MYES Bus Ticketing System

Revision History

Date	Author(s)	Description	Version
05/05/2021	Mustafa Ilıkkan Serdar Mumcu	Initial Version	v1.0

Architecture Notebook

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

2. Architectural goals and philosophy

- Modularity is an important concern for the MYES Bus Ticketing System. The software should be built as having low coupling between components. System should be modifiable, extensible and open for future changes. In summary, architecture should support extensibility and low coupling.
- We aim for a rapid and incremental development style. We will use Django web framework to achieve this goal. When we evaluated web application framework options available in the market, Django seemed the most viable option due to its support for general usage web application development as well as its big ecosystem and detailed documentation.
- After analyzing our problem domain, we came up with using a relational database management system because it was more suitable for a relational data model like ours.
- Reliability, security, performance and usability qualities will be mostly supported by the cloud platform so that we can concentrate primarily on the development of functional requirements and application logic more efficiently.

- We will not maintain our physical servers. Instead, we will use cloud resources from Amazon Web Services (AWS). We will deploy the application on an Amazon Elastic Compute Cloud (EC2) virtual machine.
- We will use Docker to containerize the application for easy and fast deployment. We will use Docker Swarm to achieve scalability and orchestration.
- We will use Docker Compose to provide a standard and sterilized development environment across development team members, and to overcome problems caused by the differences between development and deployment machines.
- We will use git tool for version control and GitHub as the git provider for collaboration during development activity.

3. Assumptions and dependencies

- We assume that MYES Bus Ticketing System will be running on the cloud as a SaaS application.
- The team is composed of experienced developers and testers. We assume that team members are familiar with all tools and technologies that will be used throughout the project.
- Technologies used for development are open source and flexible technologies.
- Cloud infrastructure on which we will deploy our application is assumed to be highly available.
- The system can be used only on web browsers, there will not be any extra installations on the user's computer.
- Since the standard HTML, CSS and JavaScript will be used on the user interface, any user with a modern browser shall use the system.
- The user should have JavaScript enabled because some user interface components are rendered by JavaScript on the browser.
- Further architectural decisions will be made after being discussed by all team members.

4. Architecturally significant requirements

Architecturally significant requirements are the non-functional requirements that are specified in the System-Wide Requirements Specification. These consist of the below system qualities.

- Usability requirements (section 3.1 in system-wide reqs.spec.)
- Reliability (section 3.2 in system-wide reqs.spec.)
- Performance (section 3.3 in system-wide reqs.spec.)
- Supportability (section 3.4 in system-wide reqs.spec.)
- Security (section 3.5 in system-wide reqs.spec.)

5. Decisions, constraints, and justifications

- We will use layered (3-tier) architecture.
- Python and Django web framework will be used for development. Django provides rapid application development, code-first approach development (database generation from code), automatic admin interface generation, object-relational mapping and database migration support, and easy form template generation. It also comes with automatic authorization and authentication mechanisms and security middleware support. Design philosophies and features of the Django web framework on which our application's architecture is based are listed below:
    - Loose coupling
    - Less code
    - Quick development

  - Do not repeat yourself (DRY)
  - Explicit is better than implicit
  - Consistency at all levels
  - Include all relevant domain logic
  - SQL efficiency
  - Powerful syntax
  - Loose coupling
  - Separate logic from presentation (template system)
  - Be decoupled from HTML
  - Safety and security
  - Extensibility

- HTML, CSS (Bootstrap library), and javascript (JQuery library) will be used for the user interface. Our target browsers will be Safari, Chrome, Firefox and Edge.
- Django's template language will be used to create necessary files for the rendering of the user interface.
- A relational database management system will be used. For the initial release, we will use SQLite database, but in the future it can be changed to other relational database management systems such as MySQL, PostgreSQL, or Oracle. (it is possible to switch the database by changing some configuration inside Django framework)
- For the table views on the user interface, we will use the Bootstrap Datatable component.
- Software will be deployed on Amazon Web Services cloud in a Docker container.
- The docker compose tool will be used for development environment setup.
- The docker machine tool will be used for cloud deployment. It helps provision AWS EC2 instances from the command line.
- The docker swarm tool will be used for container orchestration. Orchestration will provide scalability, load balancing, easy roll back, and desired state reconciliation.
- The docker hub platform will be used as the docker image repository.

