peepal

SC2006 FDAB Team 1

Soong Jun Shen
Adam Soh Shi Jie
Joyce Lee Jia Xuan
Liew Jia Wei
Tan Kai Hooi

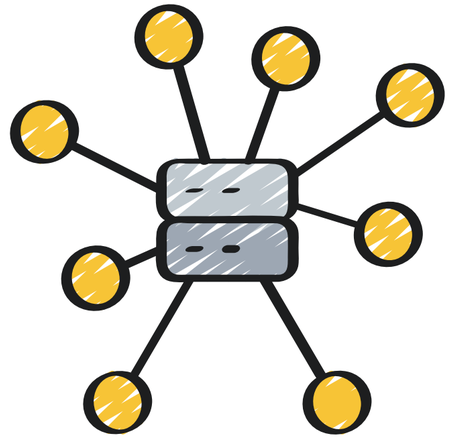
peepal

Problem Statement

Finding a **clean and accessible public toilet** is often a challenge in busy cities like Singapore. People struggle to identify nearby toilets with essential features such as **bidets, accessibility for people with disabilities, showers and sanitisers.**

peepal

What if we can use a **centralised**, **easily accessible application** to address this issue?



Introduce our solution

peepal

A Community-Driven Crowd Sourced Toilet Finder

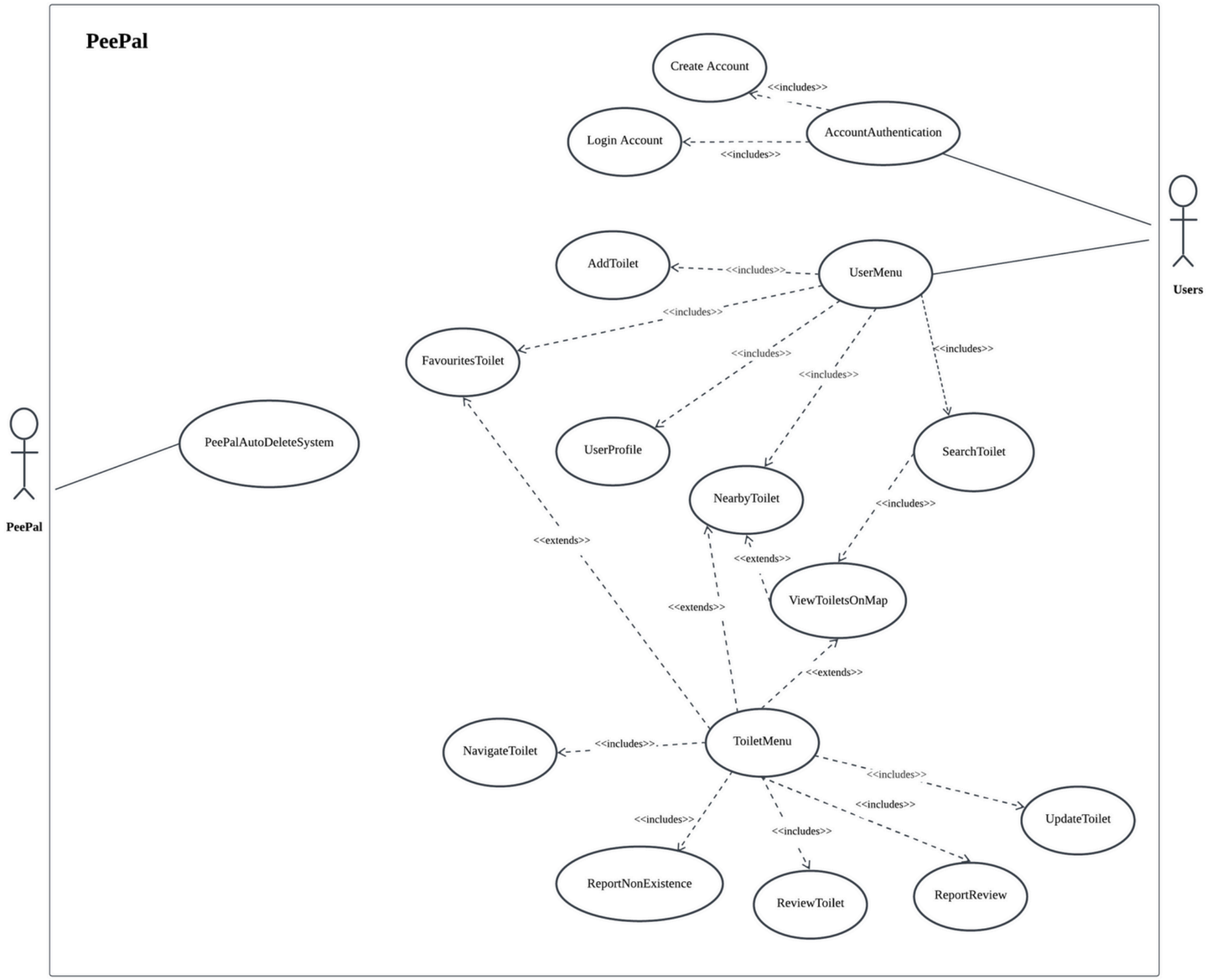
peepal

Key features:

- Find the **nearest public toilets** based on your current location and receive turn-by-turn directions to **navigate** to any toilet..
- Allow users to identify toilets with specific features, such as **bidets, showers, sanitiser, and OKU-friendly facilities** across Singapore.
- View **ratings and reviews** from other **PeePal** users to make informed decisions about toilet quality and cleanliness.
- **Community-driven data** to ensure data remains open, transparent and free for all users.

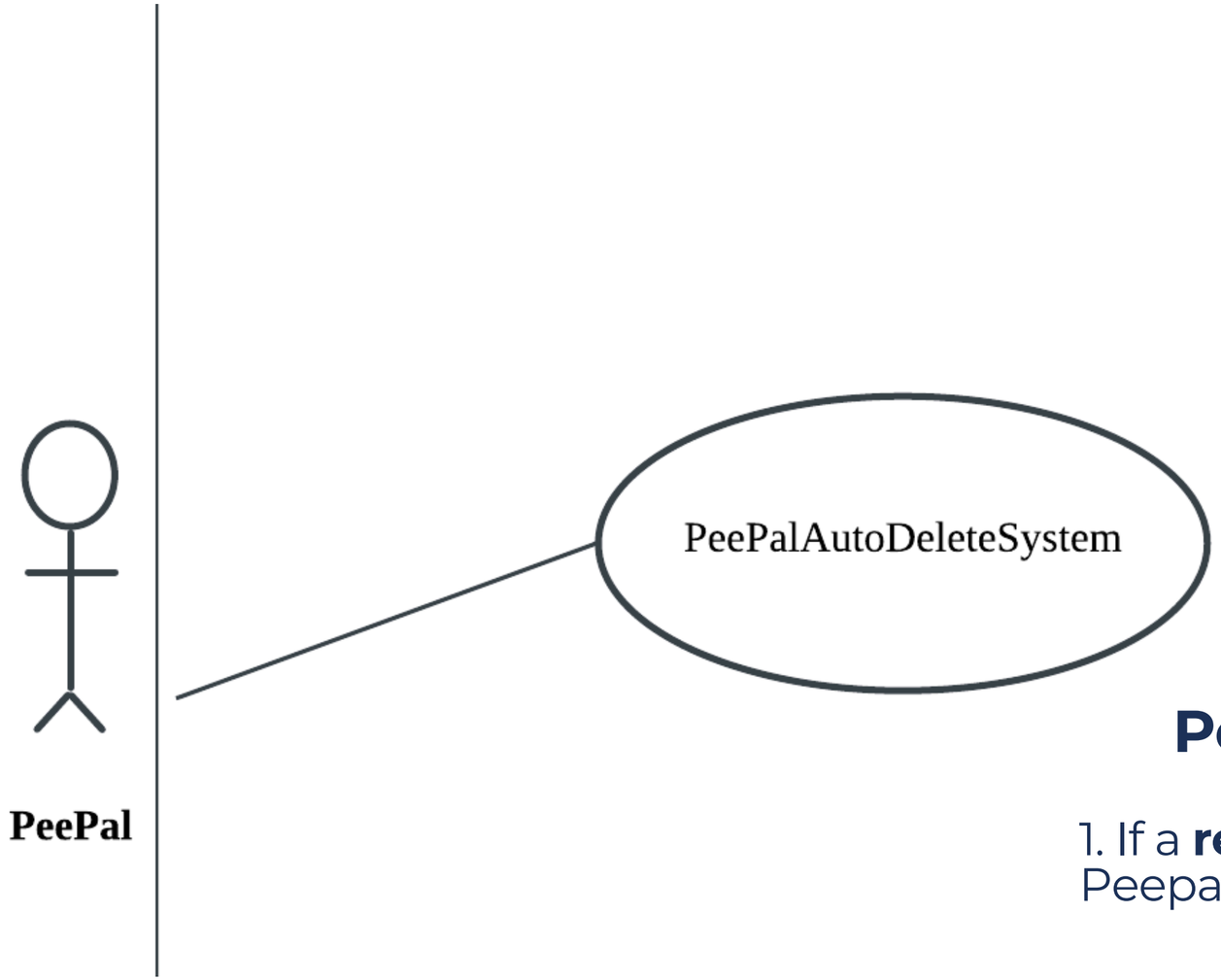
Use case diagram

peepal



Use case diagram

peepal



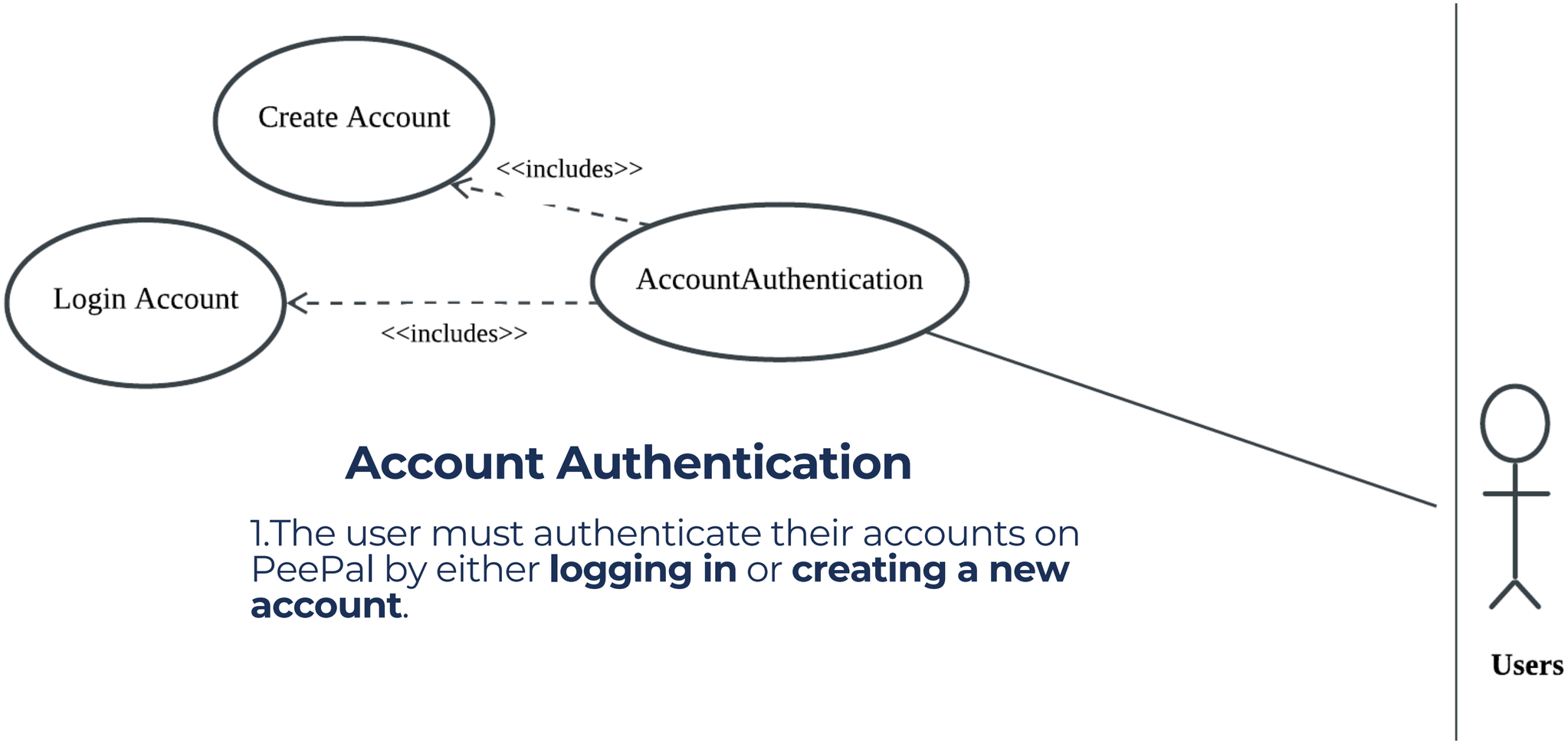
PeePal auto delete system

1. If a **review** is reported **more than 3 times**, Peepal automatically deletes it.

2.If a **toilet** is reported as **non-existent more than 3 times**, Peepal automatically deletes it.

