

**Report Title : My Contribution to The Smart Living
Community Project**

Course Code: CSE 404

Course Title: Software Engineering and ISD Laboratory

Submitted by

SHANJIDA ALAM(ID: 353)

Submitted to

Dr. Md. MUSHFIQUE ANWAR, Professor
Dr. Md. HUMAYUN KABIR, Professor



Computer Science and Engineering
Jahangirnagar University
Dhaka, Bangladesh

January 06, 2025

Contents

1	Abstract	1
2	Introduction	2
3	My Contribution to The Project	3
3.1	Profile Management	3
3.2	Submit Complaints	9
4	Measuring My Impact with Toggl Track	16
4.1	Toggl Track for Manage Profile	16
4.2	Toggl Track for Submit Complaints	18
5	Tools and Practices	20
5.1	Discord	20
5.2	Trello	21
5.3	Toggl	21
5.4	Version Control	21
5.5	Development Tools	22
6	Learning Outcomes	23
6.1	Technical Skills	23
6.2	Soft Skills	23
7	Team Collaboration and Outcome	25
8	Conclusion	26

Chapter 1

Abstract

Every project is not just a task; it offers a chance for learning, growth, and self-discovery. My journey in this project was not simply fulfilling responsibilities; it was about contributing to a collaborative effort that helped to complete the full project successfully. This project provided us with valuable opportunities to develop essential soft skills such as patience, leadership, punctuality, effective teamwork, and time management. Moreover, it fostered an environment where we could effectively solve any problem through collaboration and consultation with our teammates.

This report provides an extensive overview of my role in the team project, highlighting my key contributions, reflecting on the learning process, and evaluating team collaboration.

Chapter 2

Introduction

We live in the technological era. In this era, technology is ubiquitous. We begin and end our days with technology. We utilize technology to enhance our quality of life. In today's fast-paced world, there is a strong drive to integrate technology into every sector. However, challenges arise in managing residential communities. From delayed service requests and security concerns to a lack of communication between residents and management, these issues can disrupt the harmonious living environment within the community. The **LivSmart** community addresses these pain points by engaging technology to create an efficient and connected ecosystem.

LivSmart, a **Smart Living Community** platform, is designed to manage residential community operations seamlessly. Residents can easily submit service requests, track their progress, and stay updated on community activities through features including profile management, payments, event management, security monitoring, and complaint submission. During my involvement, I significantly contributed to developing key features, particularly **Profile Management** and **Complaint Management**, ensuring a user-friendly experience. **LivSmart** transforms residential living into a hassle-free and collaborative experience.

The main goal of the **Smart Living Community** platform is to build a user-friendly platform for the resident to make their daily life easy and comfortable. It also saves the resident time by submitting their problems in this platform. By using this platform, residents easily up-to-date their daily activities. Residents can update their profile when they feel to need. They submit their issues or problems through the service request. Even if they have any complaint about the community system, they can submit their complaint through the complaint management feature. The special feature of this system is monitoring the security log. Finally, the system ensures the user-friendly, organized and secure community.

Chapter 3

My Contribution to The Project

3.1 Profile Management

- **Integration of Core Fields :**

- Added fields like NID/Birth Number to ensure proper identification of residents.
- Enabled real time data update in Firebase-Firestore, ensuring accuracy and immediate accessibility.
- Designed and developed a profile management feature enabling residents to update personal and property information.
- If residents log in to the system after a few months and try to manage their profile in a different time zone, then the system will notify them with the proper reason.