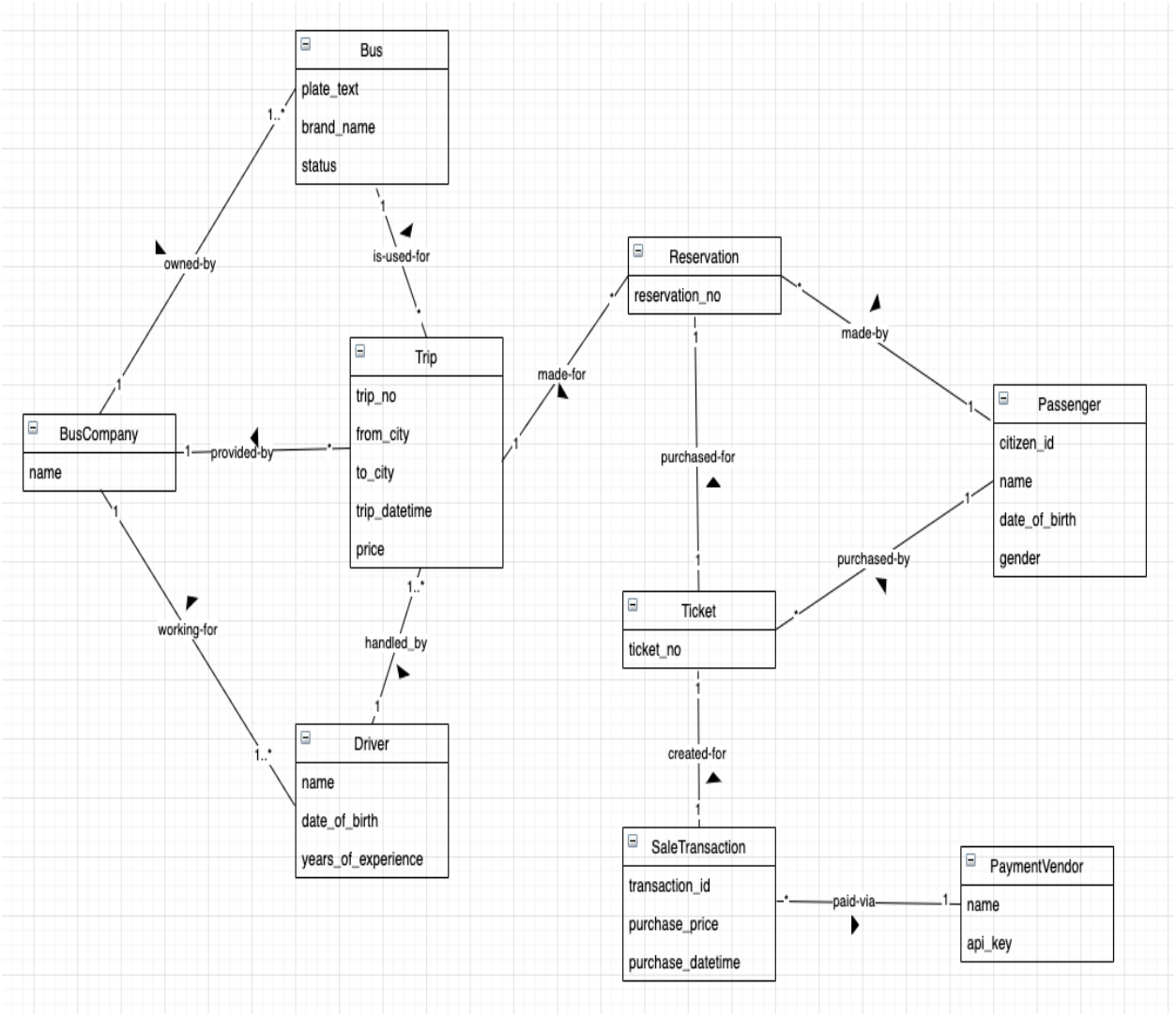
6. Architectural Mechanisms

- Persistence mechanisms:
  - SQLite relational database management system will be used as the persistence mechanism.
  - SQLite is easy to work with as it works as an embedded database on the machine that runs the application rather than working as a client-server database.

- Model-View-Template pattern:
  - Django framework uses a variation of the Model-View-Controller (MVC) compound design pattern called the Model-View-Template (MVT) pattern.
  - The Model in MVC is the same as the Model in MVT.
  - The View in MVC is the same as the Template in MVT.
  - The Controller in MVC is the same as the View in MVT.