Use case diagram

peepal

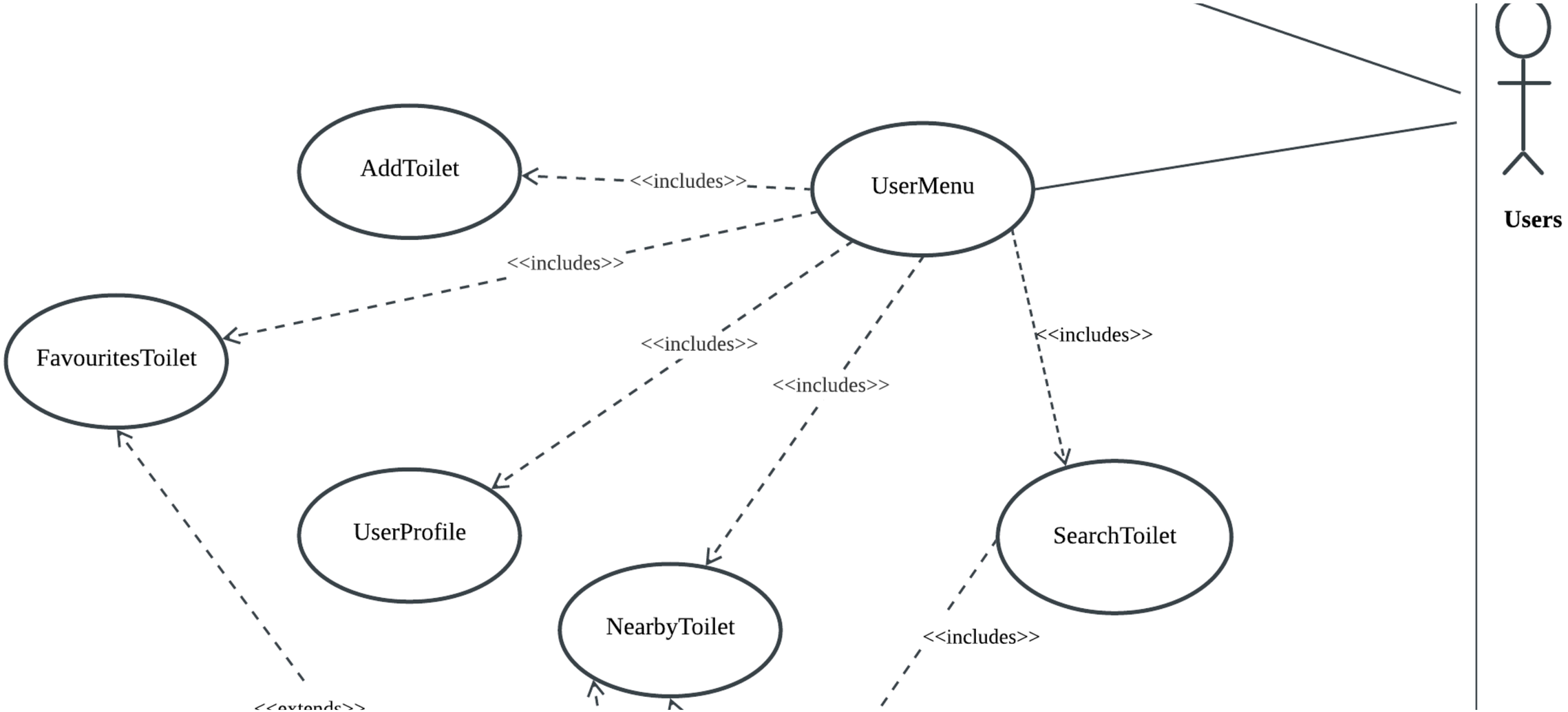


Account Authentication

1.The user must authenticate their accounts on PeePal by either **logging in** or **creating a new account**.

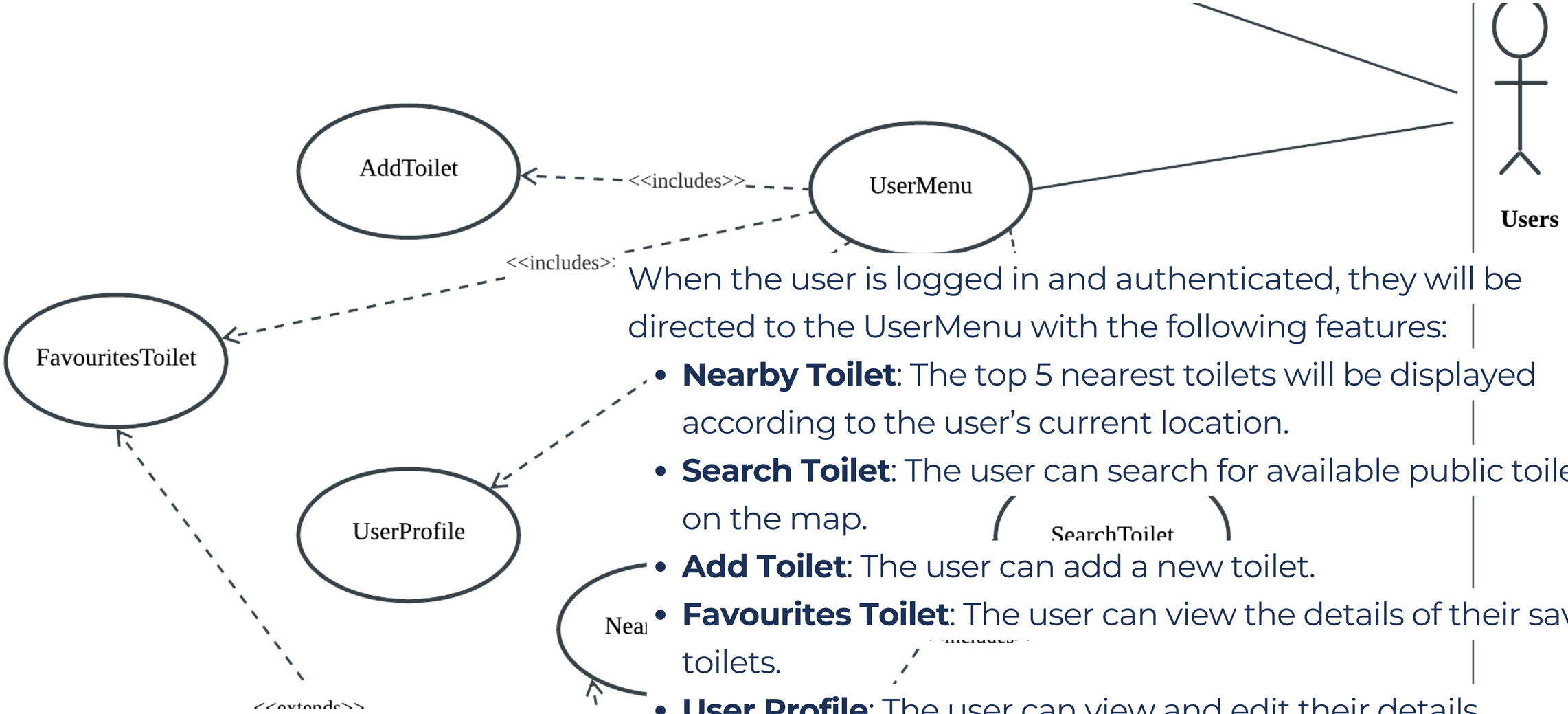
Use case diagram

peepal



Use case diagram

peepal

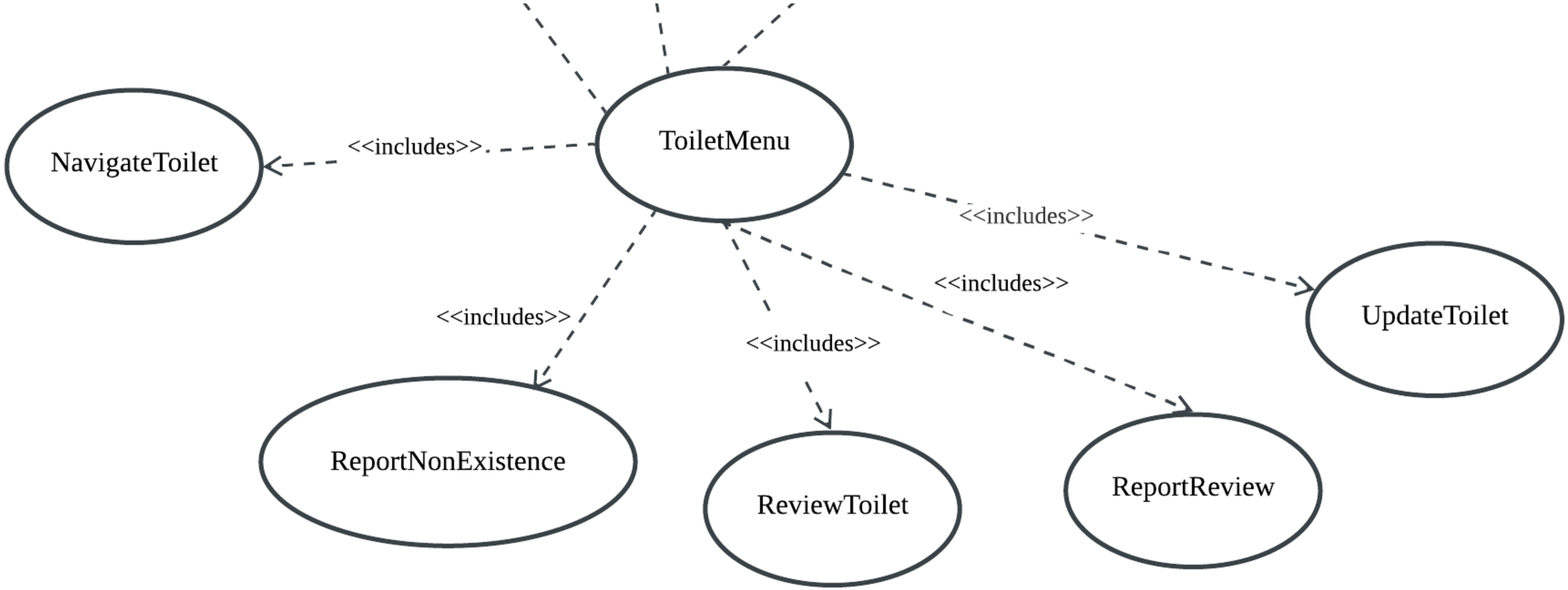


When the user is logged in and authenticated, they will be directed to the UserMenu with the following features:

- **Nearby Toilet**: The top 5 nearest toilets will be displayed according to the user's current location.
- **Search Toilet**: The user can search for available public toilets on the map.
- **Add Toilet**: The user can add a new toilet.
- **Favourites Toilet**: The user can view the details of their saved toilets.
- **User Profile**: The user can view and edit their details.

