WVUP: The College Quest

Software Requirements Document

Jennifer Mackey

CS460

Contents

[Introduction 3](#_Toc58279861)

[Overview 3](#_Toc58279862)

[User Requirements 4](#_Toc58279863)

[Functional Requirements 4](#_Toc58279864)

[End User 4](#_Toc58279865)

[Developer 4](#_Toc58279866)

[Nonfunctional Requirements 4](#_Toc58279867)

[End User 4](#_Toc58279868)

[Developer 4](#_Toc58279869)

[System Requirements 4](#_Toc58279870)

[Operating Environment 4](#_Toc58279871)

[Design and Implementation Constraints 5](#_Toc58279872)

[Hosting Requirements 5](#_Toc58279873)

[System Models 6](#_Toc58279874)

[Use Cases 6](#_Toc58279875)

[Gameplay 6](#_Toc58279876)

[The KIS 7](#_Toc58279877)

[Process Views 8](#_Toc58279878)

[Sequence Diagrams 8](#_Toc58279879)

[Activity Diagrams 10](#_Toc58279880)

[DevOps Implementation Strategy 11](#_Toc58279881)

[Objective: 11](#_Toc58279882)

[Continuous Integration 11](#_Toc58279883)

[Continuous Delivery 11](#_Toc58279884)

[Continuous Testing 11](#_Toc58279885)

[Continuous Deployment 11](#_Toc58279886)

[Appendix 11](#_Toc58279887)

[Glossary 11](#_Toc58279888)

# Introduction

West Virginia University at Parkersburg requires an interactive web-based video game to provide a fun and immersive experience to prospective students by providing an informative tour to the school via the style of a retro-styled video game. This game shall be called the **WVUP: The College Quest.**

# Overview

The main features of the WVUP: The College Quest include:

1. Provide an overall tour of the school as described in the introduction.
2. Individualized selection of the look of their character to represent who they are in the game.
3. Allow the player to select a degree to learn more about that division and the degree, allowing for various unique scenarios to allow them to tour the game.
4. Inform the prospective student of important information when they apply to the school
   1. Allow them to print out the information for future reference for when they apply to the school.

WVUP: The College Quest is NOT intended to do the following:

1. Provide admission or any forms that would otherwise be provided by OLSIS or FAFSA.
2. Fulfill any class credit that would teach similar knowledge.

# User Requirements

## Functional Requirements

### End User

1. A user will require an account to be able to access the game, which they shall be able to create.
   1. If account is already created, the system shall a user to log in to the account to access the saved data.
2. The system shall allow a user to create a **character** that represents them in what is known as the *Character Creation* scene, so they can navigate throughout the game.
   1. The system shall allow a player to, on this character creation scene, click through arrows on the UI to click through the various parts to let them choose the look they want.
      1. In regards to customization, the system shall allow Hair Style, Hair Color, Skin Color, Body Type, Expression Type, Eye Type, Brow Type, and Hat Type to be customized.
3. The system shall allow a player to earn **EXP (experience points),** which allow them to level up so that they can advance through the game.
4. The system shall allow a player to receive and completed **quests**.
   1. These quests are available through the **KIS**.
   2. Quests shall award EXP.

### Developer

## Nonfunctional Requirements

### End User

### Developer

1. A script shall be written as to allow the player to experience progress through the game.
2. All information for the game will be acquired from professional sources, such as division chair members, academic advisors, and any other necessary faculty members, etc.

# System Requirements

## Operating Environment

1. The system shall be able to run on all modern browsers.
2. Data shall be saved to a MySQL database, which will be transferred via the “client” (The game build), to the “server” (which is currently a localhost database), via a php file that will act as a middle man to provide the ability to transfer data.
3. The system will run on the Unity Game Engine.
4. The resolution of the game shall be set in a retro style, which will be 640p.
5. The application shall run on all modern PCs, regardless of OS, since it will be web based.
6. The application shall be coded effectively so that it is not too demanding on resources and data, so that the application can be used across various devices. Current goal is to be less than a 1GB of ram.

## Design and Implementation Constraints

1. All Web Interfaces shall apply to a HTML5 industry standard.
2. All art created for the game shall be in the style of an 80/90’s retro video game style.

## Hosting Requirements

1. The game shall be hosted on via an Azure Web Application server.

# System Models

## Use Cases

### Gameplay

|  |  |
| --- | --- |
| **Use Case:** | Gameplay – Character Creation |
| **Actor:** | Player |
| **Pre-Condition:** | Player has clicked the “start game” button |
| **Post Condition:** | All data as described in the description, including character customization attributes, will be saved to the database. |
| **Description** | Player is met with a register screen that is essentially the character creation screen. Here, the player can customize their character and input their information. Once the player is satisfied with customization, they can click the “play” button. |

|  |  |
| --- | --- |
| **Use Case:** | Gameplay – Completing a Quest |
| **Actor:** | Player |
| **Pre-Condition:** | Player has completed a quest as required by the tasks of the quest. |
| **Post Condition:** | All data as described in the description will be saved to the database. |
| **Description** | Player will receive EXP (and an increase in a level if they have acquired enough EXP) for successfully completing a quest, and will receive a notification upon doing so. |

|  |  |
| --- | --- |
| **Use Case:** | Gameplay – Completing the Game |
| **Actor:** | Player |
| **Pre-Condition:** | Player has successfully completed all tasks. |
| **Post Condition:** |  |
| **Description** | Once a player has completed all tasks, they will be considered done with their college quest, at least virtually. They will receive a screen that shows them they have won the game. |

### The KIS

The KIS, or the ***Knowledge Inventory System,*** is the hub of information for the player as they progress through the game, where they can access their information, and available quests for them to complete.

The KIS button is always available to the player in the upper right corner of the screen.

|  |  |
| --- | --- |
| **Use Case:** | Opening the KIS System |
| **Actor:** | Player |
| **Pre-Condition:** | User already has an account and is in the game with a player they can play as. |
| **Post Condition:** |  |
| **Description** | The player will be able to select the KIS System button in the upper right corner, and upon clicking it, the game will open up a scene called The KIS System, and the user will be able to access the applicable data that is shown within the KIS system. |

|  |  |
| --- | --- |
| **Use Case:** | Using the KIS System – Viewing a Quest |
| **Actor:** | Player |
| **Pre-Condition:** | User already has an account and is in the game with a player and has clicked the KIS button. |
| **Post Condition:** |  |
| **Description** | The player will be able to view the quests available on the right, and upon clicking a task, it will show the information on the bottom left, and they need to do to accomplish it. |

## Process Views

### Sequence Diagrams

#### Sequence Diagram #1 – Overall System Sequence



#### Sequence Diagram #2 – Login Sequence



### Activity Diagrams

#### Activity Diagram #1 – Character Creation



# System Design

## Overview – What is ECS?

As described in the UML, the architectural pattern that Unity Game Engine, and thus, The College Quest will use, is an Entity-Component-System. This paradigm models a program in a data-oriented way.

ECS (Entity Component System) overall, is a software architecture pattern that enforces the separation of data and behavior.

**Entities** They represent the individual "things" in your game or application. An entity has neither behavior nor data; instead, it identifies which pieces of data belong together. Systems provide the behavior, and components store the data.

**Components** represent the data of your game or application. Entities are identifiers that index your collections of components, while [systems] provide the behavior.

A **System**, the S in ECS, provides the logic that transforms the component data from its current state to its next state — for example, a system might update the positions of all moving entities by their velocity multiplied by the time interval since the previous update.

Source: <https://docs.unity3d.com/Packages/com.unity.entities@0.16/manual/ecs_core.html>

## Coupling

However, as one might expect, this does introduce high amounts of coupling into the design, which is generally bad design in most applications. However, it is fairly unavoidable due to the architecture, but is not necessarily a horrible approach for this type of system.

One could almost consider a **Component** to an MVC model. As mentioned above, it doesn’t necessarily provide any functionality, but it provides information/raw data to the system involved, so it would be similar to the Model-Controller coupling that is involved with the MVC architecture, where the model would show the business logic to the controller that the controller (in this case, the **System**) will mess with.

Source: <https://softwareengineering.stackexchange.com/questions/8565/is-loose-coupling-w-o-use-cases-an-anti-pattern>

## Cohesion

# DevOps Implementation Strategy

## Objective:

The objective of this section is to describe the strategy for the DevOps practices to be implement across the development of the College Quest Video Game. DevOps is a type of a practice that helps automate and speed up the software delivery process.

Overall, the deployment strategy used will be an automated process that upon pushing code to GitHub, will pull necessary build files, and upload it to the azure web application.

## Continuous Integration

On top of using a GitHub repository to allow for version control, a major goal of using continuous integration is to deploy code as quickly as possible. However, it is to also get rapid feedback if we come across any defects then they can be identified and corrected as soon as possible.

To do this, we would use GitHub’s own GitHub actions It uses a combination of YAML/YML syntax to automate building, testing, and delivering and deploying software, that is used within the continuous delivery process.

## Continuous Delivery

The most basic continuous delivery pipeline has the minimum of three stages that are used within a continuous delivery pipeline: Build, Test, and Deploy.

With GitHub actions, this is controlled within what is known as a workflow file, which contains snippets for this code that will indicate the steps taken when performing the steps.

## Continuous Testing

[To Do]

## Continuous Deployment

Continuous Deployment is the next stage after Continuous Delivery, where once the tests pass, it automatically deploys to the Azure website

# Appendix

## Glossary

**Player/Character**: Referred to as the player or character, this represents what the user will control throughout the game.

**EXP** – An abbreviation for “Experience Points”. EXP are essentially the form of numerical value that regards the experience the player has received in the game, by the means of completing a quest.

**KIS** – An abbreviation for the Knowledge Inventory System. This system is where the current save data is displayed, including the current quest list, their name, major, and current level in the game.

**Quest** – A quest is a task that a player must do to progress through the game. In this game, these are generally informative in nature, and require the player to navigate throughout the campus to get this information so that they can complete their quest.

**Top-Down**: The camera view of the game as the character navigates, since the game is two-dimensional.

**Color Depth** – refers to the number of bits per pixel on a computer monitor to represent a specific color. (Computer Hope, 2017)

**16 bit** – Refers to the color depth of the characters, which was used in 90’s video games.

**Collision** – When a game object collides with another one, it invokes an action as dynamically indicated by code. Examples include a player colliding through the wall so they can’t travel through it.