CSUDH
CALIFORNIA STATE UNIVERSITY, DOMINGUEZ HILLS

Summer 2021

Senior Project Report

Computer Science Department

California State University, **Dominguez Hills**

Software Systems, Design Patterns, and the Modern-Day Software Engineering Process

Prepared by

Jonathan Chua

In
Fulfillment of the requirements

For
Senior Design – CSC 492

Department of Computer Science
California State University, Dominguez Hills
Summer 2021

Committee Members/Approval

Dr. Mohsen Beheshti		
Faculty advisor	*Signature*	*Date*
Committee member	*Signature*	*Date*
Committee member	*Signature*	*Date*
Dr. Mohsen Beheshti		
Department Chair	*Signature*	*Date*

ACKNOWLEDGEMENTS

I would like to acknowledge the following people who have contributed directly and indirectly to the road leading up to the completion of this report. The scientific and engineering process involves a lot of guidance and hard work to employ, it would be a disservice to everyone involved in my success if I did not take this opportunity to properly convey my gratitude.

To Dr. Horace Crogman, Professor of Physics at California State University, Dominguez Hills for emphasizing the importance of asking questions and constant encouragement towards the pursuit of excellence. Dr. Bin Tang, Professor of Computer Science at California State University, Dominguez Hills for your tireless effort in finding ways to abstract the many algorithms that exist in the field of Computer Science to a more palatable complexity. Dr. Alireza Izaddoost, Professor of Computer Science at California State University, Dominguez Hills for strengthening my foundation in relational databases. Professor Malcolm McCullough, Lecturer of Professor of Computer Science at California State University, Dominguez Hills in spreading your enthusiasm of programming paradigms, as it has had a major impact in how I approach programming problems. Dr. Brad Hollister, Professor of Computer Science at California State University, Dominguez Hills for your careful curating of the topic of Software Engineering. The direction for this senior design project was heavily influenced by the principles emphasized during your lectures. Finally, to Dr. Beheshti Professor of Computer Science at California State University, Dominguez Hills for advising me in my academic journey to this point.

TABLE OF CONTENTS

Software Systems, Design Patterns, and the Modern-Day Software Engineering Process

Jonathan Chua
Department of Computer Science
California State University, Dominguez Hills
Carson, California 90747

Abstract

In software development, commercially produced software are commonly comprised of many individual systems communicating with each other to accomplish a given task. The *Go Ahead, Make My Movie Night* desktop application demonstrates the complexity of such software systems often found in commercial software applications and exposes the importance of adhering to the principles of the software engineering process to produce a scalable and efficient design of a product. The aim of this project is to show how a simple task can require thoughtful decisions and careful planning to produce an optimal solution. By using modern-day design practices and engineering doctrines, the development of the *Movie Night* desktop application follows the fundamentals of object-oriented principles, implements representational state transfer architecture, employs application interfaces, and design philosophies to produce a reusable and scalable solution to commonly occurring problems in the given context of Hypertext Transfer Protocol and development in today's World Wide Web.

Key Words: application programming interface (API); commercial software; full-stack developer; Hypertext Transfer Protocol (HTTP); loosely coupled systems; representational state transfer (REST); software stack; World Wide Web (WWW)

I. INTRODUCTION

A. *Purpose*

Current trends within the software development industry have gravitated to loosely coupled system of systems. Applications for example, are typically broken down to parts that comprise of the client-side application communicating with a server-side system. In an attempt to improve scalability and abstraction, APIs are commonly implemented to loosely couple the two parts so that each system can exist independently from the other. The expectation of this approach is that any changes within one system will not inherently affect the other, therefore requiring its own separate engineering process and batch of unit tests. The purpose of this report is to describe how this senior design project encapsulates the modern expectation of a full-stack developer and applying the software engineer process to the software stack of the desktop application.

B. *Goals*

The goal of this project, and therefore the desktop application, is to showcase my understanding of software stacks that are commonly expected in today's market of software engineers. These goals will be accomplished by implementing several design patterns to complete the desktop application. Through the design patterns, explicit use of software APIs, database management, networking, and security will be executed to accomplish the established objectives of the project. Finally, as with the goal of the desktop application itself, to successfully provide movie suggestions that the user might consider.

C. *Motivations*

Engineering is a continuously refining centuries old process of the application of scientific principles to designing and building complex constructs for society. Unfortunately, software engineering has often been viewed as a novelty or a doomed disciplined due to its inherent inability for real-world validation when attempting to provide empirical evidence during research and development [1]. As a computer programmer, majoring in computer science, the motivation to validate software engineering as an accepted and respected branch of engineering to our engineering peers is the impetus for the project.

D. *Contributions*

As contribution, to the purpose of the project, I intend to implement a movie suggestions desktop application. The application's result, to the end user, is to provide a graphical interface in which to interact with the systems that provide these suggestions. The objective of the application's result, for the purpose of this Senior Design course, is to produce all tangible work that comprises of the software engineering process:

1. Design
2. Development
3. Maintenance
4. Requirements
5. Testing

E. *Nomenclatures*

- Abstraction (Software Engineering) – the process of establishing the decomposition of a problem into a simpler and more understood primitives.
- API – acronym for Application Programming Interface.
- Application Programming Interface – a software intermediary that acts as a means for separate software systems to interact with each other.
- App – shorthand term for application.

- C# – a general purpose, strongly typed, programming language developed by Microsoft
- Class – an extensible program-code-template for creating objects, providing initial values of state and implementations of behavior.
- Concept of Operations – document describing the characteristics of a proposed system from the viewpoint of an individual who will be using the system.
- CONOPS – acronym for Concept of Operations
- Create, Read, Update, and Delete – are the four basic operations of persistent storage in computer programming.
- CRUD – acronym for Create, Read, Update, and Delete.
- Database abstraction layer – an API that communicates on behalf of the application to the database.
- Database management system – a software system that enables users to define, create, maintain, and control access to the database.
- DBAL – acronym for database abstraction layer
- DBMS – acronym for database management system
- Dependency Injection – a technique in which an object receives other objects that it depends on. These other objects are called dependencies.
- Desktop application – software that is specifically designed to run in desktop operating system.
- Function – block of organized code that is used to perform a task.
- IMDB – acronym for Internet Movie Database, an Amazon company.
- Method – in object-oriented programming, this is a member function of a class.
- MySQL – an open-source relational database management system
- Operating system – system software that manages computer hardware, resources, and services for computer programs.
- OS – acronym for Operating system.
- Python – a high level interpreted programming language designed by Guido van Rossum and developed by the Python Software Foundation
- RDBMS – acronym for relational database management system
- Representational state transfer – software architectural style used to guide the design and development of how internet-scaled distributed hypermedia systems, such as the World Wide Web, should behave.
- REST – acronym for representational state transfer.
- RESTful – informal term used to describe a web service that obeys the REST constraints
- Software Requirements Specification – description of a software system to be developed modeled in the form of business requirements specifications CONOPS
- Structured Query Language – domain specific language used in programming to communicate with relational databases
- SQL – acronym for Structured Query Language
- Technical debt - a concept in software development that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer.

II. BACKGROUND

A. Overview

Go Ahead, Make my Movie Night is a desktop application that produces a recommended movie based on the given constraints of pre-defined options, which are:

1. Completely Random Movie Suggestion
2. Movies Currently Trending
3. Movies Now Playing
4. Load saved movies

The four constraints can be described as follows:

1. Randomly picks a movie from the IMDB database, the movie suggested would be a movie that has already been released.
2. A movie that is currently receiving a lot of web traffic from the IMDB site organically, that is not a movie currently playing in theaters or a movie that has been recently released to DVD, Blu-Ray, or to streaming services.
3. A movie that is currently playing in theaters.
4. Previous movies that have been suggested in the past, which the user has saved for future consideration.

B. Application Design

The desktop app produces a list of movie suggestions based on the buttons provided. The listed movies are not available to the user upon generation, since the purpose of the app is to provide a suggestion on a case-by-case basis. The aim of the design is to prevent the user from being overwhelmed by the number of suggested movies that the app will produce, therefore promoting a better case scenario where each suggestion is considered simply by the merits of the movie and not if it is to be considered out of thousands of possible suggestions.

Each suggested movie will produce the properties that would typically make up the details of a movie, like title of the movie, the genre, the movie's MPAA audience rating (not to be confused with scoring system that attempts to rate reception), the runtime of the movie, the year it was released, the IMDb rating (scoring system that rates reception of the movie), the primary language of the movie, a list of people who directed the movie, and the list of people who are starring in the movie.

When a movie populates the user interface, the user has to decide what to do with each suggestion provided. Not deciding on the movie, for example, closing the app or clicking a different suggestion type than the one currently being used, will cause the app to possibly suggest the same movie again in the future.

A use case scenario that required consideration was the possibility of a suggestion that the user likes, but current circumstance dictated the suggestion be passed. Therefore, one of the options provided will permit the user to save the suggested movie for future consideration. If this selection is made on a movie suggestion, an event trigger will occur that cause the application to package the information of the movie to a JSON object and send a request to the REST API, that was developed for this project, to be saved in a database. Loading the user list of previously saved movies will delete the movie from the database upon load. The reasoning behind this design is for the user to not let the saved content clutter by inaction. If the previously saved suggestion continues to be a movie the user wants to continue considering in the future versus consuming now, it can be saved again to the database before suggesting another movie from the list of previously saved movie suggestions.

The other option, on what to do with the provided movie suggestion, lets the user mark the movie as a movie that user will watch. The purpose of this action is to create a record of movies that the user has viewed by virtue of this application. This record will then be used as cache as a mechanism for cross-referencing movies that IMDB has already provided as a suggestion. The record also acts as a future data pool of data points that can be used to make tailored suggestions based on the types of movies the user has stated to watch [2]. This feature mentioned is a future implementation to be considered and will be discussed in more detail in the Conclusion and Future Works chapter of this report.

The final option, on what to do with the provided movie suggestion, lets the user mark the movie as a movie that the user is not interested in. The purpose of this option, outside of producing the next suggestion, is to create a separate record of movies that the user has shown disinterest over. This discarded record, like the watched record, can be used as a cache to cross-reference movies provided by IMDB so that already suggested movies do not show up again in future suggestions. The other reason is to create a separate data pool of data points is to increase accuracy on a future feature of tailoring movie suggestions based on patterns produced by movies the user has shown disinterest in [2]. This feature mentioned is a future implementation to be considered and will be discussed in detail in the Conclusion and Future Works chapter of this report.

III. System Overview

A. Architecture

The desktop application is designed with modularity in mind. In the outset of design, the architecture was meant to mimic a simple application system with a subsystem to provide flexibility of function when it came to meeting future business requirements. While keeping initial investment costs down, where in the spirit of Senior Design can be equated to time, the direction of reusing existing systems to incorporate into the application's features was the impetus of the direction for the movie suggestion features [3].

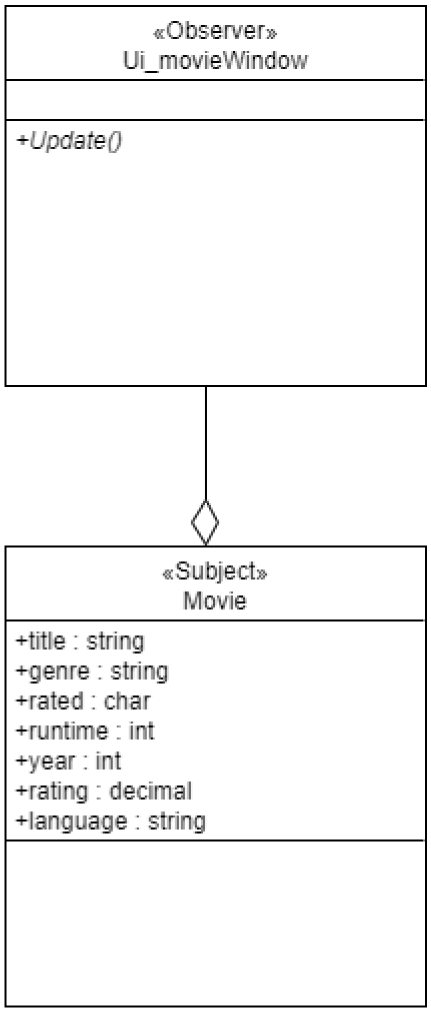


Fig. 1. UML class diagram based on Observer pattern

The user interface was designed with the Observer pattern in mind, shown in Fig. 1, during design phase of the process. Since this application has many derivatives, the engineering process dictates that the design should be derived from a common pattern that has already been determined to produce an inherently pleasing and effective solution [3],[4].

Each event trigger and functional tasks that are tied to them are loosely coupled to each other. The IMDb API is very robust, where several of the functional aspects of my desktop application was designed around the features that the API had already built in. An example of such a feature was the trending movies option, where it would have been difficult to figure out an acceptable algorithm to make such a distinction, as well as finding the metrics to plug into my algorithmic analysis that would determine if a movie was trending.

Initially, the intent was to have a more robust suggestion algorithm where the user could define the initial set of data points to help the program return more relevant movie suggestions to the user from the outset. That feature is still possible as a future integration for this application but considering that the application is built to work in conjunction with an existing system (IMDb) it was prudent to implement those first as a proof of concept for a loosely based application system.