- **Validation and Usability:**

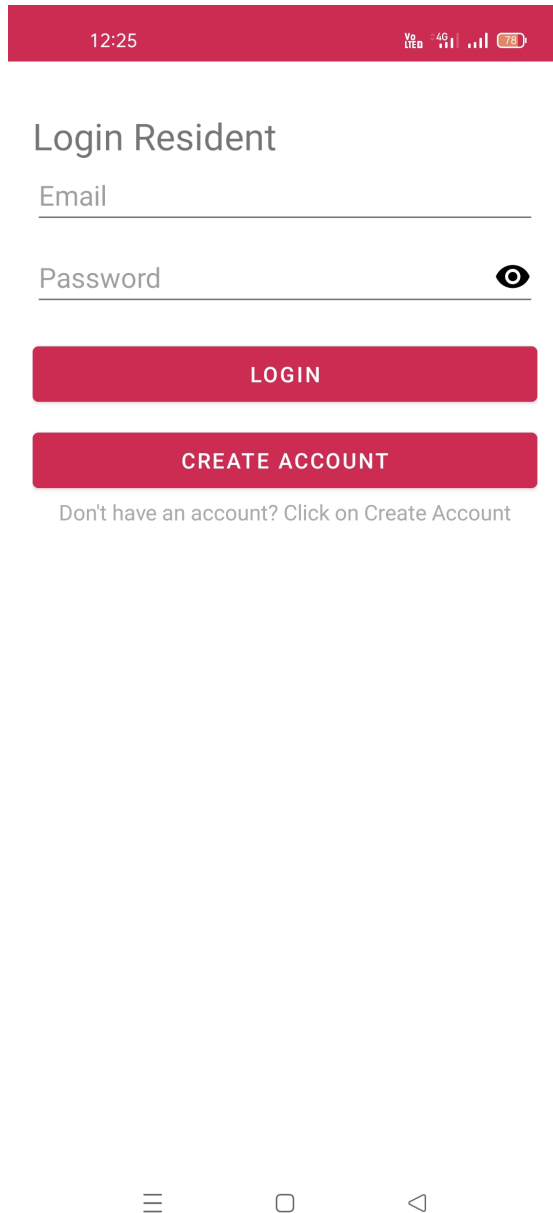
- Implemented strong validation to ensure that all mandatory fields are completed before submission.
- Enhanced the user interface to ensure residents can easily update their profiles, providing a seamless and user-friendly experience.

- **User Role for Data Handling:**

- Stored additional fields such as **unitCode**, **additionalUnitCode**, and **user-Role** in the database for backend operations, even though these were not displayed on the UI.

- **Visual Aspect of The Project:**

- **Login Page:**



The image shows a mobile application interface for a login page. At the top, there is a status bar with a red background, displaying the time 12:25, signal strength, 4G LTE, and a battery icon at 78%. Below the status bar, the title "Login Resident" is centered. Under the title, there are two input fields: "Email" and "Password". The "Password" field has a toggle icon (an eye) to its right. Below the input fields, there are two red buttons with white text: "LOGIN" and "CREATE ACCOUNT". Below the "CREATE ACCOUNT" button, there is a link that says "Don't have an account? Click on Create Account". At the bottom of the screen, there are three navigation icons: a hamburger menu, a square, and a back arrow.

Figure 3.1: This is the login page for residents. Here residents provide the registered Email and Password to login the system

