NetQuax

Group #3, Super Duck Force

Andrew Ferguson, Brendan Blais, Brandin Dennin, Jordan Gardiner, Sawyer Marchand

CSC 445 - 01

Software Design Document

Version:(5) Date:(05/03/2019)

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This software design document describes five major design components of ‘Netquax’. Firstly, the architectural design section which includes a diagram that presents the relationships between subsystems and data flow. Secondly, the design rationale, this section includes reasoning for certain design decisions, which are based on priorities. In the next section, we will review data storage details. This section involves technicalities about how data is captured and held by the system, as well as essential functions to be implemented. The later section of the document will examine human interface specifics. This will include user interactions with particular views, and what the user will encounter when they are traveling from screen to screen. Here, user experience will be discussed regarding certain components of the system. The last section will mainly consist of cross references between the segments discussed in this document and the specifications discussed in the requirements document.

1.2 Scope

Specific objectives for NetQuax include providing an easy way for users to log on, browse for a movie, choose a renting option, and check-out. Each one of these tasks needs to be simple and intuitive to an individual who does not have a technical background. The process of browsing is made easy with the use of filtering and search features.

Some goals that are less definitive that we hope to work towards achieving include;

creating safe, robust software that is not sensitive to unexpected user behavior. Also, to provide easy user interaction and decreasing occurrences where a user is confused or frustrated, as well as minimizing negative maintenance effects, (for example backing up data at the time where it least effects others.)

There are many benefits to our product. Due to the convenience of streaming from any device, there is potential for a large audience. Also, affordable pricing (much lower than average movie ticket cost) appeals to thrifty consumers. No downloading required is yet another benefit, considering data storage is needed in that case.

1.3 Overview

This document will be broken up into six main sections. Firstly, the Introduction including this document's purpose, scope, and definitions. Secondly, a brief overview of document's organization. Following, the architectural design and rationale, and data design. Lastly, closing with human interface design details and a requirements matrix that will pertain to the details discussed in the Requirement's Specification document.

1.4 Reference Material

Chapter Eight of Textbook 'Software Engineering' by Ian Sommerville, 10th edition. In this chapter, the author describes several testing methods that are to be used during the testing.

1.5 Definitions and Acronyms

UI - *User Interface* - The presentation of the final product as it is presented to the customer.

DFD- *Data Flow Diagram* - used for visualization of data processing that is shown by a simple graphical representation.

OO- *Object Oriented* - Object oriented design is based on the notion that all system objects interact with one another, each one possessing individual attributes and methods.

DB - *Database* - An organized electronically stored collection of data.

UML - *Unified Modeling Language* - a standard way of visualizing the design of a system.

ADL - *Architectural Description Language* - A conceptual model used to represent system architectures.

SSD - *System Sequence Diagram* - Diagram used to show objects where objects are presented horizontally, and interactions are represented through labeled arrows. A thin rectangle represents the time in which the object is the controlling object in the system.

State Diagram - Image used to show how objects respond to different requests from user, as well as the transitions that occur when these requests are submitted.

MVC- *Model-View-Controller :* Separates system interaction and presentation from the system data. The Model component manages the system data and associated operations.

The view manages the way that the data is presented to the user (User Interface Interactions.)

The control unit manages actions performed by users, and then passes these actions to the view and model.

2. System Overview

‘Netquax’ simulates a simplified version of a service similar to Netflix. The design is focused on making the rental transaction process to be relatively straightforward and smooth. Browsing for a particular movie will be simplified by allowing users to filter by a particular individually enacted personal request or constraint. For example, searching for movies that were filmed in the 1990‘s only, or contradictory, searching for movies that were not filmed in the 1990‘s. Importantly, the check-out process must be seamless. If it is confusing in any manner, the user has a greater chance of opting out of pay-for options and resorts to the free service.