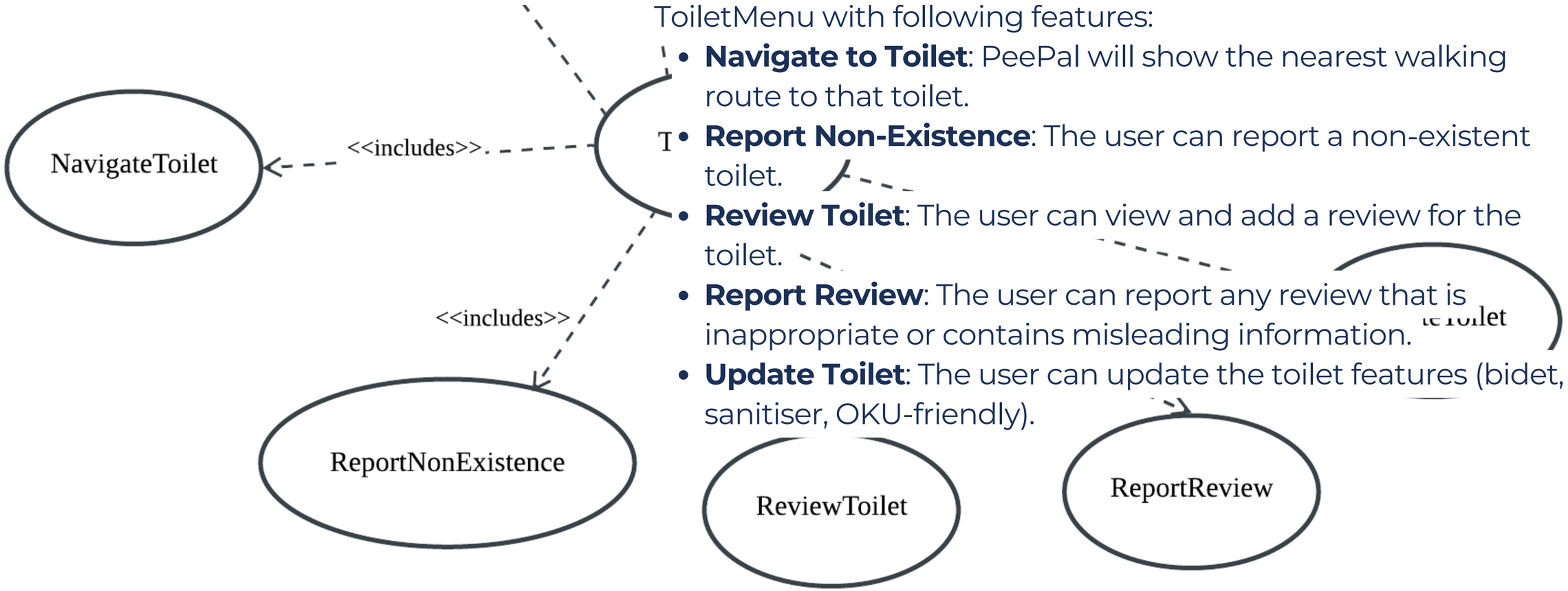
Use case diagram

peepal



Use case diagram

peepal



When the user select a specific toilet, it will be directed to the ToiletMenu with following features:

- **Navigate to Toilet**: PeePal will show the nearest walking route to that toilet.
- **Report Non-Existence**: The user can report a non-existent toilet.
- **Review Toilet**: The user can view and add a review for the toilet.
- **Report Review**: The user can report any review that is inappropriate or contains misleading information.
- **Update Toilet**: The user can update the toilet features (bidet, sanitiser, OKU-friendly).

External APIs

peepal

Google Maps API

Used the Nearby Places API, Places Photo API and Places Details API to retrieve and index the data on public toilets across Singapore.

Apple MapKit API

Provides real-time navigation and ETA matrix computations to the nearest toilets, and interactive map interface.

S3 Storage Buckets

For permanant object storage using the standard S3 Bucket APIs.

peepal

Live Demo!

peepal

Tech Stack

Flutter bloc

state management, layered architecture using repository pattern

Drizzle ORM

Hono

PostgreSQL

VITEST

JWT Based Auth

Custom Backend Stack with Test-Driven Development

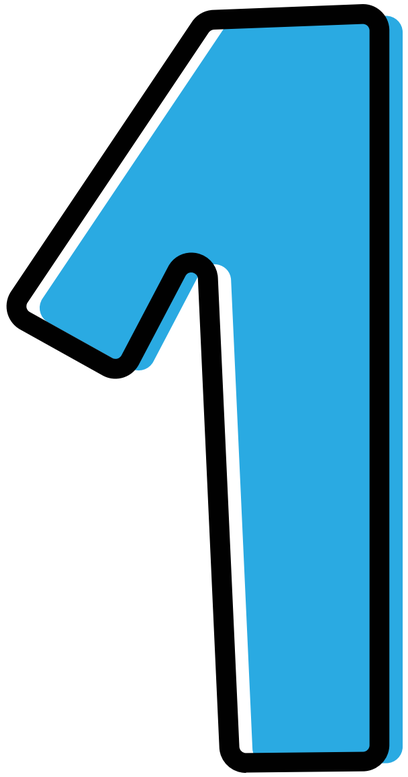
docker

GitHub Actions

Sevalla

Continuous Integration and Deployment

peepal



Good SWE Practices

peepal

Good SWE Practices

1. **Documentation**
2. **Consistent Code Style & Naming Convention**
3. **Reusability and Refactoring**
4. **Waterfall structure**

peepal

1.Documentation - README

2006-FDAB-P1

Peepal - Toilet Finder Application

Overview

Peepal is a modern toilet finder application that helps users locate nearby toilets with detailed information and reviews. The application is built using a microservices architecture with separate frontend and backend services.

Tech Stack

Backend

- **Framework**: Hono (TypeScript)
- **Database**: PostgreSQL with Drizzle ORM
- **Authentication**: JWT
- **Image Storage**: MinIO
- **Testing**: Vitest
- **Validation**: Zod
- **API Documentation**: Swagger/OpenAPI

Frontend

- **Framework**: Flutter
- **State Management**: BLoC Pattern
- **Navigation**: Flutter Navigation
- **UI Components**: Custom widgets with Material Design
- **Maps**: MapKit API
- **Location Services**: Core Location Framework

Architecture & Design Patterns

Backend Architecture

1. **RESTful API Design**
   - Clean and consistent API endpoints
   - HTTP status codes for error handling
   - JSON responses with proper validation
2. **Database Design**
   - Relational database with proper indexing
   - Foreign key constraints for data integrity
   - Separate tables for users, toilets, reviews, and favorites
3. **Authentication Flow**
   - JWT-based authentication
   - Secure password hashing with bcrypt
   - Role-based access control

Frontend Architecture

1. **BLoC Pattern**
   - Separation of concerns between UI and business logic
   - State management through BLoC classes
   - Event-driven architecture for UI updates
2. **Widget Architecture**
   - Reusable widgets for common UI components
   - Custom widgets for specific features
   - Responsive design for different screen sizes

peepal

1. Documentation - Code Comments

```
/**
 * Creates a new toilet based on the provided information.
 *
 * The request body is validated using the defined schema to ensure correct data types a
 *
 * The location field is converted to a spatial point and inserted into the database.
 *
 * Logs the creation operation and returns the created toilet information upon success.
 *
 * @param c - The context containing request and response objects, and logger.
 * @returns A JSON response with the created toilet details or an error message.
 */
toiletApi.post('/create', validator('json', createToiletSchema), async (c) => {
  const logger = c.get('logger')
  const { name, address, latitude, longitude, location, handicapAvail, bidetAvail, showerA
    = c.req.valid('json')

  // Generate a unique ID for the new toilet
  const toiletId: string = nanoid();

  const [ newToilet ] = await db.insert(toilets).values({
        id: toiletId,
        name,
        address,
        // Convert location to a spatial point IMPORTANT! Use SRID 4326
        location: sql`ST_SetSRID(ST_MakePoint(${location.x}, ${location.y}), 4326)`,
        handicapAvail,
        bidetAvail,
        showerAvail,
        sanitiserAvail,
        rating: sql`${rating}::decimal`
  }).returning()
```

backend > src > middleware > auth.ts > ...

```
import { Context, Next } from 'hono'
import jwt from 'jsonwebtoken'
import { JWTPayload } from '../types/auth'

const JWT_SECRET = process.env.JWT_SECRET || 'your-secret-key'
 ⌘I for Command, ⌘L for Cascade
/**
 * Authentication middleware that checks if a valid JWT token is present in the Authori
 *
 * If the token is present and valid, it decodes the token and sets the `user` property
 * If the token is invalid or not present, it returns a 401 response.
 *
 * @param {Context} c - The Hono Context object.
 * @param {Next} next - The next middleware function to call.
 */
export async function authMiddleware(c: Context, next: Next) {
  // Auth header format: Bearer <token>
  const token = c.req.header('Authorization')?.split(' ')[1]

  if (!token) {
    return c.json({ error: 'No token provided' }, 401)
  }

  try {
    const decoded = jwt.verify(token, JWT_SECRET) as JWTPayload
    /// Forward the request to the next handler with the decoded user
    c.set('user', decoded)
    await next()
  } catch (error) {
    /// User is not authenticated, return 401
    return c.json({ error: 'Invalid token' }, 401)
  }
}
```