The design of the RESTful web service to interface with the database, will be a CRUD REST API implemented using the dotnet framework. The web service acts as the subsystem to the application, which acts as a database abstraction layer to unify the communication between the desktop application and the sql database, as shown in Fig 2.

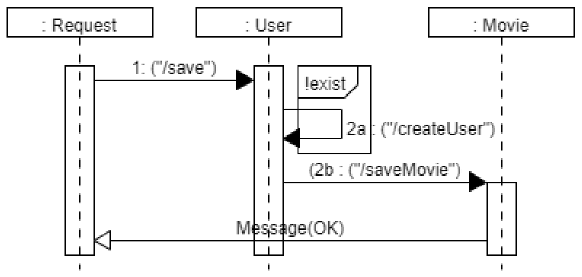
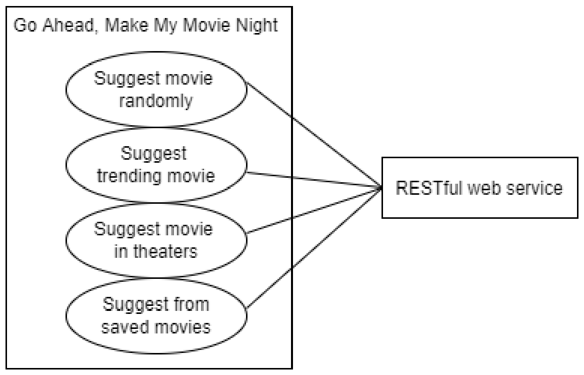


Fig. 2. UML sequence diagram of the RESTful web service on save

The purpose to adding a DBAL was to further enhance the loosely coupled design of the desktop application with its backend component. The intention was to separate the requirements that govern the desktop application to be independent of the requirements that would govern the SQL database development [5], this abstraction can be seen by the Use Case diagram in Fig. 3.



Go Ahead, Make My Movie Night
Suggest movie randomly
Suggest trending movie
Suggest movie in theaters
Suggest from saved movies
RESTful web service

Fig. 3. UML use case diagram of the application and its neighboring system relationship of the RESTful web service

The conscious decision of the design permits scaling of the desktop application while providing a consistent interface with the database that stores the application's saved data. Equally important, the desktop application becomes database-agnostic which makes unit testing the source code easier by virtue of more defined requirements from the other separate systems that are running in orchestration with the desktop application. The design result can be seen through the use case diagram from Fig 4. From the web service relationship to some abstract database.

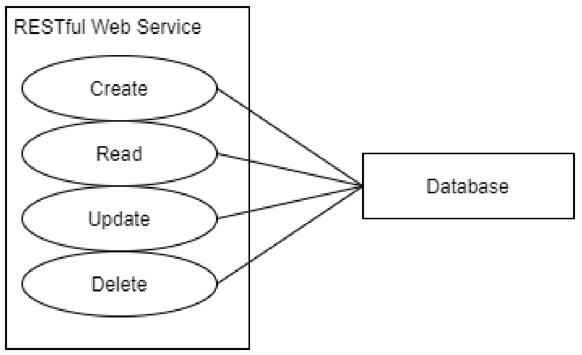


Fig. 4. UML use case diagram of the RESTful web service and its neighboring system relationship as an abstraction layer to a database

System agnostic design of the database can be accomplished from the RESTful web service by calling on stored procedures designed at the database level that reflect the basic operations that the REST API will need to implement. The relationship that exists between the systems and design can be seen on Fig. 5

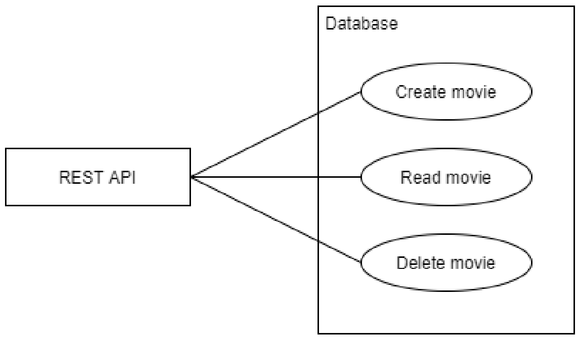


Fig. 5 *UML use case diagram of the database by the REST API. Use cases represent stored procedures*

The design of the database was conducted with the awareness that additional functions of the desktop application might include saving the analysis of the future implementation of the smarter suggestion feature. Saving the data points produced from user choices to a database did not seem like an inherently obvious design choice due to the increasing cost of the proposition without a business requirement dictating the need to the preservation of the data. Future business requirements might dictate the need for saving those data points, but the current function of the application nor the future features outlined in this report did not dictate its necessity. Thus, a cost to benefit analysis was required in determining the best design of the database schema. The result of the exercise was the conscious choice of omitting the movies records produced by the user's choices to help with future storage scalability and the separation of the movies table, which will contain movies saved by the users for future consideration, with the user table in the database design. The relationship between the two entities, as a result of the design, means that the users and movies will have a one-to-many relationship, which can be seen in Fig. 6.

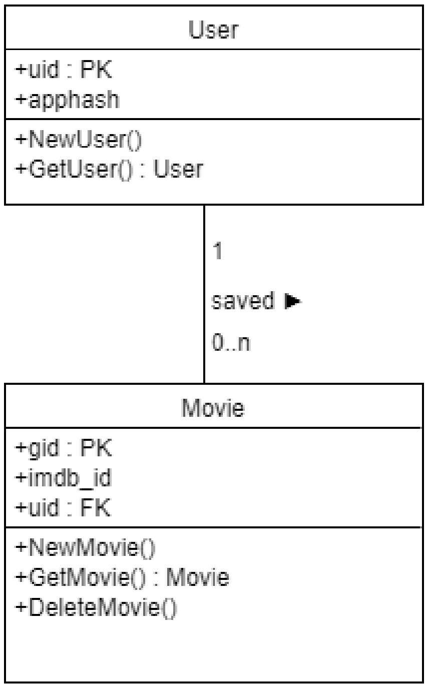


Fig. 6 Database Model in UML format. An instance of the user can have 0-to- n relationship with movies and a movie has to be related to the user who saved it

The design choice should result in less expensive queries, where query cost is equatable to I/O bandwidth and computational resources required per query, at the expense of increase data storage cost due to movies stored in the save movies table not being unique. The technical debt accrued by the exchange can be considered more than equitable due to the trends of storage costs compared to computational cost in today's modern market [6].

B. Characteristics

The desktop application is comprised of two primary classes, the interface (observer) and the movie (subject). The interface contains a *primary* method to update the content of the UI when a request and successful return response is given by the APIs used in the system. The update method is broken down in a piecewise functional manner for the purpose of code readability. Currently, the application gives the user 4 use case scenarios to dictate the suggestion, illustrated by Fig. 7.

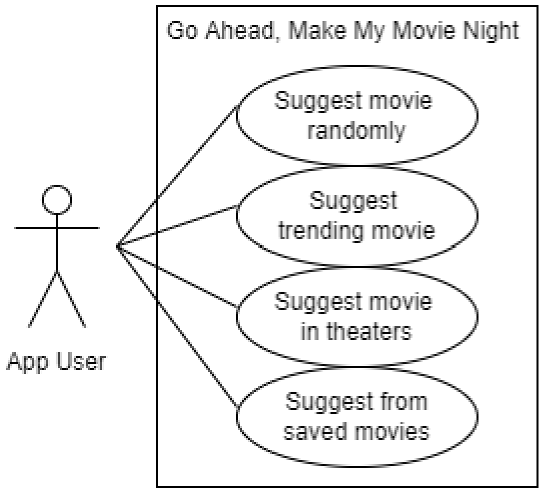

Go Ahead, Make My Movie Night
Suggest movie randomly
Suggest trending movie
Suggest movie in theaters
Suggest from saved movies
App User

Fig. 7. UML use case diagram of the application

The saved movie suggestion will derive its suggestions from movies that have been previously saved by the user. The program will communicate with the database abstraction layer to produce a suggestion for the user by pulling the oldest entry in the saved movie list. Fig. 8 captures the behavior of the web service when the program uses previously saved movies as the source for its suggestions.

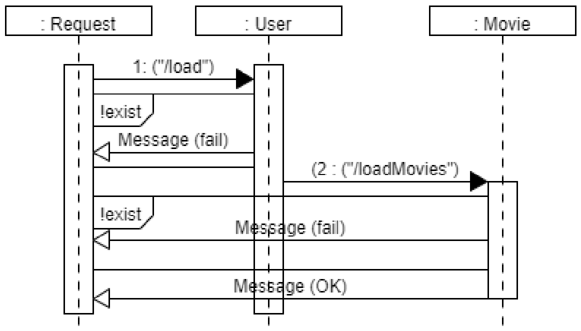


Fig. 8. UML sequence diagram of the RESTful web service on load

The supplied suggestions based of the user options:

- Suggest movie randomly
- Suggest trending movies
- Suggest movies in theaters

Are produced by the IMDb system through the use of their RESTful web service by implementing its REST API. The IMDb REST API is implemented by the program through the *ImdbApi* class, which is a static and stateless class. The *ImdbApi* class attributes are compositions of the IMDb configuration file(config.imdb.py), shown in Fig. 9. The purpose of this design was to loosely couple the IMDb system parameters from the program's source code. Therefore, any changes that occur to IMDb's REST API call parameters, to use its RESTful web service, can be made at the configuration file to satisfy the needed changes to the desktop application for continued use of the IMDb DBAL.