3. System Architecture

3.1 Architectural Design

The modular design of NetQuax allows for separation of user interface, server methods, and required data. The end-user interacts exclusively with the user interface and cannot interact with the server or the database directly. However, user interface elements contain server hooks that give the page functionality. For Example, when the user clicks on a movie their selection is posted to the server, and the server returns the movie selection view with the correct data model. This view contains data from a data-model, which is populated with information regarding the chosen movie. If the user selects the movie with ID 1, the controller method will return the view with a model populated with whichever movie matches the ID. The model connects directly to the entity classes to instantiate objects matching the value or values passed into the model constructor. This provides a separation between the view and the data, if data retrieval fails the views should still be able to connect if the correct fail safes are in place. The object classes connect directly to the database to retrieve their associated information. With these requirements

in place, the team decided to use Asp.Net MVC. Since NetQuax is a web application we will be using Microsoft's primary web application framework with integrated Microsoft SQL Server and Azure Cloud publishing support. Microsoft SQL Server is a relational database system, which will be required to keep track of entity relations and inter-leaved data. Proprietary inter-connected components were chosen over portability because of familiarity with the development and connection system, and because of time constraints placed upon the project. Static web pages were considered, put did not give the team enough development flexibility and more time consuming to insert or update new data. Additionally, XML was considered instead of a SQL relational database, but the programming time and code complexity would have been greatly increased.

3.2 Data Decomposition.

See MVC and dataflow diagram.

3.3 Design rationale

This architecture was chosen because of the powerful view manipulation and simple database connections. NetQuax must display an array of data to the end-user with multiple presentations and all the data NetQuax uses is stored in the database. Additionally, there are many operations that manipulate database entities, therefore we chose to use the architecture described above.

4. Data Design

4.1 Data Description

Our data and information files will be interfaced with the combination of mySql and C#. All of the major data and entities involved will be stored within the SQL database setup called NetQuax. We are processing in multiple ways. The netquax database will have separate tables for different data. For example, our 'movie' table will hold all the information for our movies such as but not limited to the following: movietitle, movieactor, moviedescription, etc. This will be input by hand across all columns and rows. Our user information will be stored in the table 'users' and will be gathered by asking the user to make a profile. Their input will directly be added to the netquax database with the help of C#. The 'usertypes' table will be also be determined by the user him/herself when registering with our site. If a user decides to choose a usertype greater-than or equal to full-grown duck, then there credit card information will be prompted and received in the same way as the user information, except in the 'creditcard' table. Furthermore, we decided to keep 'addresses' as a separate table, as a subcategory of user.