7. Key abstractions

Domain Model of the system is presented below. It includes the key abstractions from the problem domain.



8. Layers or architectural framework

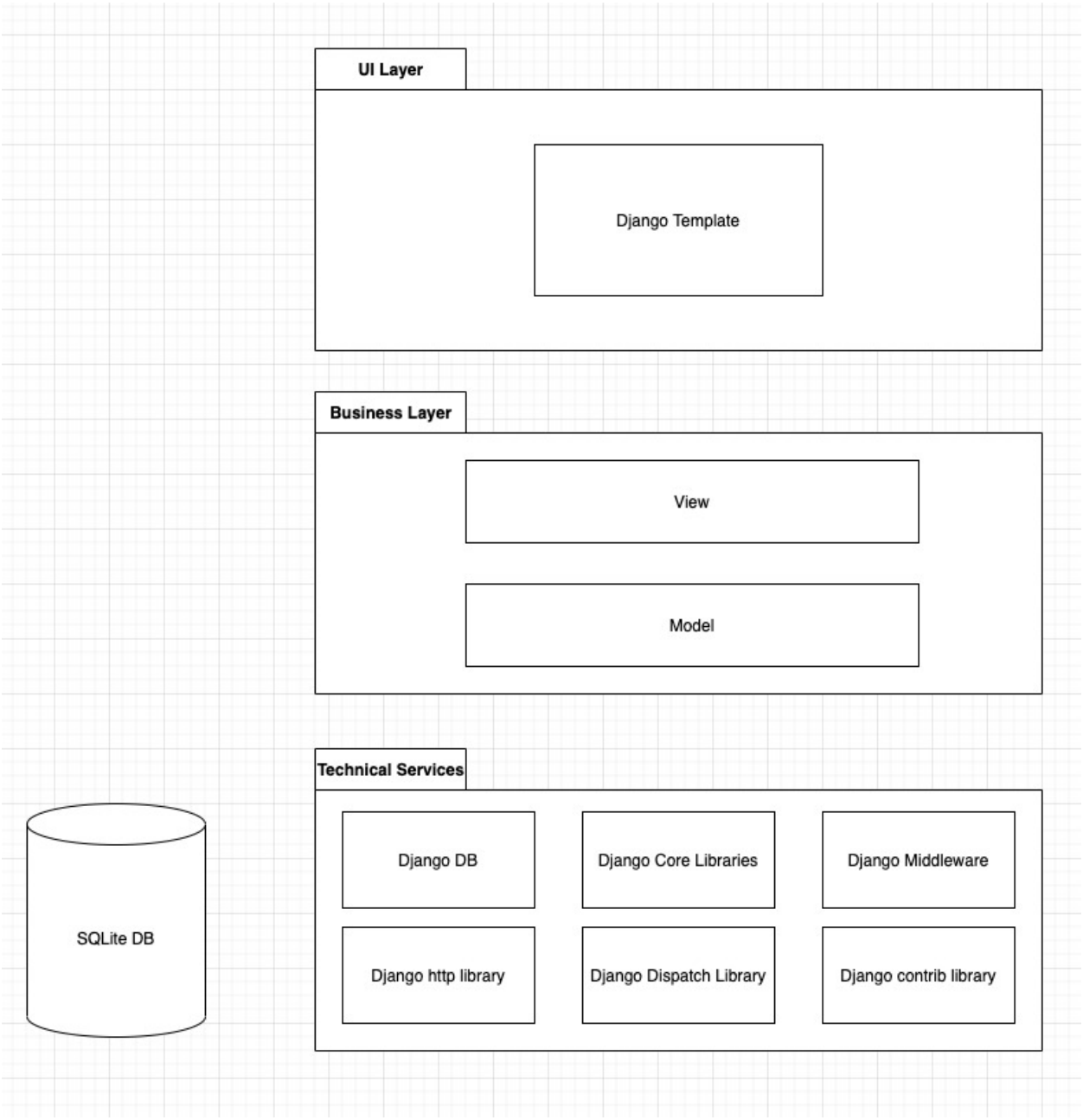
The system will be constructed using a 3-tier layered architecture. Django uses MVT pattern instead of MVC pattern as explained in section 6 above. The layers of the system is summarized below:

- User interface: These are the views presented to the user. These views are HTML files mixed that are generated from Django templates.
- Business Logic layer: This layer includes the Django views and Django models. These are equivalent to controllers.
- Technical services (Core Django framework, modules, session management, signal handlers, http handlers, form mechanism, Django middleware, templates and shortcuts, generic views, web server creation, security features, other dependencies)