– **Dashboard:**

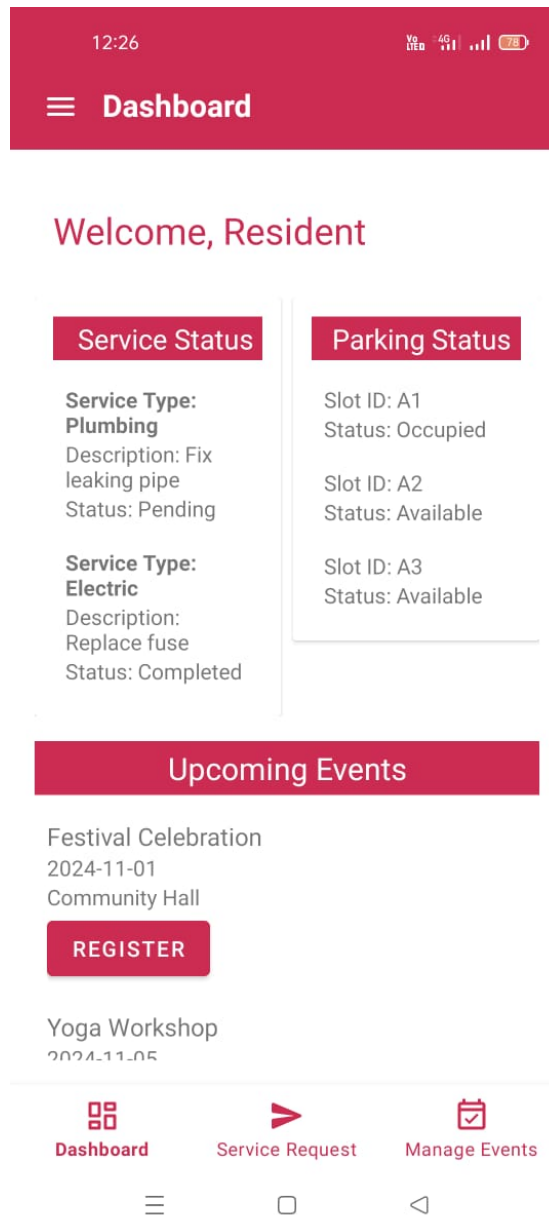


Figure 3.2: This is the dashboard page for residents. As a resident after successfully login to the system then the system turn to the resident this page. Dashboard page basically shows the overall summary of the resident at a glance.

– Feature List of The System:

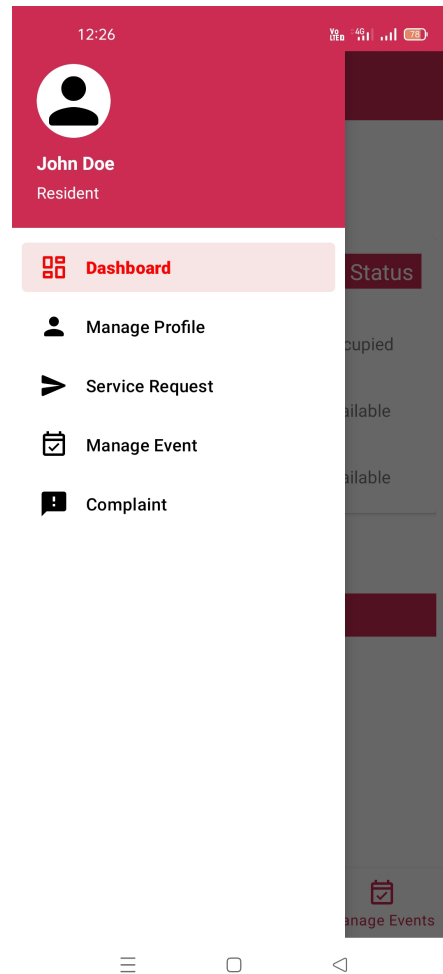


Figure 3.3: This is the side nav bar page for residents. Upon successful login, residents can access the side navigation bar for quick and easy access to essential features. This bar basically displays the basic information of the resident profile including their image, name and role. It also allows residents navigate to the dashboard, manage profile, service request, manage event and complaint.

– **Mange Profile:**

The screenshot displays the 'Manage Profile' interface of a mobile application. At the top, a red header bar contains a hamburger menu icon and the text 'Manage Profile'. Below this, the section is titled 'Resident Profile'. A circular profile picture placeholder with a pencil icon is centered. The form contains five input fields, each with a label and a value:

- Full Name:** Shanjida
- NID/Birth Number:** 7163683930493
- Email:** jucse29.353@gmail.com
- Contact Number:** 01521704753
- Emergency Contact Number:** 01877306914

The bottom of the screen features a navigation bar with three items: 'Dashboard' (with a grid icon), 'Service Request' (with a right-pointing arrow icon), and 'Manage Events' (with a calendar icon). Below the navigation bar are three standard Android navigation icons: a hamburger menu, a square home button, and a left-pointing arrow.

Figure 3.4: This is the **Resident Profile** page. Upon successful login, residents can access the manage profile page for updating their information like name, email, contact number, password etc.

The screenshot shows a mobile application interface for managing a user profile. At the top, a red header bar contains a hamburger menu icon and the text "Manage Profile". The status bar at the very top shows the time 12:26, signal strength, 4G network, and a battery icon at 78%. Below the header, the profile information is displayed in a light gray container with the following fields and values:

- Email: jucse29.353@gmail.com
- Contact Number: 01521704753
- Emergency Contact Number: 01877306914
- Profession: Doctor
- Monthly Income: 6000
- Password: Shanjida@29

At the bottom of the profile container is a red button labeled "EDIT". Below this container is a navigation bar with three items: "Dashboard" with a grid icon, "Service Request" with a right-pointing arrow icon, and "Manage Events" with a calendar icon. At the very bottom of the screen are three standard Android navigation icons: a hamburger menu, a square home button, and a left-pointing arrow for back navigation.

