COSC349 A1 - Project Report

Callum Sullivan - 7234739, Anthony Dong - 2169260

Introduction

The Flat Bills Tracker is a web application designed to overview and manage property bills and payments in shared living situations. This report details the development, implementation and architecture of the application, with a focus on how virtualisation improves the overall portability and deployment across different environments. The application addresses common challenges that are faced in shared accommodations, where tenants must manually track and split property expenses. Its goal is to promote transparency among housemates and simplify the process of splitting and managing bills. The application utilises a range of different web development technologies including HTML, CSS, JavaScript and EJS for the frontend, Node.js with Express.js for the backend, MySQL for the database management and Docker for containerization. This technology stack enables the creation of a responsive, user-friendly interface while ensuring robust data management and scalability of the application.

Application Design and Use of Virtual Machines

The Flat Bills Tracker employs a microservices architecture, consisting of three main components:

1. Tenant Service
The Tenant Service manages all tenant-related functionalities, this includes the initial registration, input validation, user authentication, bill management and property assignment. Built with Node.js, Express.js and EJS to render an intuitive user interface.

2. Manager Service
The Manager Service oversees property operations, this includes property overview and management, tenant oversight and bill assignment. Built on the same technological foundation as the Tenant Service - Node.js, Express.js and EJS. The Manager Service also shares similar functionalities for the user registration and authentication.

3. Database Service

The Database Service implements MySQL which stores and manages all application data. It includes tables for the Tenants, Property Managers, Properties and Bills, forming the backbone of data persistence for the entire application.

Each of these services is encapsulated within its own Docker container, functioning as a lightweight, isolated virtual machine. This containerization approach offers several significant advantages. Primarily, it provides a high degree of isolation between services which reduces the risk of conflicts and simplifies the debugging and maintenance processes as errors can be pin-pointed without disrupting the entire application. For example, if the Tenant Service experiences an issue, it won't affect the Manager Service or Database Service. Alongside this, the overall application becomes more portable as containers package the application and its dependencies together, ensuring that there's consistent behaviour across different environments. By implementing GitHub as our repository, we were able to facilitate version control, allowing for easier rollbacks and more efficient development as users can work on different containers at the same time and not have to worry about merge conflicts on the main branch. The last benefit of using containerization is that it enables our application to be scalable. This means individual components can be scaled horizontally if needed. For example, if the Tenant Service experiences a surge in user traffic, we can deploy multiple instances of this container to distribute the load without affecting the other services.

Automated Build Process

The application utilises Docker Compose for orchestration, enabling an unattended build and start process. To set up the application, users simply need to clone the Git repository and run `docker-compose up --build`. This command builds the necessary Docker images and starts the containers, setting up the entire application stack automatically.

The repository includes SQL scripts in the database directory (`schema.sql` and `data.sql`) that are automatically executed when the database container starts, preloading the application with test data for demonstration purposes.

The main software components used in the project include Docker Compose, Express.js, MySQL, and several npm packages such as mysql2, cors, body-parser, multar and express-session. The application also incorporates CSS files and EJS templates for the front end.

The size of the three containers varies. The manager and tenant containers, each with Node.js and various packages installed, take up approximately 900MB of space when

checked in the terminal. The MySQL database container, on the other hand, consumes around 600MB of space. These sizes provide insights into the resource requirements of each component and contribute to the overall footprint of the Flat Bills Tracker application.

The build process is largely similar for both first-time builds and subsequent redeployments. The `docker-compose.yml` file defines the structure and dependencies of the application, ensuring consistency across builds. Both the tenant and manager services use similar Dockerfiles, which set up a Node.js environment, install dependencies, and use a wait-for-it script to ensure the database is ready before starting the application.

Suggested Improvements

Two potential improvements that a developer joining the project could make are:

1. Landlord approval system for tenant property assignments.

Currently, tenants can instantly rent a property without the landlord's permission. This isn't how it would normally work in the real world so a suggested improvement to this application would be a system where tenants send a request to the landlord/manager to rent a property. This would allow the landlord/manager to either accept or deny their request. To implement this, the database would have to be changed to handle the rental requests. The backend would need new API endpoints for submitting, viewing and responding to rental requests alongside frontend changes for the tenant to send the request and the landlord to accept/decline the request.

2. File storage system for document verification

When applying to rent a property, a tenant needs to prove their identity through multiple documents. This feature would allow tenants to upload important documents such as bank statements, ID verification or proof of address to help with the application process. Implementation would require setting up file storage (potentially using cloud storage solutions), adding file upload handling to the backend services, and creating new UI components for file management.

To implement these changes, developers would need to modify the relevant server files, update the database schema, and adjust the front-end code. After making these changes,

they would rebuild and rerun the application using the `docker-compose up --build` command to ensure all services are updated with the new functionality.

Repository Structure and Development Process

The repository is organised with a clear separation between the different components of the application. The `tenant` and `manager` directories contain the respective service code and Dockerfiles. The `database` directory holds the SQL scripts for schema and initial data. The `docker-compose.yml` file in the root directory orchestrates the entire application.

The development process followed an incremental approach which is reflected through our commit history on GitHub. Originally we began development on our database as this is the foundation to our project. After we finalised the schema, we focused on the tenant service by getting both the frontend, backend and connection to the database working. Once the tenant service was functioning, we moved on to developing the manager portal as this was similar to the tenant portal but with a few adjustments. This incremental development approach allowed us to efficiently debug and understand exactly how our application works while preventing us from becoming overwhelmed by the project's full scope.

Conclusion

The Flat Bills Tracker demonstrates the effective use of virtualisation techniques to create a portable, robust and easily deployable web application. By leveraging Docker containers and adopting a microservices architecture, we developed a scalable and maintainable solution that effectively addresses common challenges in shared accommodations. This project highlights our full-stack capabilities as we implemented a range of different web technologies to create an application that focuses on containerization. The microservices architecture used in this project (tenant, database and manager services) provides a solid foundation for future enhancements and scalability. As shared living arrangements become increasingly popular, the continued development of the Flat Bills Tracker could have the potential to be an effective tool used to improve the communication between tenants and property managers.