“You May Dig”

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.

|  |  |  |
| --- | --- | --- |
| Erin T | 07/04/17 | Initial revision |
| Erin T | 12/06/17 | Updated |
| Somer H | 13/06/17 | Updated to reflect current architecture |

# 2. Architectural goals and philosophy

## Accessibility/Usability

One of the largest concerns of the application is accessibility. Due to being targeted toward the average user, the amount of people who use browse the web especially our target audience (gamers) the system will be deployed as a web application, only accessible through a browser (not a desktop program). To this end, the JavaServer Faces (JSF) framework alongside PrimeFaces was selected to assist in producing an easy to develop and manage forward facing web interface. And Tomcat7 chosen to run the JSF application due to the simple nature of deployment.

Additionally, due to the likelihood of users accessing the application on their phone, the application is employing a responsive design with the aid of Bootstrap - a CSS framework that leaves accessibility rendering decisions to the client’s browser rather than serving an entirely separate page for a mobile user than a desktop user.

## 

## 

## 

## Compatibility during development and deployment

Due to relying on multiple architectural components such as servers (tomcat, postgresql & nginx), the system is designed to be developed and deployed using Docker, which runs each server within a self contained environment within a VM, leaving the developer’s systems clean of clunky software that they may not use besides during work on this application.

Using Docker also allows quick deployment by running the exact same environment as is used during development - in order to deploy the system in a simple environment, after initial installation of git, docker and maven (and possibly java in order to run maven) only three commands are required in order to deploy the application.

Due to Docker’s nature of continuing to develop however, the application could eventually be broken by a change to Docker. The severity of this risk is relatively low due to the application being able to be run without any reliance on Docker by simply installing the components on a server (or servers) and altering some configuration files.

## Performance

An nginx server will be employed between the end user and the tomcat application server to handle larger static content requests such as images and other files while forwarding any other requests to the application server. This helps to cut down on having to use resources on the tomcat server to stream the contents of a static file that could be needed elsewhere within the application.

In the efforts of performance, the database structure of the application is also being designed to avoid expensive queries during regular usage. Updating calculated values and storing them only when needed rather than executing the calculations on every request where they may be displayed. (Such as when calculating the average rating assigned to a game, or while generating recommendations)

## Persistence

A major part of our application is data persistence and later analysis of that data - both to provide the end user with suggestions of games they might enjoy, but also to better inform customers of who exactly enjoys what kinds of games. In order to simplify the process of data persistence, EclipseLink, an implementation of JPA which is the EE standard specification of Object Relational Mapping (ORM) to persist and keep track of the various data classes within our application (Game, Genre, User, etc.)

Due to JPA using JDBC, the specific database software used is generally unimportant and the system can likely employ any of the major persistence providers that a JDBC driver exists for.

In order to populate our database of games - we will be designing and implementing different gatherers to collect data from public sources such as using Steam’s store API, the Playstation store API, web scrapers, etc. These gatherers are run at scheduled intervals by using the Quartz java scheduling engine.

# 3. Assumptions and dependencies

* A PostgreSQL database will be used for persistence
* PrimeFaces will be used where possible due to providing a simple method of creating impressive AJAX based web interfaces
* A virtual private server (VPS/cloud server) will be used to initially deploy the production application due to ease and low cost

# 4. Architecturally significant requirements

* A persistence layer is required to ensure a solid record of games, users, and the ratings users assign to games is stored to be used later with the recommendation engine to produce our user profiles and recommendations - our primary goal
* A basic authorization system is required to ensure end users aren’t provided access to the admin only section of the site where more sensitive information and controls could be found in order to ensure the end user does not find the raw profile that may be gathered on them to avoid frightening them

# 5. Decisions, constraints, and justifications

* The application will be deployed on a Tomcat server due to its lightweight nature, large amount of support if another method of overall deployment is ever sought and works well with Eclipse
* A PostgreSQL database will be used for persistence for no major reason besides being an increasing industry standard and one of the most pure SQL implementations
* JPA is used for data persistence and data integrity due to both being a crucial part of the application - our end result relies on the integrity of our persisted data (see section 4.). (And also to aid easy development)
* Nginx is used to reduce load on the tomcat server by serving static content such as images, leaving the tomcat server resources to run our application
* JSF and PrimeFaces is used to easily produce our interactive web interface - simplifying development
* Bootstrap is used to provide a responsive interface which will render well on a variety of screen sizes (such as desktop and mobile)

# 

# 

# 6. Architectural mechanisms

## Authentication and Authorization [Implemented]

The system will require users to have a valid account and be authenticated in order to rate games and receive customised recommendations. This is accomplished by standard user authentication (persisting in database, etc.).

Authorization is also a component due to the site having an admin section to perform additional tasks such as triggering manual gathers of more games from the external sources. This is accomplished by a simple “staff” flag on the user object which is also persisted. This keeps average users away from administrative sections of the site.

## Gathering [Implemented]

The system will use a variety of gathering classes to collect game data from other services to populate our own database of games and their descriptions. The collected data is persisted and the information is used to not only provide the selection of games for end users to rate, but also persists information such as genre so the application has more information on which to build a more complete analysis of a user’s likes.

## Rating [Implemented]

A crucial part of the system is for users to be able to rate the different games found on our application to provide their feedback and build a better idea of what games they may enjoy for further recommendations. This is achieved by persisting a Rating object, linked to the game and the user rating it.

## Recommendation Engine

The final mechanism of the application is the recommendation engine which takes all the information collected from the prior mechanisms and producing a profile of users in order to provide further recommendations to those users, but also to provide ourselves and our customers with a greater insight into exactly what kind of user (person) enjoys what kind of game based on our generated profiles.

# 

# 

# 7. Key abstractions

Many of the key abstractions can be found in more detail within the requirement model [here on version control](https://github.com/ITC303-Violet/ITC303/blob/master/docs/requirement-model.pdf)

# 8. Layers or architectural framework

As this is a JSF application, it will follow the Model-View-Controller framework. The View layer is all the xhtml files which incorporate what the users see. The controller is hidden by the JSF architecture – the FacesContext is involved in transporting data between the view and the Java classes behind it on the server. The Model layer is the Managed Beans which are involved in manipulating the data that is shown to the user or received from the user, along with all the backing Java classes that do more intensive work on it.

# 9. Architectural views

A component diagram displaying both a logical and operational view can be found [here on version control](https://github.com/ITC303-Violet/ITC303/blob/master/docs/diagrams/ITC303%20-%20Violet%20-%20Component%20Diagram.png)

A use case diagram can be found within the requirement model [here on version control](https://github.com/ITC303-Violet/ITC303/blob/master/docs/requirement-model.pdf)