**CS673 Software Engineering** 

**TV Bums - BUMTV**

**Software Design Document**

| **Team Member** | **Role(s)** | **Signature** | **Date** |
| --- | --- | --- | --- |
| Trevor Cardoza | Design and Implementation Leader/Backup Leader | *Trevor Cardoza* | 9/11/2023 |
| Zhe Huang | QA Leader/Backup Leader | [Zhe Huang](mailto:wsuncle@bu.edu) | 9/11/2023 |
| David Mulvihill | Configuration Management Lead/Backup Leader | *David Mulvihill* | 09/11/2023 |
| Mahim Choudhury | Security Leader/ Backup Leader | *Mahim Choudhury* | 9/22/2023 |
| Delaney Sullivan | Requirements Leader | [Delaney Sullivan](mailto:sullid@bu.edu) | 9/24/2023 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 0.1 | Team | 09/13/2023 | Initial Release |
| 1.0 | Team | 09/25/2023 | Version 1.0 of the application has been deployed to AWS |
| 1.01 | Team | 10/09/2023 | Version 1.01 has updated class diagram, db schema |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Class Diagram](#_heading=h.1fob9te)

[UI Design (if applicable)](#_heading=h.3znysh7)

[Database Design (if applicable)](#_heading=h.2et92p0)

[Security Design](#_heading=h.tyjcwt)

[Business Logic and/or Key Algorithms](#_heading=h.3dy6vkm)

[Design Patterns](#_heading=h.1t3h5sf)

[Any Additional Topics you would like to include.](#_heading=h.4d34og8)

[References](#_heading=h.2s8eyo1)

[Glossary](#_heading=h.17dp8vu)

# Introduction

The BUMTV Web Application is a sophisticated movie and TV show tracker web application that adheres to industry-standard design procedures and best practices. This Software Design Document outlines the architectural and design considerations that underpin the development of BUMTV, emphasizing modularity, security, and quality. This document serves as a guiding compass for developers, designers, and stakeholders, offering insights into the architectural components, database schema, user interfaces, and security measures that constitute the foundation of the BUMTV project.

With time and cost limitation in mind, here are some key design goals that we have set:

Maintainability:

AWS Lambda and RDS Service: We ensure maintainability by leveraging AWS Lambda for serverless execution and Amazon RDS for managed database services. This allows us to focus on code updates without worrying about server maintenance.

Modularity:

MVC Structure: BUMTV follows a modular design pattern by implementing the Model-View-Controller (MVC) architecture, separating concerns for improved maintainability.

Docker Containerization: We employ Docker containers to encapsulate application components, enhancing modularity and portability across environments.

Security:

Data Encryption: All sensitive data is encrypted to protect user information during transmission and storage.

Backup & Recovery: Automatic database backups on AWS ensure data integrity and security.

Authentication: Robust authentication mechanisms are implemented to safeguard user accounts and access control.

Database Injection Prevention: Measures such as prepared statements and input validation are in place to prevent SQL injection attacks.

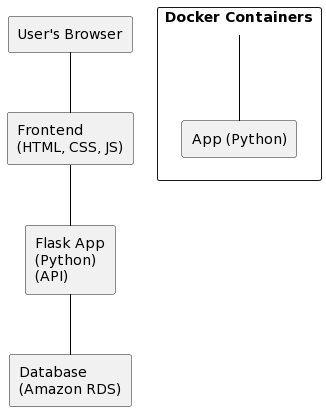
Static Application Security Testing (SAST): Regular SAST scans are conducted during the development process to proactively identify and mitigate security vulnerabilities.

Quality & Reliability:

Coding Standards: Adherence to coding standards and best practices ensures code quality and readability.

Testing: Rigorous testing, including unit tests, integration tests, and end-to-end tests, is performed to validate functionality and maintain high software quality and reliability.

# Software Architecture



Frontend Module

* Description: The frontend module handles the presentation layer of the BUMTV web application.
* Interfaces: Exposes a user interface for interaction with the application.
* Dependencies: Communicates with the Backend API for data retrieval and updates.

Backend API

* Description: The Backend API serves as the core of the application, handling business logic, data retrieval, and user authentication.
* Interfaces: Provides RESTful APIs for user account management, movie and TV show tracking, and streaming information retrieval.
* Dependencies: Relies on the Database Service for data storage and retrieval.

Database Service (Amazon RDS)

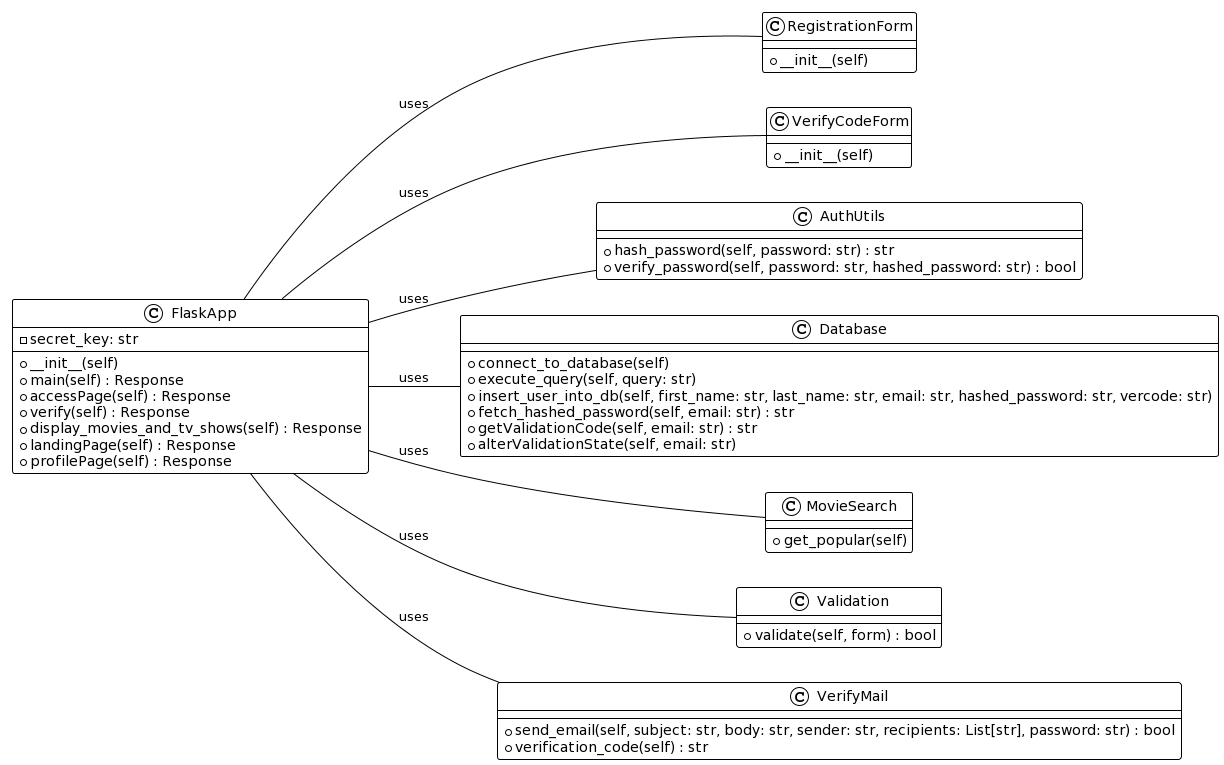
* Description: Amazon RDS is used for managing the database to store user accounts, tracked content, and streaming information.
* Interfaces: Provides a SQL interface for CRUD operations on user data and content tracking.
* Dependencies: None.

Docker Containers

* Description: Docker containers encapsulate individual components for improved modularity and portability.
* Interfaces: Containerized components interact via APIs and HTTP requests.
* Dependencies: None.

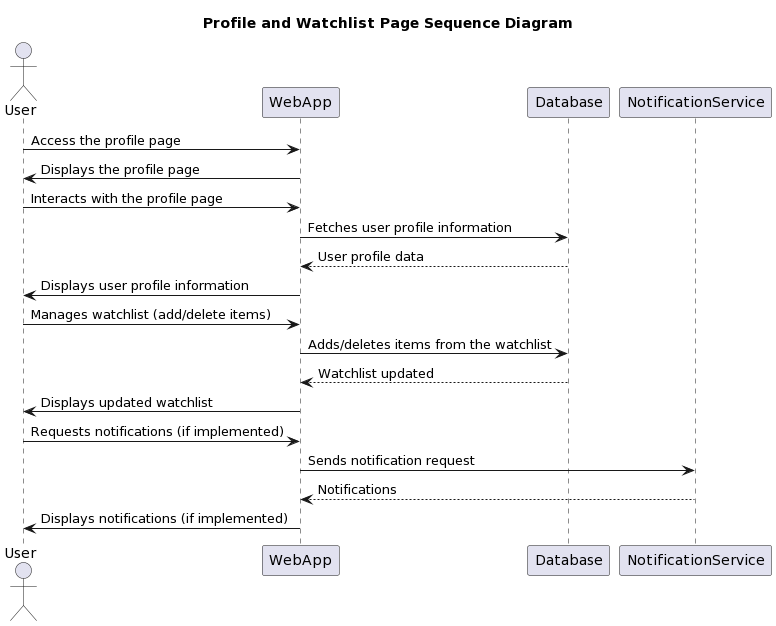
# Class Diagram

In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.

**General Class Diagram:**

**Profile Page:**

**Work in progress**

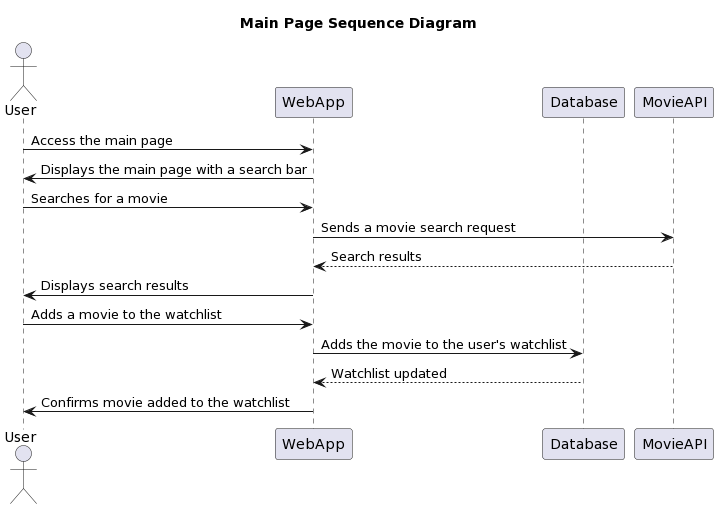
****

Watchlist movies will have the following information:

* Title
* Concatinated description
* Where it is showing
* Add/Remove from watchlist button
* STRETCH:
  + Link to IMDB page
  + Top reviews

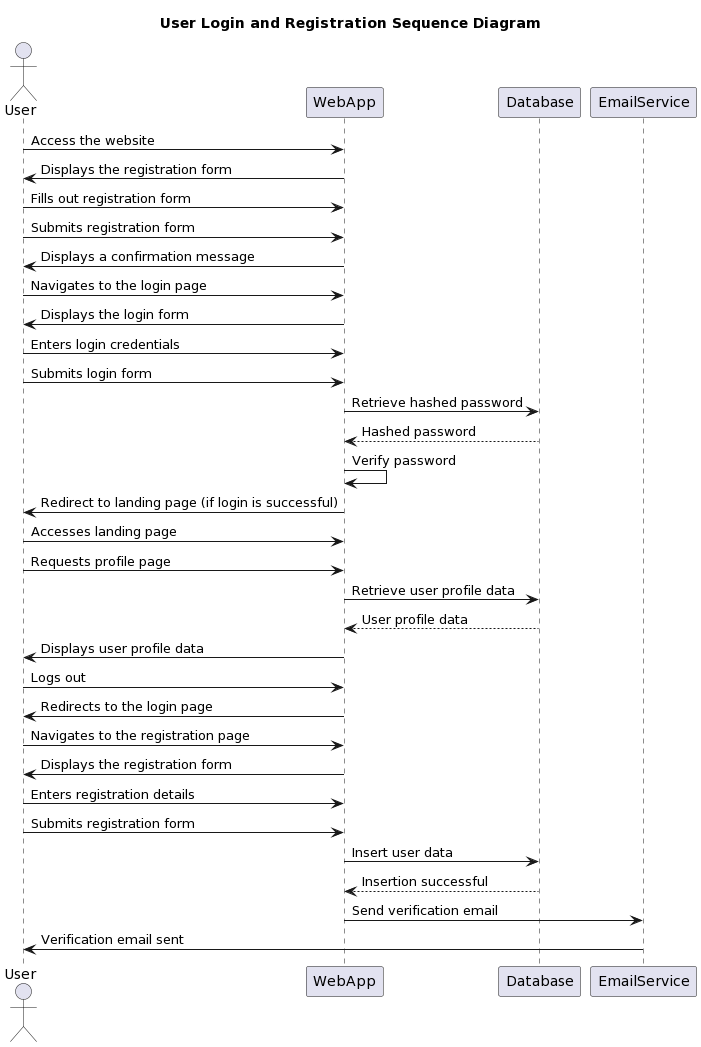
Profile will have the following information:

* Name
* Email
* STRETCH:
  + Phone Number
  + Way to change Password

**Main Page:**

Each movie displayed will have the following information:

* Title
* Description
* Where it is showing
* Add/Remove from watchlist button
* STRETCH:
  + Link to IMDB page
  + Top reviews

**Login:**

# Database Design

The application will be using a Aurora-MySQL RDS Serverless V2 cluster. This cluster will consist of the following AWS resources:

* RDS cluster
* RDS cluster instance
* RDS subnet group
* RDS parameter group
* secrets for RDS credentials
* security group
* KMS key for data encryption
* The following is the current Terraform code block for the RDS:

###################

## KMS Resources ##

###################

resource "aws\_kms\_key" "flask\_app\_db\_kms" {

description = "flask app db kms"

deletion\_window\_in\_days = 10

}

resource "aws\_kms\_alias" "flask\_app\_db\_kms\_alias" {

name = "alias/my-key-alias"

target\_key\_id = aws\_kms\_key.flask\_app\_db\_kms.key\_id

}

########################

## RDS Resources ##

########################

resource "aws\_rds\_cluster" "flask\_app\_db\_cluster" {

cluster\_identifier = "flask-app-db-cluster"

db\_subnet\_group\_name = aws\_db\_subnet\_group.flask\_app\_subnet\_group.name

engine = "aurora-mysql"

engine\_mode = "provisioned"

engine\_version = "8.0.mysql\_aurora.3.02.0"

database\_name = "flask\_app\_db"

master\_username = "tvbum\_admin"

master\_password = "od9KN7pOhEV32oz"

vpc\_security\_group\_ids = [aws\_security\_group.flask\_app\_sg.id]

serverlessv2\_scaling\_configuration {

max\_capacity = 1.0

min\_capacity = 0.5

}

}

resource "aws\_rds\_cluster\_instance" "flask\_app\_db" {

cluster\_identifier = aws\_rds\_cluster.flask\_app\_db\_cluster.id

db\_subnet\_group\_name = aws\_db\_subnet\_group.flask\_app\_subnet\_group.name

identifier = "flask-app-rds-cluster"

instance\_class = "db.serverless"

engine = aws\_rds\_cluster.flask\_app\_db\_cluster.engine

engine\_version = aws\_rds\_cluster.flask\_app\_db\_cluster.engine\_version

publicly\_accessible = true

}

resource "aws\_db\_subnet\_group" "flask\_app\_subnet\_group" {

name = "flask-app-subnet-group"

subnet\_ids = [aws\_subnet.flask\_app\_subnet\_1a.id, aws\_subnet.flask\_app\_subnet\_1b.id, aws\_subnet.flask\_app\_subnet\_1c.id, ]

tags = {

Name = "flask-app-subnet-group"

}

}

resource "aws\_db\_parameter\_group" "flask\_app\_pg" {

name = "rds-pg"

family = "mysql8.0"

parameter {

name = "character\_set\_server"

value = "utf8"

}

parameter {

name = "character\_set\_client"

value = "utf8"

}

}

###############################

## Secrets Manager Resources ##

###############################

resource "aws\_secretsmanager\_secret" "flask\_app\_db\_user" {

name = "flask-app-db-user"

}

resource "aws\_secretsmanager\_secret" "flask\_app\_db\_pass" {

name = "flask-app-db-pass"

}

##############################

## Security Group Resources ##

##############################

resource "aws\_security\_group" "flask\_app\_sg" {

description = "Security group for flask app rds"

name = "flask-app-sg"

vpc\_id = aws\_vpc.flask\_app\_vpc.id

}

* The following is the database schema from MySQL Workbench:
* **Tables:**

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(32) NOT NULL,

last\_name VARCHAR(32),

email VARCHAR(64) NOT NULL UNIQUE,

hashed\_password VARCHAR(128) NOT NULL,

);

CREATE TABLE movies\_shows (

mov\_show\_id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);

CREATE TABLE reviews (

review\_id INT AUTO\_INCREMENT PRIMARY KEY,

mov\_show\_id INT NOT NULL,

FOREIGN KEY (mov\_show\_id) REFERENCES movies\_shows(mov\_show\_id)

);

CREATE TABLE locations (

locations\_id INT AUTO\_INCREMENT PRIMARY KEY,

mov\_show\_id INT NOT NULL,

FOREIGN KEY (mov\_show\_id) REFERENCES movies\_shows(mov\_show\_id)

);

CREATE TABLE shares (

share\_id INT AUTO\_INCREMENT PRIMARY KEY,

review\_id INT NOT NULL,

FOREIGN KEY (review\_id) REFERENCES reviews(review\_id)

);

CREATE TABLE watch\_list (

watch\_list\_id INT AUTO\_INCREMENT PRIMARY KEY,

mov\_show\_id INT NOT NULL,

FOREIGN KEY (mov\_show\_id) REFERENCES movies\_shows(mov\_show\_id)

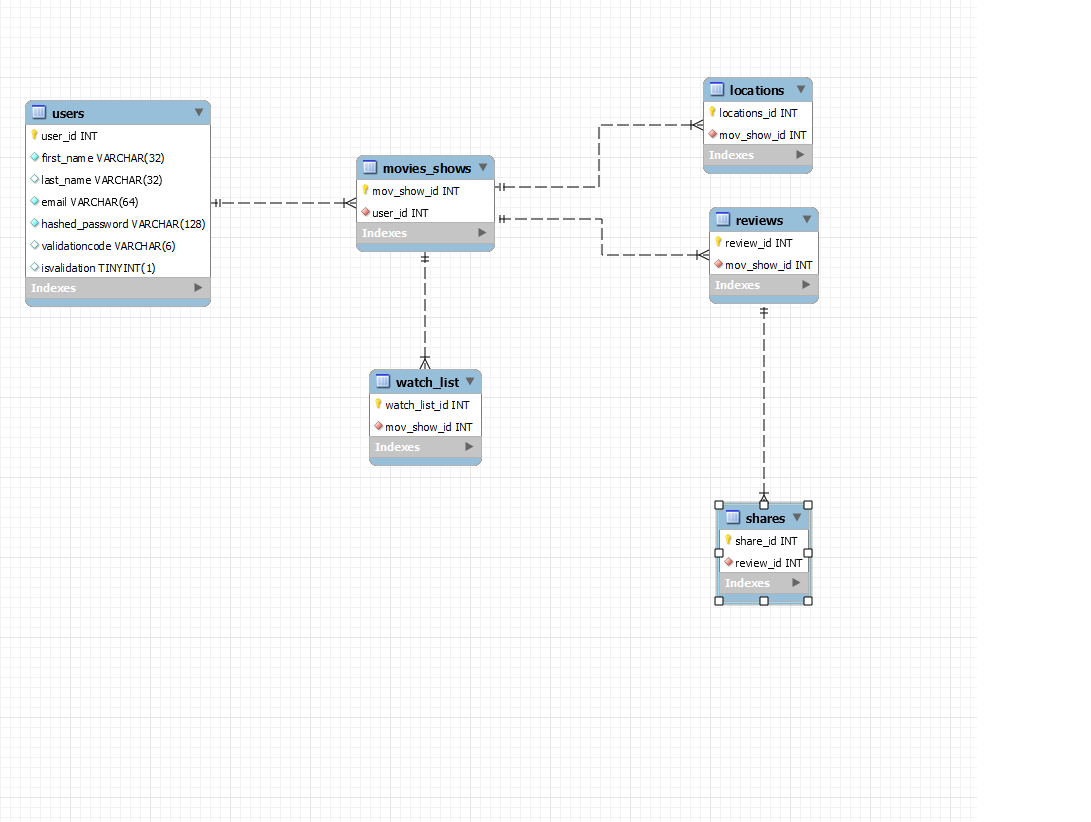
)

ALTER TABLE users

ADD COLUMN validationcode VARCHAR(6) AFTER hashed\_password;

ALTER TABLE users

ADD COLUMN isvalidation TINYINT(1) AFTER validationcode;

* **DB relationships:**

# Security Design

With limited time and cost in mind, we will implement the following security design principles into our design to ensure comprehensive security to our BUMTV.

**Data Encryption:**

Security Principle: The use of industry-standard encryption protocols like HTTPS/TLS for data in transit aligns with the principle of keeping security features as simple as possible (Economy of Mechanism). These protocols are well-established and provide a straightforward yet effective means of securing data during transmission. Not to mention, user passwords will also be hashed before they are stored securely.

**Static Application Security Testing (SAST) Scanning:**

Security Principle: The integration of SAST scanning into the development process follows the principle that security should not depend on the secrecy of its design or implementation (Open Design). It embraces transparency by proactively identifying and addressing security issues in the source code.

**Network & Web Security:**

Security Principle: The proactive approach to designing the application with security in mind (Security by Design) demonstrates a commitment to addressing potential vulnerabilities before they become threats. Frequent code reviews, following coding standards to ensure we can avoid web vulnerabilities.

**Backup and Recovery:**

Security Principle: We will be using aws rds as our choice of database to store user data. As part of the aws service, they are committed to do regular backups of the db. The practice of performing regular backups and securely storing them in separate, encrypted formats aligns with the modularity principle (Modularity). Backups are treated as separate components, ensuring that they can be restored independently

# Business Logic and/or Key Algorithms

Business logic is being handled in the business logic folder which hold the API calls and the notifications.

# Design Patterns

We decided upon Modal View Controller Scheme

# Any Additional Topics you would like to include.

N/A

# References

* Terraform Registry - AWS

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

* ECS Documentation - AWS

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html>

# Glossary

N/A