BinBuddy

Presented By Group 3

The Problem

- Climate change + E-waste being frequently discarded + people wanting to be more environmentally friendly → government has set up E-waste disposal bins around the country.

- In 2023, Singapore produced about 60,000 tonnes of E-waste, but only 16,0000 tonnes were recycled. Low E-waste recycling rate because of lack of awareness or inconvenience.

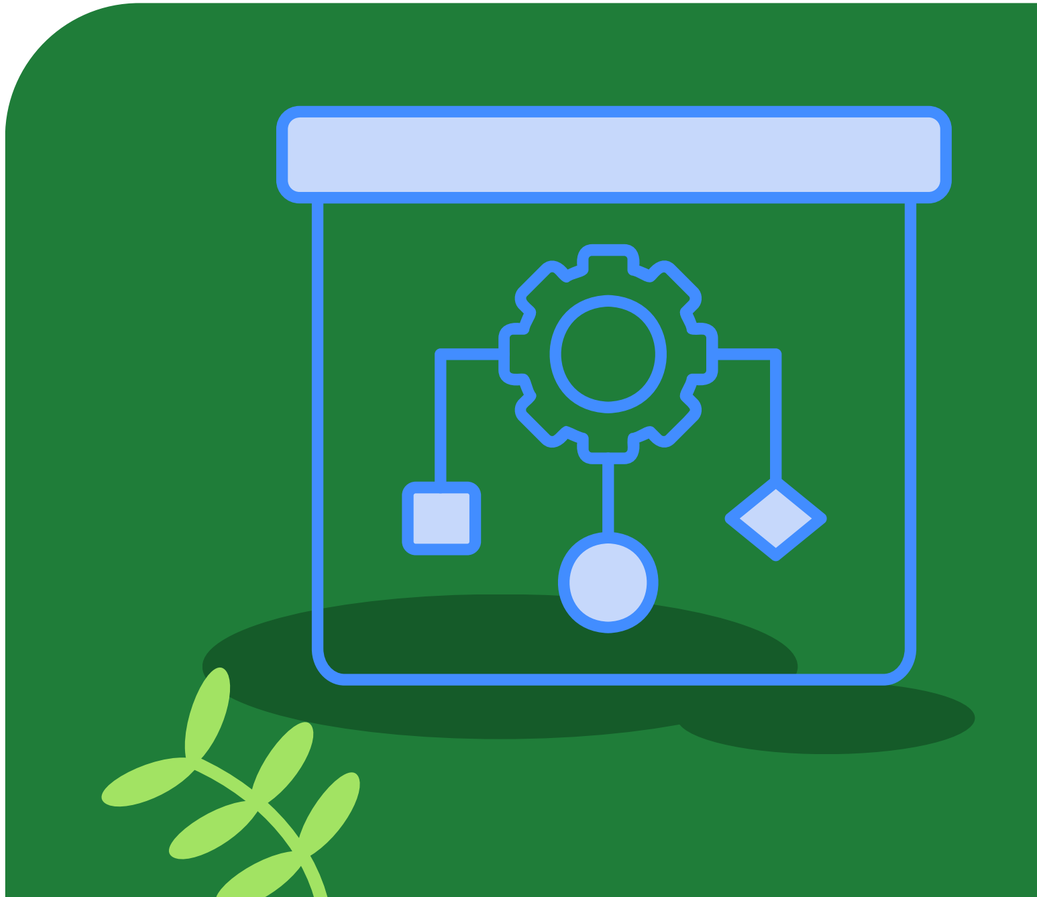
Our Solution

- Making it easier and quicker for people to locate recycling bins → increased convenience
- Users directed to the nearest bins
- Users have access to educational guidelines
- Users can set reminders to recycle
- Users can send queries to admins

User features

- Used NEA’s dataset on E-waste disposal bins around SG from data.gov.sg

Live Demo



Tech Stack

Frontend Rendering & Templates

- Handlebars: JS templating language for webdev to dynamically generate HTML for reusable and flexible web pages.

Tech Stack

Backend (Server-Side Logic & Frameworks)

- Node.js & Express.js: routing, middleware, and request/response handling
- Multer: Node.js middleware for handling form data
- Flash Messenger: To notify user about the state of actions he/she made or show info to users.
- Cookie Parser: Parse the Cookie header on incoming HTTP requests and expose the cookie data as the property req.cookies

Tech Stack

Database Layer

- MongoDB: NoSQL Database
- Mongoose: An Object Data Modeling (ODM) library → bridge between Node.js application and MongoDB database, simplifying interactions

Tech Stack

Security & Authentication

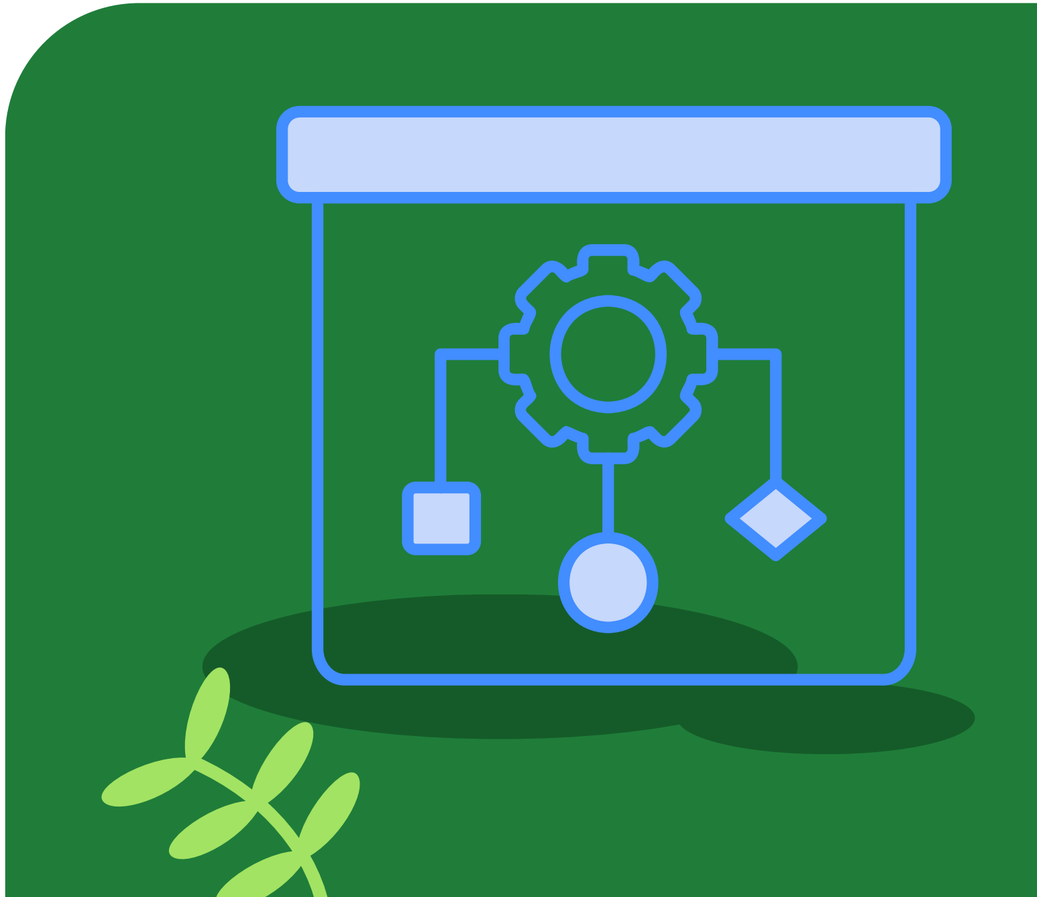
- Regex: For username and password validation
- Bcrypt: Password-hashing function for security
- Email (password recovery): SMTP → Simple Mail Transfer Protocol, handles moving an email from the sender's server to the recipient's server