Figure 3.5: When the 'Edit' button is clicked, the fields become editable, allowing residents to modify their information.

3.2 Submit Complaints

- **Model Development:**

- Designed a structured **ComplaintModel** to include fields like complaintID(auto generate in the database), unitCode, emailAddress, role, phoneNumber and problemDescription.
- The following code represents the ComplaintModel class used for complaint handling:

```
1 package com.example.smartlivingcommunity.data.model;
2
3 import java.text.SimpleDateFormat;
4 import java.util.Date;
5 import java.util.Locale;
6 import java.util.UUID;
7
8 /**
9  * ComplaintModel class representing the user's complaint
10  * information.
11  * <p>
12  * This class contains essential fields required for
13  * complaint registration.
14  * It includes the complaint ID, unit code, user details,
15  * contact information, and the complaint description.
16  * The model also provides methods for generating unique
17  * complaint IDs and accessing or modifying complaint-
18  * related data.
19  * <p>
20  * This class is also used for Firebase data mapping, and
21  * therefore includes no-argument constructors.
22  *
23  * @author Shanjida Alam
24  * @version 1.0
25  * @since 2024-11-15
26  */
27 public class ComplaintModel {
28     /**
29      * Unique identifier for the complaint.
30      */
31     private String complaintId;
32     /**
33      * Unit code associated with the complaint.
34      */
35     private String unitCode;
```

```
30     /**
31      * User name associated with the complaint.
32      */
33     private String userName;
34     /**
35      * User role associated with the complaint.
36      */
37     private String userRole;
38     /**
39      * Phone number associated with the complaint.
40      */
41     private String phoneNumber;
42     /**
43      * Email address associated with the complaint.
44      */
45     private String emailAddress;
46     /**
47      * Description of the complaint.
48      */
49     private String complaintDescription;
50
51     /**
52      * No-argument constructor required for Firebase data
53      * model mapping.
54      * <p>
55      * This constructor generates a unique complaint ID upon
56      * object creation.
57      */
58     public ComplaintModel() {
59         this.complaintId = generateUniqueId();
60     }
61
62     /**
63      * Constructs a ComplaintModel with specified details.
64      * <p>
65      * This constructor allows setting the complaint's unit
66      * code, user details, contact information, and description
67      * .
68      * The complaint ID is generated automatically.
69      *
70      * @param unitCode Unit code associated with the
71      * complaint.
72      * @param userName User name associated with the
73      * complaint.
74      * @param userRole User role associated with the
75      * complaint.
```

```
69      * @param phoneNumber Phone number associated with the
complaint.
70      * @param emailAddress Email address associated with the
complaint.
71      * @param complaintDescription Description of the
complaint.
72      */
73      public ComplaintModel(String unitCode, String userName,
String userRole,
74                          String phoneNumber, String
emailAddress, String complaintDescription) {
75          this.complaintId = generateUniqueId();
76          this.unitCode = unitCode;
77          this.userName = userName;
78          this.userRole = userRole;
79          this.phoneNumber = phoneNumber;
80          this.emailAddress = emailAddress;
81          this.complaintDescription = complaintDescription;
82      }
83
84      /**
85       * Generates a unique ID for the complaint based on the
current timestamp and a random suffix.
86       * <p>
87       * The ID follows the pattern "
COMPyyyyMMddHHmmssSSS_randomSuffix", ensuring uniqueness
using a timestamp and UUID.
88       *
89       * @return Unique ID for the complaint.
90       */
91      private String generateUniqueId() {
92          SimpleDateFormat dateFormat = new SimpleDateFormat("
yyyyMMddHHmmssSSS", Locale.getDefault());
93          String timestamp = dateFormat.format(new Date());
94          String uniqueSuffix = UUID.randomUUID().toString().
substring(0, 4);
95          return "COMP" + timestamp + "_" + uniqueSuffix;
96      }
97
98      /**
99       * Get the complaint ID.
100      *
101      * @return The complaint ID.
102      */
103      public String getComplaintId() {
104          return complaintId;
```

```
105     }
106
107     /**
108      * Set the complaint ID.
109      * <p>
110      * This method is necessary for Firebase mapping to set
111      * the complaint ID manually.
112      *
113      * @param complaintId The complaint ID.
114      */
115     public void setComplaintId(String complaintId) {
116         this.complaintId = complaintId;
117     }
118
119     /**
120      * Get the unit code associated with the complaint.
121      *
122      * @return The unit code.
123      */
124     public String getUnitCode() {
125         return unitCode;
126     }
127
128     /**
129      * Get the user name associated with the complaint.
130      *
131      * @return The user name.
132      */
133     public String getUserName() {
134         return userName;
135     }
136
137     /**
138      * Get the user role associated with the complaint.
139      *
140      * @return The user role.
141      */
142     public String getUserRole() {
143         return userRole;
144     }
145
146     /**
147      * Get the phone number associated with the complaint.
148      *
149      * @return The phone number.
150      */
```

```
150     public String getPhoneNumber() {
151         return phoneNumber;
152     }
153
154     /**
155      * Get the email address associated with the complaint.
156      *
157      * @return The email address.
158      */
159     public String getEmailAddress() {
160         return emailAddress;
161     }
162
163     /**
164      * Get the complaint description.
165      *
166      * @return The complaint description.
167      */
168     public String getComplaintDescription() {
169         return complaintDescription;
170     }
171
172     /**
173      * Set the unit code associated with the complaint.
174      *
175      * @param unitCode The unit code.
176      */
177     public void setUnitCode(String unitCode) {
178         this.unitCode = unitCode;
179     }
180
181     /**
182      * Set the user name associated with the complaint.
183      *
184      * @param userName The user name.
185      */
186     public void setUserName(String userName) {
187         this.userName = userName;
188     }
189
190     /**
191      * Set the user role associated with the complaint.
192      *
193      * @param userRole The user role.
194      */
195     public void setUserRole(String userRole) {
```

```
196         this.userRole = userRole;
197     }
198
199     /**
200      * Set the phone number associated with the complaint.
201      *
202      * @param phoneNumber The phone number.
203      */
204     public void setPhoneNumber(String phoneNumber) {
205         this.phoneNumber = phoneNumber;
206     }
207
208     /**
209      * Set the email address associated with the complaint.
210      *
211      * @param emailAddress The email address.
212      */
213     public void setEmailAddress(String emailAddress) {
214         this.emailAddress = emailAddress;
215     }
216
217     /**
218      * Set the complaint description.
219      *
220      * @param complaintDescription The complaint description
221      *
222      */
223     public void setComplaintDescription(String
224     complaintDescription) {
225         this.complaintDescription = complaintDescription;
226     }
227 }
```

