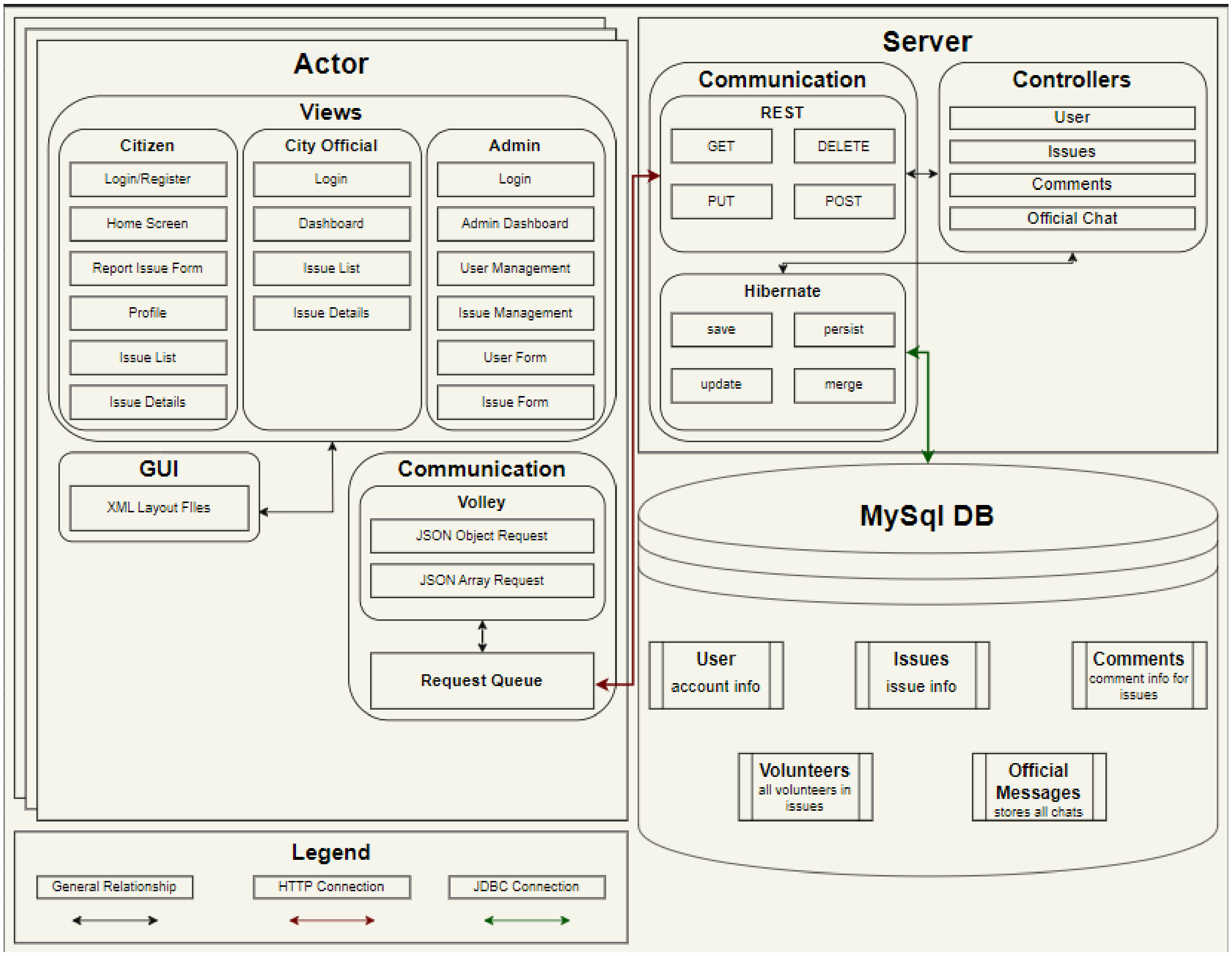
Design Document for CityWatcher

Group 2_jabir_2

Nicholas Morrow: 95%

Sam Hostetter: 5%

Sam Rowland: 0%


Actor
Views
Citizen
Login/Register
Home Screen
Report Issue Form
Profile
Issue List
Issue Details
City Official
Login
Dashboard
Issue List
Issue Details
Admin
Login
Admin Dashboard
User Management
Issue Management
User Form
Issue Form
GUI
XML Layout FIles
Communication
Volley
JSON Object Request
JSON Array Request
Request Queue
Server
Communication
REST
GET
DELETE
PUT
POST
Hibernate
save
persist
update
merge
Controllers
User
Issues
Comments
Official Chat
MySql DB
User
account info
Issues
issue info
Comments
comment info for issues
Volunteers
all volunteers in issues
Official Messages
stores all chats
Legend
General Relationship
HTTP Connection
JDBC Connection

Actor:

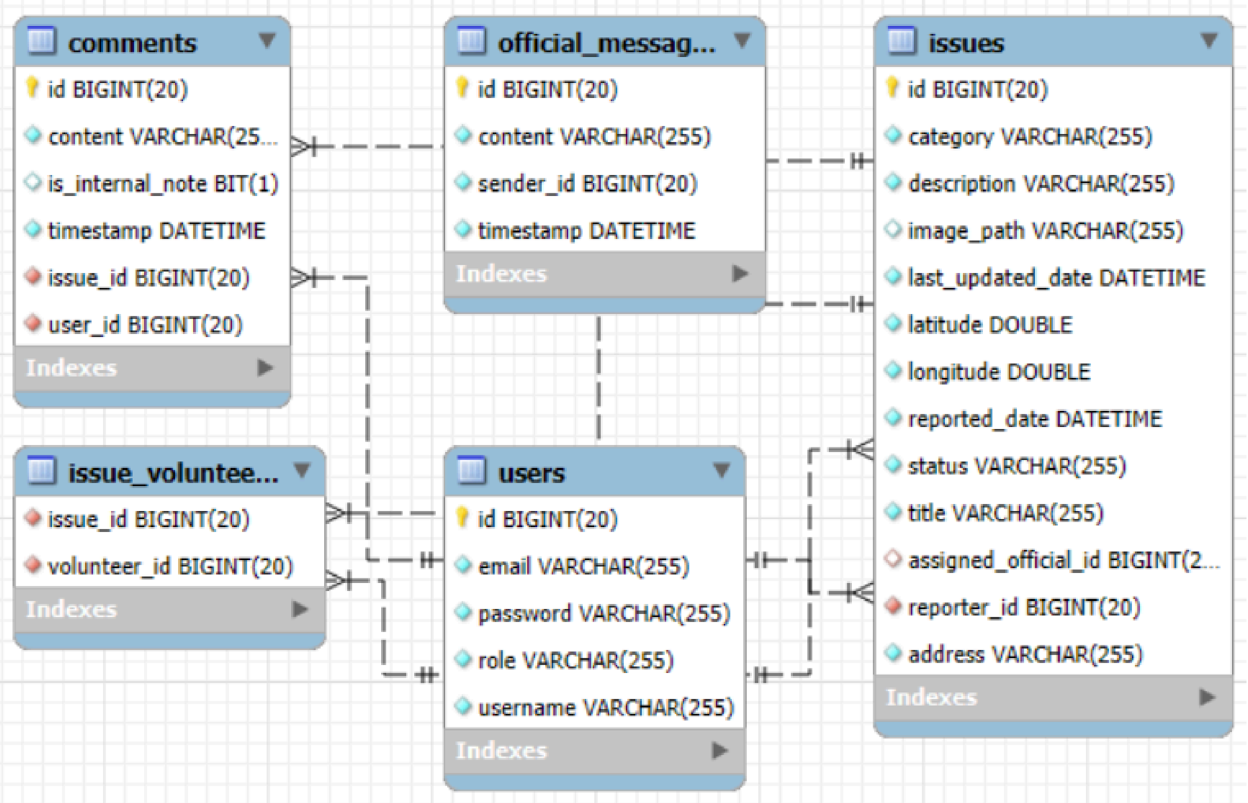
The Actor interacts with the Android application through various views that handle user input and display the UI. The main screens include functionalities for issue reporting, issue tracking, and user management. For example, when a user reports an issue, they fill out a form on the issue reporting screen. Upon submission, the view calls the presenter responsible for that action, which in turn triggers an API request to the server via the ApiService. Once the server processes the request and responds (e.g., confirming the issue was successfully reported), the presenter updates the view accordingly. This flow is consistent across other actions, such as logging in, signing up, and managing user profiles. The app's architecture follows the Model-View-Presenter (MVP) pattern, ensuring a clean separation of concerns between the UI, business logic, and data handling.

Server:

The server is built using Spring Boot and follows a layered architecture. It consists of Controllers, Services, and Repositories. Controllers handle incoming API requests and map them to the correct service methods. Services contain the core business logic, such as handling issue assignments, managing issue statuses, or handling user authentication. Repositories manage the data access logic, interacting with a MySQL database to fetch or store data. For instance, when a user submits a new issue, the IssueController receives the request, passes it to the IssueService for processing, which in turn interacts with the IssueRepository to store the issue in the database. The server also handles more complex tasks, such as managing relationships between different entities like users and issues, ensuring data consistency.

Database:

The database is powered by MySQL and serves as the central repository for all system data. It stores key entities such as users, messages, issues, and comments. The Users table captures information about citizens, city officials, and administrators, while the Issues table manages user-reported problems like potholes, graffiti, or other city-related issues. Each issue is linked to a category and has a status (e.g., Under Review, Resolved) to track its progress. The database schema is designed to maintain referential integrity across relationships, ensuring that data remains consistent and accurate as users interact with the system.



comments
- id BIGINT(20)
- content VARCHAR(25...
- is_internal_note BIT(1)
- timestamp DATETIME
- issue_id BIGINT(20)
- user_id BIGINT(20)
- Indexes

official_messag...
- id BIGINT(20)
- content VARCHAR(255)
- sender_id BIGINT(20)
- timestamp DATETIME
- Indexes

issues
- id BIGINT(20)
- category VARCHAR(255)
- description VARCHAR(255)
- image_path VARCHAR(255)
- last_updated_date DATETIME
- latitude DOUBLE
- longitude DOUBLE
- reported_date DATETIME
- status VARCHAR(255)
- title VARCHAR(255)
- assigned_official_id BIGINT(2...
- reporter_id BIGINT(20)
- address VARCHAR(255)
- Indexes

issue_voluntee...
- issue_id BIGINT(20)
- volunteer_id BIGINT(20)
- Indexes

users
- id BIGINT(20)
- email VARCHAR(255)
- password VARCHAR(255)
- role VARCHAR(255)
- username VARCHAR(255)
- Indexes