4.2 Data Dictionary

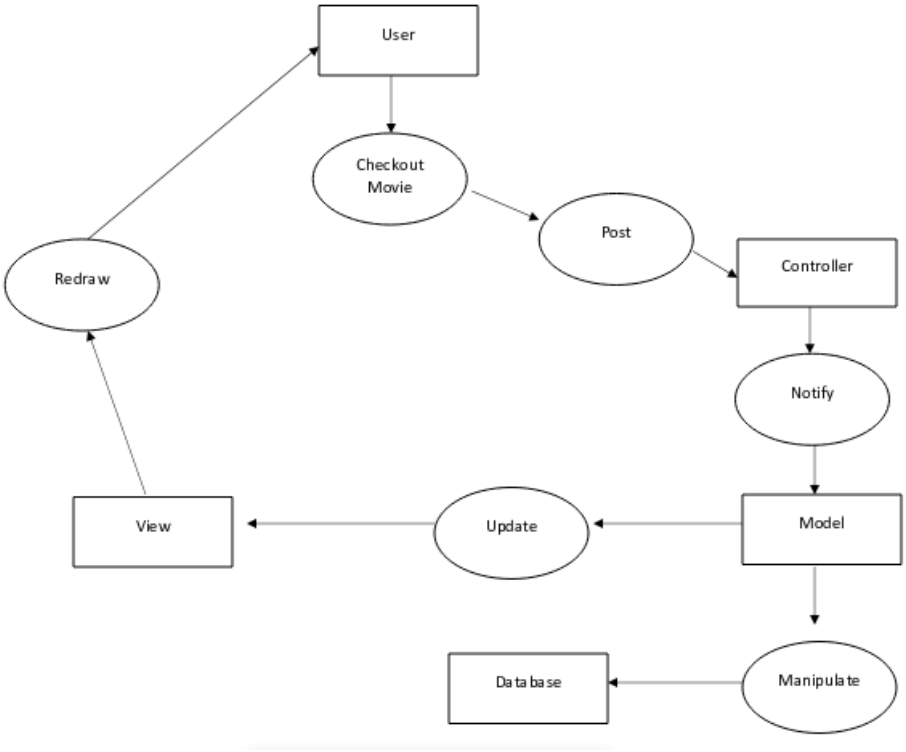

Figure 8.

Data Flow Diagram that describes the stream of actions that allow for data to be retrieved and accessed within our system.

5. Component Design

5.1 Use Case: Login

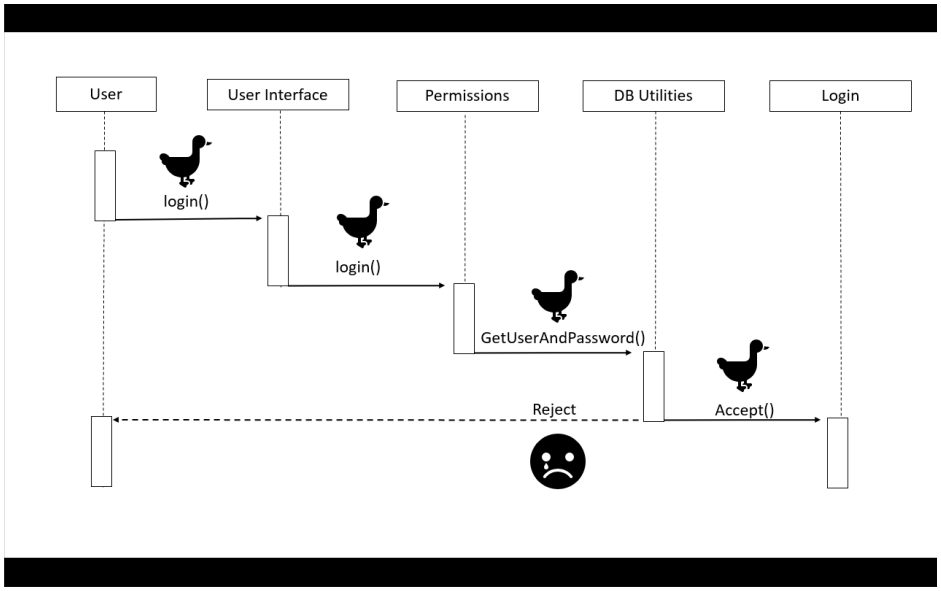

Figure 1.

A diagram of how the user and the system interact to a unvalidated login to NetQuax.