peepal

2.Consistent Code Style & Naming Convention

Front-end

Dart

Prettier

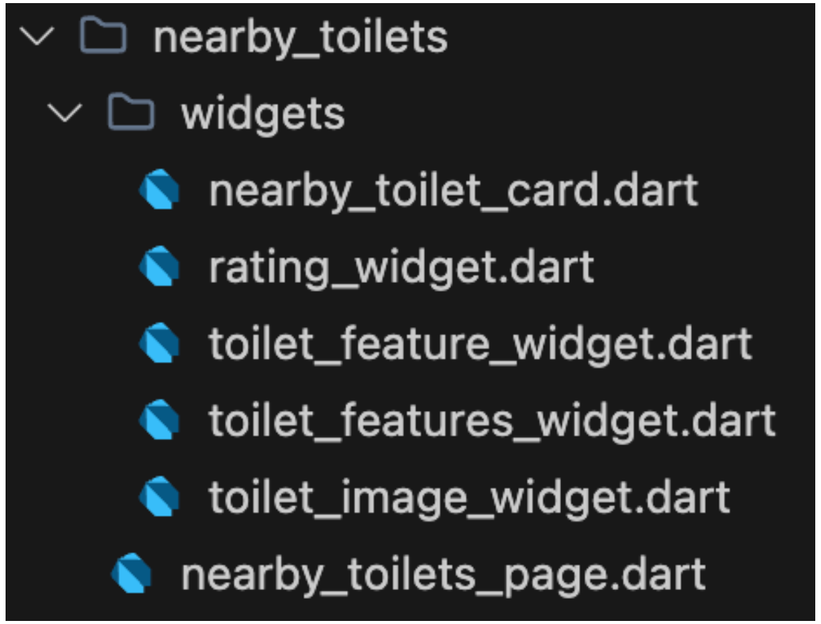
Back-end

ESLint

Ensures that Code is readable, and maintainable, and understandable

peepal

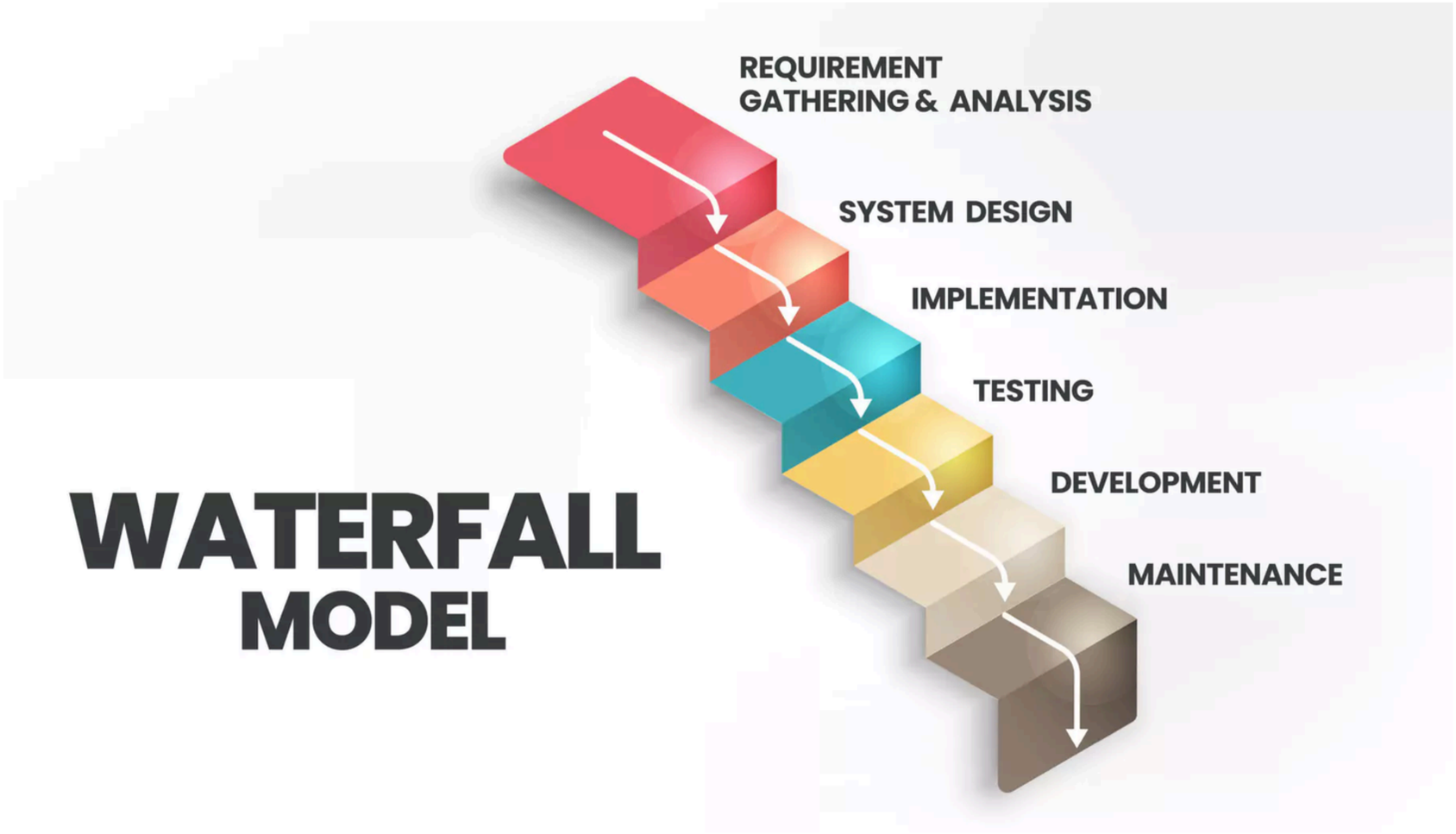
3. Reusability & Refactoring




```dart
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
    ConstrainedBox(
      constraints: BoxConstraints(maxWidth: 240.0),
      child: AutoSizeText(
        toilet.name,
        style: const TextStyle(
            fontSize: 22.0, fontWeight: FontWeight.bold),
        maxLines: 2,
        overflow: TextOverflow.ellipsis,
      ),
    ),
    RatingWidget(rating: toilet.rating),
  ],
),
SizedBox(height: 5.0),
Text(
  toilet.address,
  maxLines: 2,
  style: TextStyle(
      height: 1.6,
      color: const Color(0xFF2D2D2D),
      fontSize: 14.0),
),
SizedBox(height: 12.0),
ToiletFeatureWidget(
  hasBidet: toilet.bidetAvail,
  hasOku: toilet.handicapAvail,
  hasShower: toilet.showerAvail,
  hasSanitizer: toilet.sanitiserAvail,
),
const Spacer(),
```