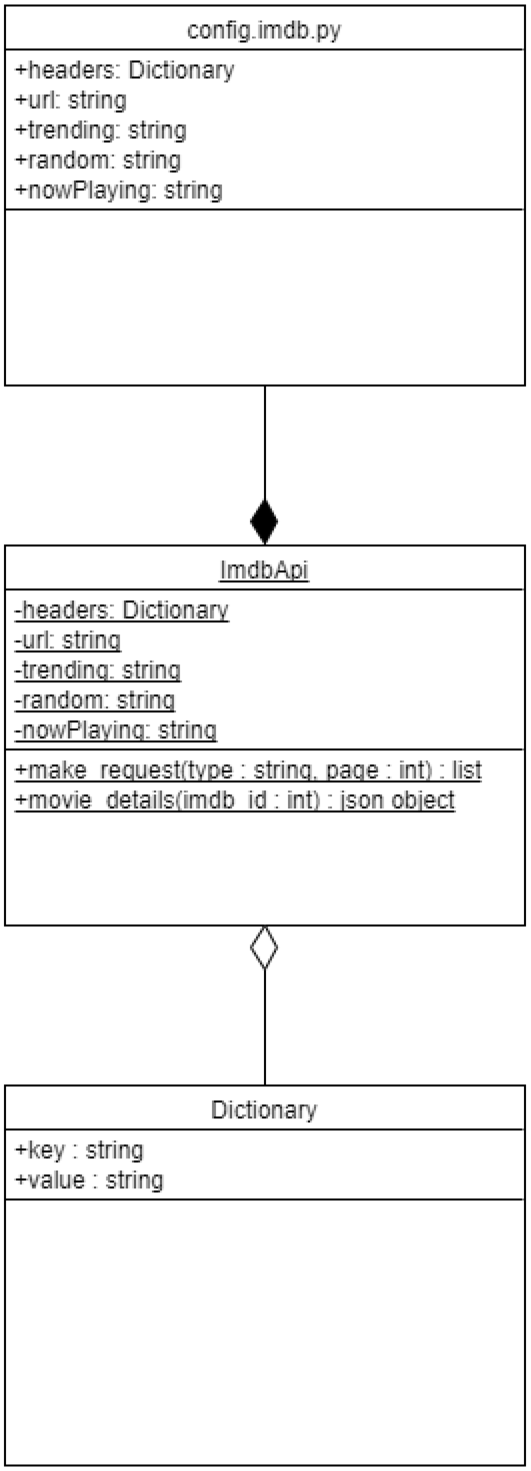


Fig. 9 – Class diagram of ImdbApi and its respective relationships

IV. System Context

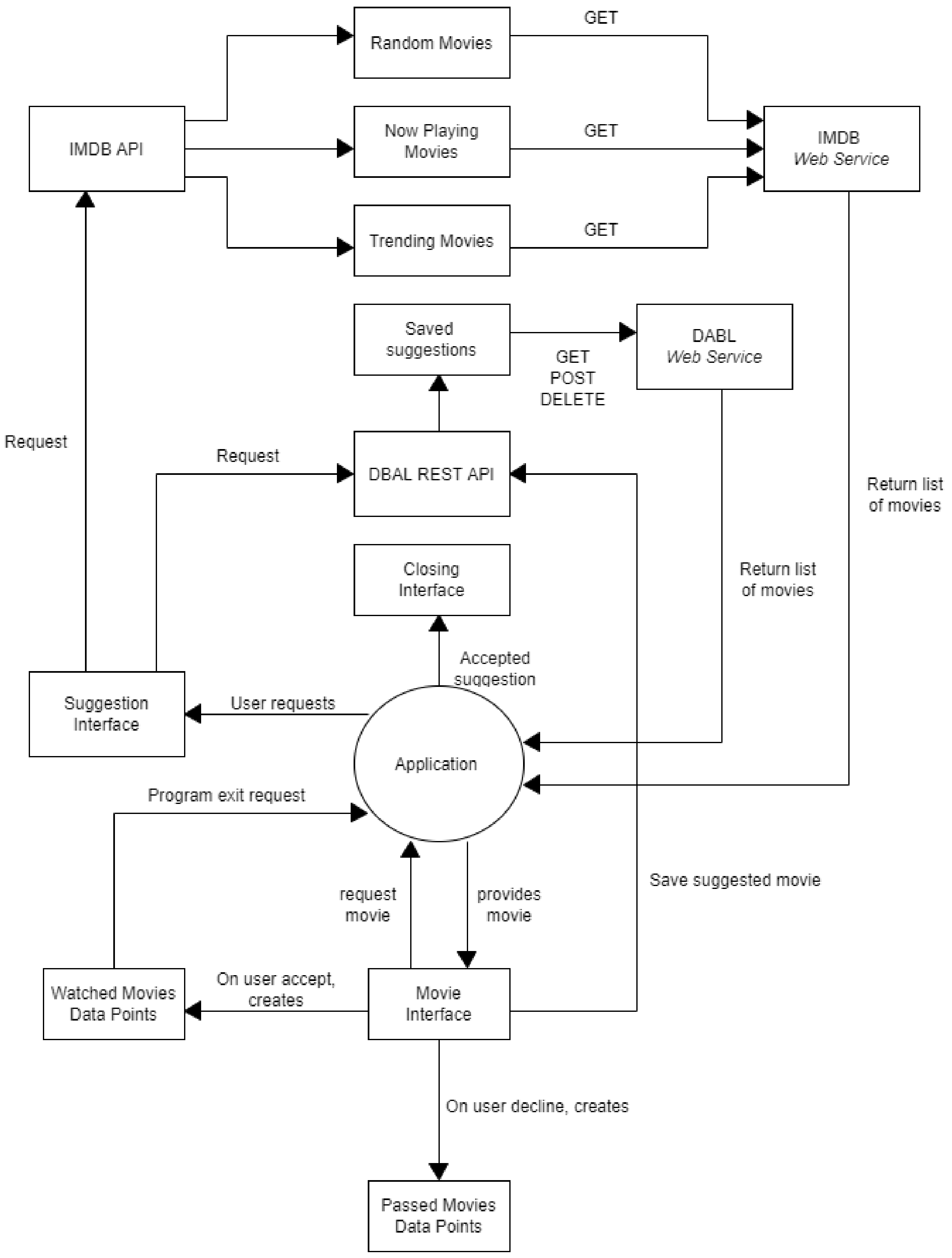


Fig. 10 – System Context Diagram

A. *Context Diagram*

The fundamentals of software engineering abstraction were exercised during design, which is visualized through the multiple entities displayed in Fig. 10, page 14. The context diagram is still considered to be a very high abstraction layer of the system's design. By breaking down the application's systems to its most primitive forms, the complexity in defining the requirements and implementing the design decreased for every split made into smaller abstractions. Information management is one of the goals of abstraction. Complex features of one abstraction are simplified into another abstraction. Good abstractions can be especially useful while bad abstractions can be very harmful, this obvious distinction emphasizes the importance of the engineering process in software development. A good abstraction leads to reusable components. Reusable system components imply generality, well designed generality implies loose coupling, which was identified as one of the primary goals of the project.

The desktop application developed represents a larger software system that would be normally employed to create a similar conceptual program in today's modern product development model. The application can be decomposed to three primary complex abstractions, which can be visualized by Fig. 11, page 16:

1. The graphical interface and its corresponding functional responses to events; the compiled application that runs at the client side. This can be abstracted as the integration of the user experience within the system.
2. The database abstraction layers; the RESTful web services that facilitate the communication of the backend systems of the application. This can be abstractly viewed as the integration of the system's core behavior.
3. The server-side database that is storing the user's data at the client-side level. This can be abstractly viewed as the integration of the backend functionality of the system.

The client-side application is an abstraction of the functions, data, and processes of the user experience. Thus, requirements and goals are independent of the other systems that run-in conjunction of the application. Therefore, the only thing of import is the accuracy of the results that the client-side application enacts as a consumer of other functional abstractions within the software system.

B. *Software Development Life Cycle*

Agile development was the SDLC applied for this project to simulate the approach in addressing how the business practices behind the development influence the building of the software. Today's modern consumer provides ever evolving requirements coupled with the expectation of immediate results. The necessity of producing a working application in the confines of a 5-week academic period required the ability to rapidly deploy an application while being able to emphasize the project's goal of employing the engineering process.

Rapid Application development, a derivative approach to Agile, was practiced at the graphical interface level due to the simulated attempt in capturing mercurial market expectations. Less emphasis on design was implemented in the development of the front end and instead, was driven by feedback from visual response to data that the application was required to provide. The result of the practice meant many changes in direction of the user interface, from one window to four, and several iterations of the four separate interfaces to align with the emphasis of the functional requirements that was dictated by the project proposal.

The functional portion of the system adhered more towards Agile modeling due to its framework lending to easier adherence of several importance doctrines in the engineering process (e.g., variations of modeling, continuous documentation, priority in requirements). The hybrid approach gave more flexibility for software systems where values and principles can vary for each system, while keeping with several edicts that are emphasized in Agile development.

C. *Computer Programming Standards and Best Practices*

The desktop application layer was developed in Python using object-oriented programming paradigms, with emphasis on loosely coupled objects. The Observer design pattern made for the obvious choice due addressing the one-to-many dependency that would exist between the interface (observer) and the movies objects (subject) without making the two objects tightly coupled. Adherence to most of the PEP 8 style guide was made through the pythonic conventions of Code Lay-out, Variable naming, Method naming, Dunders, imports, globals, and class names. Liberal application of block comments was used as an abstraction of documentation. Inline comments were used sparingly and used explicitly to make points of emphasis that were not immediately obvious. The technique of dependency injection was implemented with the interfaces to create a better separation of concerns in the construction of each interface class. The object that, represented the dependency, was the credential object required for the application to communicate with the RESTful web service. This technique helped create levels of independence between the classes and that object by means of configuration files, located in“./config”. The technique also included the additional benefit of addressing the separation of web services that the application is employing by supporting different configurations.

The RESTful web service was developed in C#, also using object-oriented programming paradigms, with use of the dotnet framework and dapper. This code base also employs the use of dependency injection, which is represented as the service (to the database) to the object (the API). The dapper framework is a lightweight ORM that maps the incompatible data type between the systems, the sql database for this example, by using objects to represent information. Several layers of obfuscation would normally be made to create more abstraction from the particulars of the database as a practice of encapsulation, but for the purpose of readability of the code base for this project, this was intentionally omitted.

D. *External System: IMDb*

The IMDb REST API was used to acquire the movie assets for the desktop application. The functional aspect of determining randomness, trending, or now playing suggestions is accomplished through the algorithm that the IMDb system employs. The design deficiencies of the API resulted in inconsistent data models being returned for each call, which required flexibility in the movie interface class and movie class design. The movie entity in the IMDb database lacked consistency between objects, which required additional logic to be implemented to provide a more consistent user experience.

V. SYSTEM DESIGN

A. *Software Development Tools*

The project can be broken down into three sections:

[1] Desktop Application Layer
[2] Web Services
[3] SQL Database

To simplify development, a singular code editor was used in lieu of using several IDEs to accomplish the project. Visual Studio Code provided a light weight platform with extensible features to add specific tools required to code-build-debug each code base without the complex workflows of having three separate IDEs.

Qt 5 Designer was used in designing the graphical user interface for the desktop application. The GUI uses the PyQt5 library, a GPL licensed product, is accessible through PIP3 and can be used freely towards non-commercially produced software.

B. *Software Requirements*

The desktop application is a layer of the system that has the following software requirements specification

I. Purpose: provide movie suggestions based of distinct classifiers for movies. Through user interaction with the interface, create data points of the user's movie preference.

II. Overall Description

- Product Perspective: the product should display relevant movie information of the suggestion. The option to save the movie for later consideration, to pass on the suggestion, or to watch the movie now should be provided to the user. Data points should be recorded for each action to refine the movie suggestions provided to improve user experience.
- Design Constraints: the product should accomplish creating a user data point while obfuscating the identity of the user. The application is meant for non-commercial purposes, therefore, monetizing the user is not part of the goal of this application.
- Constraints, assumptions, and dependencies: the application will make use of several web services to acquire movie related assets and to store user related assets. User's personal information should not be used as a dependency in constructing identifiers of the user assets.

III. Specific Requirements

- Logical database requirements: a relational database is required for persistent data storage of the user assets of the application.
- System Attributes: privacy is the highest priority; any user identifiers should be hashed to obfuscate any details and encapsulation should be employed. Maintainability is the second emphasis of the system. System configurations should be distinct and easily accessible for which the desktop application is dependent in using.
- Functional requirements: When a movie suggestion is saved by the user, the movie asset should be stored in a separate database in which it acquired the asset. If the user loads the saved asset, the user should be forced to discard or save it again for further consideration. When the movie is watched, create a data point that can be used to determine user habits. When a movie is discarded, create a separate data point that can be used to determine user habits.

The RESTful web service is the other layer of the system, its distinct requirements are as follows:

I. Purpose: provide the database an abstraction layer for systems that use it as persistent storage. Provide an application programming interface for consumers of the database.

II. Overall Description

- Product Perspective: the service should route relevant requests from consumers of its service to the proper route, so the data provided is properly identified and expected response can be given to the consumer.
- Design Constraints: optimize data assets by storing the minimum required attributes to identify the asset for the consumer.
- Constraints, assumptions, and dependencies: the web service will make use of a relational database as its persistent storage solution. Obfuscation of the database particulars should be accomplished by use of abstractions.

III. Specific Requirements

- Performance requirements: the web service needs to be lightweight. Scalability will be accomplished through containers and orchestration tools to manage multiple deployments.
- Functional requirements: when a GET User is requested, the provided credential by the consumer should yield the matching credential identity to its persistent storage provider. When a GET Movie is requested, the provided credential by the consumer is its identity in the persistent storage, the response should be a movie tied to the identity. When a POST User request is made, the web service should expect a unique credential that does not exist in persistent storage, the response upon creation is the identifier of the credential from the persistent storage solution. When a DELETE movie request is made, the web service should receive a credential identifier and an IMDb_id that acts as a composite key of the movie asset in persistent storage. The result of the request is the removal of the movie assets.

The database is the final layer to the system, its distinct requirements are as follows:

I. Purpose: provide a persistent relational storage solution to the consumer of the system.

II. Overall Description

- Product Perspective: the database should store data in a consistent and object relational manner.
- Design Constraints: user entity has one-to-many relationship with the movie entity.
- Constraints, assumptions, and dependencies: if a hashed credential already exist on a POST request, rehash the credential, and use the result as the hashed credential.

III. Specific Requirements

- Functional requirements: create stored procedures for the web services that fulfills its application programming interface requirements.