5.2 Use Case: Search

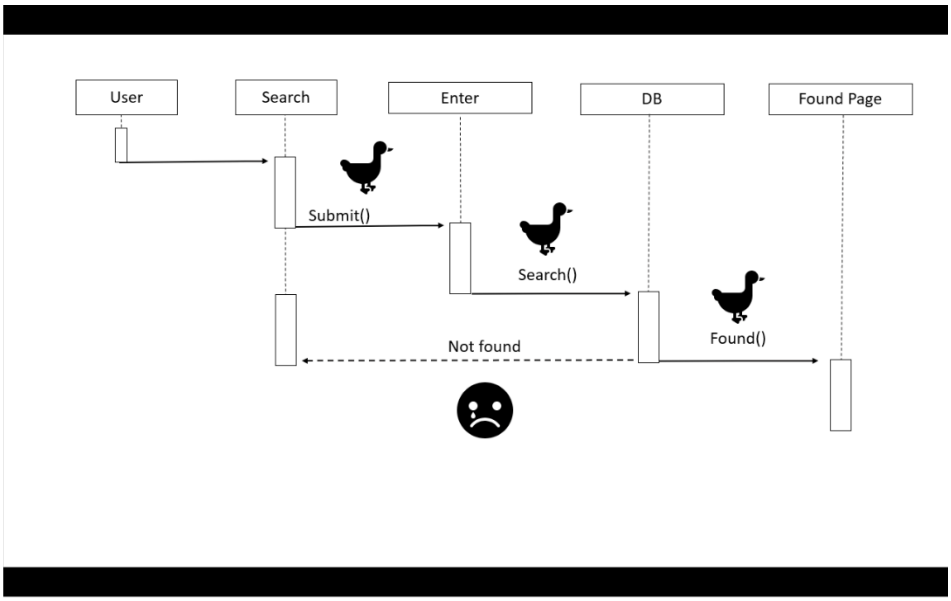

Figure 2.

A diagram to show how the user and system interact with the search functionality in the case where the data being queried is not found.

5.3 Use Case: Browse

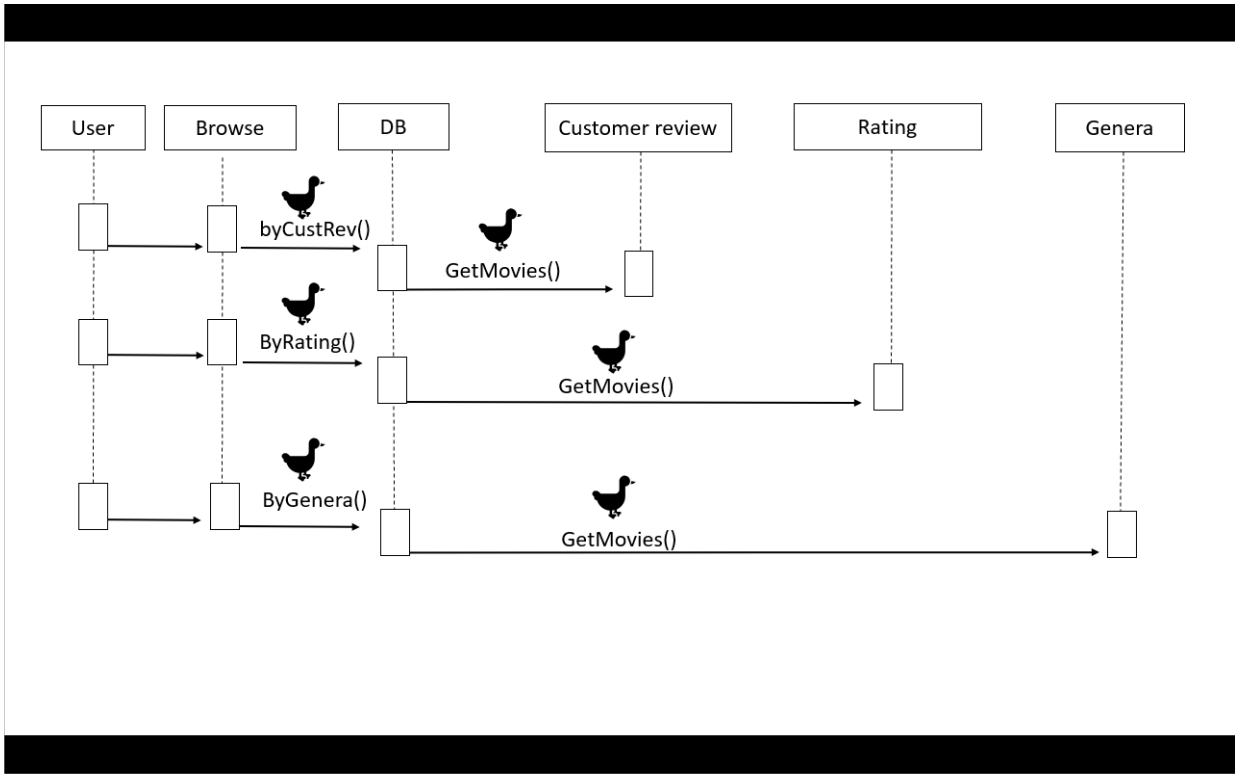

Figure 3.