peepal

4. Waterfall Structure

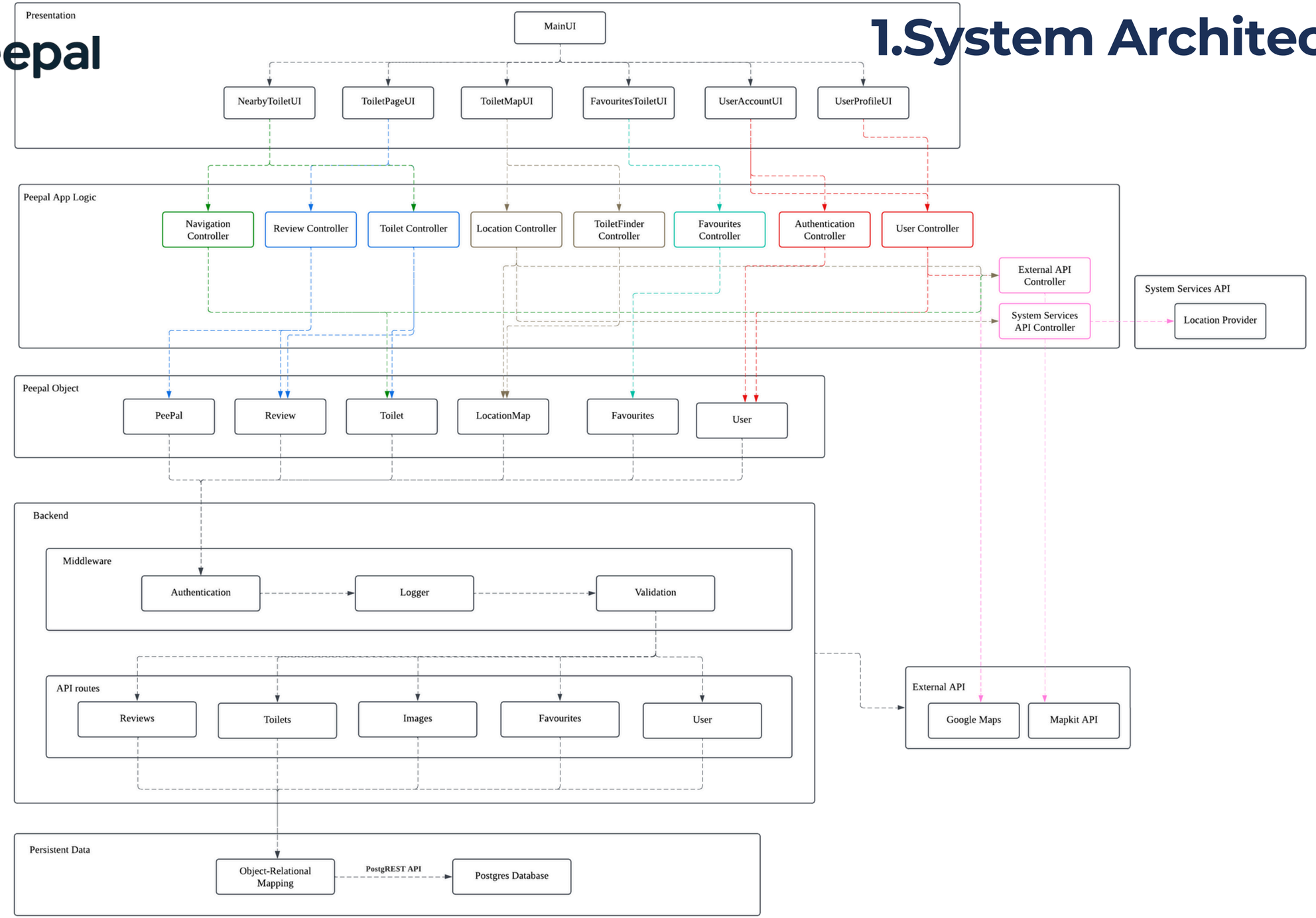


peepal

System Design

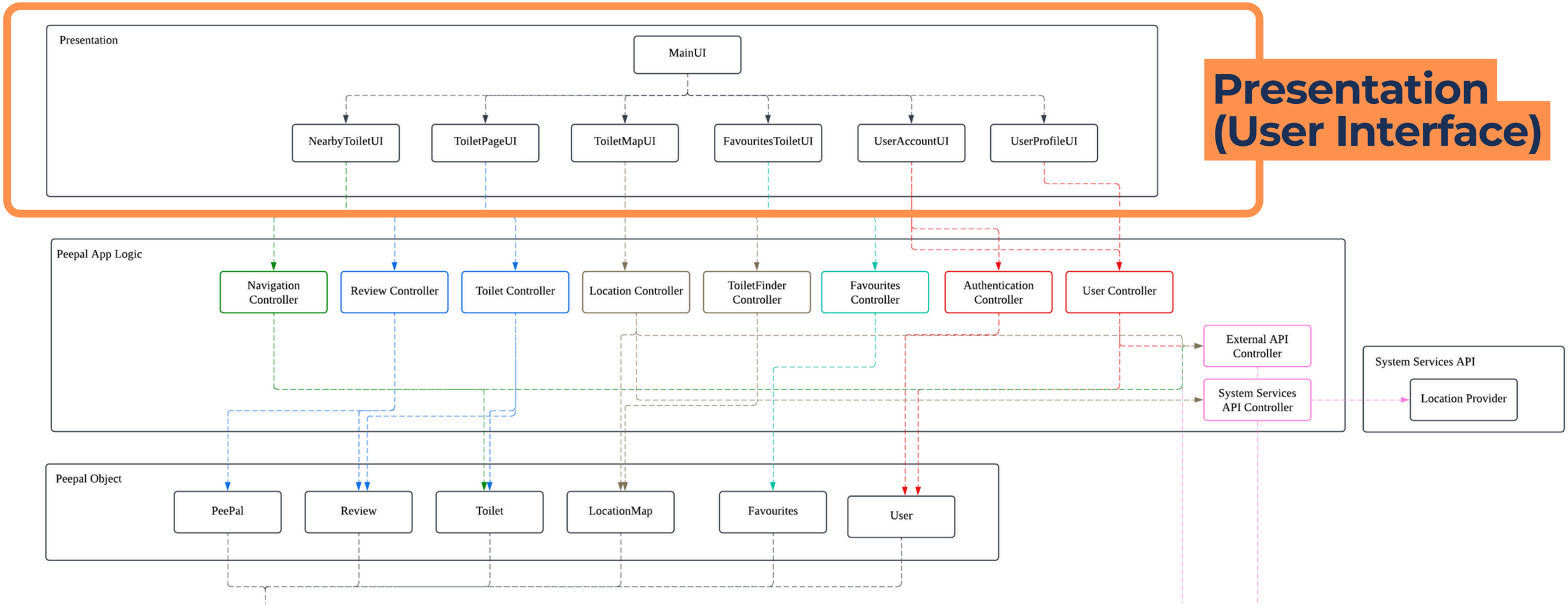
peepal

1.System Architecture



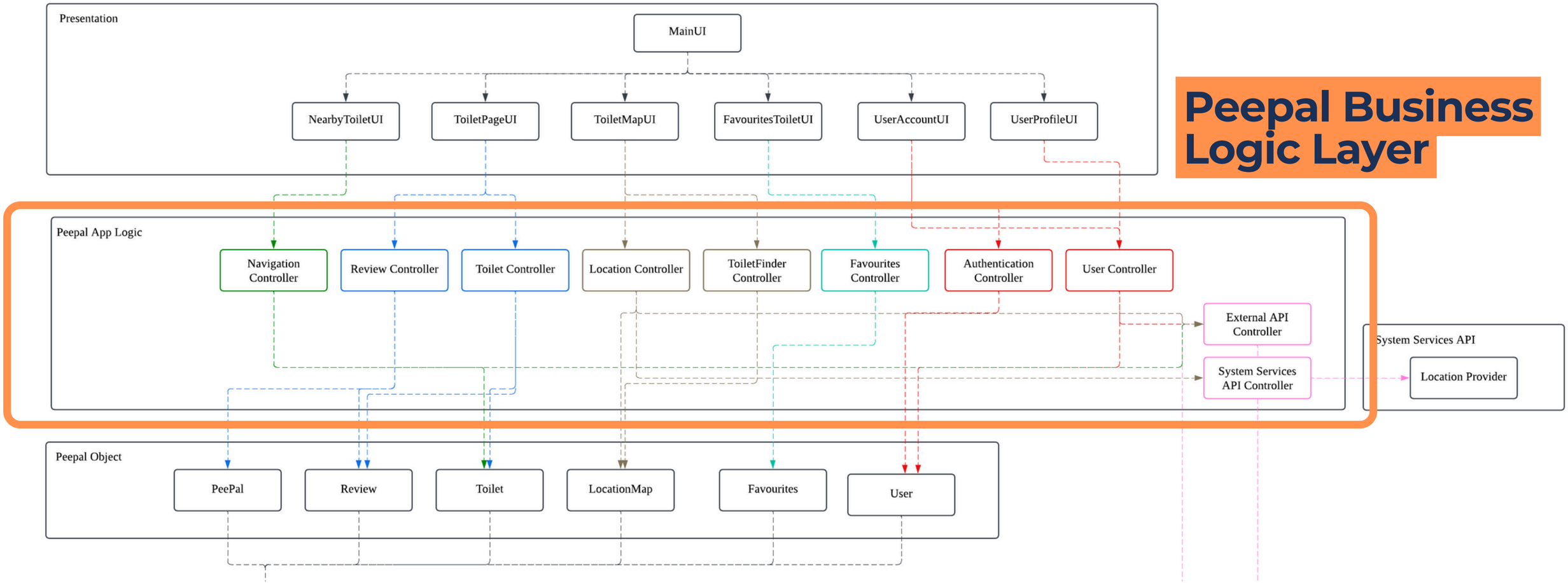
peepal

1.System Architecture



peepal

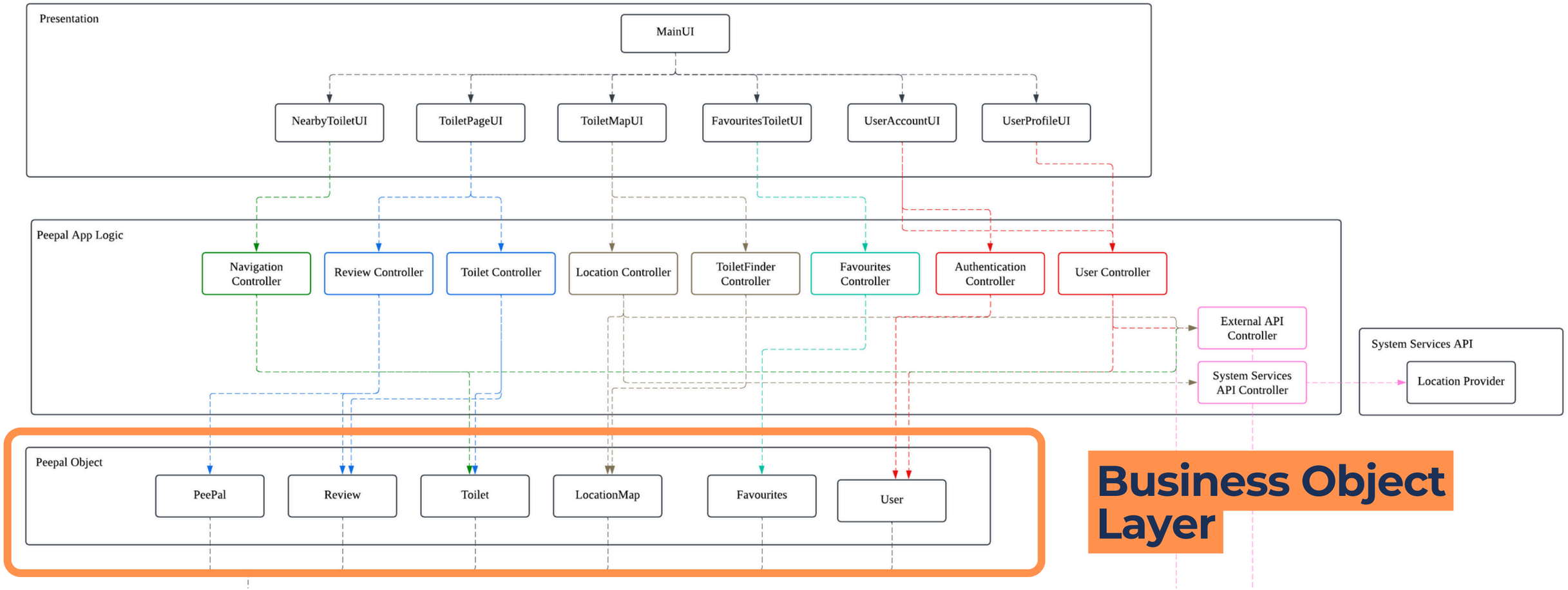
1.System Architecture



Peepal Business Logic Layer

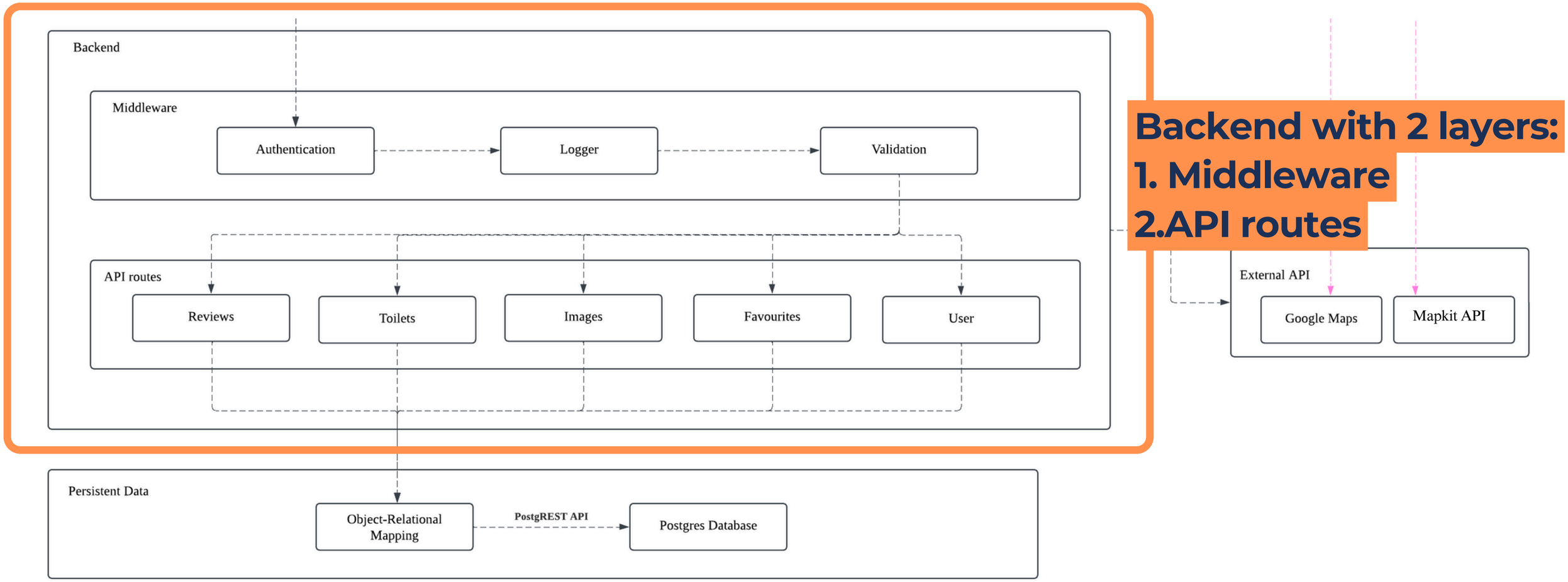
peepal

1.System Architecture



peepal

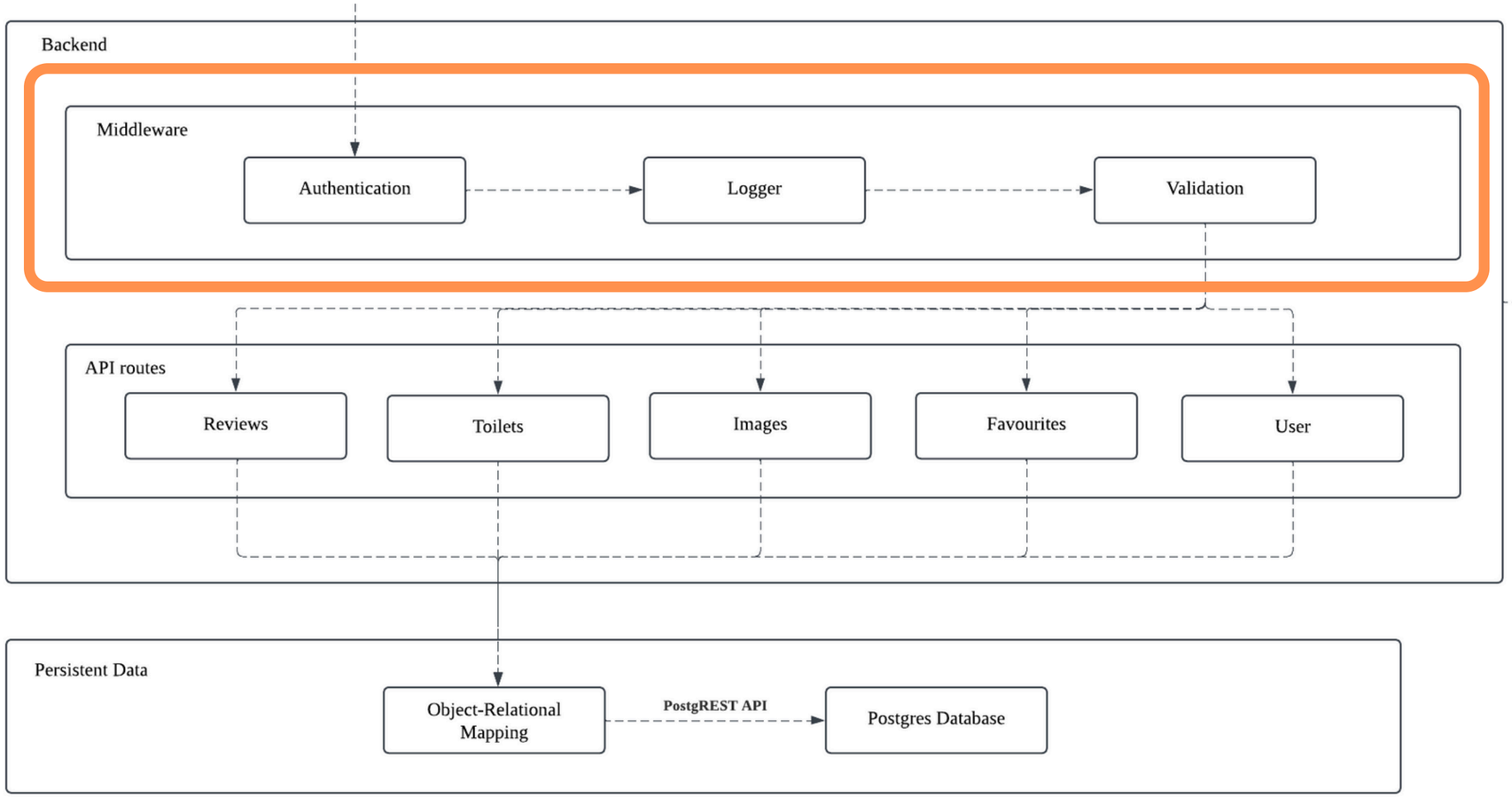
1.System Architecture



Backend with 2 layers:
1. Middleware
2.API routes

peepal

1.System Architecture



Middleware handle

1.Authentication

2.Logging