C. Application System Diagram

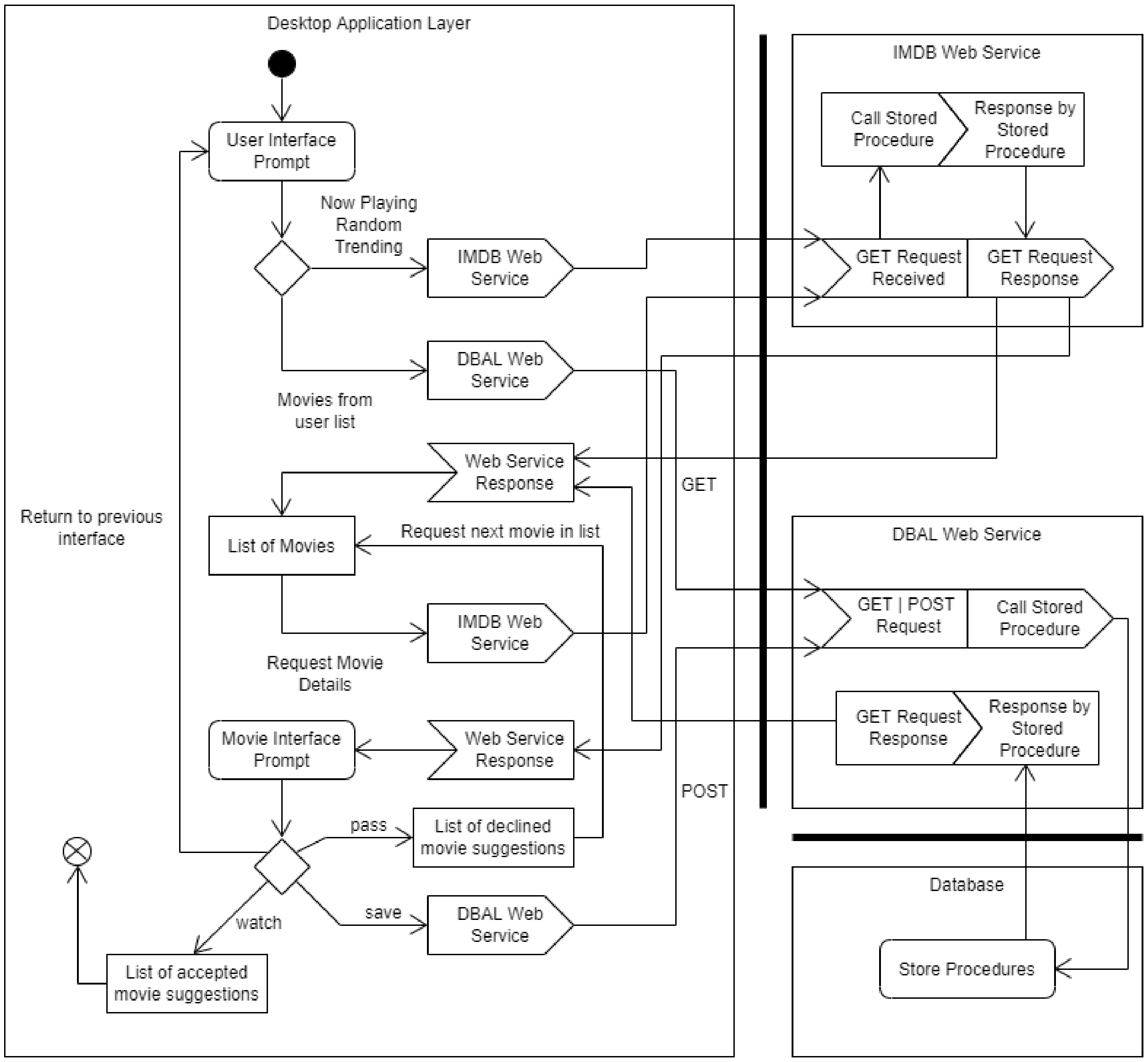


Fig. 11 – System Activity Diagram

The desktop application layer contains 9 distinct class files. The primary classes that interact with the user is the SuggestionUI, represented as the User Interface Prompt entity in Fig. 11, and the MovieUI, represented as Movie Interface Prompt in Fig. 11. Depending on the user's choice at the SuggestionUI, the interface will call on the methods within the static class ImdbApi or the methods within the static class of DbalApi class. Each class implements the respective web service API that it is named after. The MovieUI class also accesses the ImdbApi class methods when it acquires the movie details of the suggestion, which exists as list of 20xIMDb_id. If the list is empty, the MovieUI requests an additional list of 20xIMDb_id from the IMDb web service. The MovieUI class accesses the DbalApi class if the user requested, from the SuggestionUI, for suggestions from the user's saved list. The DbalApi will request a movie from the user's saved list saved in the Database, by consuming the proper store procedure, and upon receipt by the desktop application, the MovieUI will request for the movie details by using the ImdbApi class methods to communicate with the IMDb Web Service. The MovieUI class also communicates with the DbalApi class when the user chooses to save a suggestion that currently populates the MovieUI. If the user makes the request to save, the MovieUI will access the DbalApi class method to request a POST method in the DBAL Web Service. The DBAL Web Service will route the request and request the proper stored procedure from the Database to store the movie asset passed from the desktop application. If the user accepts any movie suggestion by the application, then the desktop application will close.

D. *SuggestionUI.py*

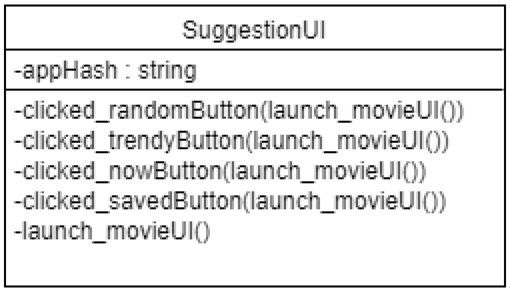

SuggestionUI
-appHash : string
-clicked_randomButton(launch_movieUI())
-clicked_trendyButton(launch_movieUI())
-clicked_nowButton(launch_movieUI())
-clicked_savedButton(launch_movieUI())
-launch_movieUI()

Fig. 12 – SuggestionUI Class Diagram

The SuggestionUI, seen in Fig. 12, will have the appHash string injected to it when the use starts the program from the opening splash screen. The SuggestionUI class has five methods, of which four are tied to the events of the buttons in this class. If the random, trending, or now playing options are selected by the user, the respective method tied to it will be called. In these respective methods contain the particular assignments that will be given to the dependency injection object that will be passed to the MovieUI. The attributes include the type of suggestion that was requested, so it can be continued in the MovieUI object, including the first batch of movie suggestions that the MovieUI will show to the user. The clicked_savedButton method holds particular attributes that are unique to the web service of the DBAL and will pass the dependency injection to the MovieUI that pertains to accessing that particular web service. The launch_movie method instantiates the MovieUI Class and provides it with the proper dependency injection sent from the button that invoked it.

E. *MovieUI.py*

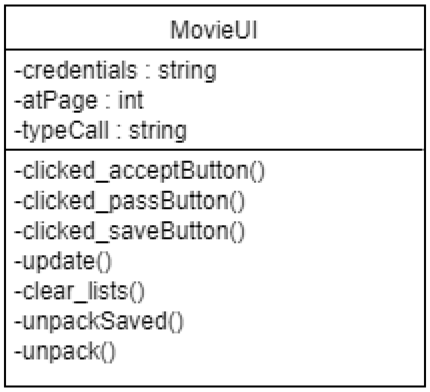


Fig. 13 – MovieUI Class Diagram

The MovieUI, seen in Fig. 13, will have a first set of movies passed to it by the SuggestionUI, the type of suggestion was selected by the user, and the credentials from the appHash. The clicked_acceptButton method will record the movie to a file as a data point for a future feature implementation and exit the program. The clicked_passButton method will record the movie to a file as a future data point for a feature down the road and call on the unpack method or unpackSaved method depending on the suggestion choice involves interacting from a list of movies saved in the project's database or movie suggestions being provided by the IMDb web service. The update method populates the MovieUI with the next movie suggestion and the clear_lists method clears the UI components that are list widgets, so the next movie suggestion does not concatenate with the previous movie's list.

F. *Movie.py*

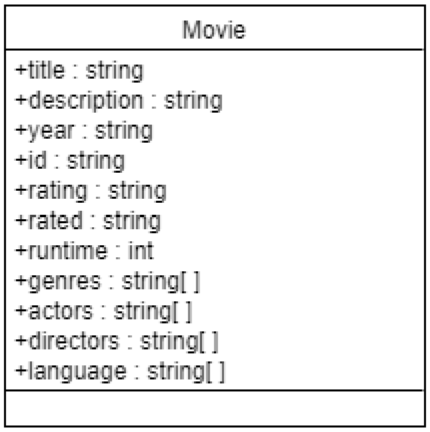


Fig. 14 – Movie Class Diagram

The Movie class, seen in Fig. 14, is instantiated each time the unpack method or unpackSaved method is called by the MovieUI object. The object is comprised of details that were returned by the ImdbApi movie_details method when unpack or unpackSaved method makes the request. This instantiated Movie object is what the MovieUI update method uses to fill the contents of the user interface.

G. *ImdbApi.py*

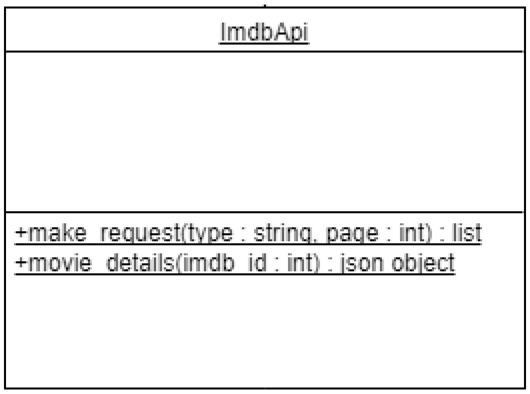

ImdbApi
+make_request(type : string, page : int) : list
+movie_details(imdb_id : int) : json object

Fig. 15 – ImdbApi Class Diagram

The ImdbApi class is a static class shown in Fig. 15, it is also stateless, like the web service. This class facilitates the call to the IMDb Web Services to generate the movie suggestions and retrieve the movie details of a suggested movie. The configuration to properly call the IMDb Web Service are found and accessed by this class at './config/imdb.py' folder in the desktop application. All changes that pertain to properly calling on the web services are made in the imdb.py file and not in the code contained in this class file.

H. *dbalapi.py*

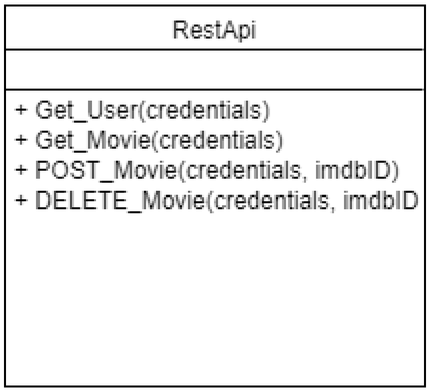

RestApi
+ Get_User(credentials)
+ Get_Movie(credentials)
+ POST_Movie(credentials, imdbID)
+ DELETE_Movie(credentials, imdbID

Fig. 16 – RestApi Class Diagram

The RestApi class, seen in Fig. 16, located in the dbalapi.py file, is also a stateless static class. This class facilitates the calls to the database abstraction layer that was implemented in this project for the sql database. The Get_User method is called by the desktop application at the start of the application and makes a GET request to the web service to retrieve the ID in the database that corresponds to the applications credential that is generated by the application's hashing function. The Get_Movie method is called by the desktop application when the MovieUI is instantiated with the parameters of using previously saved movies for its source of suggestions. The MovieUI will use the Get_Movie method continuously if the user passes or saves on the provided movie suggestion until the user accepts the suggested movie or returns to the previous menu. The POST_Movie method is called by the MovieUI class when the user chooses to save the movie object that is currently showing in the MovieUI object. Finally, the DELETE_Movie is called by the MovieUI object when it is instantiated to get its source of movies from the previously saved movies in the database. For everyone GET_Movie request made by the MovieUI object, a corresponding DELETE_Movie request will be made by the MovieUI object to delete the retrieve movie asset from the database. This methodology was a result of finding a way to creating a queue of resaved movies being pulled so the movies suggested in each instantiation of MovieUI, using previously saved movies, would not always suggest movies in the same order in which they were first saved. Rather this approach treats the table as an abstract implementation of a queue data type.

I. *DataController.cs*

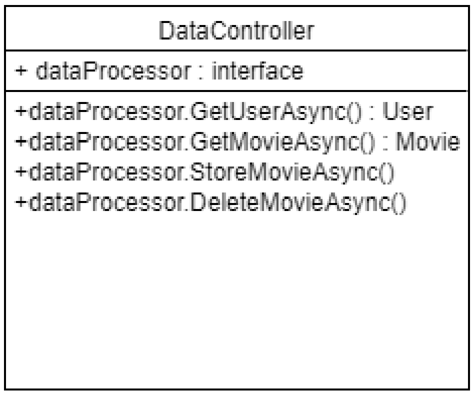

DataController
+ dataProcessor : interface
+dataProcessor.GetUserAsync() : User
+dataProcessor.GetMovieAsync() : Movie
+dataProcessor.StoreMovieAsync()
+dataProcessor.DeleteMovieAsync()

Fig. 17 – DataController Class Diagram

The DataController class, seen in Fig. 17, is one of two primary classes in the RESTful Web Service. It is the object that is instantiated at the start of the program that processes the requests given to the program at port 5000 or port 5001 in the routes stipulated in the class file. Based on the route that the request received at the port, the DataController will use the interface dataProcessor, which the interface of the DataProcessor class, to make the proper method call assigned to the route. GetUserAsync is used when the request is a GET Method with a route '<domain>/{appHash}' where appHash is the credential that is passed by the RestApi class Get_User method. The GetMovieAsync is used when the request is a GET Method with the route '<domain>/{uid}/movie', where {uid} is the identifier returned from the RestApi class' Get_User method and is then used by the same class' Get_Movie method. StoreMovieAsync is used when the request is a POST Method with a route '<domain>/{uid}/{movieID}' where {uid} is the identifier mentioned earlier and the {movieID} is the imdb_id passed to this web service by the RestApi POST_Movie method. Finally, the DeleteMovieAsync is used when the request is a DELETE Method with a route '<domain>/{uid}/{movieID}' where {uid} is the identifier mentioned earlier and the {movieID} is the imdb_id passed to this web service by the RestApi POST_Movie method.

J. *DataProcessor.cs*

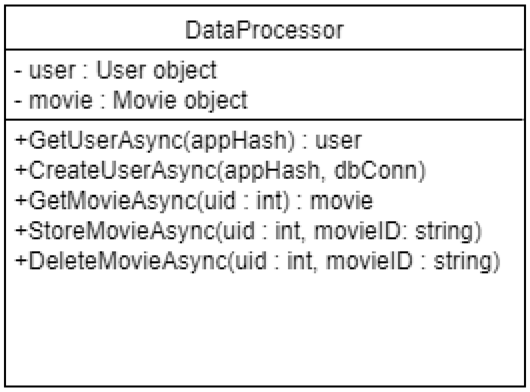

DataProcessor
- user : User object
- movie : Movie object
+GetUserAsync(appHash) : user
+CreateUserAsync(appHash, dbConn)
+GetMovieAsync(uid : int) : movie
+StoreMovieAsync(uid : int, movieID: string)
+DeleteMovieAsync(uid : int, movieID : string)

Fig. 18 – DataProcessor Class Diagram

The DataProcess class, shown in Fig. 18, facilitates the communication between the web service the sql database implemented in this project as its persistent data solution for the desktop application. The User object and Movie object is modeled after the entities that it is based off as a means for the object-relational mapping that the dapper framework implements. GetUserAsync is the method that facilitates the call of the stored procedure that retrieves the user identifier based off the credential sent to the web service from the desktop application layer. If the User does not exist in the database, the DataProcessor will run the CreateUserAsync method which facilitates the call of the stored procedure in the database that creates a user when a credential is paired with the call. GetMovieAsync is the method that facilitates the stored procedure call that returns the "top of the list" movie that exists in the User's saved list of movies.