A diagram to show the user ad system interaction for browsing functionality.

5.4 Use Case: Checkout

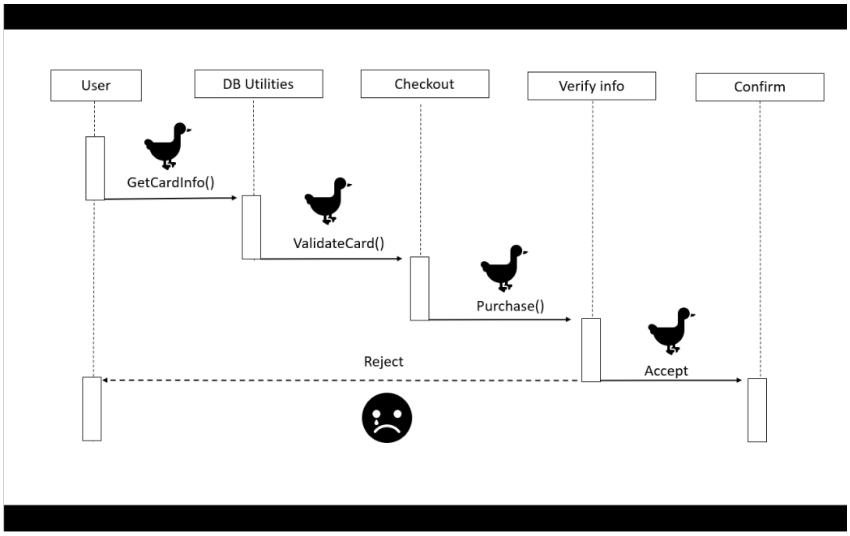

Figure 4.

A diagram to show user and system interaction for check out functionality.

6. Human Interface Design

6.1 Overview of User Interface

When the user goes to the website, they go directly to the home page. The hamburger at the left will have the options for logging in, account settings. Banner at top will be of our logo, below that will have banner style buttons for, from left to right, Home, Browse, Search. Home will have the same functionality as the logo, taking the user to the landing page. Browse will bring up a page/ of genres with checkboxes/radio boxes to allow filtering by genre. Search will allow the user to search movies by name, director,actor,genre and alphabetical. The movies themselves will be displayed side by side and allow the user to scroll through. This is known as carousel format. Each movie will be displayed by it's cover art. Each movie will have it's own page, with the display for the movie being top center, as standard video players are positioned. Below there will be an informational section about the movie listing the following: Director, Genre, Actor(s), Parental Rating, Ratings, Release Date. The log in button will take the user to an account settings page. The option to upgrade accounts will be directly in the hamburger drop down or inside of the account settings page. On the account settings page, there will be options to change email, change password, change billing information, upgrade account version and parental controls.

6.2 Screen Images

Figure 5.

User Interface for Main Home Screen Page. Clicking each of these will open up a pop-up or take you to a new page for the desired press. The movies will be displayed in a carousel format. They will be side by side in a row, and allow scrolling to the left or right through mouse clicks or movements.

6.3 Screen Objects and Actions

The top left button will be a drop down menu style button. When clicked this will extend into a drop down menu over the left side of the screen. This menu will stay extended until the user stops hovering over the menu, clicks away, or clicks a button on the menu. The buttons on this menu will include Log In/Out, Account Settings, and Saved Movies. Clicking each of these will open up a pop-up or take you to a new page for the desired press. The movies will be displayed in a carousel format. They will be side by side in a row, and allow scrolling to the left

or right through mouse clicks or movements.