Listing 3.1: ComplaintModel Class in Java

- **Validation and Usability:**

- Implemented strong validation to ensure that all mandatory fields are completed before submission.
- Enhanced the user-friendly interface to ensure residents can easily submit their problems in this platform.

- **Visual Aspect of The Submit Complaints:**

- **Complaint Page:**

This screenshot shows the 'LivSmart Complaint Form' page. At the top, there's a red header with a hamburger menu icon and the word 'Complaint'. Below the header, the title 'LivSmart Complaint Form' is centered. Underneath the title is a circular icon containing a speech bubble with an exclamation mark. Below the icon, the text 'Do you have a complaint?' is centered. The form consists of five stacked input fields: 'Name of the Resident', 'Unit Code', 'Resident email address', 'Role', and 'Phone Number'. At the bottom, there's a navigation bar with three items: 'Dashboard' (with a grid icon), 'Service Request' (with a right-pointing arrow icon), and 'Manage Events' (with a calendar icon). Each item has its name written below the icon.

Figure 3.6: This is the complaint page. Upon successfully logging in, if a resident has any complaints about the community system, they can navigate to this page and submit their complaint.

This screenshot shows the same complaint form as Figure 3.6, but with an additional red button labeled 'SUBMIT FORM' located below the 'Problem Description' field. Below the button, there is a warning message: 'Never submit sensitive information such as passwords.' At the bottom, the navigation bar is identical to the one in Figure 3.6, showing 'Dashboard', 'Service Request', and 'Manage Events' with their respective icons.

Figure 3.7: When residents have a complaint about the community system, they must fill out all required fields in the complaint form and then click the '**Submit Form**' button to save the information in the database.