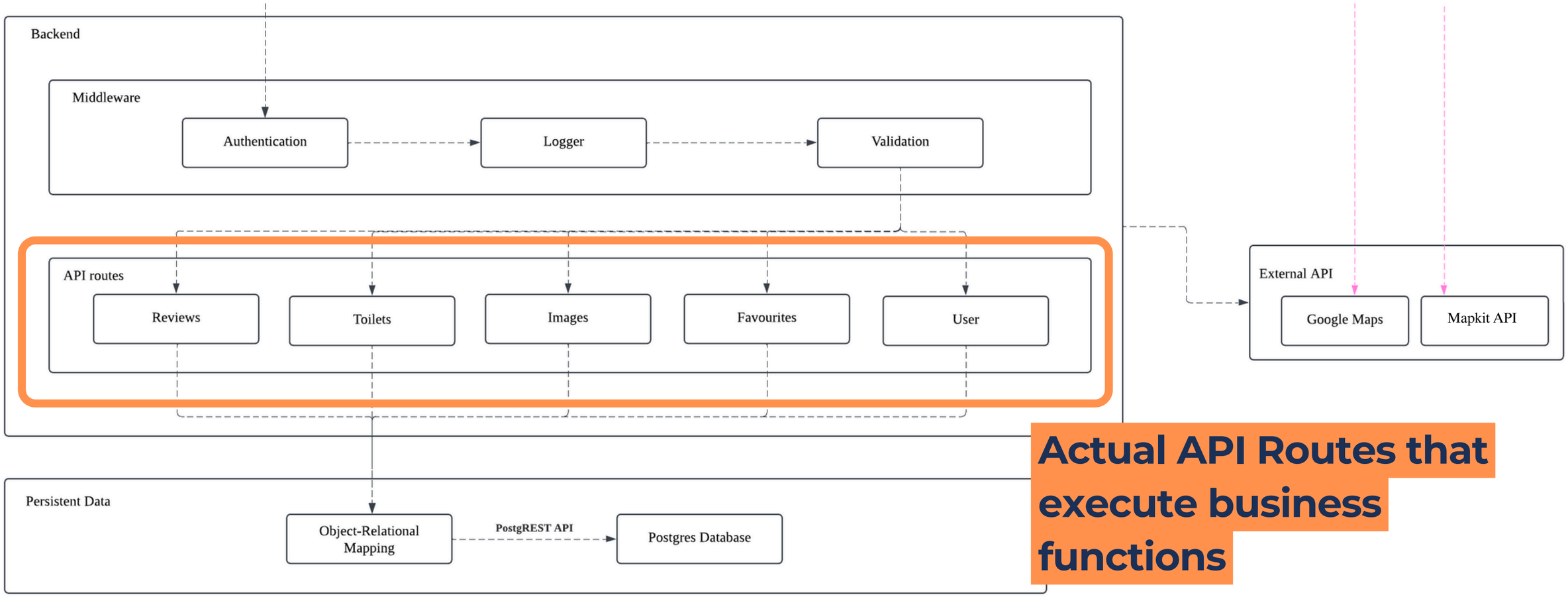
3.Data Validation

Google Maps

Mapkit API

peepal

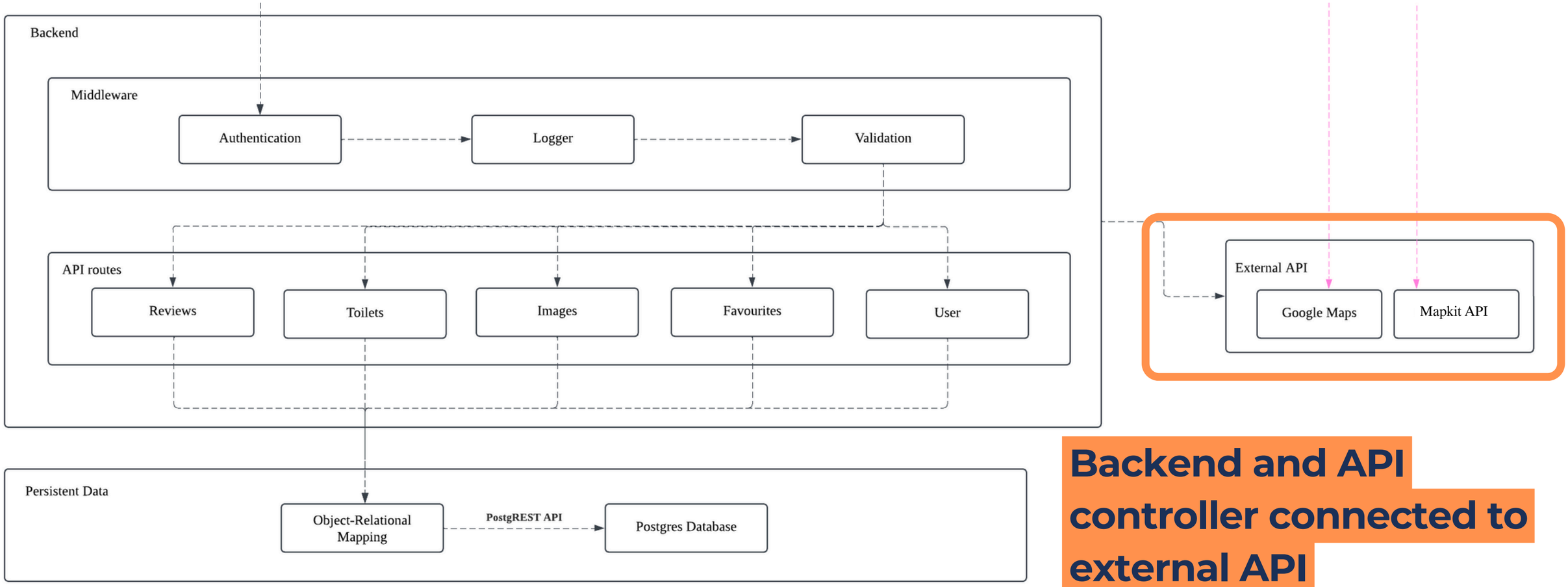
1.System Architecture



Actual API Routes that execute business functions

peepal

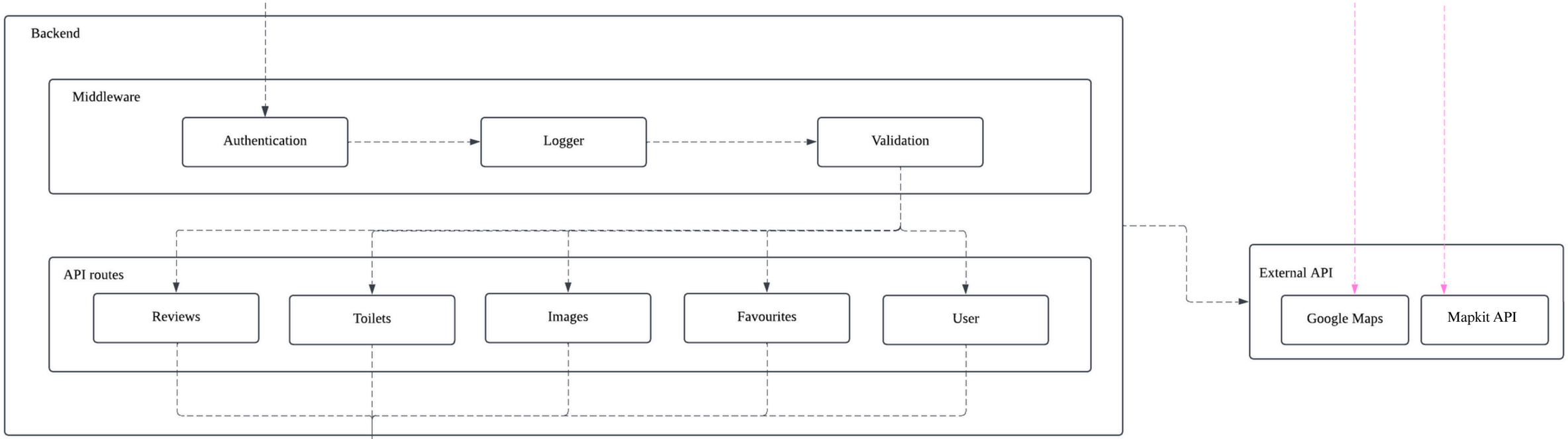
1.System Architecture



Backend and API controller connected to external API

peepal

1.System Architecture

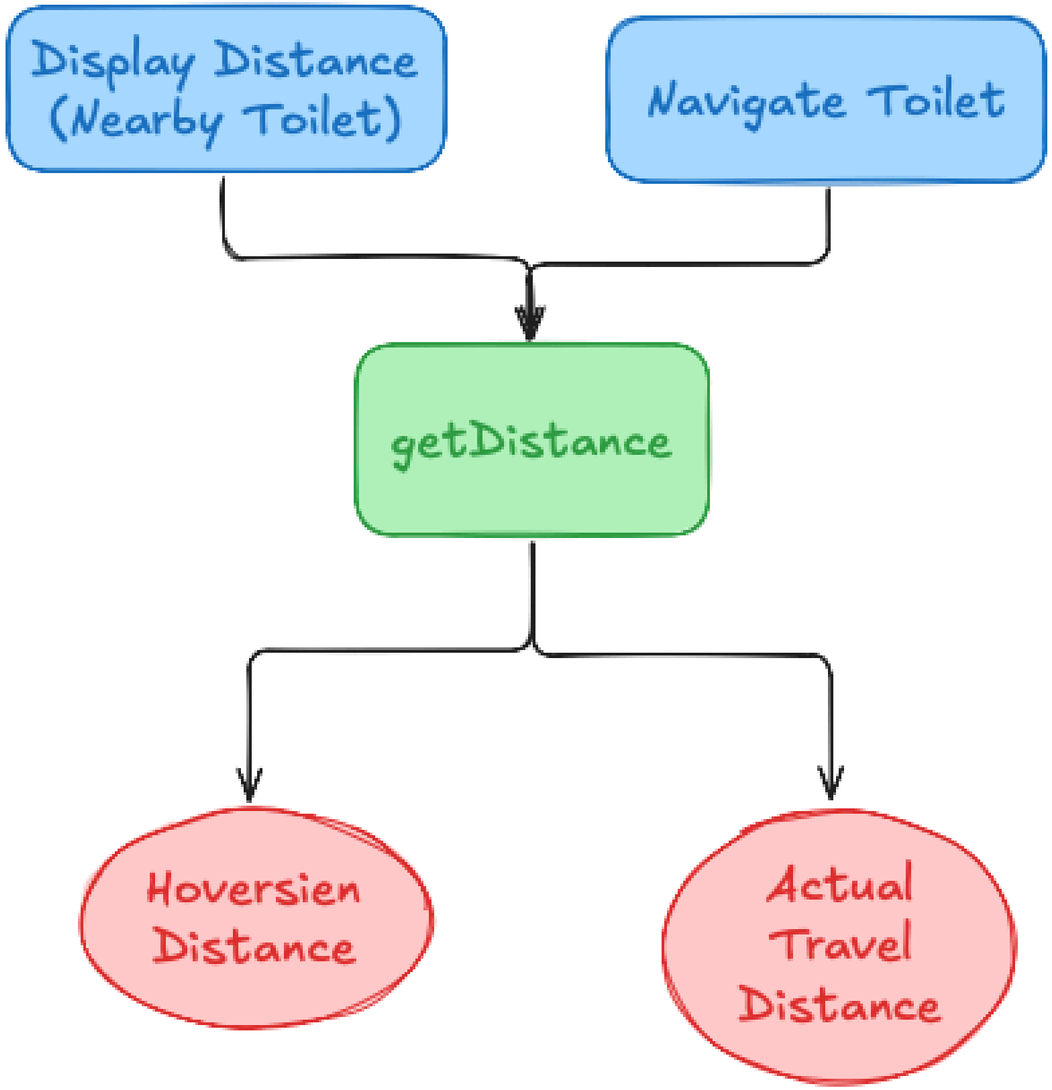


Persistent Data layer handle object-relational mapping to Postgres Database

peepal

Design Patterns

peepal

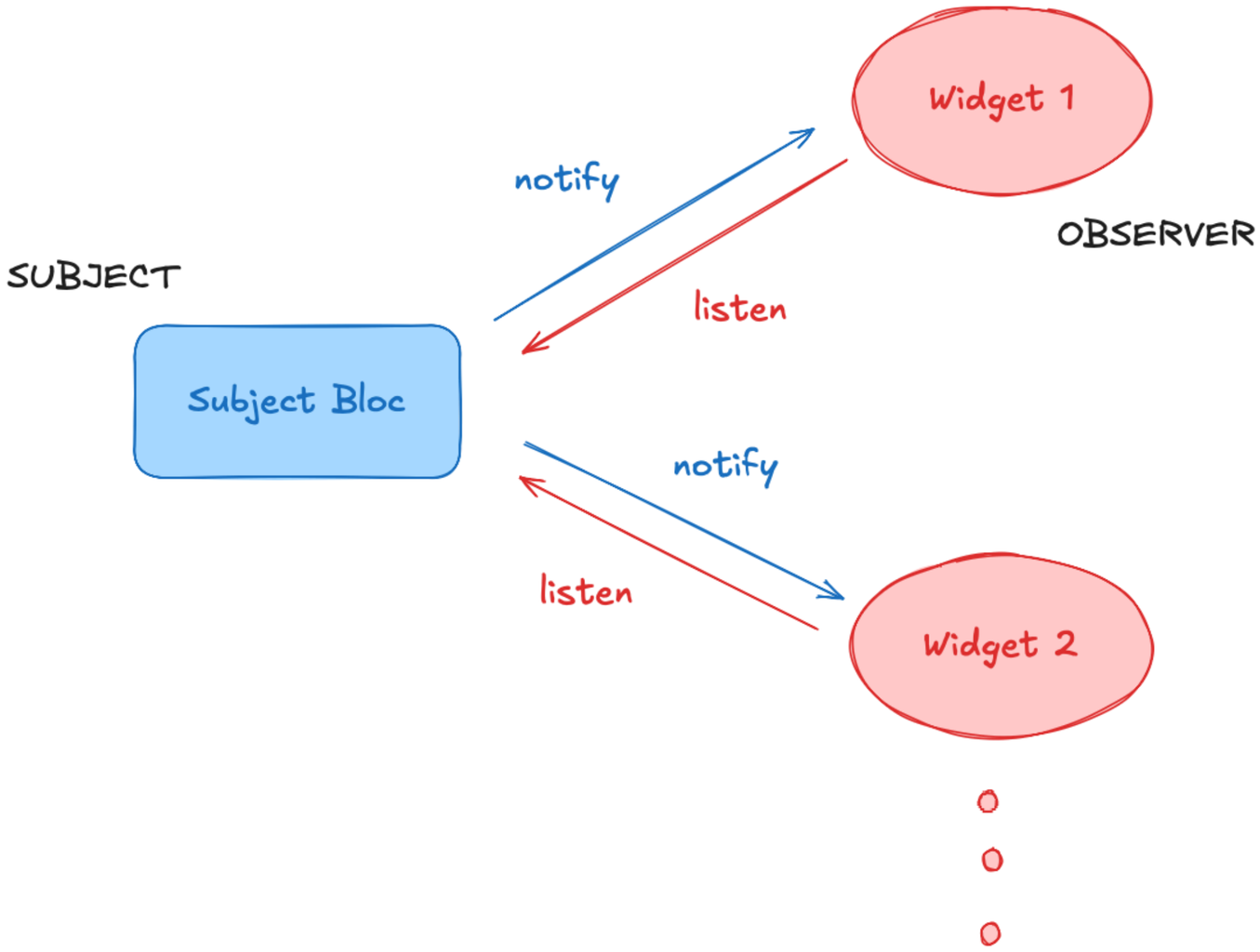


1. Strategy Pattern

- The system will get the location distance according to the widget specification.