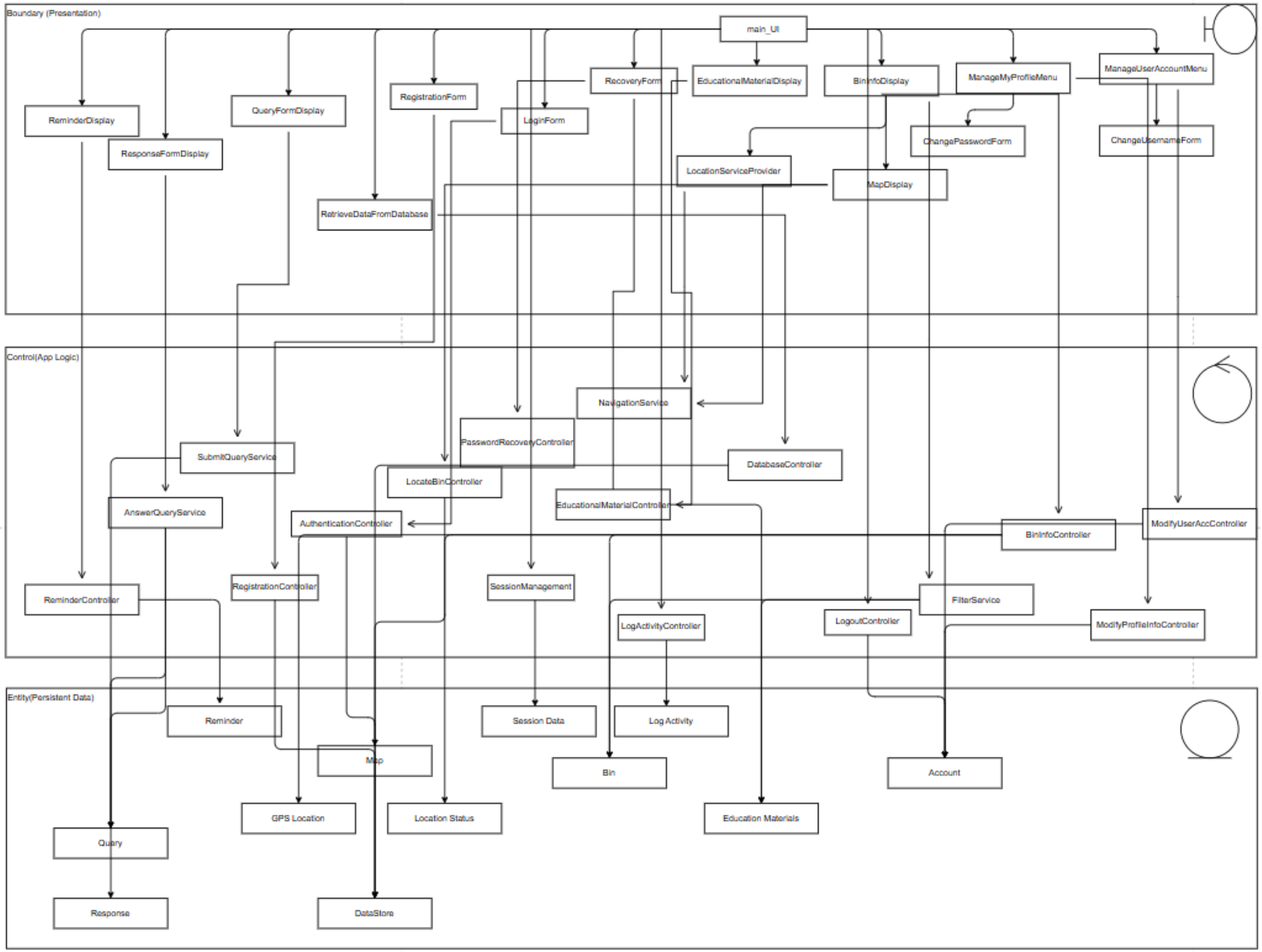
Tech Stack

APIs & External Integrations

- Google Maps API
- SMTP (used via Nodemailer): Integration for sending emails from our app.

System Design




Boundary (Presentation)
main_UI
ReminderDisplay
ResponseFormDisplay
QueryFormDisplay
RegistrationForm
LoginForm
RecoveryForm
EducationalMaterialDisplay
BinInfoDisplay
ManageMyProfileMenu
ManageUserAccountMenu
ChangePasswordForm
ChangeUsernameForm
LocationServiceProvider
MapDisplay
RetrieveDataFromDatabase
Control(App Logic)
NavigationService
PasswordRecoveryController
SubmitQueryService
DatabaseController
LocateBinController
AnswerQueryService
EducationalMaterialController
AuthenticationController
BinInfoController
ModifyUserAccController
ReminderController
RegistrationController
SessionManagement
FilterService
LogActivityController
LogoutController
ModifyProfileInfoController
Entity(Persistent Data)
Reminder
Session Data
Log Activity
Map
Bin
Account
GPS Location
Location Status
Education Materials
Query
Response
DataStore

Software Engineering Practices

Scrum

- Broke our work down into sprints
- Work to be done individually before a team meeting after each sprint (of 1-2 weeks)
- Sprint Planning: Meeting 1-2 weeks to plan before the start of the next sprint
- Sprint Review: Updating product backlog based on priorities, time constraints and functional features remaining
- Sprint Retrospective: How to improve process

Documentation

- **Updated README**

🛠️ Tech Stack

- Backend: Node.js, Express.js
- Database: MongoDB (Mongoose ODM)
- Templating Engine: Handlebars (hbs)
- Mail Service: SendGrid SMTP
- Session Management: express-session + connect-mongo

👷 Setup Instructions Follow these steps to set up the project on your own system or server.

1️⃣ Clone the repository

git clone https://github.com/softwarelab3/2006-SCSI-28.git

cd 2006-SCSI-28

2️⃣ Install dependencies Make sure you have Node.js (v18+) and npm installed. Then install all required packages:

npm install

3️⃣ Configure Environment Variables Create a .env file in the project root and add the following keys:

- PORT=3000
- MONGO_URI=mongodb+srv://<your_mongodb_connection_string>
- SESSION_SECRET=your_secret_key_here

Security and Reliability

- Hash passwords (e.g., with bcrypt).

```
const bcrypt = require('bcryptjs');
```

```
// Hash password before saving
UserSchema.pre('save', async function(next) {
    if (!this.isModified('password')) {
        return next();
    }
    try {
        const salt = await bcrypt.genSalt(10);
        this.password = await bcrypt.hash(this.password, salt);
        next();
    } catch (error) {
        next(error);
    }
});
```

```
// Method to check password
UserSchema.methods.matchPassword = async function(enteredPassword) {
    return await bcrypt.compare(enteredPassword, this.password);
};
```

Security and Reliability

- Used crypto
- When user requests password reset, cannot just send their user ID or email in the reset link (insecure). Anyone with that link could reset the password.
- Instead, generate a random token that is unique, unpredictable, and temporary.
- This token is stored in the database along with an expiry time.

```
const crypto = require('crypto');
```

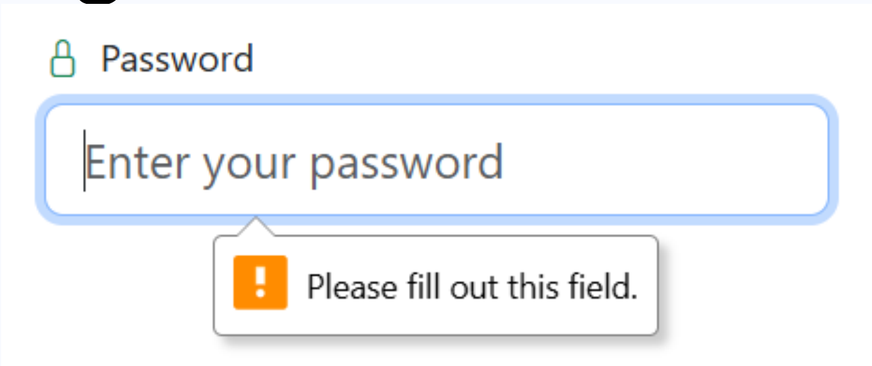
```
async function handleForgotPassword (req, res) {
    const { email } = req.body;
    try {
        const user = await User.findOne({ email });
        if (!user) return res.redirect('/login');

        const token = crypto.randomBytes(32).toString('hex');
        const expiry = Date.now() + 3600000; // 1 hour

        user.resetToken = token;
        user.resetTokenExpiry = expiry;
        await user.save({ validateBeforeSave: false });
```

Security and Reliability

- Validated all inputs on both client and server sides.
- Stored credentials in .env files, not in code.
- Handled errors gracefully: always provide user-friendly error messages and log errors internally (through flash messenger)



Testing and Debugging

- Unit testing: test critical functions individually.
- Integration testing: verify modules work together.
- Used try-catch blocks and meaningful error logs

```
// View all queries for a given user
async function viewMyQueries(req, res) {
  try {
    const queries = await Query.find({ userID: req.session.user.id }).lean();
    res.render('queryList', { title: 'My Queries', queries });
  } catch (err) {
    console.error(err);
    alertMessage(req, 'error', 'There was an error retrieving your queries. Please try again.');
    res.redirect('/query/queryDashboard');
  }
}
```

Version Control and Collaboration

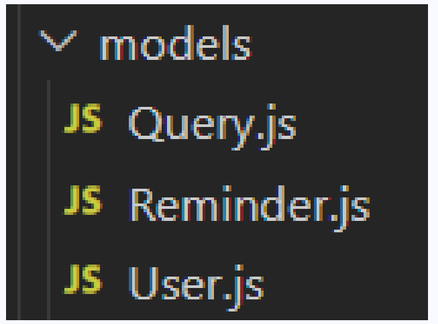
- Small, meaningful commits regularly
- Branching strategy: developed features in separate branches and merge via pull requests.
- Used .gitignore to exclude node_modules, .env
- Documented setup steps clearly in README.

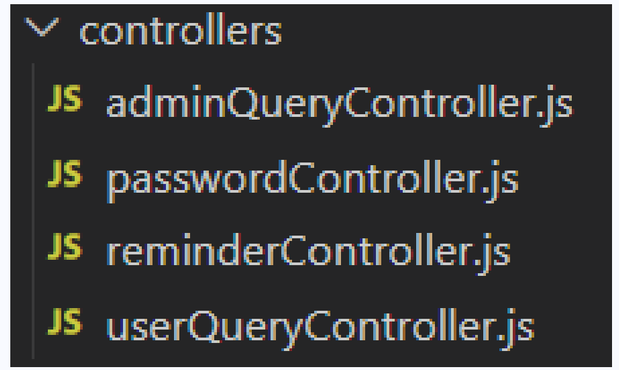
Code Quality and Maintainability

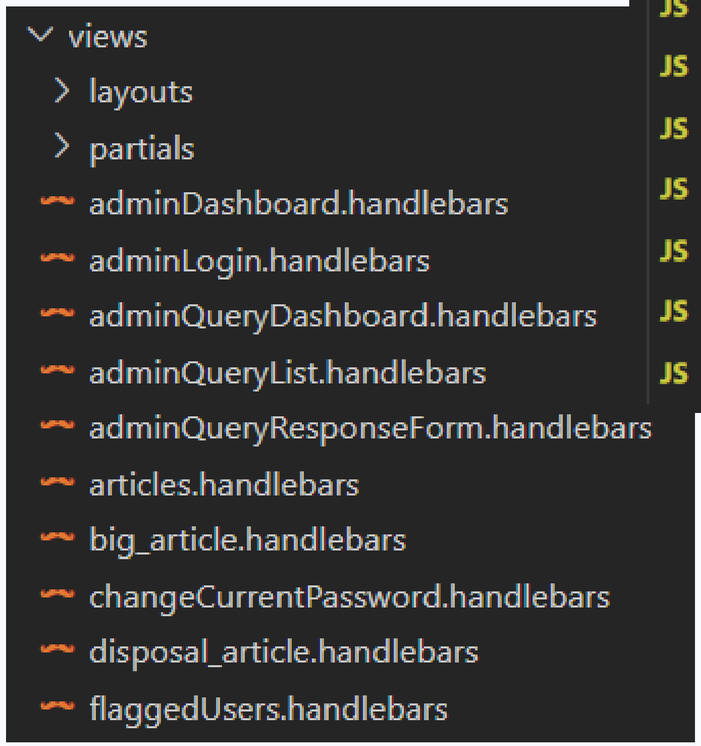
- Meaningful naming: variables, functions, and files were self-explanatory.
- Helpful comments
- Refactored regularly: cleaned up redundant or inefficient code when discovered.

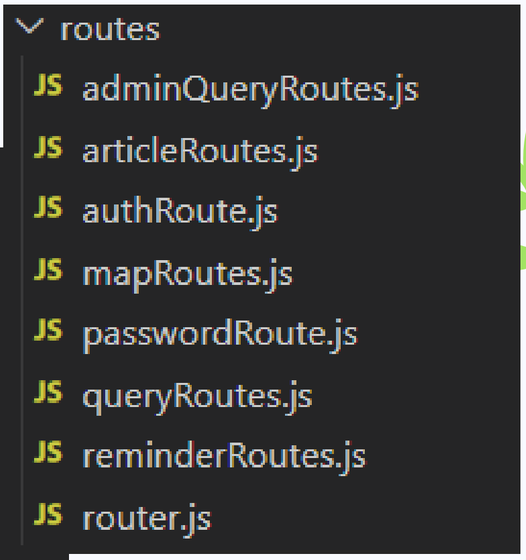
Code Design and Architecture

- **Used MVC pattern (Model–View–Controller): separate logic, data, and presentation.**
- **Modularization: broke code into reusable, single-purpose functions.**
- **Scalability in mind: Structured folders and functions so new features can be added easily (controllers, models, views, routes)**





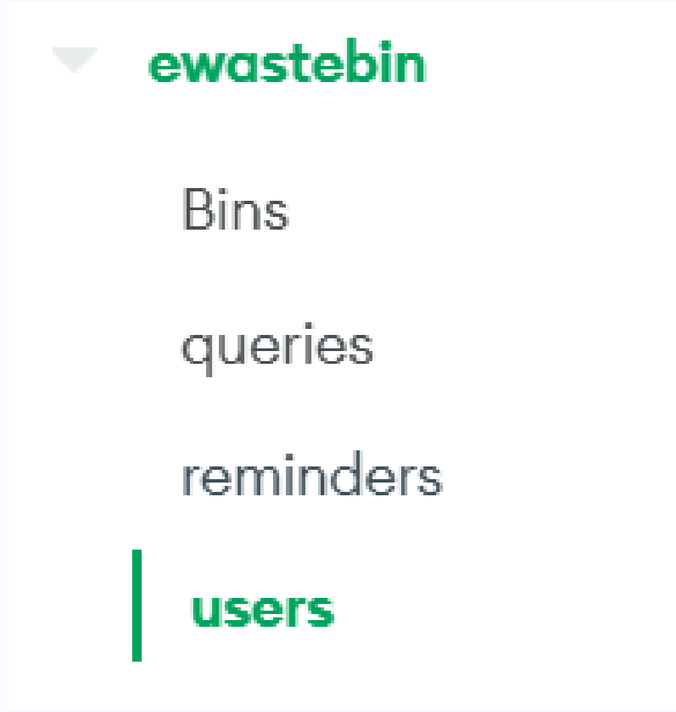




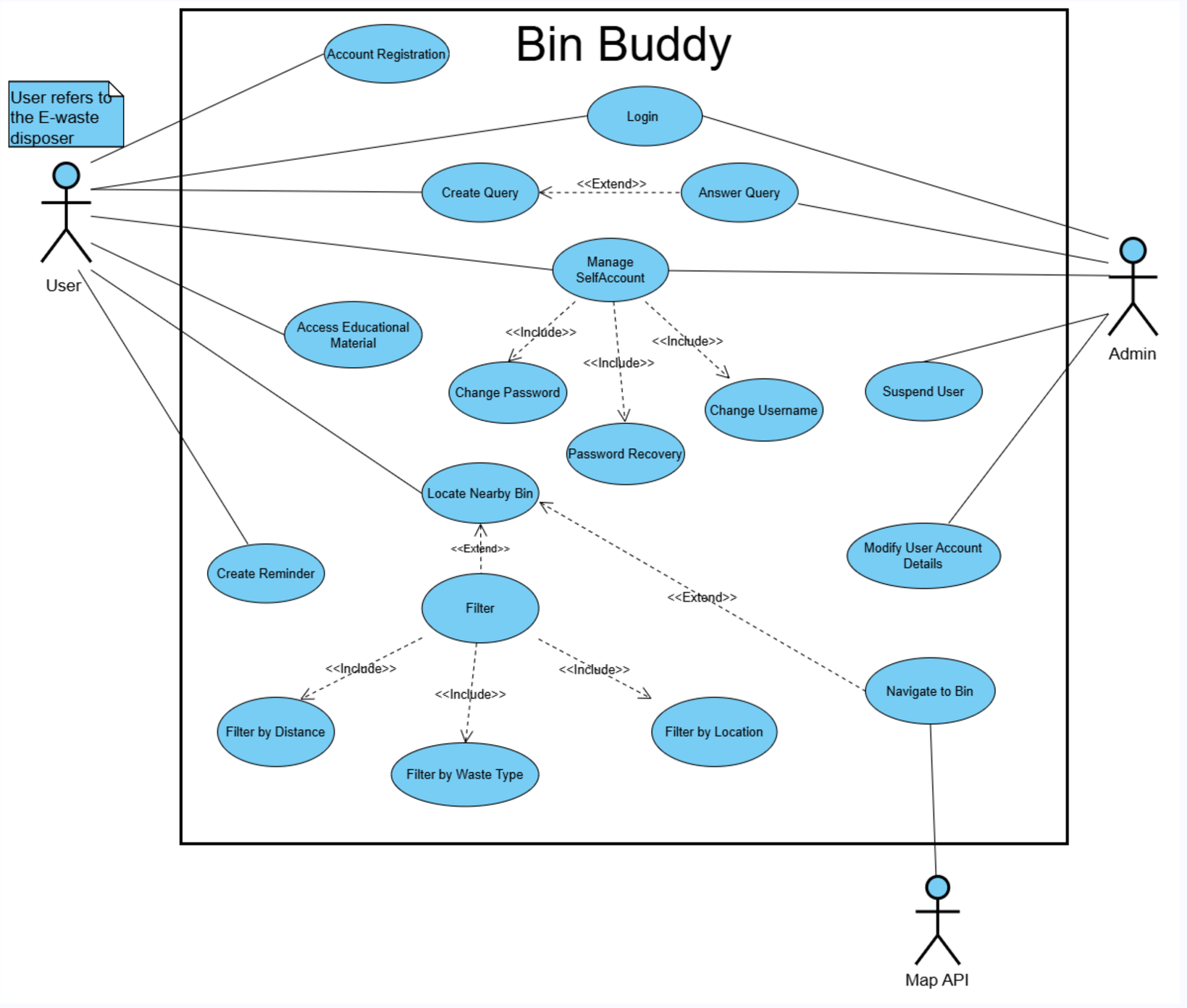
Design Patterns

Singleton Pattern

- Applied in our database connection to ensure only one instance of the connection is maintained.



Diagrams, Traceability and Testing

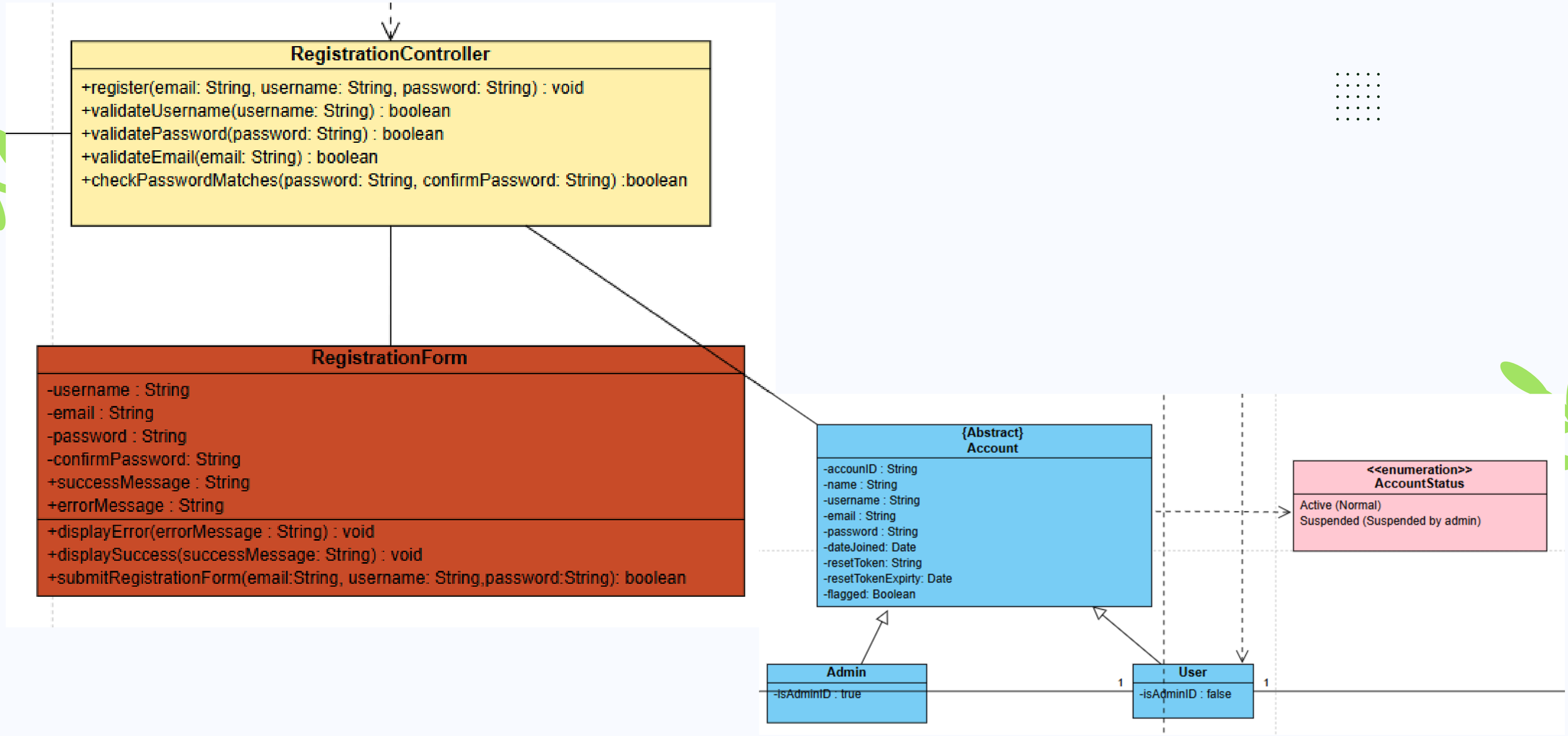

Bin Buddy
Account Registration
User refers to the E-waste disposer
Login
Create Query
<<Extend>>
Answer Query
User
Manage SelfAccount
Admin
Access Educational Material
<<Include>>
<<Include>>
<<Include>>
Change Password
Change Username
Suspend User
Password Recovery
Locate Nearby Bin
Create Reminder
<<Extend>>
Modify User Account Details
Filter
<<Extend>>
<<Include>>
<<Include>>
<<Include>>
Navigate to Bin
Filter by Distance
Filter by Location
Filter by Waste Type
Map API

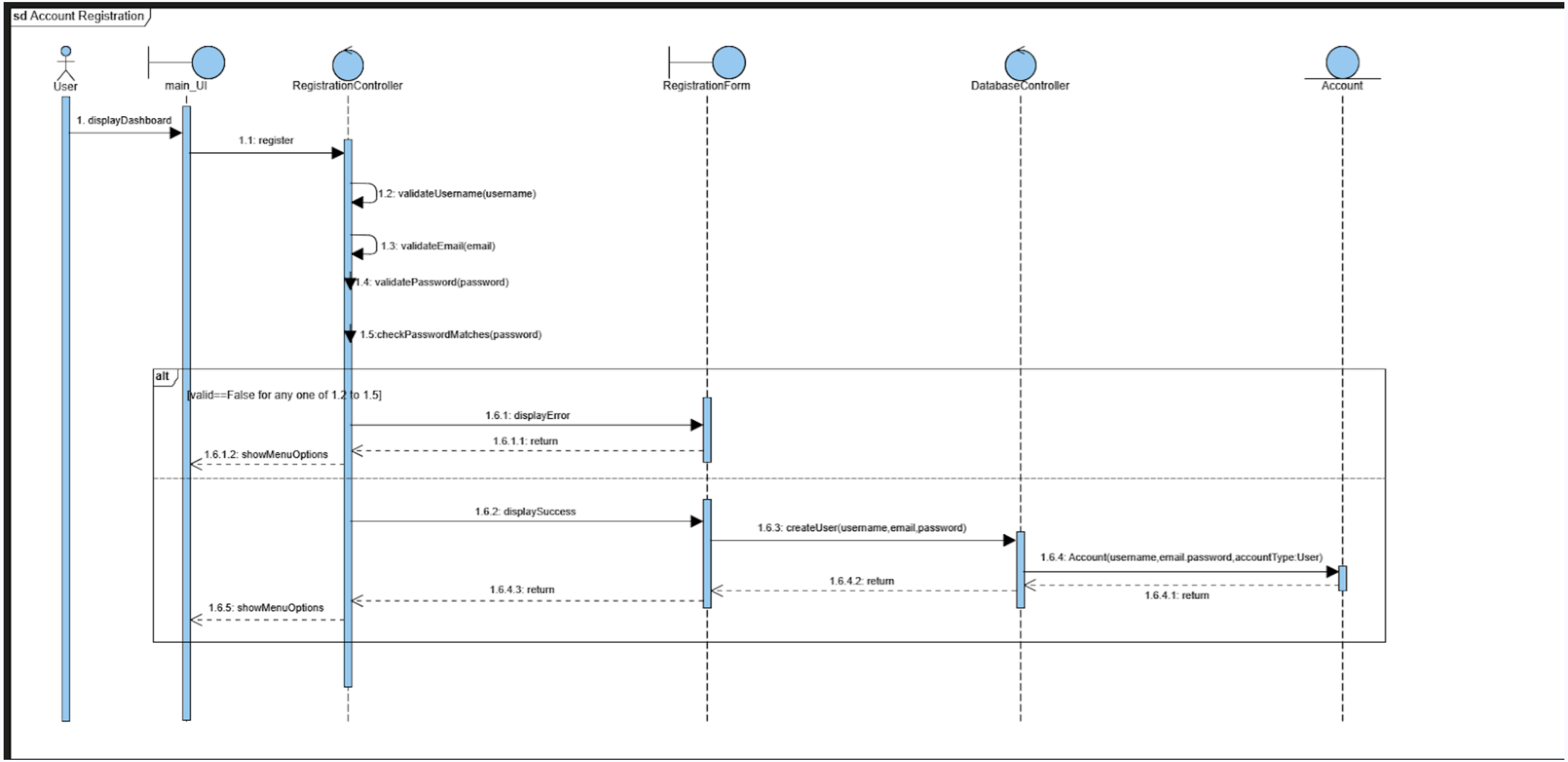
Documentation for Traceability

1. Account Registration

Use Case ID:	UC1		
Use Case Name:	Account Registration		
Created By:	Htoo Myat Noe	Last Updated By:	Htoo Myat Noe
Date Created:	24 Aug 2025	Date Last Updated:	28 Aug 2025

Actor:	User (E-waste disposer)
Description:	The E-waste disposer user may register for a new account within the E-waste recycling system in order to gain personalized access and convenience features. Registration enables users to securely store their personal details and home/work location, making it easier to locate nearby recycling bins without having to repeatedly input their information. By creating an account, the user gains access to a persistent profile that can be used for future interactions such as checking nearby bins, receiving notifications to remind them on their next recycling trip, and bin availability in their vicinity. During registration, the user provides basic details such as their username and password credentials. The system ensures that user information is valid, unique (no duplicate username), and compliant with password security requirements. The account creation process is designed to be simple and user-friendly while maintaining strict security standards. The successful creation of an account establishes the user's identity within the system, enabling them to log in with their username.
Preconditions:	• The user can access the application. • The user's device has internet connectivity. • The user does not already have an account (for the same email/username).
Postconditions:	• A new account is created and stored in the database. • The user can log in using the newly created credentials.
Priority:	High
Frequency of Use:	Low (usually only done once for every account to be created)
Flow of Events:	1. The user opens the portal. 2. The user selects "Register / Create new Account." 3. The user enters required information: = Email


RegistrationController
+register(email: String, username: String, password: String) : void
+validateUsername(username: String) : boolean
+validatePassword(password: String) : boolean
+validateEmail(email: String) : boolean
+checkPasswordMatches(password: String, confirmPassword: String) :boolean
RegistrationForm
-username : String
-email : String
-password : String
-confirmPassword: String
+successMessage : String
+errorMessage : String
+displayError(errorMessage : String) : void
+displaySuccess(successMessage: String) : void
+submitRegistrationForm(email:String, username: String,password:String): boolean
{Abstract}
Account
-accounID : String
-name : String
-username : String
-email : String
-password : String
-dateJoined: Date
-resetToken: String
-resetTokenExpirty: Date
-flagged: Boolean
<<enumeration>>
AccountStatus
Active (Normal)
Suspended (Suspended by admin)
Admin
-isAdminID : true
User
-isAdminID : false
1
1


sd Account Registration
User
main_UI
RegistrationController
RegistrationForm
DatabaseController
Account
1. displayDashboard
1.1: register
1.2: validateUsername(username)
1.3: validateEmail(email)
1.4: validatePassword(password)
1.5:checkPasswordMatches(password)
alt
[valid==False for any one of 1.2 to 1.5]
1.6.1: displayError
1.6.1.1: return
1.6.1.2: showMenuOptions
1.6.2: displaySuccess
1.6.3: createUser(username,email,password)
1.6.4: Account(username,email,password,accountType:User)
1.6.4.2: return
1.6.4.1: return
1.6.4.3: return
1.6.5: showMenuOptions

Black Box Testing:

- Equivalence class testing: Finding valid and invalid equivalence classes that partition the input values for a specific use case. This aims to reduce the number of test cases needed while maintaining thorough test coverage.

- Valid Equivalence Class: Full Name, Username, Email Address, Password, Confirm Password fields are all valid.

- Invalid Equivalence Class: Full Name, Username, Email Address, Password, Confirm Password fields with invalid formats or incorrect length.

No.	Test Input	Expected Output	Actual Output	Expected Result
1.	**(All Valid Inputs)** Name: Tan Tan Username: Lim123 Email Address: binbuddy3@gmail.com Password: 12345! Confirm Password: 12345!	You are now registered and can log in	 You are now registered and can log in	Pass
2	**(All Valid Inputs except Name)** Name: “”	App will highlight the “Full Name” field and ask the user to fill in the field	 App will highlight the “Full Name” field and ask the user to fill in the field	Reject
3	**(All Valid Inputs except Username)** Username: Lim123 (already registered)	Username is already taken	 Username is already taken	Reject
4	**(All Valid Inputs except Username)** Username: Li (Length is 1-2 characters)	Please lengthen this text to 3 characters or more (you are currently using xx characters)	Please lengthen this text to 3 characters or more (you are currently using xx characters)	Reject
5	**(All Valid Inputs except Username)** Username: “” (Length is 0 characters)	Please fill out this field	Please fill out this field	Reject
6	**(All Valid Inputs except Email)** Email: binbuddy3 (no @ in input)	Please include an ‘@’ in the email address. ‘binbuddy3’ is missing an ‘@’.	Please include an ‘@’ in the email address. ‘binbuddy3’ is missing an ‘@’.	Reject
7	**(All Valid Inputs except Password)** Password: 12345 (less	Please match the requested format	 Please match the requested format	Reject

White Box Testing

Cyclomatic Complexity: |decision points|+ 1 = 6+1=7

Basis Paths

Baseline Path: 1,2,3,5,7,9,11,12,14,17

Basis Path 2: 1,2,3,4,16

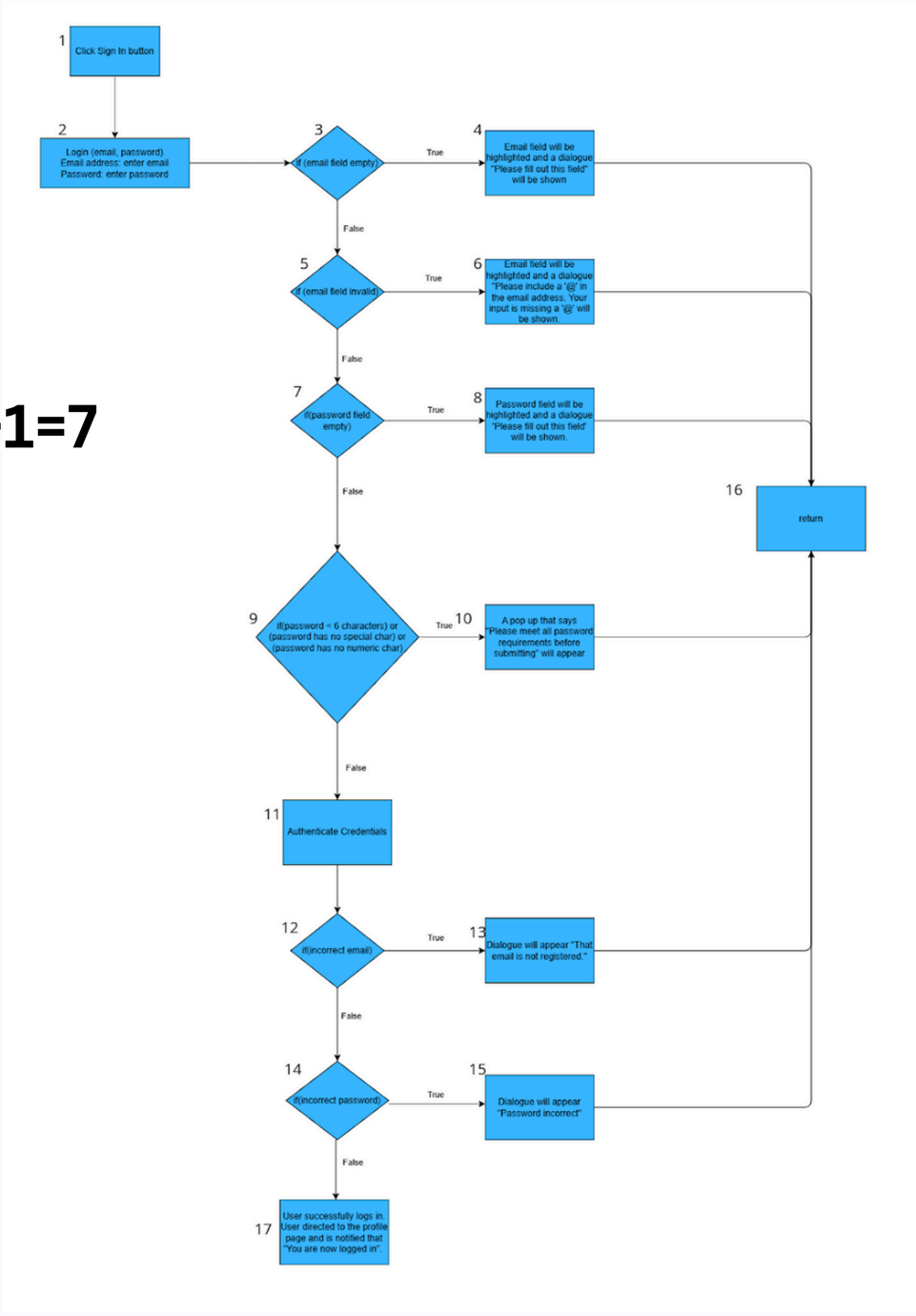
Basis Path 3: 1,2,3,5,6,16

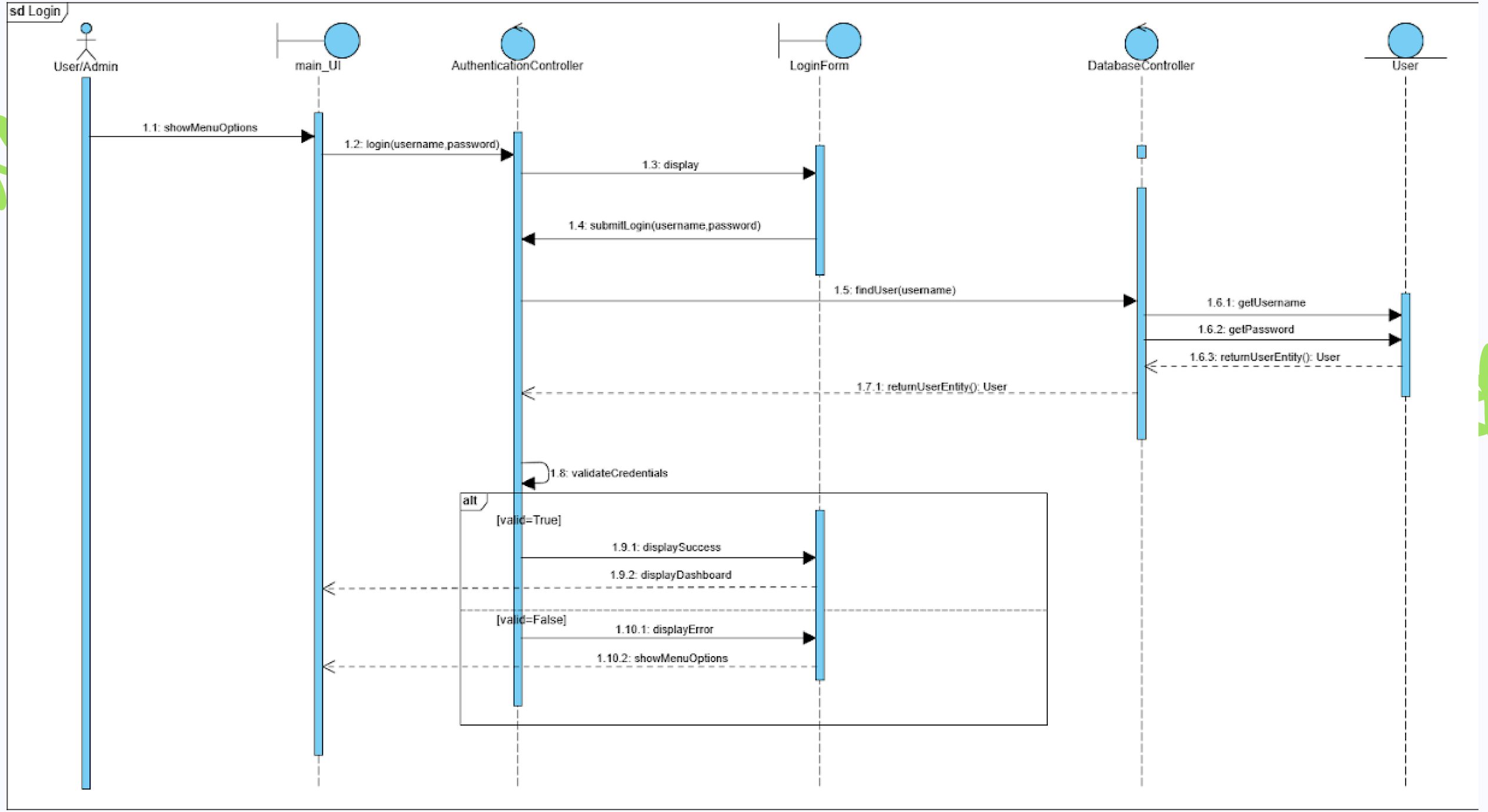
Basis Path 4: 1,2,3,5,7,8,16

Basis Path 5: 1,2,3,5,7,9,10,16

Basis Path 6: 1,2,3,5,7,9,11,12,13,16

Basis Path 7:1,2,3,5,7,9,11,12,13,14,15,16




sd Login
User/Admin
main_UI
AuthenticationController
LoginForm
DatabaseController
User
1.1: showMenuOptions
1.2: login(username,password)
1.3: display
1.4: submitLogin(username,password)
1.5: findUser(username)
1.6.1: getUsername
1.6.2: getPassword
1.6.3: returnUserEntity(): User
1.7.1: returnUserEntity(): User
1.8: validateCredentials
alt
[valid=True]
1.9.1: displaySuccess
1.9.2: displayDashboard
[valid=False]
1.10.1: displayError
1.10.2: showMenuOptions

Future Plans

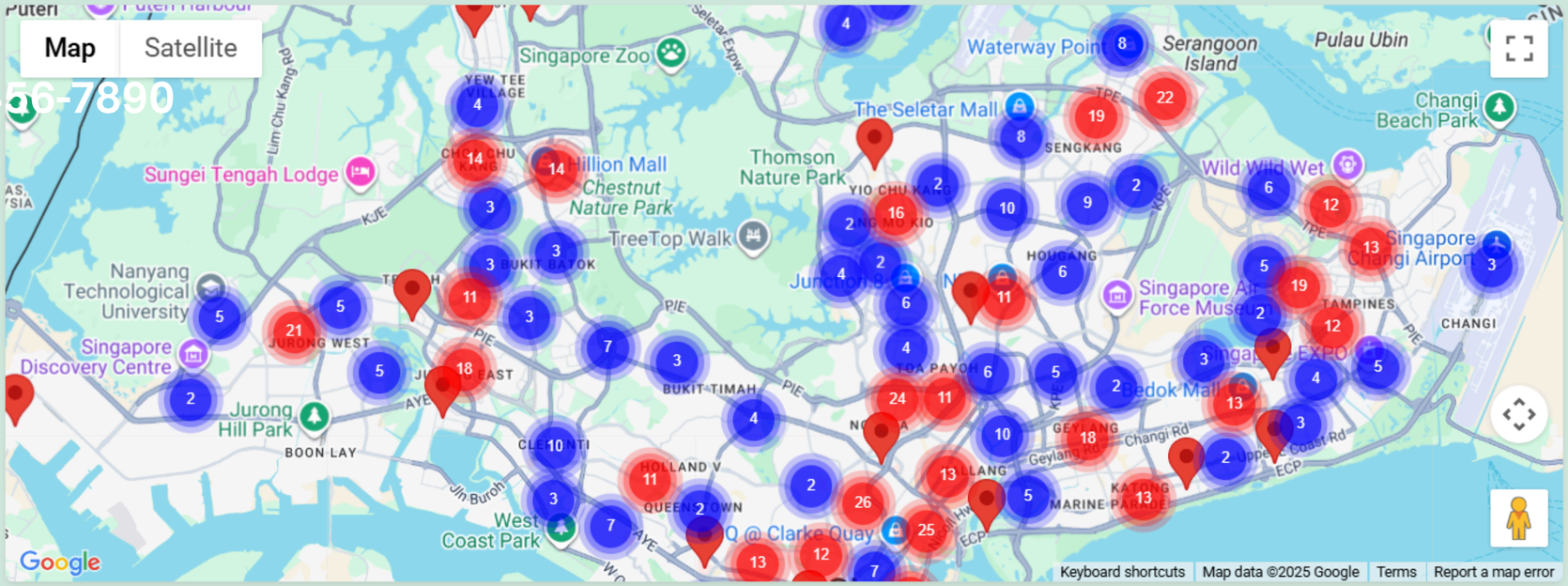
- AI-powered chatbot
- Filter bins by occupancy/capacity (need ground data from NEA through sensors on bins)
- Admins managing bin data (such as occupancy) in real time

Thank You

Get Started

E-Waste Bins Near You

Full Map



123-456-7890