Chapter 4

Measuring My Impact with Toggl Track

Toggl Track, a versatile and user-friendly time-tracking tool, empowers individuals and teams to boost productivity and optimize project management. By allowing us to effortlessly track tasks and activities, Toggl Track simplifies time management. In the context of this project, Toggl Track proved invaluable by helping me accurately track the time spent on various tasks. This not only improved my time management but also provided a clear and objective record of my contributions to the team's success. Below, I have attached some screenshots of my Toggl Track usage:

4.1 Toggl Track for Manage Profile

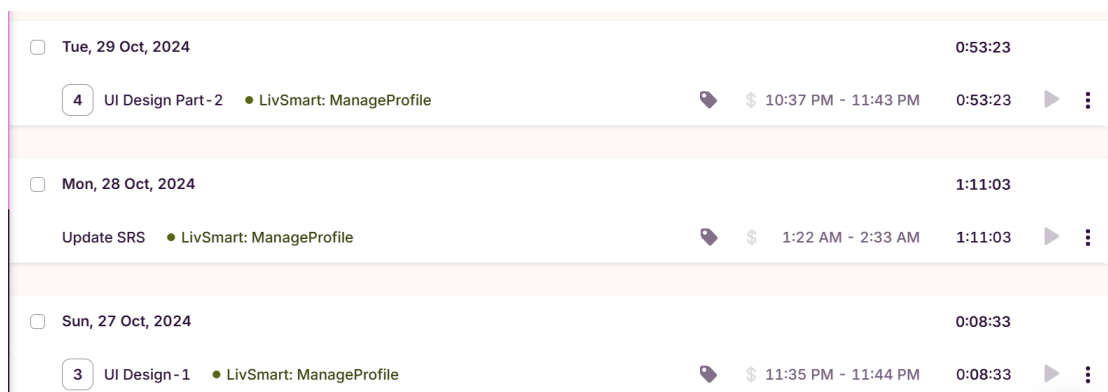


Figure 4.1: This image shows the time-tracking data for the **Manage Profile** feature. It displays three different work sessions across consecutive days in October 2024. The total duration of these three sessions is 2 hours 12 minutes and 59 seconds.

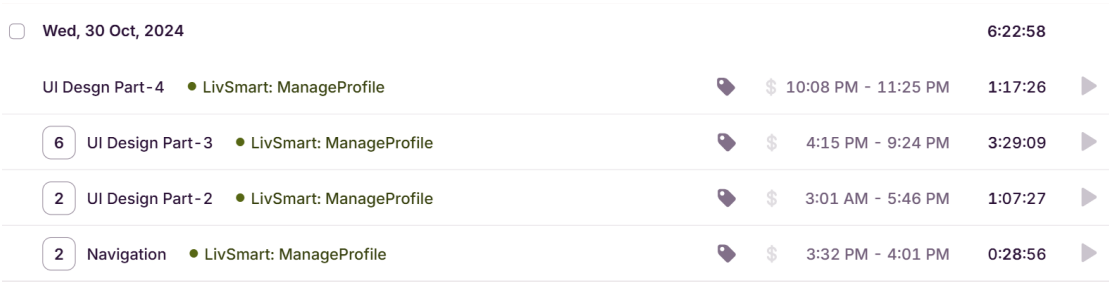


Figure 4.2: This image displays time-tracking data for the **Manage Profile** feature. It shows four different work sessions completed on Wednesday, 30th October 2024. The sessions are labeled as **UI Design Part-4**, **UI Design Part-3**, **UI Design Part-2** and **Navigation** with respective durations of 1 hour 17 minutes and 26 seconds, 3 hours 29 minutes and 9 seconds, 1 hour 7 minutes and 27 seconds and 28 minutes and 56 seconds. The total time spent on this day is 6 hours 22 minutes and 58 seconds.