The StoreMovieAsync is the method that facilitates the stored procedure call that saves the movieID tied user’s identifier. Finally, the DeleteMovieAsync is the method that facilitates the stored procedure call that deletes a movieID that is tied to the user’s identifier.

K. Unit Testing

To complete the implementation of the engineering process, a semblance of empirical tests must be implemented. The RESTful web service contains a unit test called DataControllerTest.cs which creates mock objects to verify expected responses of the methods contained in the class. The unit test provided is not meant to represent a thorough example of the unit testing portion, but rather a representation of this part of the engineering process.

VI. Conclusion And Future Work

a. Future Improvement

Future iterations of the desktop application contain scaffolding to implement smarter suggestions given to the user at the desktop application layer. There is no control on the IMDb side to filter previous suggestions from being provided or filter suggestions based on evolving criteria. The purpose of the saved data points in the file is to parse the frequency of particular attributes that happens to occur more frequently that results in the user declining a suggestion vs. watching a suggestion. It did not seem prudent to implement some feature that accomplishes that feat. without some empirical data that backs the reasoning to the results. With the proper team backing the project, a data scientist can extrapolate an acceptable link between the phenomena of accepting and passing in relation to the movie attributes like actors, directors, genres, runtime, ratings, rated, and language. If a data model is made that can convey the level of importance of particular actors, genres, et al. as a generalization, then I believe the application creates the necessary foundation to implement the model.

Another feature that I would like to implement is for the desktop application to launch the movie's trailer if it is available in YouTube. The feature should be an easy integration to the application layer, but the idea of adding this to the project arrived late in the process and I was not confident that I could complete the integration of the feature before the end of the semester.

b. Conclusion

The academic journey to attain a Bachelor of Science in Computer Science lays ample foundation of the application of the engineer process. With emphasis on design and thoughtful consideration of the software development life cycle, our major has provided the proper curriculum to succeed in today's software engineering job market. The emphasis of engineering in this project is a result, in what I believe, is the regression of title software engineer by the market unable to meet the demand. The frequent attempts of, would be, tech entrepreneurs have generated tremendous need of technologists that it has reduced the criteria that would qualify a programmer as an engineer [7]. I believe my own professional journey as a reflection of the reduction of such standards. The realization was impetus for going back and getting my degree as well as the topic of emphasis that I chose to cover in this senior design project.

In the end, seeing all that I have covered to simply cover a portion of what is typically expected to produce a professionally engineered product, I have come to appreciate the level of work it requires to produce the amazing technology that we use daily but take for granted the amount of effort in case and work it took to develop it. My hope is that a measure of that is shown by the careful approach I took in the development of the software system employed by the application, the supporting diagrams and documentations, and how it is integrated to this project report.

VII. References

[1] Dijkstra, E.W. (1988), “On the Cruelty of Really Teaching Computer Science”, *EWD-1036 E.W. Dijkstra Archive*, Center for American History, University of Texas at Austin

[2] N. Bin, "Research on Methods and Techniques for IoT Big Data Cluster Analysis," *2018 International Conference on Information Systems and Computer Aided Education (ICISCAE)*, 2018, pp. 184-188, doi: 10.1109/ICISCAE.2018.8666889.

[3] I. Sommerville, “Design and implementation,” in *Software Engineering*, 9th ed. New York City, NY: Pearson Education, 2010, pp. 189-195

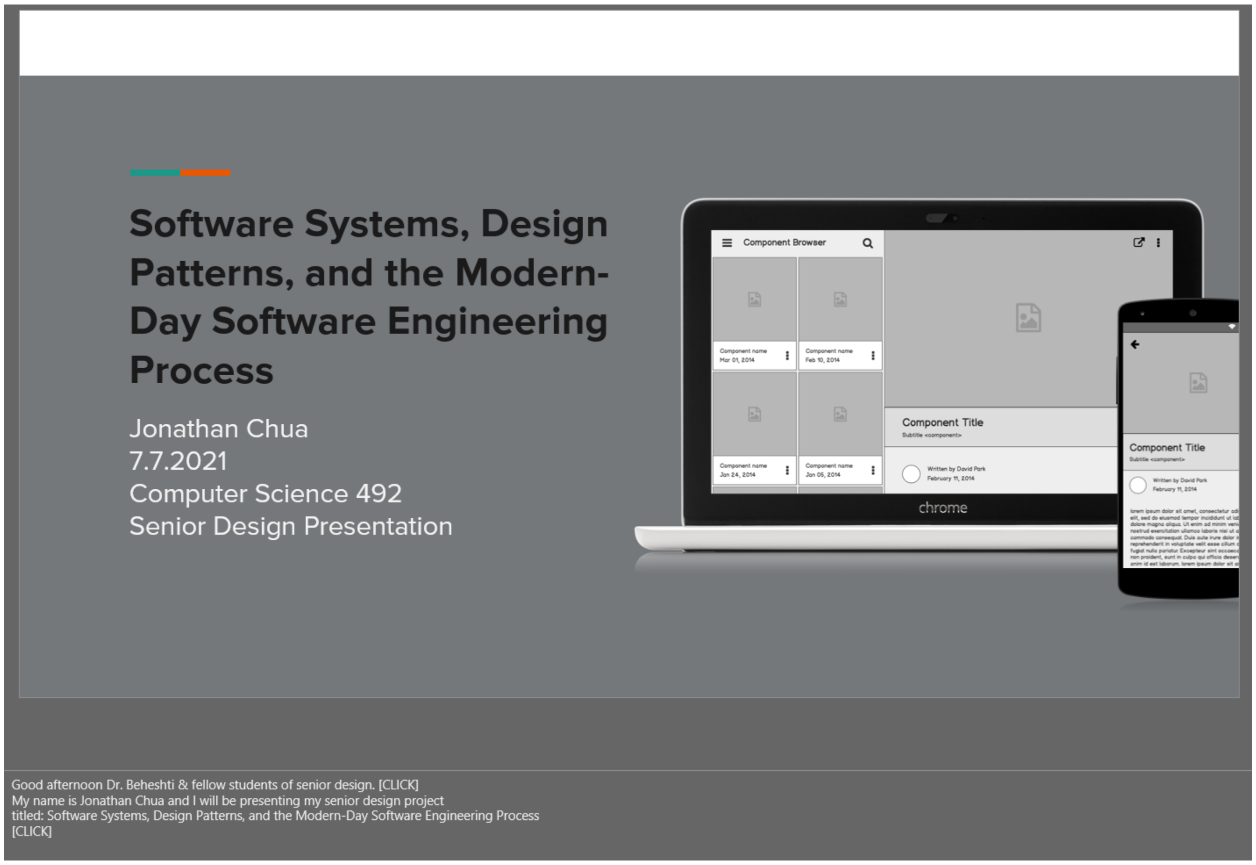
[4] E. Freeman and E. Robson, “The Observer Pattern,” in *Head First Design Patterns: Build Extensible and Maintainable Object-Oriented Software*, 2nd ed. Sebastopol, CA: O’Reilly, 2020 pp. 52-54

[5] S. Preibisch, “API Gateways.” in *API Development: A Practical Guide for Business Implementation Success*, New York City, NY: Apress, 2018 pp. 125-139

[6] M. Albarak and R. Bahsoon, "Prioritizing Technical Debt in Database Normalization Using Portfolio Theory and Data Quality Metrics," *2018 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 2018, pp. 31-40.

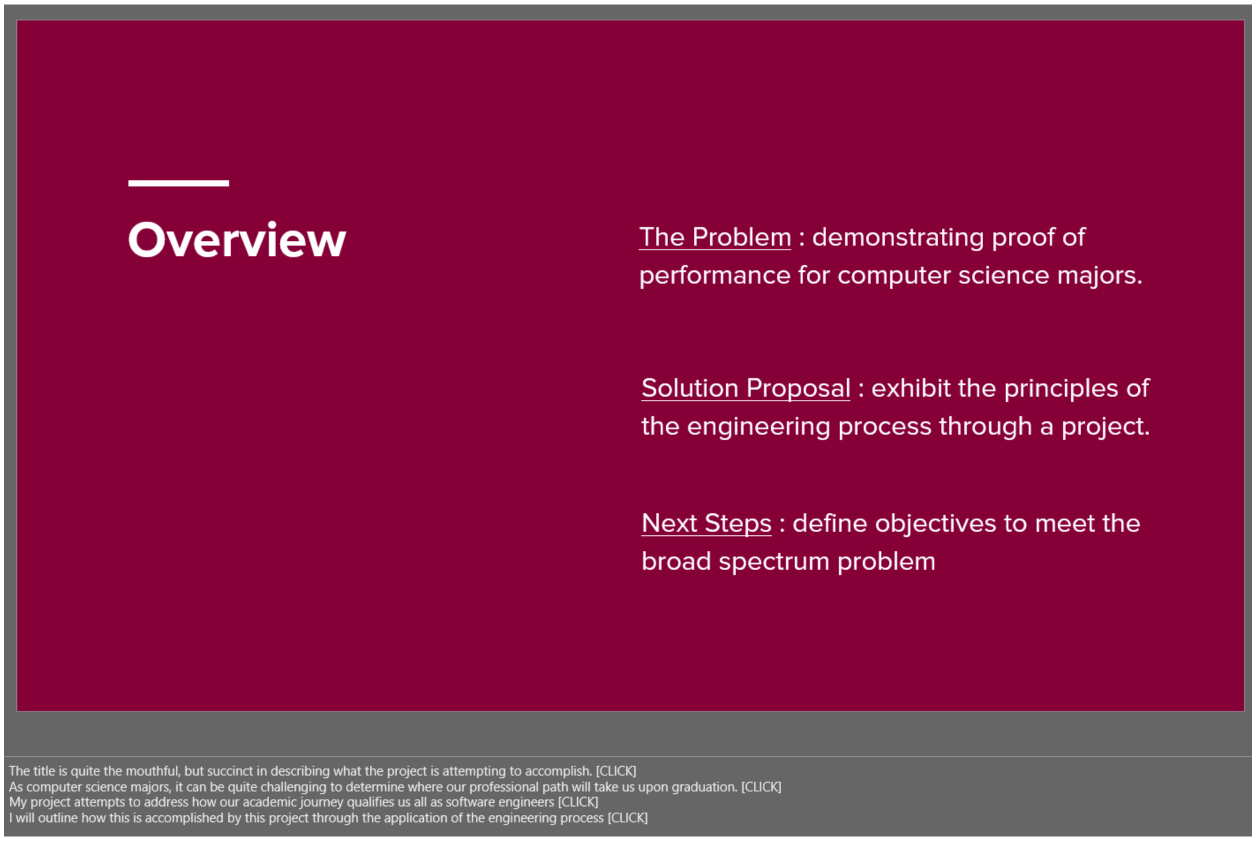
[7] E. Klotins *et al*., "A Progression Model of Software Engineering Goals, Challenges, and Practices in Start-Ups," in *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 498-521, 1 March 2021, doi: 10.1109/TSE.2019.2900213.