- There are different implementations and calculation methods to get distance.

peepal

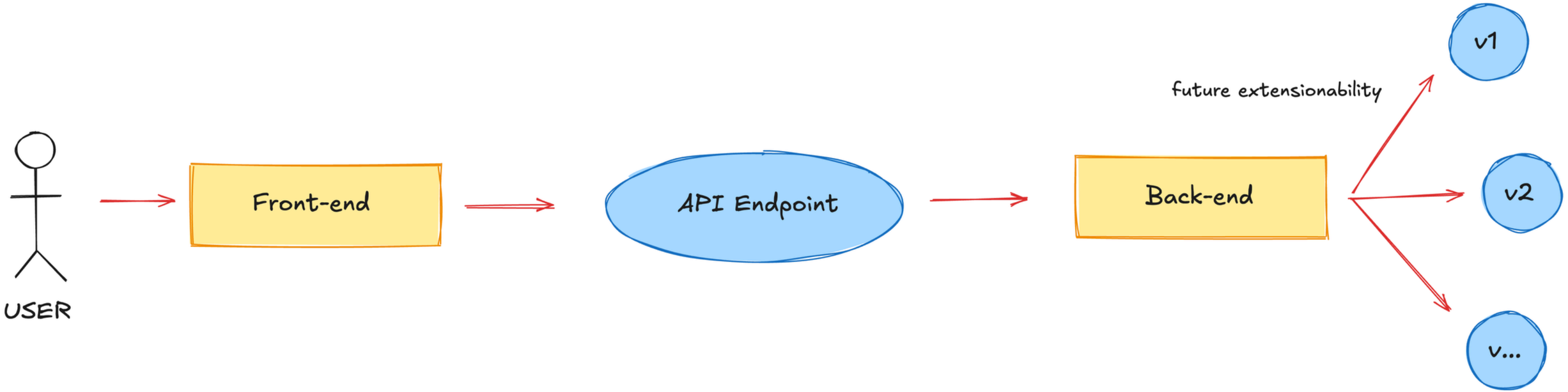


2. Observer Pattern

- UI components (Observers) subscribe to the bloc using **BlocBuilder** or **BlocListener.**
- When an event occurs, the bloc processes it.
- All subscribed widgets are automatically rebuilt with the state change.

peepal

3. Facade Pattern



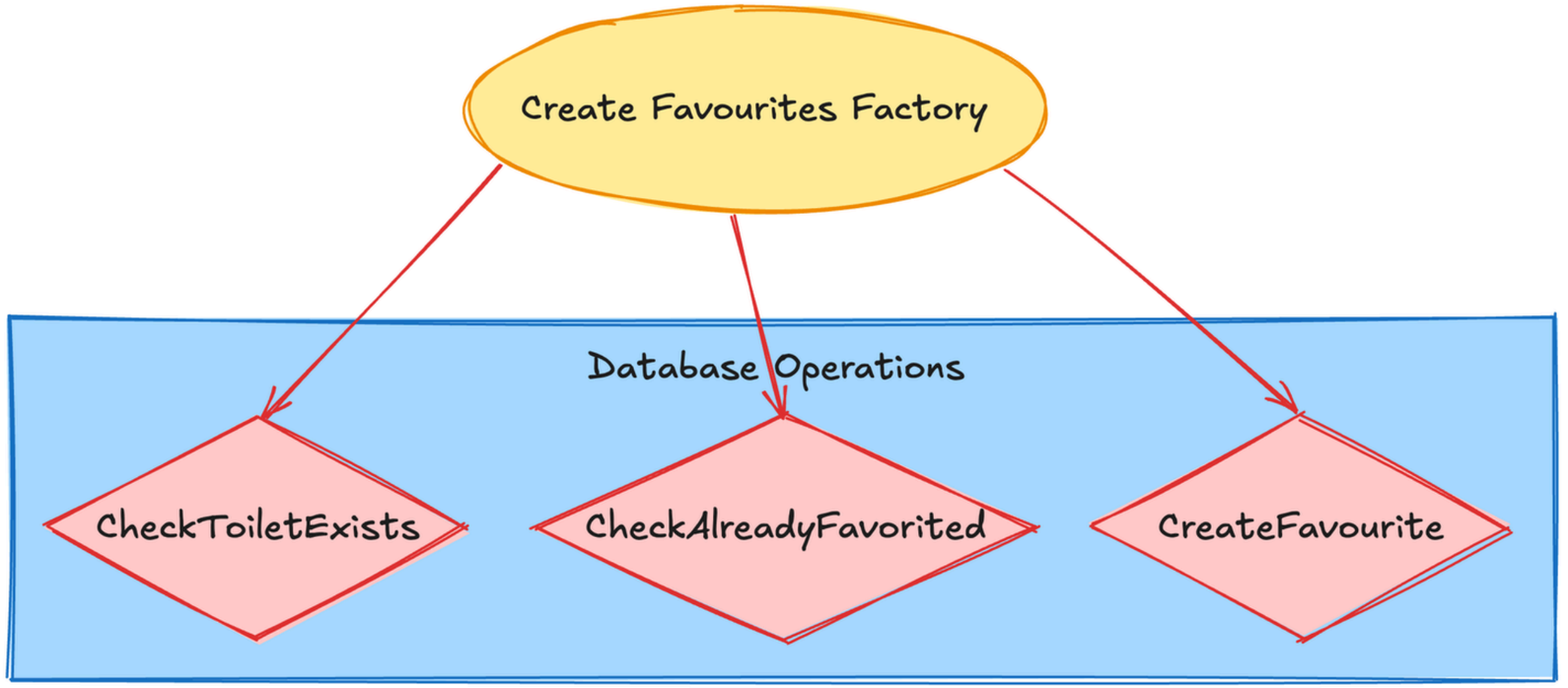
- When a client makes a POST request to /toilets/nearby, the Toilets API facade handles the request through handleGetNearby(). Behind the scenes, it coordinates multiple operations.

- The client only interacts with the simple Toilets API interface, unaware of the complex operations behind the scenes.

- This pattern improves code maintainability by encapsulating complexity and enhancing usability by providing a clean, simple interface for complex operations.

peepal

4. Factory Pattern



- The CreateFavoriteFactory manages favorite creation through the Factory Pattern.
- It first checks if the toilet exists, then verifies if it's already favorited, and finally creates the new favorite.
- This pattern hides complexity, maintains a single entry point, and makes modifications easy without affecting other parts of the system.

peepal

Traceability in Project Deliverables

peepal

NearbyToilet

Use-case Description

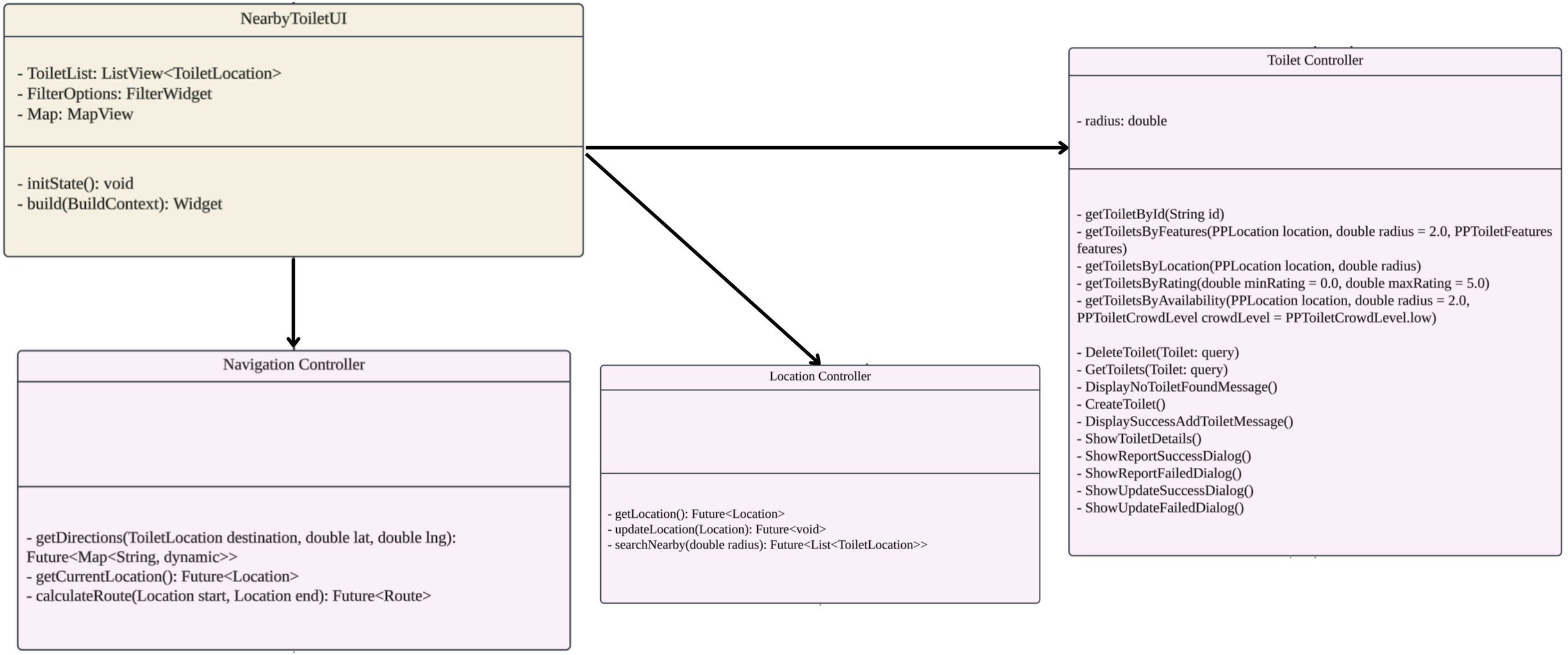
Use Case ID:	#4-7		
Use Case Name:	NearbyToilet		
Created By:	Jia Wei	Last Updated By:	Jia Wei
Date Created:	13/04/2025	Date Last Updated:	13/04/2025

