: While updating, there are optional fields that they could leave as-is, or update them to improve their experience better. For instance, they can update their password, which will need to be confirmed. Moreover, they have the option to update their preferences and the importance of those parameters. The importance will be validated against a list of choices, with the same process applying to the preferences themselves.

Priority: Medium

## 5.2 Non-Functional Test Cases

### 5.2.1 Performance Testing

ID: 43

Test Type: Performance Testing

Procedure of Testing steps:

- Set up a sizable database with a certain amount of data.
- Send queries to the backend with increasing complexity and data size.
- Measure the response time for each query.
- If the response time is above 500 ms, investigate and optimize the query or the database performance.
- Once the data size reaches a certain upper threshold, ensure that the response time is at most 1000 ms.

Expected Result: The backend should respond to a query within 500 ms at most with a sizable database size. If the stored data reaches a certain upper threshold, the response time should be 1000 ms at most.

Priority: Critical

### 5.2.2 Usability Testing

ID: 44

Test Type: Usability Testing

Procedure of Testing steps:

- Recruit human users to test the application.
- Observe how they use the application and take note of any difficulties or confusion they experience.
- Ask users to perform specific tasks and monitor their progress.
- Evaluate the ease of use, logical transitions between pages, and the quality of error messages.
- Assess the consistency and informativeness of the GUI.

Expected Result: The usability of the application will be evaluated by human users. They should be comfortable using the application. To ensure this, the application should be easy-to-use, have logical transitions between pages, and give meaningful error messages. The GUI of the application should be informative and consistent.

Priority: Major

### 5.2.3 Stress Testing

#### 5.2.3.1 High Load

ID: 45

Test Type: Stress Testing

Procedure of Testing steps:

- Simulate simultaneous access by all Bilkent users to BilMate
- Monitor server uptime using AWS tools
- Verify that table partitioning and load balancing have ensured the server stays up

Expected Result: Assuming all Bilkent users access BilMate simultaneously, table partitioning and load balancing offered by AWS will ensure that our server stays up. This is a worst-case scenario; a more realistic scenario will not have any issues regarding server uptime.

Priority: Critical

#### 5.2.3.2 High Stress

ID: 46

Test Type: Stress Testing

Procedure of Testing steps:

- Use bots to flood the server
- Verify that the spam detection system catches the behavior and rejects further attempts to create/delete posts until a timer expires

Expected Result: The spam detection system should be effective in catching bot behavior and preventing server overload. If a user uses a bot to flood the server, our spam detection system will catch the behavior and reject further attempts to create/delete posts until a timer expires.

Priority: Major

## 5.2.4 Security Testing

ID: 47

Test Type: Security Testing

Procedure of Testing steps:

- Attempt to gain unauthorized access to BilMate's GitHub repository, MongoDB Atlas instance, or AWS EC2 server using various methods, such as brute force attacks, injection attacks, and cross-site scripting attacks.
- Monitor whether the respective service alerts all team members and automatically secures the login portal to stop any further attempts of breaking in.
- Verify whether the system uses JWT authentication tokens to ensure that only authorized users can perform protected actions, such as deleting or updating personal information.

Expected Result: In case an adversary attempts to gain unauthorized access to either BilMate's GitHub repository, MongoDB Atlas instance, or AWS EC2 server, the respective service will alert all team members and automatically secure the login portal to stop any further attempts of breaking in. The system also uses JWT authentication tokens to ensure that only authorized users can perform several protected actions, such as deleting or updating personal information.

Priority: Critical

## 5.2.5 Portability Testing

ID: 48

Test Type: Portability

Testing Procedure of Testing steps:

- Install BilMate on Android and IOS platforms.
- Use the application on both platforms to verify its functionalities.
- Check if the application works in the same way on both platforms and doesn't give platform-dependent errors.

Expected Result: Users may have different operating systems, Android or IOS. The application should work in the same way and should not give platform-dependent errors. The functionalities of the application should not be affected by the platform in which it is used.

Priority: Major

## 5.2.6 Maintenance Testing

ID: 49

Test Type: Maintenance

Testing Procedure of Testing steps:

- Add new functionalities and modify existing ones in the application
- Run maintenance testing on the altered and added features
- Check that the altered and added features work as expected and do not cause any side-effects for the unaltered features

Expected Result: After the launch of the application, it will require several updates for adding new functionalities and altering the existing ones. Those updates are supposed not to affect the work of unaltered features and cause any side effects. Maintenance testing ensures that the altered and added features work as expected and do not cause any side effects for the other parts of the system.

Priority: Major

#### 5.2.6.1 Confirmation Testing

ID: 50

Test Type: Confirmation

Testing Procedure of Testing steps:

- Modify functionalities and features in the application
- Run confirmation testing on the modified functionalities and features
- Check that the modified functionalities and features work properly and expectedly

Expected Result: Confirmation testing tests the modified functionalities and features to ensure they work properly and expectedly.

Priority: Major

#### 5.2.6.2 Regression Testing

ID: 51

Test Type: Regression

Testing Procedure of Testing steps:

- Modify functionalities and features in the application
- Run regression testing on the remaining system with unmodified functionalities and features
- Check that the modifications do not affect the running of the system and do not cause any side-effects

Expected Result: Regression testing tests the remaining system with unmodified functionalities and features to ensure modifications do not affect the running of the system and do not cause any side-effects.

Priority: Critical

#### 5.2.7 Recovery Testing

ID: 52

Test Type: Recovery



Testing Procedure of Testing steps:

- Simulate different failure types, including power interruption, network failures, database overload, and hardware failures
- Check that the system terminates gracefully without data corruption in case of failure

Expected Result: In the case of failure, the system should terminate gracefully without data corruption. Recovery testing contains simulations of different failure types, including power interruption, network failures, database overload, hardware failures...

Priority: Critical

## 5.2.8 Reliability Testing

ID: 53

Test Type: Reliability

Testing Procedure of Testing steps:

- Test the system in a particular environment for a fixed period of time
- Ensure that the system works properly without any failure in that environment
- Ensure that the system yields the same output in the same setup to ensure its reliability
- Test the system according to its maximum workload, the proper execution of each operation and feature, and the working of an entire system after a bug fix

Expected Result: The system should work properly without any failure in a particular environment for a fixed period of time, and it should yield the same output in the same setup to ensure its reliability. The system should be tested according to its maximum workload, the proper execution of each operation and feature, and the working of an entire system after a bug fix.

Priority: Major

# 6. Maintenance Plan and Details

Regarding the financial maintenance and sustainability of the project, it is planned to implement a donation-based approach in the future, where each user is kindly asked to spare a small

amount of money if they are satisfied with the application. Note that donation is only optional and provides no advantage to users who decide to donate money. This way, the aim is to compensate for the hosting and web services' cost.

To achieve maintenance from the view of the users, updates regarding the UI or functionality of the application will be released at regular intervals. Noteworthy that these updates will be done in accordance with the feedback of our user base.

Initially, regarding the regular intervals, unless a major bug or issue arises, it is planned to release a patch every two weeks. If an error that impedes the usage of the application occurs, regardless of this regulation, the application will be reverted back to the older version as swiftly as possible. The versions of the application will be stored via version control software.

Lastly, each team member will continue in their current position, i.e., backend and frontend developer, and each will be responsible for their corresponding codes' monitoring and testing. This maintenance is planned to be kept until the end of the fall semester of 2023-2024, and future maintenance plans of the application will be updated according to the statistical data collected heretofore.

## 7. Other Project Elements

### 7.1 Consideration of Various Factors in Engineering Design

This section outlines some factors that might impact the project's development.

#### 7.1.1 Public Health

Issue of public health does not affect the project. We expect users' own control in case of a health-related situation. A house-sharer and house-seeker must plan according to their current health situation.

#### 7.1.2 Public Safety

To ensure the safety of users, a reporting feature will be added to the project. In case of encountering a malicious post, a user can report the post with a comment explaining the reasoning behind the report. This request will be taken and sent to a list of reported posts,

where each post contains a counter displaying the number of times that post is reported. An admin of the application will observe each post and corresponding comments, and if feasible, the reported user and their posts will be deleted and will be banned for an indefinite duration. Also, as part of the project's terms, no user data will be publicly shared or with an untrusted third-party app.