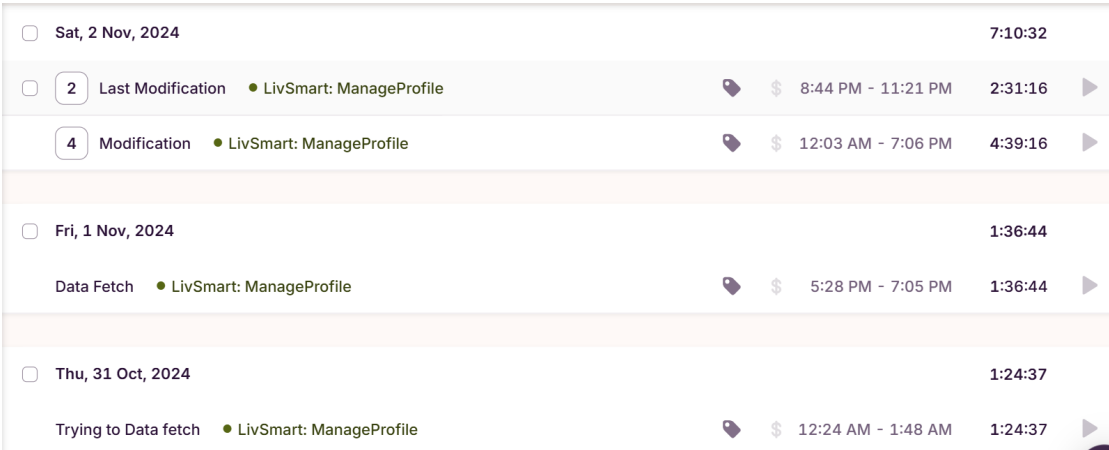


Figure 4.3: This image displays detailed time-tracking data for the **Manage Profile** feature across three consecutive days: 31st October 2024, 1st November 2024 and 2nd November 2024. The total duration of these three days is 10 hours 11 minutes and 53 seconds.

I spent a total of 18 hours 47 minutes and 50 seconds completing the **Manage Profile** feature. This valuable tool helped me track how much time I spent on the project and provided clear data about my contribution to it.

4.2 Toggl Track for Submit Complaints

The screenshot shows a Toggl Track log for the 'Submit Complaint' feature. It is organized by date, with each day's tasks listed below a date header. Each task entry includes a task name, a project name, a status icon, a currency icon, a time range, a total time, and a play button icon.

<input type="checkbox"/> Fri, 8 Nov, 2024		4:19:07
3 UI Design (Sprint - 2)	LivSmart: Create Complaints	7:09 PM - 12:33 AM 3:28:36 ▶
3 TDD	LivSmart: Create Complaints	3:08 PM - 4:10 PM 0:50:31 ▶
<input type="checkbox"/> Thu, 7 Nov, 2024		0:23:05
TDD	LivSmart: Create Complaints	12:56 PM - 1:19 PM 0:23:05 ▶
<input type="checkbox"/> Tue, 5 Nov, 2024		1:36:29
2 TDD	LivSmart: Create Complaints	9:14 PM - 11:57 PM 1:36:29 ▶

Figure 4.4: This Toggl Track log details the time spent on various tasks related to the **Submit Complaint** feature during sprint-2 in November 2024. It covers three consecutive days and includes tasks associated with both User Interface (UI) design and Test Driven Development (TDD). The total time logged for these tasks is 6 hours, 18 minutes, and 41 seconds.

The screenshot shows a Toggl Track log for the 'Submit Complaint' feature. It is organized by date, with each day's tasks listed below a date header. Each task entry includes a task name, a project name, a status icon, a currency icon, a time range, a total time, and a play button icon.

<input type="checkbox"/> Thu, 14 Nov, 2024		4:09:22
3 TDD	LivSmart: Create Complaints	9:52 PM - 2:28 AM 4:09:22 ▶
<input type="checkbox"/> Wed, 13 Nov, 2024		1:21:00
UI Design Part - 2	LivSmart: Create Complaints	11:33 AM - 12:54 PM 1:21:00 ▶
<input type="checkbox"/> Sat, 9 Nov, 2024		1:35:52
2 UI Design Part - 2	LivSmart: Create Complaints	12:33 AM - 3:29 AM 1:35:52 ▶

Figure 4.5: This Toggl Track log also describes the time spent on different tasks related to the **Submit Complaint** feature during sprint-2 in November 2024. It covers three days and includes tasks associated with both User Interface (UI) design and Test Driven Development (TDD). So, the total time spent over these three days was 7 hours 6 minutes and 14 seconds.

□ Fri, 15 Nov, 2024		4:22:31	
TDD	● LivSmart: Create Complaints	📍 \$ 7:58 PM - 10:33 PM	2:34:31
4	UI Design Part-2 ● LivSmart: Create Complaints	📍 \$ 12:20 PM - 6:19 PM	1:48:00

Figure 4.6: This is a Toggl time tracking log showing work done on Friday, November 15, 2024 for the **Submit Complaint** feature. The total time for the day is shown as 4 hours 22 minutes and 31 seconds.