9. Architectural views

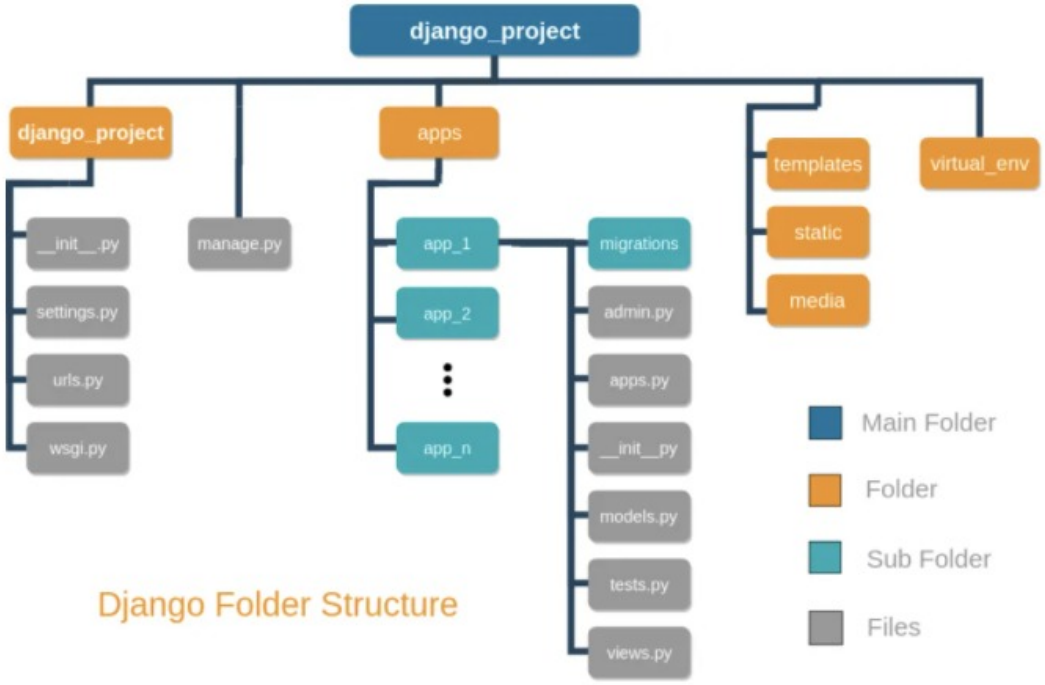
a. Logical View:

Logical view is presented below. It shows how MYES Bus Ticketing Application's different components are logically placed inside the 3-tier architecture.



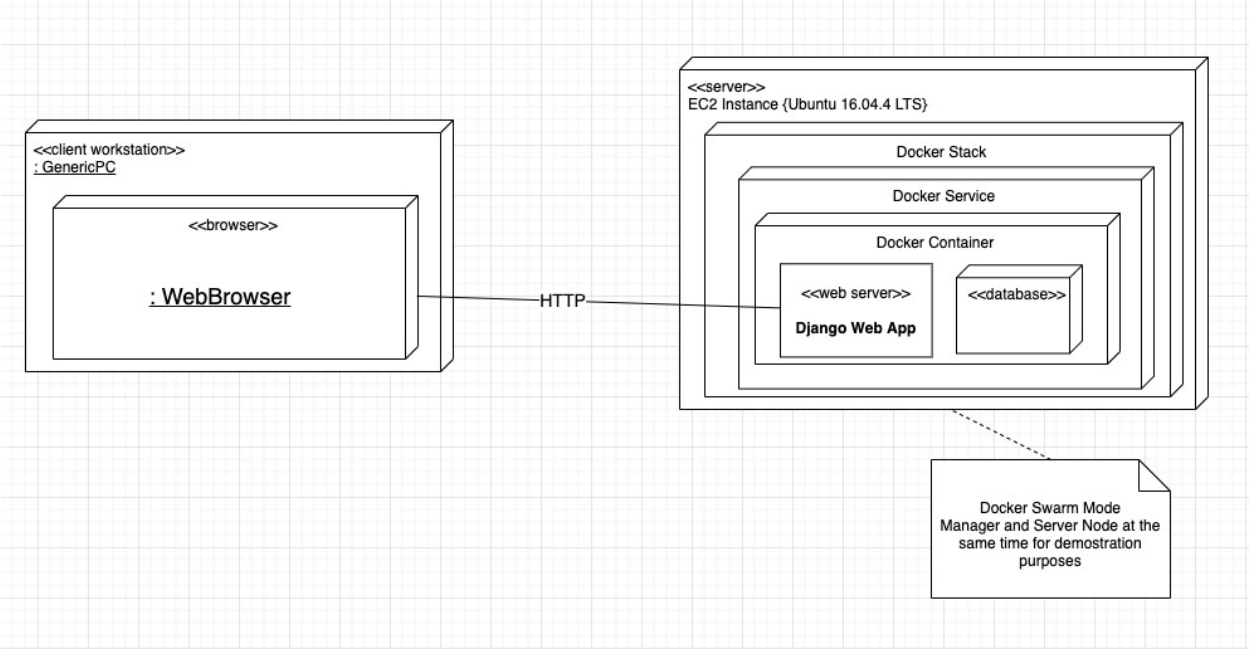
b. Implementation View:

Django web framework provides a standard folder structure for every web application. In our project source code, “django_project” is named as “myes” and “app” is named as “busticket”.



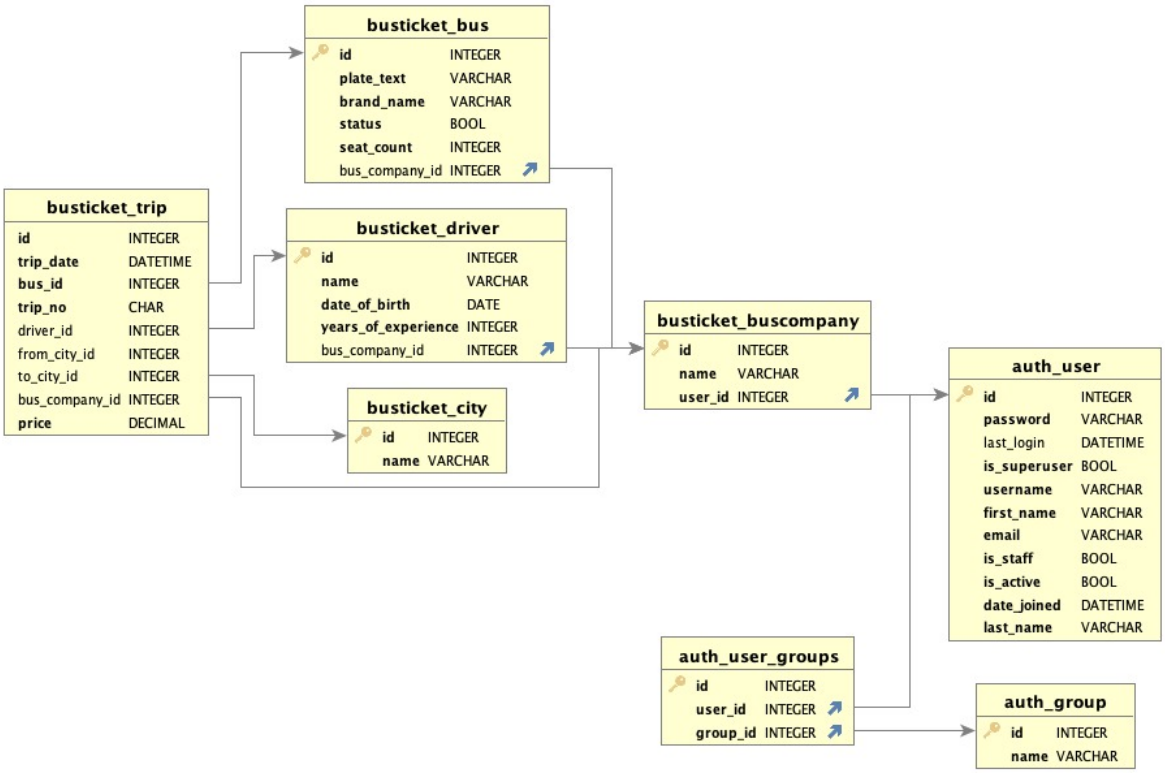
c. Deployment View:

Physical deployment of the application is shown in the below diagram. A use accesses the system using a web browser. Http requests made by the browser reaches the application that is running on a container which is running in an AWS ECS instance. Database is also placed in the same container.



d. Data View:

Entity-relationship diagram shows the data view of the application below.



e. Use Case View: