

# The Orchid Arcade

## Contents

1. Project description .....	2
2. Overview .....	2
2.1 Traceability matrix .....	2
3. System Architecture .....	3
4. Software components .....	4
4.1 User .....	4
4.2 Customer .....	4
4.3 Developer .....	5
4.4 Game .....	5
4.5 Transaction .....	5
4.6 Review .....	6
4.7 Security Architecture .....	6
5. Software UI components .....	7
6. Data components .....	12
7. Test cases .....	13
7.1 Manage user account .....	13
7.2 Search and Buy Games .....	13
7.3 Purchase a Game .....	14
7.4 Manage Game Library .....	14
7.5 Submit a Game Review .....	15
8. References: .....	16

## Table of Figures

Figure 1 UML diagram .....	3
Figure 2 Library view .....	8
Figure 3 Shop view .....	9
Figure 4 Game detail view .....	9
Figure 5 Write review view .....	10
Figure 6 Developer view .....	11
Figure 7 Create game view .....	11

## 1. Project description

The Orchid Arcade (TOA) is a web-based game distribution platform specifically designed for casual gamers seeking cozy and relaxing games. It serves as a marketplace where users can purchase, download, and play games while developers can publish and manage their games.

## 2. Overview

### Core Features:

- **User Management:** Users can create accounts, manage their profiles and view purchase history.
- **Game Store:** Offers browsing, searching, and purchasing of games. Users can filter by genre, price, popularity, or developer.
- **Game Library:** Users can manage their purchased games, including downloading, installing, and updating games.
- **Game Publishing for Developers:** Developers can upload and manage their games, track sales, and interact with users through reviews and feedback.

### Technology Stack:

The application will use ASP.NET Core to handle the server-side logic, including user authentication, game management, and payment processing. The application will follow the MVC pattern, with models representing data, views for the user interface, and controllers to handle the logic and user interactions as described by Ardalís (2023) in Architect modern web applications with ASP.NET Core and Azure chapter 4.

### Functional Scope:

- Users: Account creation, game browsing, purchasing, and managing a personal game library.
- Developers: Game publishing, updating, and sales management.
- The application will support secure payments and regular game updates and include user reviews.

### 2.1 Traceability matrix

SRS Requirement	SDD Module
3.2.1 Manage User Account	4.1 User, 4.2 Customer, 4.3 Developer

3.2.2 Browsing and Purchasing Games	4.2 Customer, 4.4 Game, 4.5 Transaction
3.2.3 Game Library Management	4.2 Customer, 4.4 Game, 4.5 Transaction
3.2.4 Game Publishing and Management	4.3 Developer, 4.4 Game
3.2.5 Sales and Revenue Management	4.3 Developer, 4.4 Game, 4.5 Transaction
3.2.6 Review and rate games	4.2 Customer, 4.4 Game, 4.6 Review, 4.5 Transaction

### 3. System Architecture

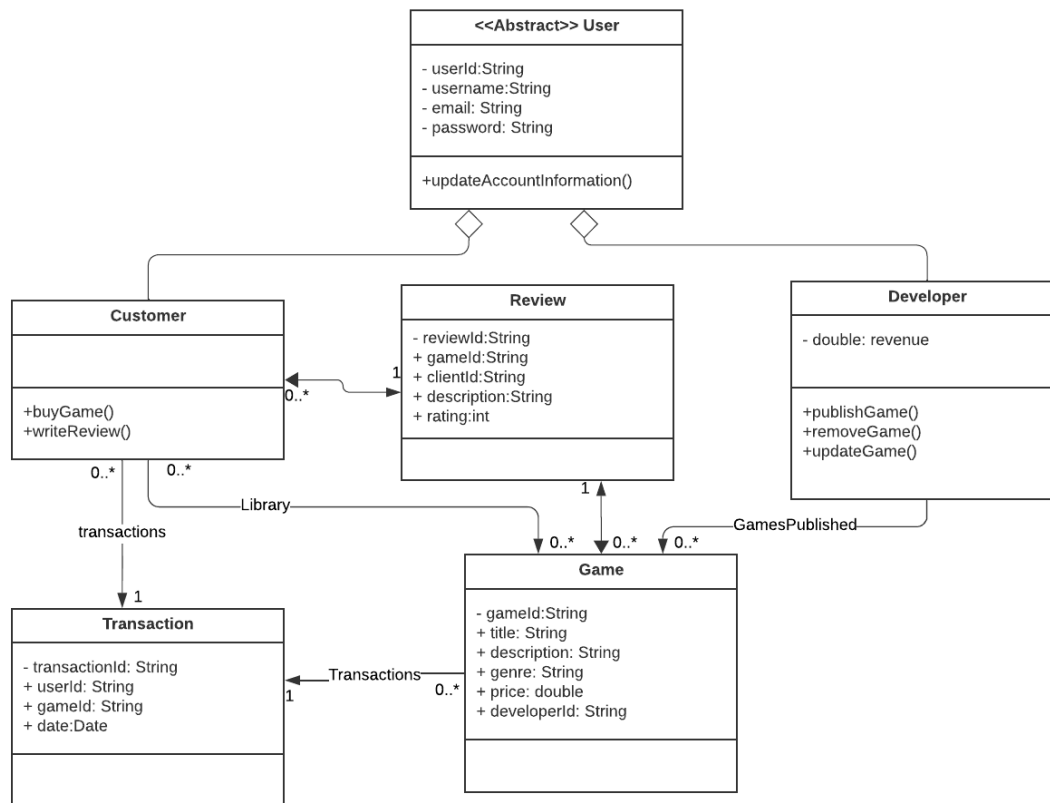
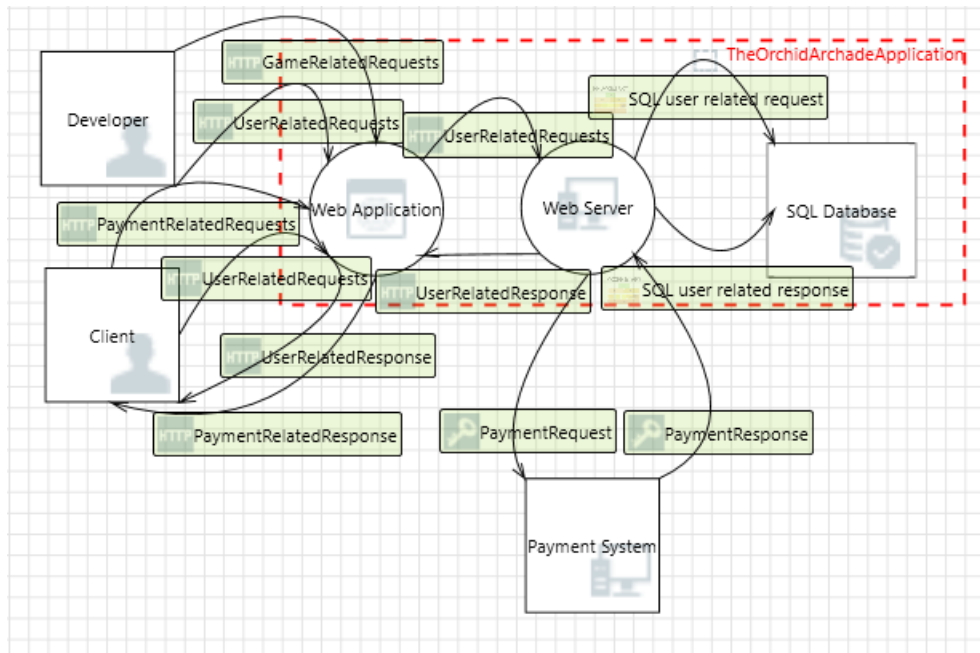


Figure 1 UML diagram

The UML diagram illustrates the relationships between the key entities in the application. There is an abstract User class that contains basic information, and it is inherited by both Customer and Developer classes. A Customer can purchase a game, creating a transaction that adds the game to their library. Afterward, the customer can submit a review for a particular game. Meanwhile, a Developer can publish, update, or remove a game, and they have access to revenue reports for all games they have published. The architecture of the system is the following:



#### 4. Software components

Since we're using the ASP.NET Core MVC architecture, the models will correspond to the entities of the application denoted on the UML diagram of the previous section. A more detailed description of this components is:

##### 4.1 User

- **Description:** The User class serves as the base class for all user types in the system. It contains common properties like UserID, Username, Email, and Password. This abstract class enforces shared behavior for all users, whether they are customers or developers.
- **Key Responsibilities:**
  - Centralizes shared user information.
  - Provides the foundation for user-related actions such as logging in and managing profiles.
  - Enables inheritance for more specialized user types (e.g., Customer and Developer).

##### 4.2 Customer

- **Description:** The Customer class extends from the User class and represents a customer who interacts with the platform to purchase and manage games.

- Key Responsibilities:
  - Manages the customer's game library, which stores the games they've purchased.
  - Facilitates transactions and enables users to purchase games.
  - Allows users to submit reviews and ratings for games.

#### **4.3 Developer**

- Description: The Developer class extends from the User class and represents developers or publishers who upload and manage games on the platform.
- Key Responsibilities:
  - Manages published games, including creating, updating, and removing game listings.
  - Provides access to financial and revenue reports for their published games.

#### **4.4 Game**

- Description: The Game model represents a game listed on the platform. It includes properties like GameID, Title, Description, Genre, and Price. Each game is associated with a developer who publishes it.
- Key Responsibilities:
  - Stores and retrieves game-related information.
  - Connects to customer purchases and reviews.
  - Provides pricing and genre details to display in the store.

#### **4.5 Transaction**

- Description: The Transaction model represents a financial transaction that occurs when a customer purchases a game. It tracks the TransactionID, the UserID of the customer, and the GameID of the purchased game.
- Key Responsibilities:
  - Logs purchases made by customers.
  - Provides transactional history for both customers and developers.
  - Helps in managing refunds and future transaction-based functionalities.

## 4.6 Review

- **Description:** The Review model represents a review or rating submitted by a customer for a specific game. It contains properties like ReviewID, GameID, ClientID, Description, and Rating.
- **Key Responsibilities:**
  - Stores and retrieves customer reviews for games.
  - Associates' customer feedback with specific games, allowing other customers to see ratings and reviews.
  - Connects to the customer who submitted the review and the game being reviewed.

Every single one of these models will have their respective views as mentioned in the UI section and controllers to handle the application logic.

## 4.7 Security Architecture

As specified on the non-functional requirements of the application requirements document, the security architecture of **The Orchid Arcade** includes several mechanisms to protect user data, ensure application integrity, and prevent unauthorized access.

### 1. Authentication and Authorization

- **Design Approach:** Implement ASP.NET Core Identity with support for two-factor authentication (2FA) for enhanced access security. Identity roles and claims-based authorization will control user access to various application areas.
- **Components:** Integrate an AuthController to manage user authentication workflows, including login, registration, and 2FA configuration.

### 2. Input Sanitization

- **Design Approach:** Use built-in ASP.NET Core model binding and input validation for all user inputs to prevent SQL injection and XSS attacks. The application will employ a Content Security Policy (CSP) to further protect against XSS.
- **Components:** Each controller, particularly those managing user-generated content (e.g., ReviewController, GameController), will apply input validation logic to sanitize data before it reaches the database.

### 3. Logging and Auditing

- **Design Approach:** Use ASP.NET Core logging middleware to track and store logs of critical events, including login attempts, account changes, and transactions. Logs will be stored securely and periodically reviewed for anomalous behavior.
- **Components:** An AuditLogService will handle log storage and ensure that sensitive information is omitted, maintaining both security and traceability.

### 4. Rate Limiting

- **Design Approach:** Implement rate limiting using ASP.NET Core's IThrottlePolicy or a similar middleware to prevent abuse, including excessive login attempts or repeated actions on resources.
- **Components:** Configurations in the AuthController and other endpoints will set rate limits to prevent DoS attacks or brute-force attempts on login endpoints.

### 5. Session Management

- **Design Approach:** Configure ASP.NET Core to manage secure sessions, using cookies with SameSite, Secure, and HttpOnly flags to prevent session hijacking. Sessions will expire after a defined period of inactivity, requiring re-authentication.
- **Components:** The SessionController will manage session creation, renewal, and invalidation for secure handling of active user sessions.

### 6. Data Encryption

- **Encryption in Transit:** Enforce HTTPS for all network communications using ASP.NET Core's HTTPS redirection.
- **Encryption at Rest:** Sensitive data, such as PII and payment details, will be encrypted in the database. ASP.NET Core's Data Protection API or an external service will manage encryption keys.
- **Components:** The SecurityConfig module will configure encryption settings for both in-transit and at-rest data protection.

## 5. Software UI components

- Library view:

## My library



Figure 2 Library view

This view shows the library's view from the perspective of a customer. This is the homepage of a user, and it includes a list of the user's games, a search bar, a button to go to the shop, a button to go to their account and buttons for each game to Play/Download a game and to write a review.

- Show view:



## Shop

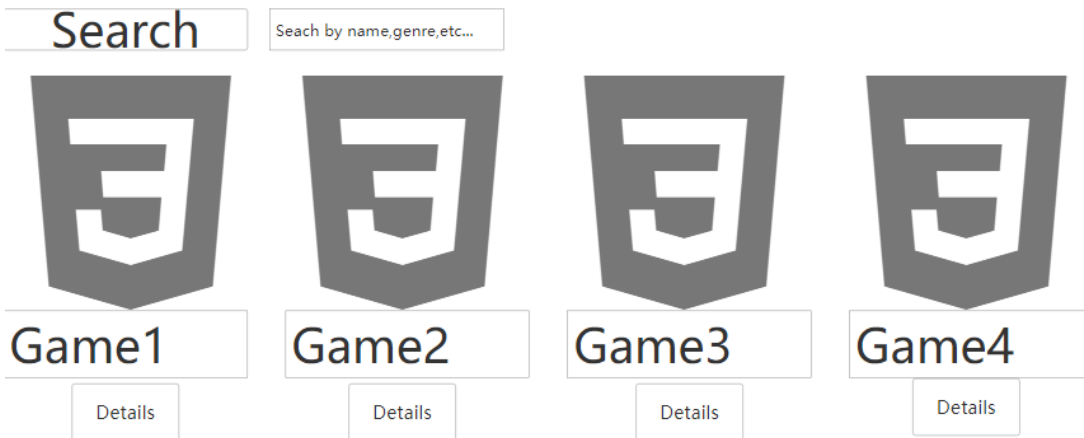


Figure 3 Shop view

This view includes a list of games available to buy, a search bar, a button to return to the library and a button for each game to see the details of the games and buy them.

- Game detail view:

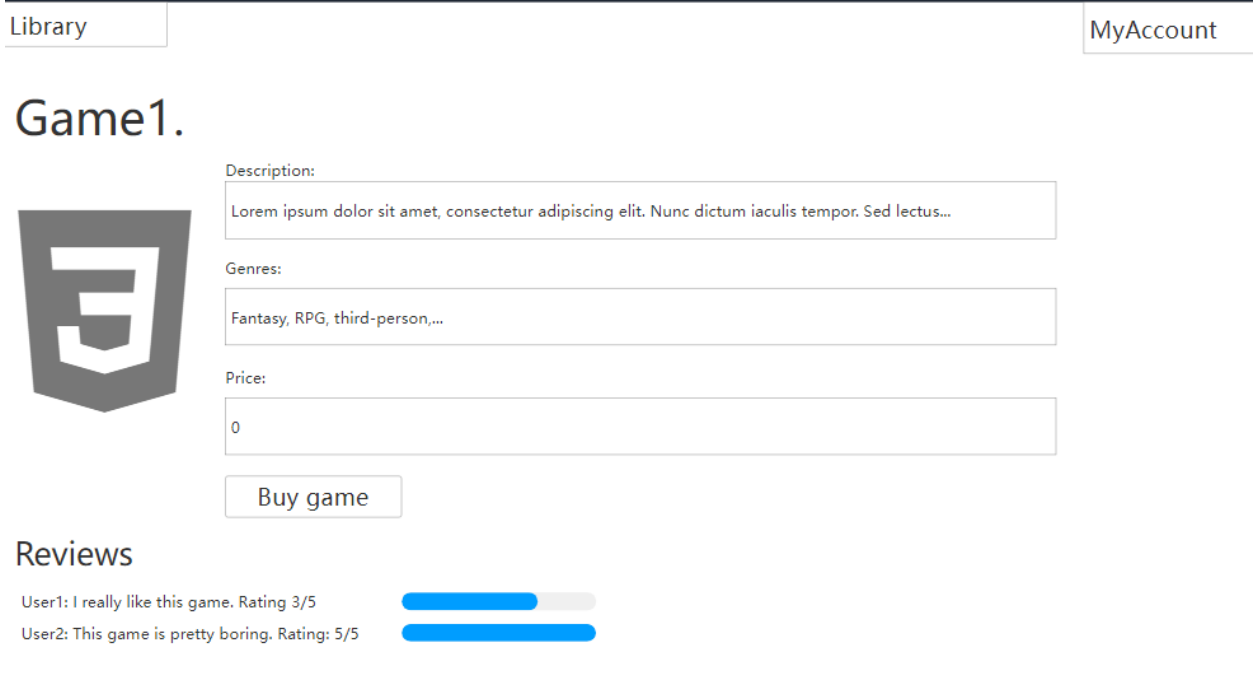


Figure 4 Game detail view


This view Includes the details of a selected game, their name, image, description and price and the ability to buy the game. It also has a list of reviews made by other users that own the game.

- Write review view:

Library

MyAccount


Game1



Write a review for the game:

This game is pretty good!

Give the game a rating



Publish review

Go back

*Figure 5 Write review view*

This view allows the user to write a review for a particular game they own; they can write the review description and a rating from 1 to 5.

- Developer view:

## Developed games

  
**Game1**

  
**Game2**

  
**Game3**

  
**Game4**

Figure 6 Developer view

This view allows a developer to look at their published games, search for them and view the details of a particular game.

- Create game view:

**Game1. (Generated revenue = \$0)**



Description:

Genres:

Price:

**Reviews**

User1: I really like this game. Rating 3/5

User2: This game is pretty boring. Rating: 5/5

Figure 7 Create game view

This view allows a developer to create a new game with its cover image, title, description and check the user reviews. This is also the same view used to update an existing game.

## 6. Data components

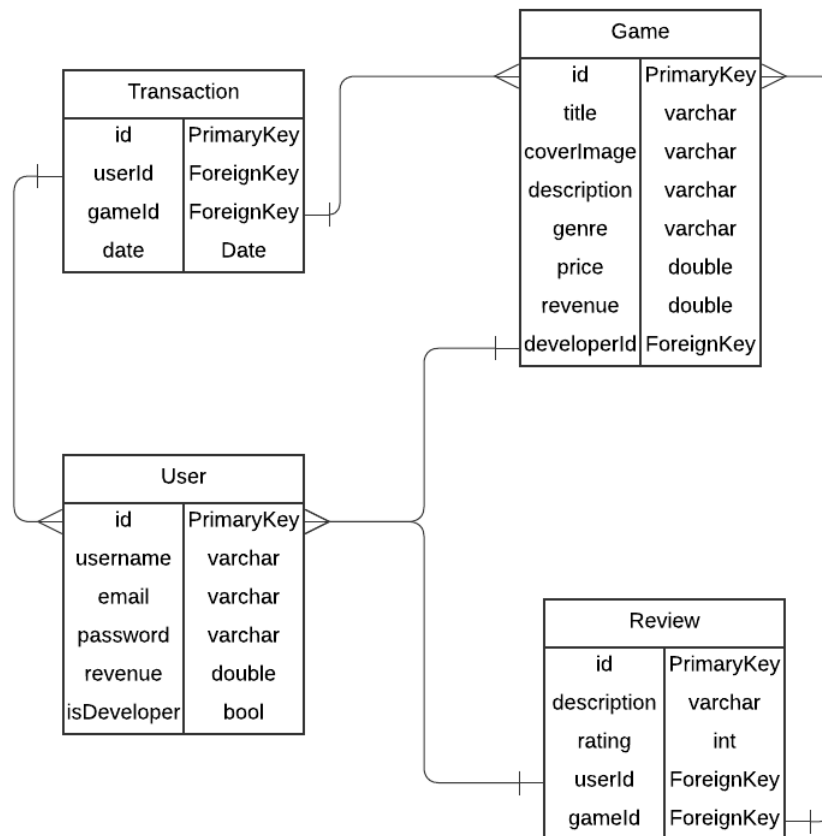


Figure 8 Database diagram

This entity-relationship diagram was built considering the previous UML diagram made. It includes the following tables:

- User table: This table has all the user relevant information like username, email, password and includes a field to see if the user is a developer or a customer. If the user is a developer this table also has the respective revenue field.
- Game table: This table includes all the information of a game like the title, description, cover image, genre, price, generated revenue and a foreign key reference to the user that uploaded the game.