VIII. Appendix


Software Systems, Design Patterns, and the Modern-Day Software Engineering Process

Jonathan Chua
7.7.2021
Computer Science 492
Senior Design Presentation

Good afternoon Dr. Beheshti & fellow students of senior design. [CLICK]
My name is Jonathan Chua and I will be presenting my senior design project
titled: Software Systems, Design Patterns, and the Modern-Day Software Engineering Process
[CLICK]



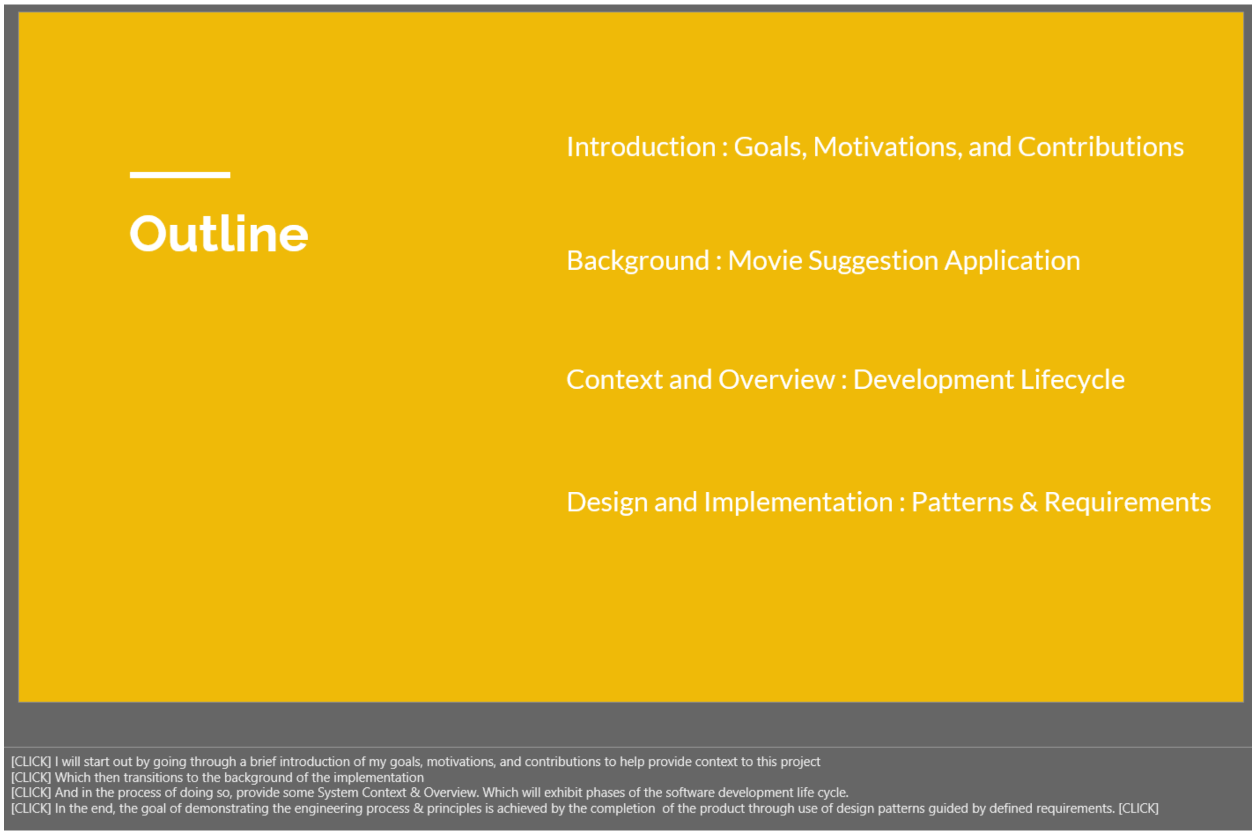
Overview

The Problem : demonstrating proof of performance for computer science majors.

Solution Proposal : exhibit the principles of the engineering process through a project.

Next Steps : define objectives to meet the broad spectrum problem

The title is quite the mouthful, but succinct in describing what the project is attempting to accomplish. [CLICK]
As computer science majors, it can be quite challenging to determine where our professional path will take us upon graduation. [CLICK]
My project attempts to address how our academic journey qualifies us all as software engineers [CLICK]
I will outline how this is accomplished by this project through the application of the engineering process [CLICK]



Outline

Introduction : Goals, Motivations, and Contributions

Background : Movie Suggestion Application

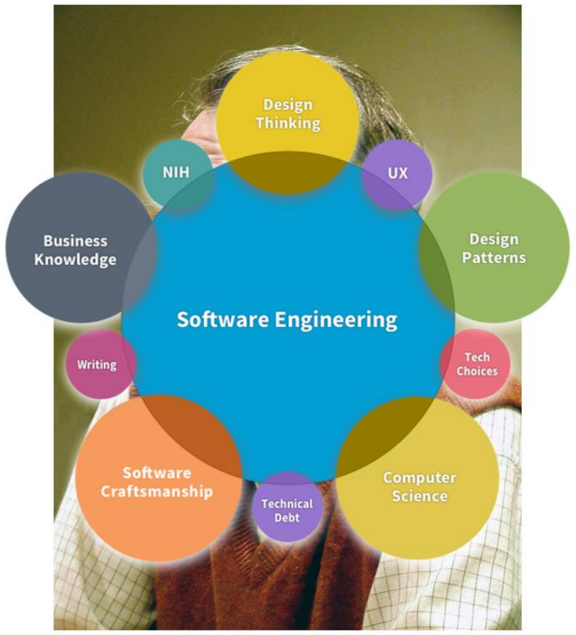
Context and Overview : Development Lifecycle

Design and Implementation : Patterns & Requirements

[CLICK] I will start out by going through a brief introduction of my goals, motivations, and contributions to help provide context to this project
[CLICK] Which then transitions to the background of the implementation
[CLICK] And in the process of doing so, provide some System Context & Overview. Which will exhibit phases of the software development life cycle.
[CLICK] In the end, the goal of demonstrating the engineering process & principles is achieved by the completion of the product through use of design patterns guided by defined requirements. [CLICK]

Introduction

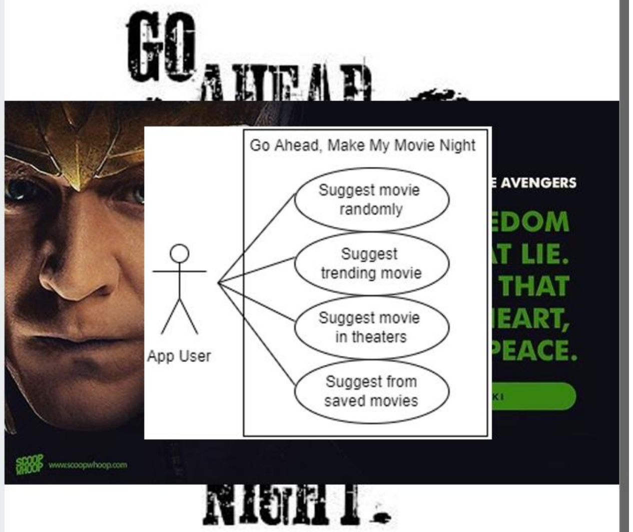
The Goals, Motivations, and Contributions of this senior design project



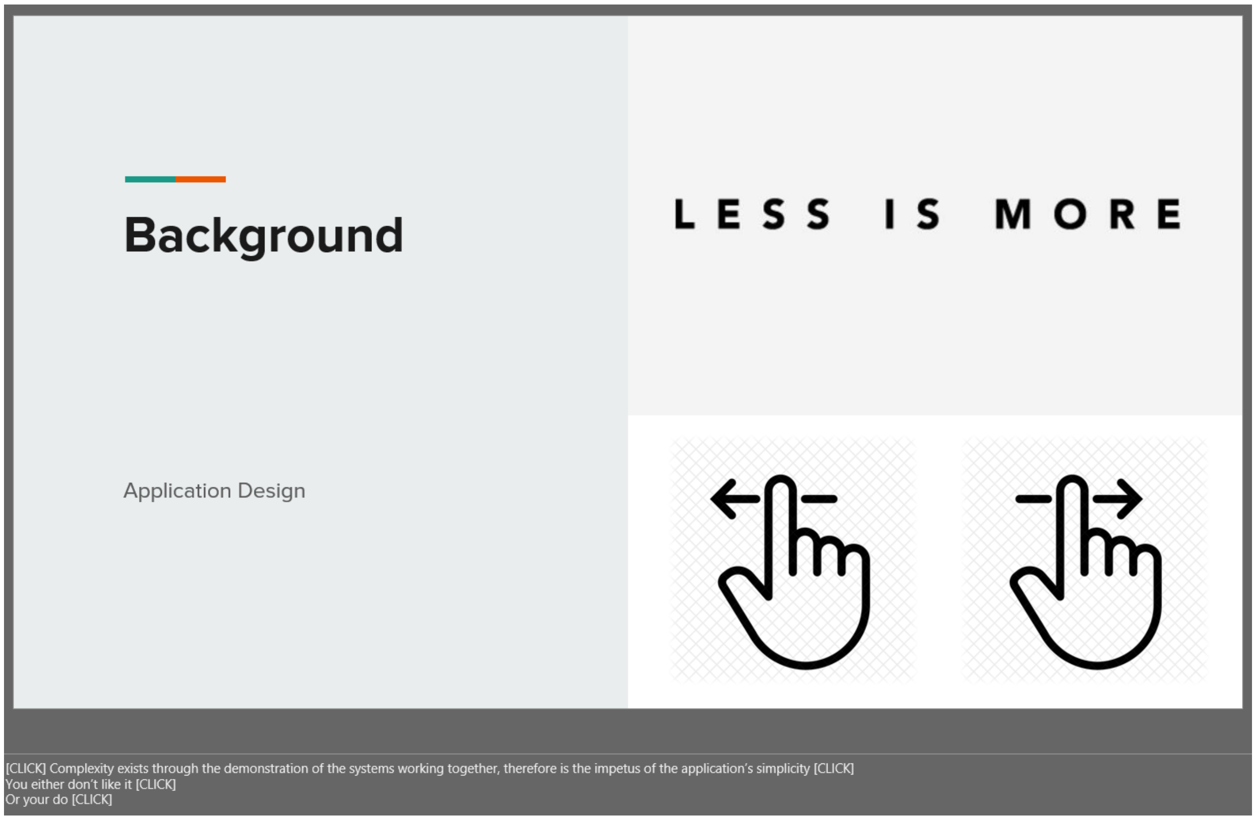
[CLICK]
Edsger Dijkstra, whom many of us computer science majors should know with a level of reverence, wrote a paper called "On the Cruelty of Really Teaching Computer Science." where he mentions how software engineering is a doomed discipline. He famously derided software engineering as "how to program, when you cannot."
[CLICK]
My response to that is with another popular engineering quote that goes, "anyone can build a bridge that stands; but it takes a civil engineer to make one that barely stands!" This statement, I feel best encapsulates how I view one who can program and a software engineer who programs. At the core of all engineering, is the principle of optimization. My goal is the employment of the software engineering process and doctrines, motivated by the reduction made by dijkstra, to demonstrate its necessity in today's evolving developer market. Therefore, my contribution will essentially be the entirety of the engineering process used to create the app. [CLICK]

Background

Application Overview


Go Ahead, Make My Movie Night
Suggest movie randomly
Suggest trending movie
Suggest movie in theaters
Suggest from saved movies
App User

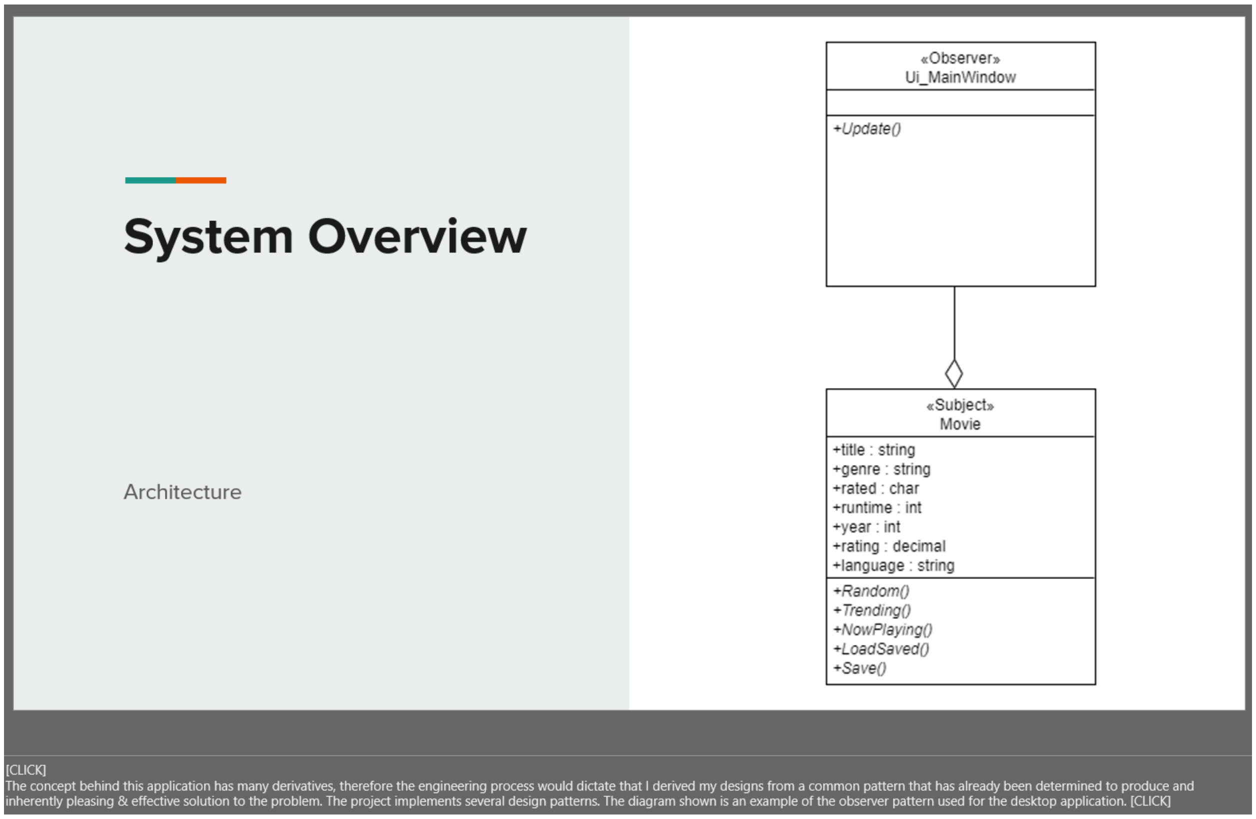
[CLICK] The desktop application, is a vehicle to demonstrate the software system that was built for this senior design project.
By design, the app is simple, both in how it works and the options the user is provided.
[CLICK]
The goal for the app is similar to Tinder, where a lot of data exists, but how it interfaces with the user is simplistic. You aren't shown a list of movies, just a movie.
[CLICK]
This is purposeful, the limit of choice and information is an attempt to keep the focus to the merits of just the suggestion, not amongst all possible suggestions nor all possible ways of generating suggestions. [CLICK]



Background

Application Design

[CLICK] Complexity exists through the demonstration of the systems working together, therefore is the impetus of the application's simplicity [CLICK]
You either don't like it [CLICK]
Or your do [CLICK]



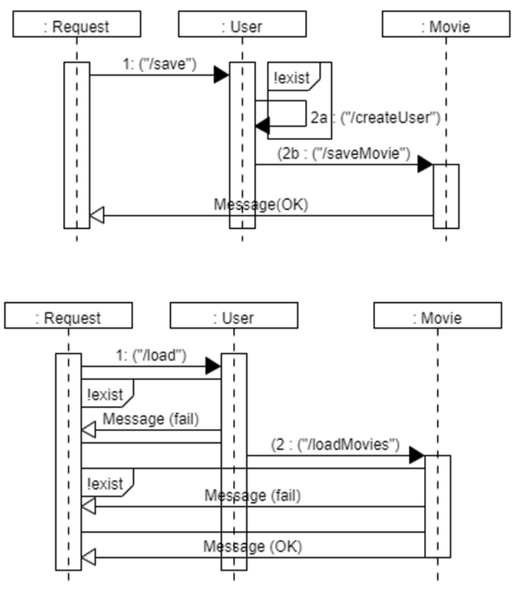
System Overview

Architecture

[CLICK]
The concept behind this application has many derivatives, therefore the engineering process would dictate that I derived my designs from a common pattern that has already been determined to produce and inherently pleasing & effective solution to the problem. The project implements several design patterns. The diagram shown is an example of the observer pattern used for the desktop application. [CLICK]

System Overview

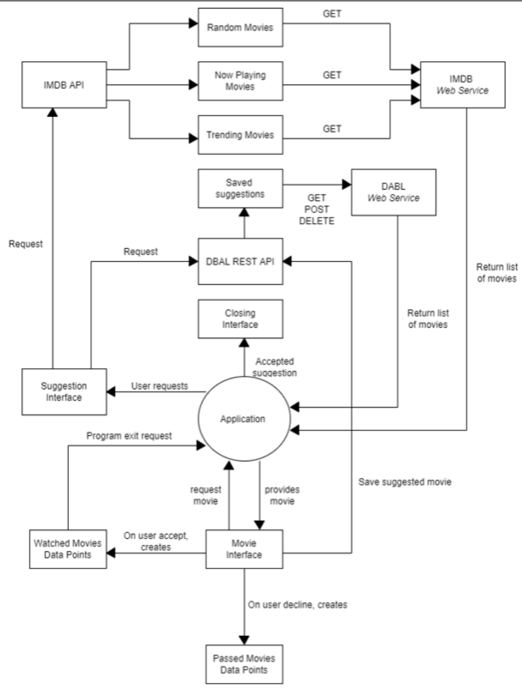
Characteristics



[CLICK]Generality is achieved by having a database abstraction layer in the form of a RESTful web service, which is used by the application through a REST API.[CLICK]
This design expands on the loose coupling approach and permits the application to be database-agnostic. That direction was guided by the agile principle of variability to preserve options. Other benefits include expanding distinction for requirements between the systems, resulting in more defined unit tests, and increasing security of the data by encapsulation of the sql system by virtue of abstraction. [CLICK]

System Context

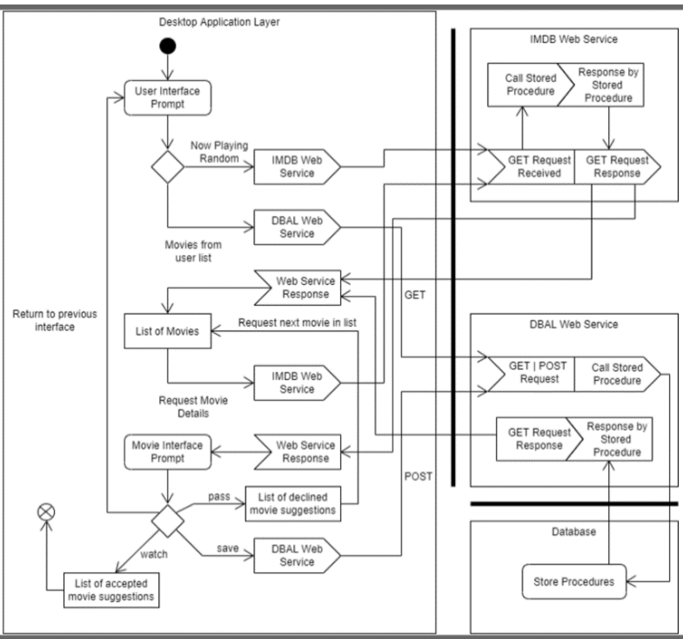
Context Diagram



Prominent British computer scientist, Sir Charles Anthony Richard Hoare stated, "the major achievement of modern science is to demonstrate the links between phenomena at different levels of abstraction and generality."

To a much lesser scale, my senior design project attempts to establish such links in every implementation of higher level concepts in terms of its lower primitive forms.[CLICK]

Thus, complexity of this project cannot be defined by the goal of the desktop application, which is to provide movie suggestions. Rather, it is the complexity of the software system developed to apply its underlying features, which can be seen by the context diagram. The application interacts with several systems through different environments to achieve fundamental doctrines of computer science and engineering. The high level context diagram, which was modeled from the requirements of each system, decomposes abstraction through use of points of views, generalization, coupling, and classification. [CLICK]



Activity Diagram

It has been said, "a picture is worth a thousand words." So I hope you forgive me in adding another thousand words to this presentation by virtue of another diagram [CLICK]

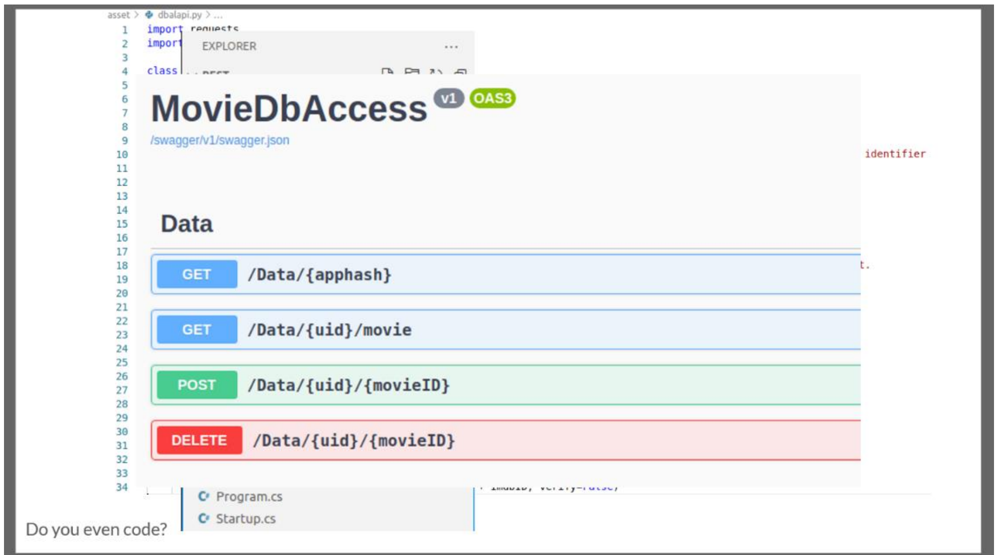
The process of establishing the decomposition of a problem into simpler and more understood primitives is basic to science and software engineering.

An abstraction is a model; the process of transformation between the context diagram to this activity diagram is part of the refinement process that helps split the problem into smaller abstractions.

The activity diagram attempts to link the abstract point of view of the context diagram to a more concrete observation.

Diagram conveys that the desktop application is an abstraction of the functions, data, and processes that facilitate the user experience. Thus, its requirements and goals are independent of the other systems that run-in conjunction with it. The primary import is the accuracy of the results that the desktop application enacts as a consumer of other functional abstractions within the software system, which is visualized by the vertical black bar as it communicates with the two database abstraction layers.

Similarly, the web services have their own set of requirements and goals that are independent of the database, e.g. scalability & asynchronous-ity, and thus only requires accuracy of results as a consumer of the functional abstractions that exist within the stored procedures of the database.



MovieDbAccess v1 OAS3

/swagger/v1/swagger.json

Data

GET /Data/{apphash}

GET /Data/{uid}/movie

POST /Data/{uid}/{movieID}

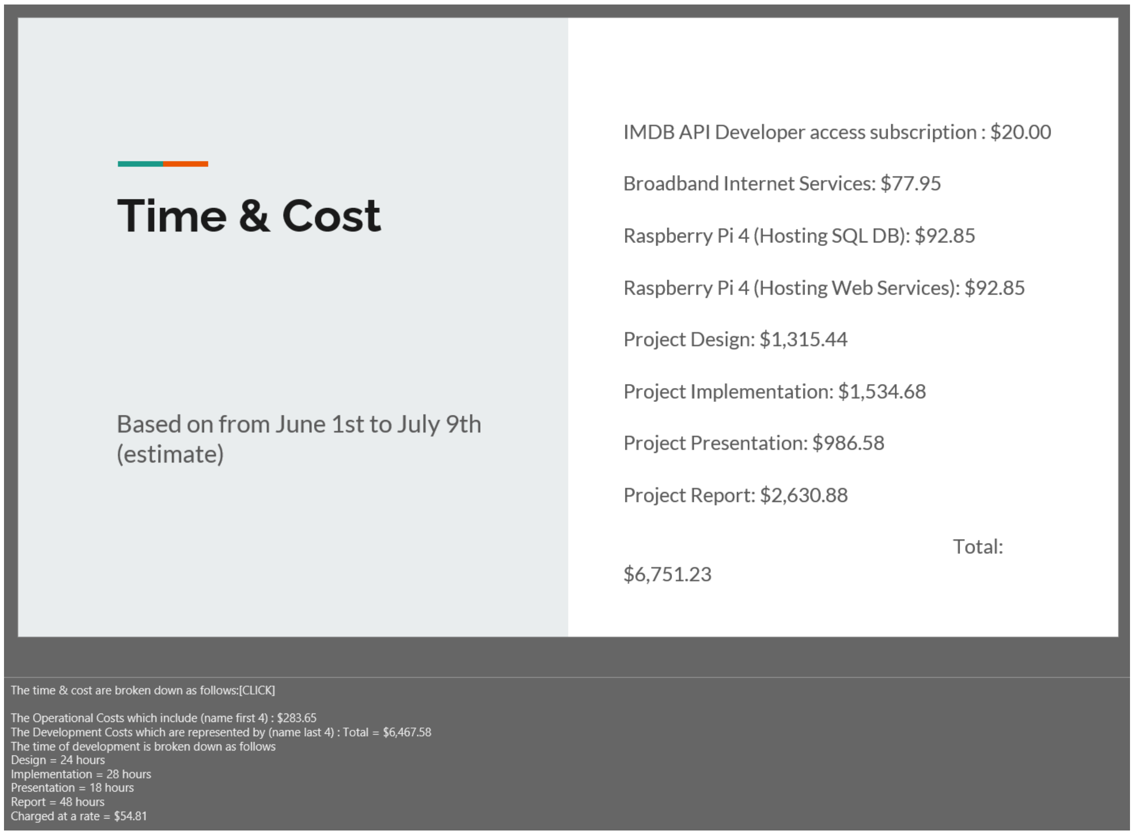
DELETE /Data/{uid}/{movieID}

Do you even code?

I will reserve this slide for results that won't be as evident during the demonstration portion of this presentation.

[CLICK]
I was successful with the integration of the IMDB web service to the desktop application by use of the IMDB REST API
[CLICK]
I was successful with the integration of the web service acting as the DBAL to the desktop application by use of the developed REST API
[CLICK]
The desktop application accomplishes its goals from the requirements set forth in the proposal and is comprised of 8 classes and 3 separate configuration files. There exists 2 additional configuration files that is a result of scaffolding for future features of the application.
[CLICK]
The RESTful web service runs as expected and successfully communicates with the Pi 4 Hosting the DB
[CLICK]
The REST API was developed using dapper and the dotnet 5.0 framework; it is comprised of a Controller that handles the routes, the Domains that represents the object models of the database, and the Services that are rendered when a request is made.

MySQL was used for the database. The database contains 2 tables with 6 stored procedures to obfuscate itself from any database abstraction layer that interacts with it.
[CLICK]



Time & Cost

Based on from June 1st to July 9th (estimate)

IMDB API Developer access subscription : $20.00

Broadband Internet Services: $77.95

Raspberry Pi 4 (Hosting SQL DB): $92.85

Raspberry Pi 4 (Hosting Web Services): $92.85

Project Design: $1,315.44

Project Implementation: $1,534.68

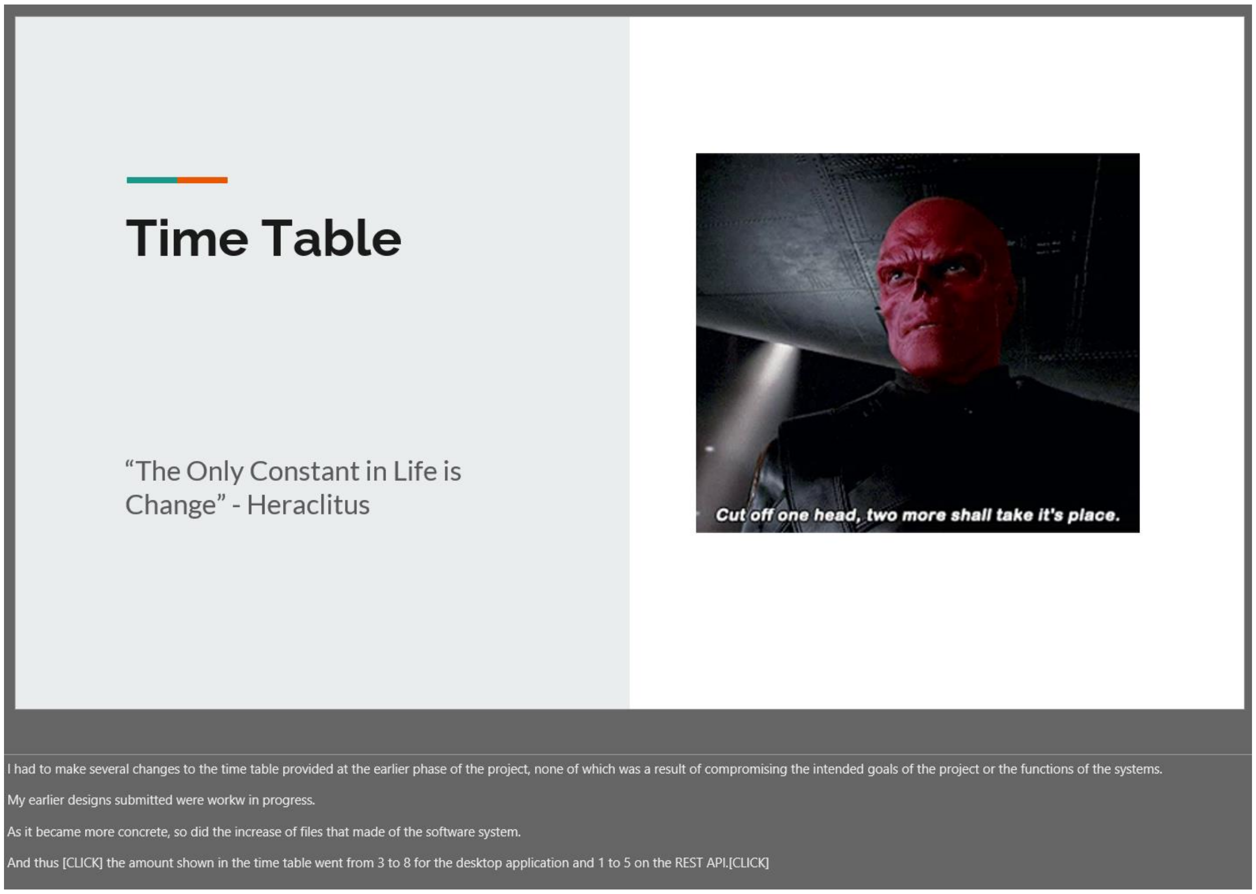
Project Presentation: $986.58

Project Report: $2,630.88

Total: $6,751.23

The time & cost are broken down as follows:[CLICK]

The Operational Costs which include (name first 4) : $283.65
The Development Costs which are represented by (name last 4) : Total = $6,467.58
The time of development is broken down as follows
Design = 24 hours
Implementation = 28 hours
Presentation = 18 hours
Report = 48 hours
Charged at a rate = $54.81



Time Table

“The Only Constant in Life is Change” - Heraclitus

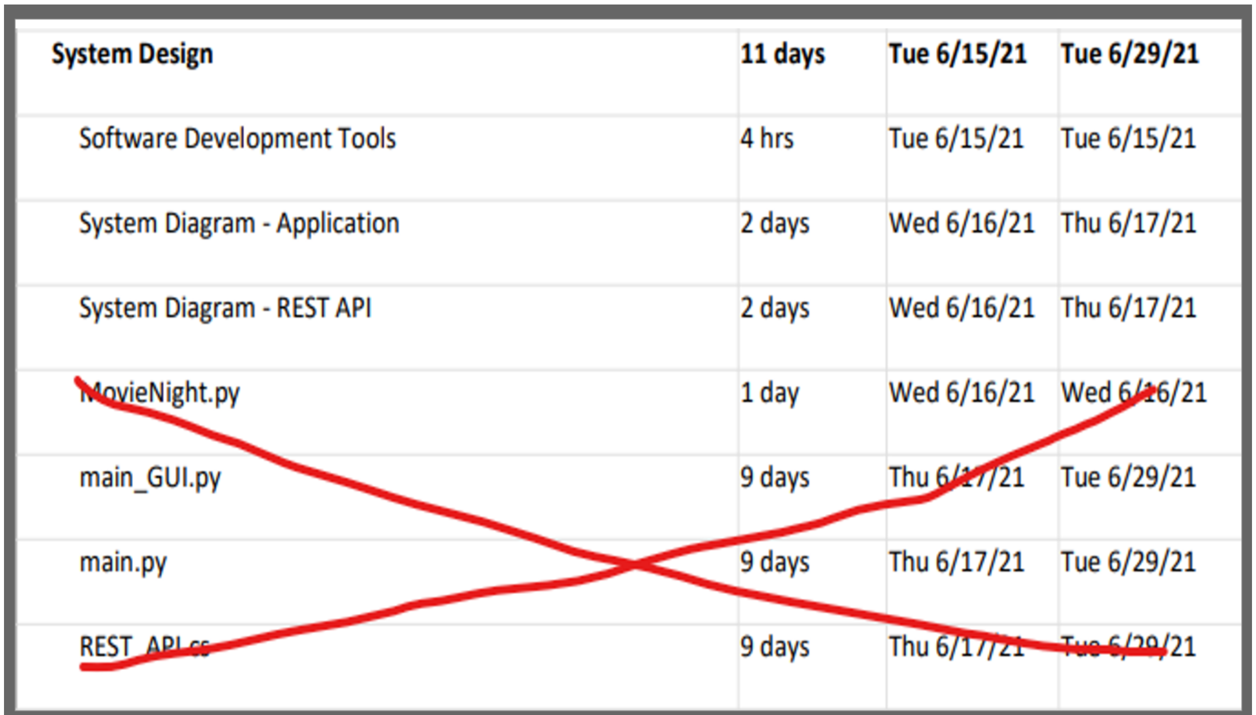
Cut off one head, two more shall take it's place.

I had to make several changes to the time table provided at the earlier phase of the project, none of which was a result of compromising the intended goals of the project or the functions of the systems.

My earlier designs submitted were workw in progress.

As it became more concrete, so did the increase of files that made of the software system.

And thus [CLICK] the amount shown in the time table went from 3 to 8 for the desktop application and 1 to 5 on the REST API.[CLICK]



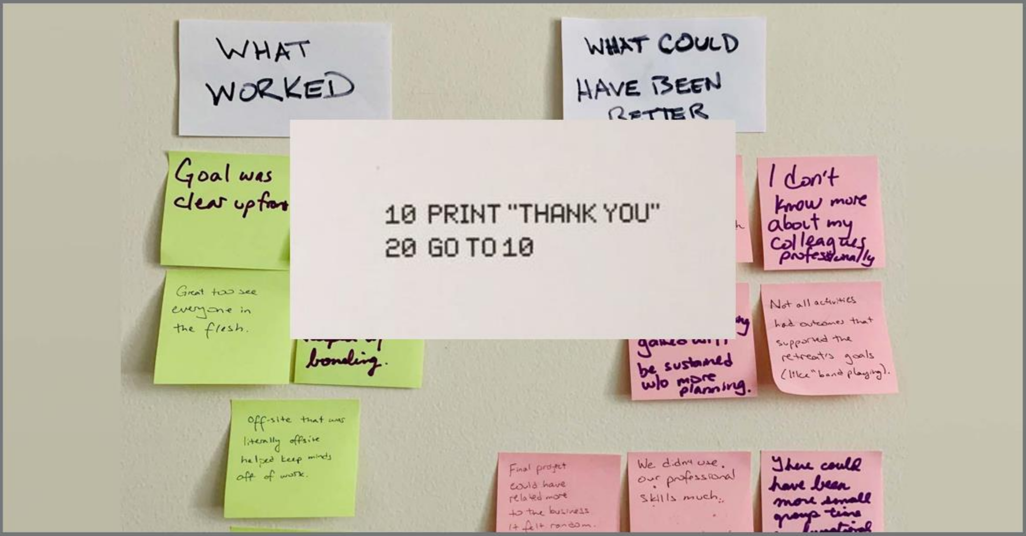
System Design	11 days	Tue 6/15/21	Tue 6/29/21
Software Development Tools	4 hrs	Tue 6/15/21	Tue 6/15/21
System Diagram - Application	2 days	Wed 6/16/21	Thu 6/17/21
System Diagram - REST API	2 days	Wed 6/16/21	Thu 6/17/21
~~MovieNight.py~~	~~1 day~~	~~Wed 6/16/21~~	~~Wed 6/16/21~~
~~main_GUI.py~~	~~9 days~~	~~Thu 6/17/21~~	~~Tue 6/29/21~~
~~main.py~~	~~9 days~~	~~Thu 6/17/21~~	~~Tue 6/29/21~~
~~REST_API.cs~~	~~9 days~~	~~Thu 6/17/21~~	~~Tue 6/29/21~~

Therefore, the time table submitted on June 14th is a deficient representation of the deliverables that will be submitted on July 9th. [CLICK]

To Summarize

The purpose of my senior design project was to demonstrate how Computer Science [CLICK] properly equips us to be Software Engineers.
[CLICK] This was not accomplished by submitting a program, [CLICK] but by following a process during the development of a product, [CLICK] and in that process following fundamental doctrines of science and engineering.

In Conclusion


WHAT WORKED
WHAT COULD HAVE BEEN BETTER
10 PRINT "THANK YOU"
20 GO TO 10
Goal was clear upfront
Great to see everyone in the flesh.
bonding.
Off-site that was literally offsite helped keep minds off of work.
I don't know more about my colleagues professionally
be sustained w/o more planning.
Not all activities had outcomes that supported the retreat's goals (like band playing).
Final project could have related more to the business it felt random.
We didn't use our professional skills much.
There could have been more small group time

In retrospect
[CLICK]
I would have created some separation in thought when jumping between code bases. In my attempt to follow the agile principle of working software being the primary measure of progress. I made it my priority to make sure each required feature was functional before moving to the next.

This led to a lot of stupid moments like forgetting how SQL is not case sensitive, by default. [CLICK] And failing to understand why this procedure kept returning True for the better part of 30 to 40 minutes.

The greatest takeaway from this senior design project was
[CLICK]
the amount of work it requires to properly, emphasis on properly, produce an engineered product in a professional manner. Hopefully, a measure of that was successfully displayed today in this final presentation.
[CLICK]

Code contributions are located in GitHub:

Senior Design App : chizuo/senior-design-app: CSC492 Senior Design Project - Application (github.com)
Senior Design Rest API : chizuo/senior-design-api: CSC492 Senior Design Project - REST API (github.com)

SQL Scripts Created

```
DELIMITER $$

CREATE PROCEDURE CheckUser(
   IN app_hash VARCHAR(255),
     OUT result BOOLEAN
)
BEGIN
   SET result = (EXISTS(SELECT * FROM User WHERE apphash =
app_hash));
END $$

DELIMITER ;



DELIMITER $$

CREATE PROCEDURE GetUser(
   IN appHash VARCHAR(255),
     OUT uid INT
)
BEGIN
   SELECT uid
     FROM User
     WHERE apphash = appHash;
END$$

DELIMITER ;
DELIMITER $$

CREATE PROCEDURE NewUser(
   IN appHash VARCHAR(255)
)
BEGIN
   INSERT INTO User(apphash)
    VALUES(appHash);
END$$

DELIMITER ;



DELIMITER $$

CREATE PROCEDURE GetMovie
   IN user_id VARCHAR(255)
)
BEGIN
   SELECT imdb_id FROM Movie
   WHERE uid = user_id
   ORDER BY gid
   LIMIT 1;
END$$

DELIMITER ;
DELIMITER $$

CREATE PROCEDURE NewMovie(
   IN user_id INT,
   IN imdb VARCHAR(255)
)
BEGIN
   INSERT INTO Movie(uid, imdb_id)
   VALUES (user_id, imdb);
END$$

DELIMITER ;
```

```
DELIMITER $$

CREATE PROCEDURE DeleteMovie(
   IN user_id INT,
   IN imdb VARCHAR(255)
)
BEGIN
   DELETE FROM Movie
   WHERE uid = user AND imdb_id = imdb;
END$$

DELIMITER ;
```