I spent a total of 17 hours 47 minutes 26 seconds completing the **Submit Complaint** feature. Toggl is a valuable tool for tracking and measuring my contributions to the project.

Chapter 5

Tools and Practices

5.1 Discord

- Discord is a community-building tool for open communication. It allows users to engage real-time voice, video and text-chats. It's become a flexible tool that's utilized by many different communities, such as those that are interested in business, education, social connection, or hobbies.
- Discord makes short meetings and conversations possible without requiring formal scheduling, making it ideal for urgent problems, brainstorming sessions, and everyday stand-ups. It allows private conversations between specific team members for the specific purpose. It is specially useful for different development teams like front-end development, back-end development, SQA, etc. because it has dedicated channels, which help organize the discussions by project, team, or topic-keeping conversations. It facilitates to easy sharing of documents, code snippets and other resources directly within the relevant channels.
- Discord has collaboration tools for integration. It connects with other tools like Google drive, Trello, GitHub to optimize workflows and reduce the needs to switch between multiple platforms. By using Discord, a software company can enhance its internal communication, improve project management, support remote work, and foster a strong company culture, all while maintaining cost efficiency and ease of use.

5.2 Trello

Trello is a simple and effective tool for project management. It is a popular project management platform that uses a visual, card-based system to help individuals and teams organize tasks, track progress, and manage workflows. In our project we use trello for,

- Set up our trello board and define various list to represent different stages of tasks progress. For example, **'To Do'** for tasks that have not yet been started, with deadlines and assigned members; **'Doing'** for tasks currently being worked on and **'Done'** for completed tasks.
- Create a card for each task or feature.
- Move cards across lists as task progress, for example, from To Do to Doing to Done.
- Use the comment section for mentioning the team members for instant notifications.
- Set due dates for each task.

5.3 Toggl

Toggl Track, a versatile and user-friendly time-tracking tool, empowers individuals and teams to boost productivity and optimize project management. By allowing us to effortlessly track tasks and activities, Toggl Track simplifies time management. In the context of this project, Toggl Track proved invaluable by helping us accurately track the time spent on various task. In our project we use Toggl for,

- Track time spent for completing the full project.
- Analyze productivity.
- Monitor time management for both individual and team contributions.

5.4 Version Control

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. And GitHub is a web-based platform that provides version controls and collaboration features for software development projects, utilizing Git as its underlying version control system. Git and GitHub are related to each other.

- Git for source code management.
- GitHub for collaboration and code reviews.
- Branch management and merge all of the branch.
- use pull requests for code review.
- Use GitHub Actions for automating with CI/CD.

5.5 Development Tools

We use various development tools and practices in our project they are:

- Android Studio Koala for android development.
- JUnit for unit testing.
- Dokka for documentation.

Chapter 6

Learning Outcomes

The main goal of this project is not necessarily to complete the entire project perfectly, but provide us with the opportunity to learn many valuable skills that are essential for project development. They are:

6.1 Technical Skills

- Acquired knowledge in Android studio for designing and developing mobile applications.
- Learned Java programming for backend logic and XML for creating user interfaces.
- Gained hands-on experience with Firebase Firestore for real-time database management.
- Learned to use Git for version control, including branching, merging, and resolving conflicts.
- Learned how to work with the Agile software model.
- Learned about unit testing and test driven development.

6.2 Soft Skills

- Enhanced team collaboration abilities.
- Resolved technical challenges.
- Developed time management skills.

- Improved technical communication.
- Enhanced problem-solving skills.

Chapter 7

Team Collaboration and Outcome

Our team successfully maintained high productivity and quality standards through:

- Actively participated in team meetings to align feature requirements and progress.
- Regular code reviews and pair programming sessions.
- Clear communication channels and documentation.
- Agile methodologies and sprint planning.
- Collaborative problem-solving approach.

Chapter 8

Conclusion

My contribution to the **LivSmart** project focused on delivering the Profile Management and Complaint User systems. Through this experience, I've not only enhanced my technical skills but also learned valuable lessons about team collaboration and project management. The successful implementation of these critical components has contributed significantly to the overall functionality and user experience of the **LivSmart** platform.