- Review table: The review table includes the information of a review like the description, the rating, a foreign key referencing the user that created the review and a foreign key referencing the game that is being reviewed.
- Transaction table: This table serves as a table that relates every user with the games they have bought by adding a transaction with the Date, a foreign key of the user that bought the game and a foreign key of the game that was bought.

## 7. Test cases

### 7.1 Manage user account

Test Case ID	TestCase1
Test Case Name	Create User Account
Description	Tests the ability of a user to create an account.
Precondition	The user has accessed the registration page.
Test Steps	1. Navigate to the registration page. 2. Input a valid email, username, and password. 3. Click "Create Account."
Expected Outcome	The user account is successfully created, and a confirmation message is displayed.
Postcondition	The user is redirected to their profile page or a confirmation screen.
Pass/Fail Criteria	Pass: The account is created and saved to the database. Fail: An error message appears, or the user cannot proceed.

### 7.2 Search and Buy Games

Test Case ID	TestCase2
Test Case Name	Search for Games
Description	Tests the ability to search for a game.
Precondition	The user is logged in and has accessed the game store.
Test Steps	1. Enter a game title or keyword in the search bar. 2. Press the "Search" button.

<b>Expected Outcome</b>	A list of matching games is displayed.
<b>Postcondition</b>	The user can view game details or purchase the game.
<b>Pass/Fail Criteria</b>	Pass: The correct games matching the keyword appear. Fail: No results appear or irrelevant games are shown.

### 7.3 Purchase a Game

<b>Test Case ID</b>	<b>TestCase3</b>
<b>Test Case Name</b>	Purchase Game
<b>Description</b>	Tests the ability to complete a game purchase.
<b>Precondition</b>	The user is logged in and has selected a game to purchase.
<b>Test Steps</b>	1. Click "Buy Now" on the game page. 2. Input payment details. 3. Confirm the transaction.
<b>Expected Outcome</b>	The purchase is completed, and the game is added to the user's library.
<b>Postcondition</b>	A confirmation message is displayed, and the transaction is logged.
<b>Pass/Fail Criteria</b>	Pass: The game appears in the user's library. Fail: The transaction fails, or an error message appears.

### 7.4 Manage Game Library

<b>Test Case ID</b>	<b>TestCase4</b>
<b>Test Case Name</b>	Download a Purchased Game
<b>Description</b>	Tests the ability to download a purchased game.

<b>Precondition</b>	The user has purchased at least one game.
<b>Test Steps</b>	1. Go to the game library. 2. Click "Download" next to the purchased game.
<b>Expected Outcome</b>	The game download begins.
<b>Postcondition</b>	The game is downloaded and installed on the user's machine.
<b>Pass/Fail Criteria</b>	Pass: The game is installed without error. Fail: The download is interrupted or cannot start.

### 7.5 Submit a Game Review

<b>Test Case ID</b>	<b>TestCase5</b>
<b>Test Case Name</b>	Submit Game Review
<b>Description</b>	Tests the ability to leave a review for a game.
<b>Precondition</b>	The user has purchased and played the game.
<b>Test Steps</b>	1. Go to the game page. 2. Click "Leave Review." 3. Input review text and rating. 4. Submit the review.
<b>Expected Outcome</b>	The review is submitted and displayed on the game page.
<b>Postcondition</b>	The review is stored in the database and visible to other users.
<b>Pass/Fail Criteria</b>	Pass: The review is created correctly. Fail: The review is not saved or does not display.

## 8. References:

- Lane, G. K. C. (2023, January 17). How to write an SRS Document (Software Requirements Specification Document). Perforce Software. <https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document>
- TianyaoHan. (n.d.). GitHub - TianyaoHan/Steam-Recommendation-System: Steam Database Design and Game Recommendation System. GitHub. <https://github.com/TianyaoHan/Steam-Recommendation-System>
- Ardalis. (2023, February 21). Architect modern web applications with ASP.NET Core and Azure - .NET. Microsoft Learn. <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/>
- *MockPlus*. (n.d.). <https://www.mockplus.com/free-wireframing-tool/>
- *LucidChart | Diagramming powered by Intelligence*. (n.d.). <https://www.lucidchart.com/pages/>
- 829-2008 - IEEE Standard for Software and System Test documentation. (2008, July 18). IEEE Standard | IEEE Xplore. <https://ieeexplore.ieee.org/document/4578383>