### 7.1.3 Public Welfare

Primary aim of the project is to connect students willing to share their house with students willing to stay in a student house to reduce their costs and enjoy a better accommodation environment. Besides this user life improvement, to enhance the app's usability, the application's lightness, and a fluent interface will be a primary design concern. Thus, comfort for users will be provided both on the application side and in real life.

### 7.1.4 Global

As mentioned in the previous subsection, the project's target audience is university students who own a house and/or are willing to stay in one of those houses. Most of the universities in Turkey teach in English, but to extend the scope and ease the use for those who do not know much English, in addition to the application's primary language, which is English, Turkish might be added in later stages of development.

### 7.1.5 Cultural

For Turkish and many other international students, staying in a student house is an accepted habit; thus, we do not expect many issues from the cultural side. However, further preference options could be added to the preferences list, such as users' religion, diet, etc., to enhance the output of the matching algorithm.

### 7.1.6 Social

The project expects a user to provide their email and phone number, which will be used to connect a house-seeker and sharer. Unless specified, both contact information of a user will be displayed publicly, but a user can hide their phone number, which is more prone to malicious acts. If they do, their email addresses will be the only way to connect a house-sharer and seeker. This way, even though there might be a little drop in UX, users' security will be provided.

### 7.1.7 Environmental

Since some students will be allocated to houses rather than dormitories, this will enhance the capacity of the student dorms. This way, those who are in actual need can stay in dorms, and others can stay with some house-sharer by increasing their accommodation comfort while reducing their bills.

### 7.1.8 Economic

BilMate will be a free-to-use application since its launch. However, donation methods, such as “buy me a coffee” might be added to provide a way of income to developers.

	Effect Level	Effect
Public Health	0	We expect users' own intervention in case of a public health situation
Public Safety	8	Forbiddance of users with malicious intents, Protection of users' data
Public Welfare	7	Lightweight application, Outstanding User Interface along with User Experience
Global	2	Extend the language of the project
Cultural	1	Adding extra required preferences to the preference list
Social	8	Hiding a user's phone number, Providing a way of communication in case of

		hidden user phone number
<b>Environmental</b>	0	Scope of the environment is not of part of the project
<b>Economic</b>	1	Add donation section for a way of income

**Table 1: Factors that can affect analysis and design**

## 7.2 Ethics and Professional Responsibilities

- We will follow the ACM Code of Ethics and Professional Conduct throughout the project.
- Since this application is designed to be used by students, login of non-students should not be allowed by verification of being a student. This verification will be done by enabling signup with only university emails.
- The users' data should not be shared with third-party companies without the user's permission.
- In the case of abuse and misuse of the program, such as acts of violence and posting irrelevant postings, the involved users are banned from the system and will not be allowed to use the application again.
- The data of the users should be considered sensitive data and be kept secure using appropriate encryption algorithms, and the application should use a “data protection by design and default” strategy.

## 7.3 Judgements and Impacts to Various Contexts

During the course of our project's design and implementation, we have made several decisions that have had far-reaching implications in global, economic, environmental, and societal spheres.

Context	Impact Level	Explanation
Public Safety	9/10	We care about the safety of our application, so we created a reporting system to ban users with malicious intents
Global	3/10	We may add different language options in the later stages

Ethical	10/10	The privacy of the users should be ensured. User reports should be examined carefully.
Environmental	3/10	Sharing the same place is a way of resource-sharing and consequently reduces carbon emissions.
Economic	1/10	In the later stages of the project, we may add a donation section
Societal	7/10	We respect people's lifestyles and add extra preferences to find the best match. We provide a solution to housing challenges, one of the societal problems of university students.

## 7.4 Teamwork Details

### 7.4.1 Contributing and functioning effectively on the Team

As a team, we managed to construct a collaborative working environment. Everyone in our group took on the responsibility and acted accordingly. Group meeting sessions were held when required, and further discussions regarding new responsibilities and work division were made. During the meeting sessions, each person was required to voice his/her opinion regarding decisions such as adding new features or removing out-of-scope functionalities in addition to the software architecture approach. This ensured that everyone contributed equally to the success of our project.

Ahmet, Ebrar, and Atasagun are part of the team responsible for the development of the project's back-end, while Javid and Onuralp were responsible for the front-end section of the project. Ahmet's primary contribution was designing the backend's workflow, setting up AWS for future hosting, creating the database schema, and preparing interfaces for front developers to access data. Ebrar's primary responsibility was to develop the security (authentication and authorization) section of the APIs, namely, providing JSON Web Tokens (JWT) along with some other enhancements regarding the back-end's security. Meanwhile, Atasagun's responsibilities

were to provide user interface design of the project, and logo design, ensuring fast response times when retrieving and storing images in AWS S3, and creating the admin portal and the reporting system.

Per the provided UI design of Atasagun, Javid and Onuralp divided the project pages and assigned their sections accordingly. Due to Onuralp's experience in Android development, he always shed light for Javid, a novice in this field. Both, in coordination and communication, developed many pages of the project in accordance with the object-oriented software development principles, such as classes, objects, inheritance, etc. Note that the front-end team used third-party libraries to enhance the functionalities of the front-end section.

#### 7.4.2 Helping to create a collaborative and inclusive environment

As mentioned in the previous subsection, we managed to construct a collaborative environment where each individual was assigned a task, and with it comes responsibility. In addition, an inclusive environment, where each team member can freely express their ideas regarding any aspect of the project, was ensured from the beginning. All group members being in the same department helped a lot in the process of this environment's creation. Additionally, the majority of the members' connection from the past also contributed to this construction.

Ahmet, the primary organizer of the group, mostly helped in terms of the establishment of the group's internal systems. Onuralp, for the most part, acted as a mentor in the front-end section's development process for Javid, taking all queries from him seriously. Other members contributed much to ensure that the group has a friendly tone.

All in all, our group is a collaborative and inclusive environment where all individuals' ideas and opinions are taken seriously and welcomed, regardless of their gender, religion, race, or any other aspects of their life. In this project, everyone is equal and shares the same responsibility. The success of the project is attributed to everyone on the team.

#### 7.4.3 Taking the lead role and sharing leadership on the team

Ever since the beginning of the project, we agreed that we wouldn't have a leader or leaders in the traditional sense. As we are all adults with full autonomy and ideas that will contribute to the

project's success, we have decided to take a more flexible approach to manage the work division and the general organization of the project.

The team divided the 5-member team into two sub-teams, one responsible for the back-end development (API, Database, AWS integration) and the other responsible for the front-end development. In order to avoid micromanagement<sup>1</sup>, an unofficial leader for each group was chosen by consensus and their merit in their respective fields. But it is important to reiterate that all the team members are equal and have to contribute equally to the project's success.

For instance, in the front-end team, Onuralp, being the one with the most experience in Android development, carried out the tasks of a team lead by choosing the technology and setting up the project according to object-oriented practices. It follows that Onuralp was mainly concerned with setting up a strong foundation for the project. Meanwhile, Javid was crucial in managing communication between the front-end and back-end teams. This role is crucial as a miscommunication error could result in severe delays and possible financial losses.

On the other side, in the back-end team, each member had a specialty. As such, depending on the topic at hand, one person's opinion had more merit as they had more experience. For example, Atasagun specializes in UI design and ensuring a smooth UX experience, so he was tasked with designing the UI, and we all agreed that he was the fittest out of all the team members. While Ebrar had years of experience designing secure APIs and web systems, it was in the project's best interest to delegate the job of the app's security to Ebrar as she was the most competent. Finally, Ahmet had some experience in developing full-stack applications and database design, as well as experience with AWS services. As such, we agreed to assign the job of setting up the project, designing the database, and integrating it with AWS to him.

#### 7.4.4 Meeting objectives

We succeeded in establishing a safe and user-friendly house-sharing program for college students over the entire BilMate project. We created reliable backend systems, put in place strong security measures, and created a simple user interface. The functionality, performance, and dependability of the application were all guaranteed by our stringent testing procedures. We also created a supportive and inclusive team climate that encourages candid dialogue and

---

<sup>1</sup> <https://www.investopedia.com/terms/m/micro-manager.asp>



equitable involvement. By achieving these goals, we have set the stage for offering students a dependable platform to interact, share housing, and improve their university experience.

## 7.5 New Knowledge Acquired and Applied

BilMate was a great opportunity for each team member to learn many software engineering practices. To remind the used technologies, Kotlin is used for frontend development, and FastAPI, along with MongoDB, is used for backend development. In addition to these, AWS is used for hosting, and Google Play Store to deploy and launch the application.

Some learned practices in Android development are usages of fragments, lifecycle principles, management, communication with the server, encapsulation and usage of retrieved data, encapsulation and dispatch of data to the server side, UI elements and their usage, and many other functionalities. All these newly learned knowledge are applied to develop the BilMate application as a professional level Android application.

Regarding the backend, API development, integration with NoSQL database, error handling and notification, integration with AWS and maintenance, security and identification, and many other applications regarding server side development are learned, in which ultimately resulted in a stable and running backend.

In addition to mentioned technical knowledge, as a team, we learned some soft skills, such as taking responsibilities, work division, teamwork, deadline management, and other skills that are required to effectively sustain a team with a shared goal. Noteworthy that the mentioned skills are applied whenever deemed necessary. All in all, we learned a lot of new skills, and with no doubt that these skills will have a major impact in the upcoming professional working life.

## 8. Conclusion and Future Work

In conclusion, our team's experience with the BilMate project has been both fascinating and demanding. We created and successfully launched a house-sharing application exclusively for

college students during the development process. We have discussed a number of project-related topics, such as functional and non-functional requirements, testing, maintenance schedules, moral ramifications, collaboration dynamics, and the discovery and use of new information.

The success of the project can be attributed in large part to the inclusive and collaborative culture we built inside our team. We have been able to properly delegate tasks and make choices by appreciating each team member's knowledge and contributions. As a result of this strategy, we have improved the quality of our work while simultaneously advancing our personal and professional development by learning more about software engineering best practices, Android programming, backend development, and project management.

We have made sure, through thorough testing, that the BilMate application satisfies the functional requirements and operates dependably in a range of circumstances. In order to confirm the application's functionality, user experience, security measures, and general robustness, we have also carried out non-functional tests, such as performance, usability, stress, security, portability, maintenance, recovery, and reliability testing. Our trust in the stability and functionality of the program is based on the favorable results of these tests.

We have considered public benefit, worldwide reach, cultural variety, social dynamics, and ethical obligations while assessing how the project will affect various environments. We have made sure that our users are secure, happy, and inclusive by putting in place tools like a reporting mechanism, data privacy safeguards, and preferences to improve user experience. Additionally, BilMate encourages resource sharing and lowers carbon emissions by increasing the capacity of student housing, therefore the project has had a positive environmental impact.

There are various opportunities for more work and improvements to the BilMate application in the future. To accommodate people who do not speak English well, one area of concentration could be to increase the number of available languages. Turkish is one of the new languages we've added, and by doing so, we can increase the application's user base.

We can also look into further customization possibilities to improve the matching algorithm. We can increase the matching process' accuracy and give consumers more specialized

house-sharing options by taking into account other preferences like religion, dietary restrictions, or other pertinent criteria.

The application's long-term success and viability depend on ongoing maintenance and updates. According to our maintenance plan, we will issue regular upgrades to the application's UI, functionality, and security based on user feedback and changing demands. The application will continue to be current, user-friendly, and safe thanks to these upgrades.

Regarding monetary sustainability, our proposal to develop a donation-based strategy can assist in defraying the expenses related to hosting and web services. We may guarantee the application's ongoing functioning and improvement by encouraging users to voluntarily pay a small sum of money if they are satisfied with the application.

Additionally, the project's codebase can provide as a starting point for additional study and advancement in the area of house-sharing software. We may continue to innovate and improve the BilMate application as technology advances and new features become available in order to satisfy evolving customer demands and expectations.

In conclusion, our team has enjoyed working on the BilMate project because it has given us the chance to put our knowledge to use, discover new technologies, and effectively interact. We have created a house-sharing program that promotes user experience, safety, and inclusion while addressing the unique needs of university students. We have laid a strong foundation for the further development and success of the BilMate application by taking into account varied circumstances, ethical obligations, and the learning of new knowledge. We want to offer students an outstanding house-sharing experience that will help them save money, improve their housing alternatives, and create a welcoming community through ongoing upkeep, renovations, and prospective expansions.