Actor:	User
Descriptions:	Display the information of the nearest 5 toilets in the card format.
Preconditions:	1. The User must be logged in and authenticated.
Postconditions:	1. Successfully display the nearby list of toilets.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The system will detect the user's current location. 2. The system will find the user's nearest 5 toilets. 3. The system will display the information in the card format. 4. The user can select any toilet card and the system will invoke **ToiletMenu** use case. 5. If the user selects the "Navigate" button, the system will invoke **ViewToiletOnMap** use case.
Alternative Flows:	None
Exceptions:	None
Includes:	1. ToiletMenu 2. ViewToiletsOnMap
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

peepal

NearbyToilet

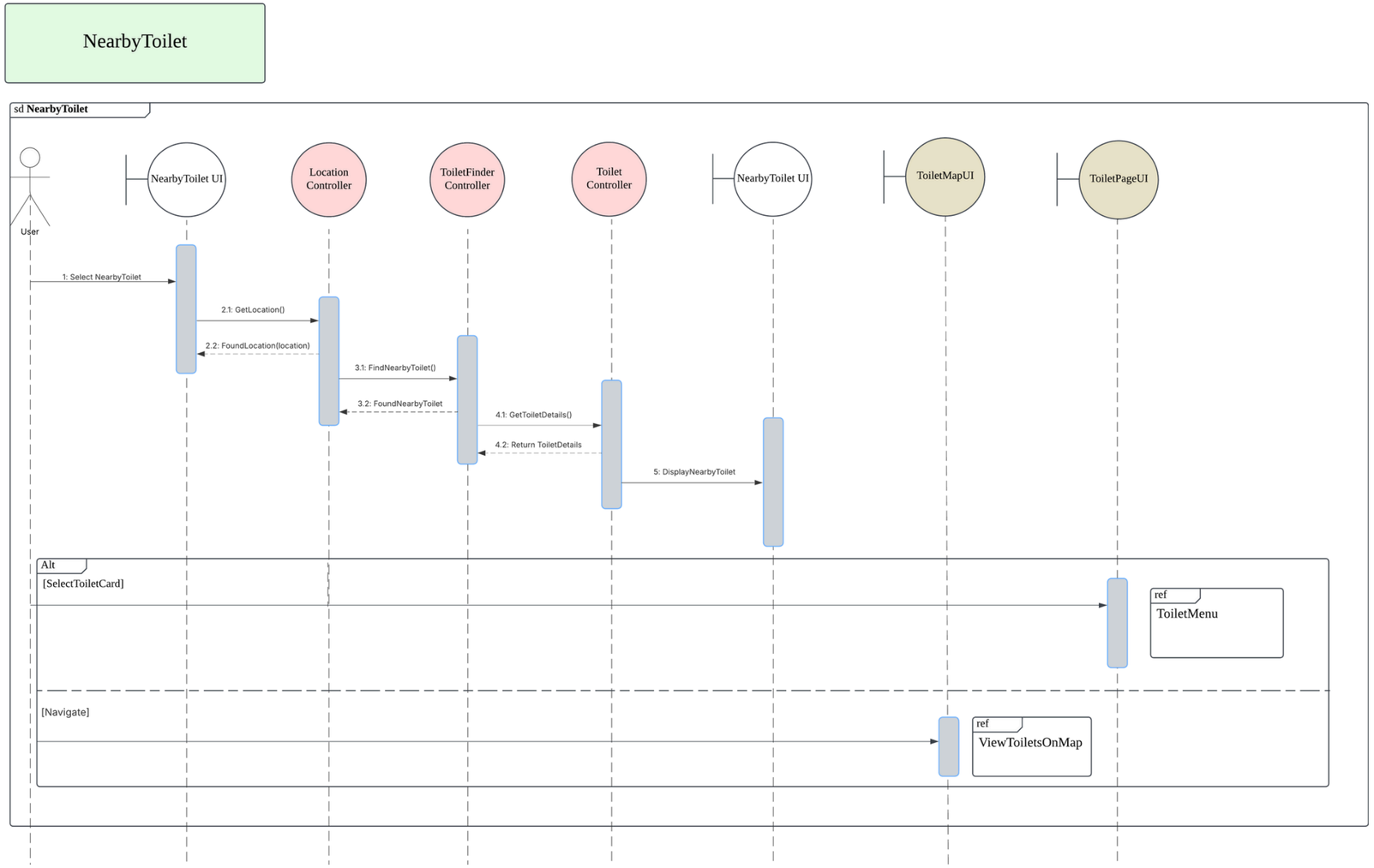
Relevant Class Diagram



peepal

NearbyToilet

Relevant Sequence Diagram



peepal

NavigateToilet - Good Designs Applied

Facade Pattern

```
356  validator('json', navigateToiletSchema), async (c) => {
381
382    const mkRoute: MKRoute = directionsResponse.data.routes[0]
383    const mkSteps: MKStep[] = directionsResponse.data.steps
384    const mkStepPaths: MKCoordinate[][] = directionsResponse.data.stepPaths
385
386    const stepPathTransformed: LatLng[][] = mkStepPaths.map(stepPath => stepPath.map(coord => ({ lat: coord.latitude, lng: coord.longitude })))
387    const stepPathPolylines: string[] = stepPathTransformed.map(path => encode(path))
388
389    const directions: RouteDirection[] = []
390
391    for
392      di
393        distance: formatDistance(step.distanceMeters),
394        duration: formatDuration(step.durationSeconds),
395        polyline: stepPathPolylines[step.stepPathIndex],
396        start_location: stepPathTransformed[step.stepPathIndex][0],
397        end_location: stepPathTransformed[step.stepPathIndex][stepPathTransformed[step.stepPathIndex].length - 1],
398        instructions: step.instructions
399      })
400    }
401
402    const overviewPolylinePoints: {
403      x: number;
404      y: number;
405    }[] = stepPathTransformed.flat().map(coord => ({ x: coord.lng, y: coord.lat }))
406
407    const overviewPolyline: string = encode(simplify(overviewPolylinePoints, 0.0001, false).map(coord => [coord.y, coord.x]))
408
409    const distanceString: string = formatDistance(mkRoute.distanceMeters)
410    const durationString: string = formatDuration(mkRoute.durationSeconds)
411
412    const route: Route = {
413      overview_polyline: overviewPolyline,
414      start_location: {
415        lat: directionsResponse.data.origin.coordinate.latitude,
416        lng: directionsResponse.data.origin.coordinate.longitude
417      },
418      end_location: {
419        lat: directionsResponse.data.destination.coordinate.latitude,
420        lng: directionsResponse.data.destination.coordinate.longitude
421      },
422      distance: distanceString,
423      duration: durationString,
424      directions: directions
425    }
426
427    return c.json({ route: route }, 200);
```

(property) RouteDirection.distance: string

distance: string;

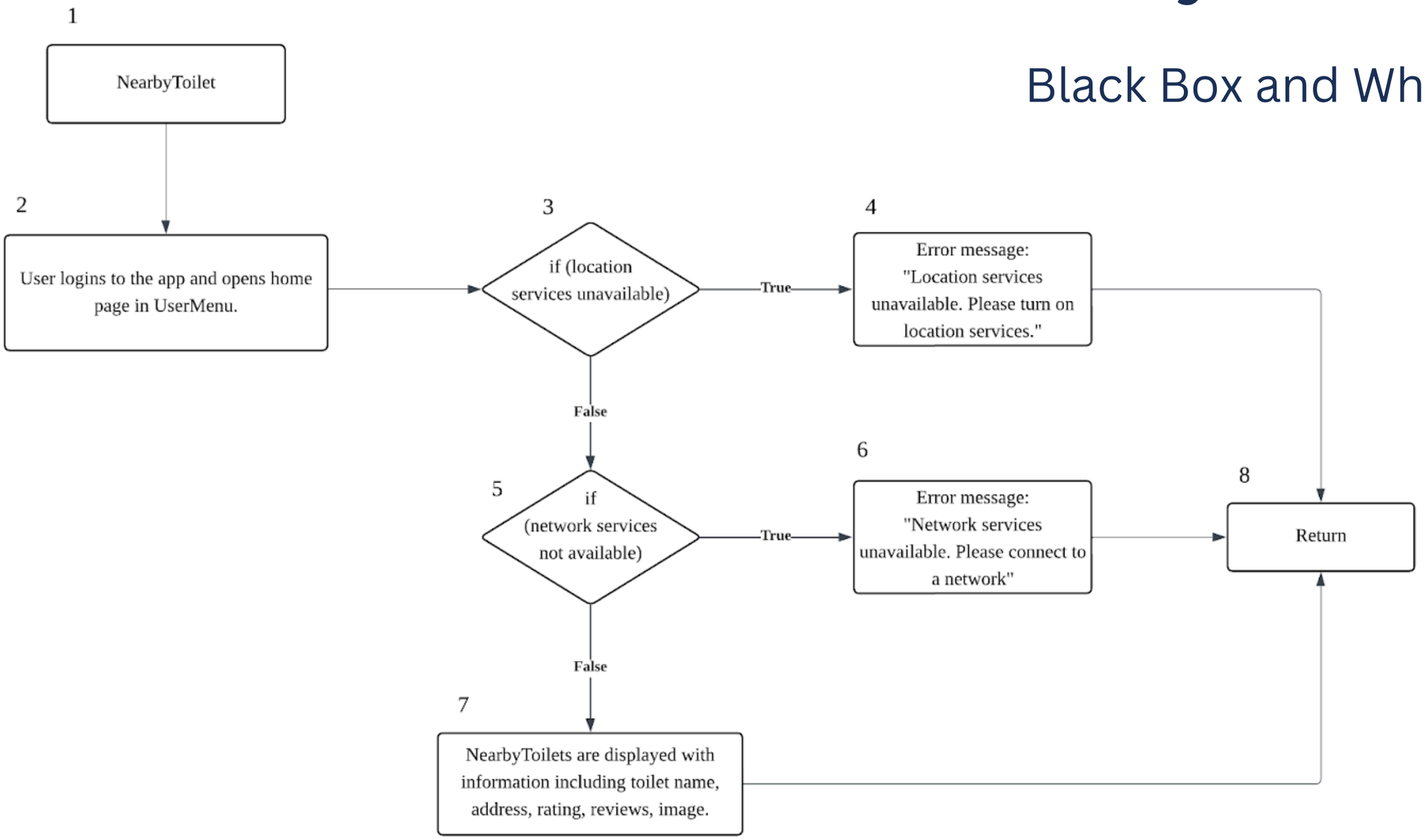
The single NavigateToilet API route handles many complex function calls:

1. Getting the MapKit **access token**
2. Getting the **directions**
3. **Parsing** the directions data
4. **Encoding** the data into polylines and routes that can be easily interpreted by the client.

peepal

NearbyToilet - Tests

Black Box and White Box



peepal

NearbyToilet - Tests

Black Box and White Box

I.II Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = 2 + 1 = 3

Basis Paths

1. Baseline path: 1,2,3,5,7,8
2. Basis path 2: 1,2,3,4,8
3. Basis path 3: 1,2,3,5,6,8

peepal

NearbyToilet - Tests

Black Box and White Box

I.III Test Cases and Results

NearbyToilet

No.	Test Input	Expected Output	Actual Output	Pass?
1.	Location services available. Network services available.	Nearby Toilets displayed successfully.	Nearby Toilets displayed successfully.	Yes
2.	Location services unavailable. Network services available.	Error message: "Location services unavailable. Please turn on location services."	Error message: "Location services unavailable. Please turn on location services."	Yes
3.	Location services available. Network services unavailable.	Error message: "Network services unavailable. Please connect to a network"	Error message: "Network services unavailable. Please connect to a network"	Yes

peepal

Automated Unit Testing

```
 Test Files  5 passed (5)
      Tests  50 passed (50)
   Start at  00:19:54
   Duration  2.13s (transform 359ms, setup 0ms, collect 3.24s, tests 2.58s, environment 1ms, prepare 351ms)

 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

peepal

Continuous Integration/Deployment (CI/CD)

peepal-backend-deployment Private

Unwatch 1

main | 1 Branch | 0 Tags | Go to file t | Add file | Code

github-actions[bot] deployment — 28bb283 · 2 hours ago — 2 Commits

.vscode	deployment	2 hours ago
drizzle	deployment	2 hours ago
scripts	deployment	2 hours ago
src	deployment	2 hours ago
.dockerignore	deployment	2 hours ago
.gitignore	deployment	2 hours ago
README.Docker.md	deployment	2 hours ago
README.md	deployment	2 hours ago
compose.yaml	deployment	2 hours ago
dockerfile	deployment	2 hours ago
drizzle.config.ts	deployment	2 hours ago
package-lock.json	deployment	2 hours ago
package.json	deployment	2 hours ago
tsconfig.json	deployment	2 hours ago
vitest.config.ts	deployment	2 hours ago

Deployments

Visit App

History

Deploy now

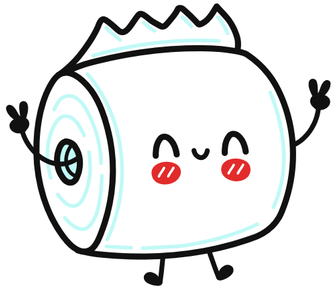
Source	Trigger	Date & Deploy time
deployment 3b061de main	Automatic (on commit)	Apr 16, 2025, 6:49 AM · 1m 0s
deployment 3bff943 main	Automatic (on commit)	Apr 16, 2025, 6:34 AM · 1m 9s
deployment 95666fe main	Manual (on Sevalla) Bryan Soong	Apr 16, 2025, 6:30 AM · 21s
deployment 95666fe main	Automatic (on commit)	Apr 16, 2025, 6:27 AM · 1m 53s
deployment 0144445 main	Automatic (on commit)	Apr 16, 2025, 6:09 AM · 1m 6s
deployment 97c7e79 main	Manual (on Sevalla)	Apr 16, 2025, 4:50 AM · 22s
deployment		Apr 16, 2025, 4:45 AM

Deployment is in progress. Cancel

peepal

Future Plans

Health Notifications
Integrating official government health announcements and push notifications for users.



HappyToilet
Collaborating with the Restroom Association (Singapore) to feature data ratings from their Happy Toilet program.

Expand Globally
Transitioning from a Singapore-focused toilet database to a global platform.

peepal

Thank you!

Your personal toilet buddy.

peepal

Video Demo