7. Requirements Matrix

Requirements Matrix					
Requirements Document		Design Document		Purpose	Status
Outline #	Requirement	Outline #	Requirement	Components	
2.1.2	Interfaces	6.1	Screen Objects	Login Buttons, Iframe Videos	In Progress
		6.1	Screen Actions	Button Functionality	In Progress
		6.2	Visual Page	Home Page, Login Page	Partially Completed
3.2	Functions	4.2	Addresses	Stores and Retrieves Address From Database	Implemented Front End Code, Back End in Progress
			Users	Stores and Retrieves Users From Database	Implemented Front End Code, Back End in Progress
			Movies	Stores and Retrieves Movies From Database	In Progress
			Credit Cards	Stores and Validates Credit Card From Database	In Progress
3.2.2	Login Function	3.1 & 5.1	Architecture & Component Design	Enables Users To Enter Username and Password And Validates	In Progress
3.2.3	Play Movie	3.1	Architecture	Enables Users to Play and Pause Video By Using I Frame	Implemented Front End Code, Back End in Progress
3.2.4	Rate Movie	3.1	Architecture	Enables Users To Rate A Movie and Stores	In Progress
3.2	Checkout	3.3	Architecture & Component	Enables Purchasing of Movie and Changing of Billing Type	In Progress

Figure 6.

Requirements Matrix- describes the current status and purpose of main functionalities.

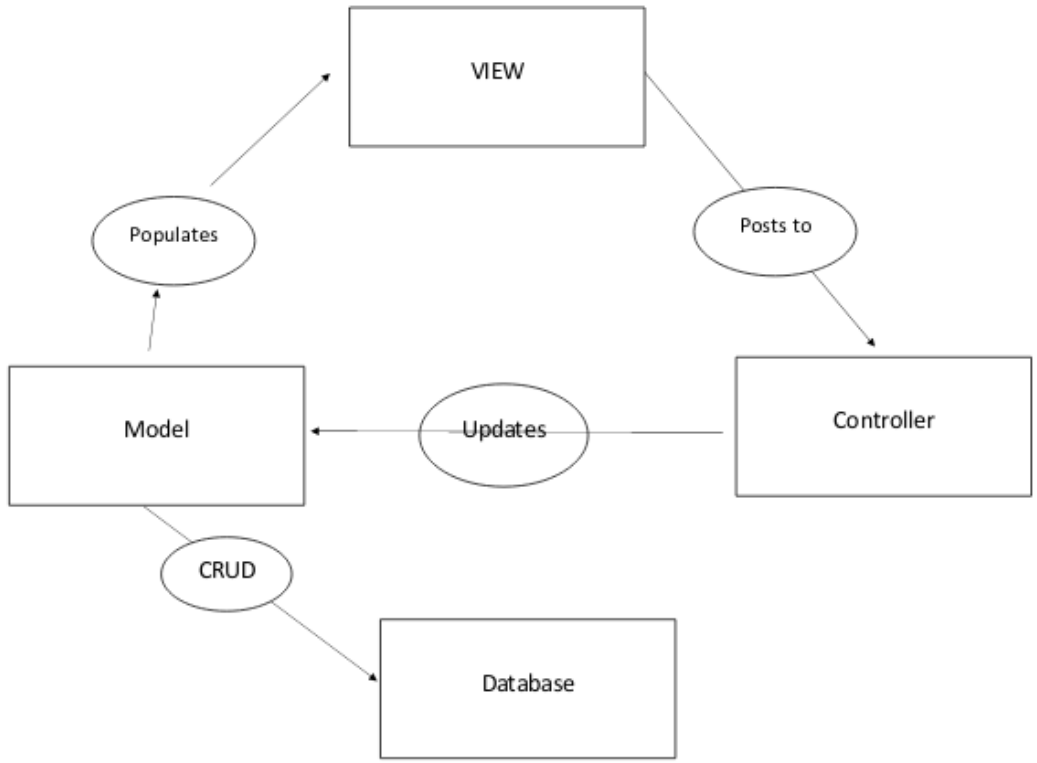

Figure 7

Model View Controller: simple diagram that shows how the framework interacts with each component.