**CS673 Software Engineering** 

**TV-Bums: BUMTV**

**Project Proposal and Planning**

| **Team Member** | **Role(s)** | **Signature** | **Date** |
| --- | --- | --- | --- |
| Mahim Choudhury | Security/Backup Leader |  | 9/9/2023 |
| Zhe Huang | QA Leader/Backup Leader | *Zhe Huang* | 9/10/2023 |
| Delaney Sullivan | Requirements Leader/Backup Leader | [Delaney Sullivan](mailto:sullid@bu.edu) | 9/11/2023 |
| Trevor Cardoza | Design and Implementation Leader/Backup Leader | *Trevor Cardoza* | 9/11/2023 |
| David Mulvihill | Configuration Management Lead/Backup Leader | *David Mulvihill* | 09/11/2023 |

**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 includes search functionality and basic implementation of CSS |
|  |  |  |  |

[Overview](#_heading=h.gjdgxs)

[Related Work](#_heading=h.30j0zll)

[Proposed High level Requirements](#_heading=h.1fob9te)

[Management Plan](#_heading=h.3znysh7)

[Objectives and Priorities](#_heading=h.2et92p0)

[Risk Management (need to be updated constantly)](#_heading=h.tyjcwt)

[Timeline (need to be updated at the end of each iteration)](#_heading=h.3dy6vkm)

[Configuration Management Plan](#_heading=h.1t3h5sf)

[Tools](#_heading=h.4d34og8)

[Deployment Plan if applicable](#_heading=h.2s8eyo1)

[Quality Assurance Plan](#_heading=h.17dp8vu)

[Metrics](#_heading=h.3rdcrjn)

[Code Review Process](#_heading=h.26in1rg)

[Testing](#_heading=h.lnxbz9)

[Defect Management](#_heading=h.35nkun2)

[References](#_heading=h.1ksv4uv)

[Glossary](#_heading=h.44sinio)

# Overview

BUMTV is a modern web application tailored to elevate the way users track, explore, and enjoy their favorite movies and TV shows. With the increasing number of OTT platforms, it is getting harder and harder to keep track of movies and TV shows that are released. This is why the TV-Bums came up with a game changing, yet simple to use idea called BUMTV. Built with a secure and user-friendly approach, BUMTV offers a comprehensive set of features to help users seamlessly manage their entertainment choices.

BUMTV is designed to cater to a diverse user base, including movie enthusiasts, streamers, casual viewers, and security-conscious individuals. Movie enthusiasts could rely on the application to access comprehensive information and reviews, streamers and renters can use it to discover where to watch content, casual viewers would enjoy its user-friendly interface, and security-conscious users will trust its robust authentication and data protection measures.

The core functionality of BUMTV revolves around secure user registration and login, seamless search and information retrieval, optional "Where to Watch" features, and watchlist management. Users can create accounts securely, search for movies and TV shows, access detailed data from trusted sources, discover streaming options, and keep their custom watchlists.

The web app will be built with the following components:

* frontend - JS (React or VUE Framework)
* Backend - Flask as the backend

In addition, the TV-Bums are determined to use robust software engineering processes during the build process. The application will be deployed to AWS, and the team intends to use the following tools:

* Jira
* GitHub
* GitHub Actions for CI/CD
* SAST scanner
* Docker
* AWS Document DB

# Related Work

During our research for similar apps online, we came across several similar apps that aim to provide more or less the same features. Movie Tracker and Trakt were a few apps that looked impressive in terms of their appearance and functionality.

While these apps share common objectives, we anticipate that BUMTV will offer distinct advantages that set it apart in the following areas:

* Subscription: Unlike some competing apps that may require a subscription or payment for access to premium features, BUMTV is committed to delivering a rich and user-friendly experience at no cost to the user.
* User experience and interface: BUMTV places a strong emphasis on simple, easy to use, and secure user experience. We aim to provide a seamless and visually appealing platform that ensures users can easily navigate and explore their favorite content. Ensuring a consistent, secure, and pleasant user experience is one of our top priorities.

# Proposed High level Requirements

* 1. Functional Requirements  
     (For each functional requirement, please give a feature title and a brief description using the following format: As (a role), I want to (action), so that (value).)
     1. Essential Features (the core features that you definitely need to finish):

(For each essential features, please give a rough estimation in terms of person hours or an range of person hours)

1. User Watch List: As a user, I want to be able to track shows I’ve watched through a watch list, so that I can see my historical information. (Person hours: 3-5)
2. User Account Creation: As a user, I want to be able to create an account, so that I can login and use BUMTV. (Person hours: 2)
3. Website Account Security: As a website, I want someone to be able to access their account only, so all information stays secured. (Person hours: 3)
4. Review Access: As a user, I want to be able to access reviews, so I can collect information on shows and movies. (Person hours: 4)
5. Data Updates: As a website administrator, I want to be able to add data, so that the site stays relevant and useful. (Person hours: 5)
6. User Search: As a user, I want to be able to search over all shows/ movies, so that I can find ones to watch. (Person hours: 4-7)
7. User Account Usage: As a user, I want to be able to login to BUMTV, so that I can view my personal account. (Person hours: 1)

Desirable Features (the nice features that you really want to have too):

1. Where to Watch: As a user, I want to be able to find where to watch a show or movie, so that I know where I can watch it. (Person hours: 6)
2. Notifications: Notifications on change of movie or TV Show information like streaming service, movie theater showings, etc (Person Hours 3-5)
   * 1. Optional Features (additional cool features that you want to have if there is time):
        1. User Rating Sharing: As a user, I want to be able to rate shows/ movies and share those with friends, so they can see my thoughts. (Person hours: 8)
   1. Nonfunctional Requirements
      1. Security requirements

* Authentication Security: As a user, I want to register/login in a system which uses strong authentication methods to protect my accounts and sensitive data(Person hours: 2-4)
* Secure Communication (SSL): As a user, I want to use a system which uses secure data transmission through the HTTPS protocol(Person hours 2)
* Static Application Security Testing (SAST): As a Security Leader, I want to integrate SAST scans into the development process so that proactively identify and address security vulnerabilities in the source code efficiently(Person hours 1-3)
* Database SQL Injection Prevention: As a Security Leader, I want to employ measures like prepared statements, parameterized queries, and input validation to prevent SQL injection attacks and maintain the security of the database(Person Hours 4)

# Management Plan

## Objectives and Priorities

(Please describe your project objectives with highest priority first. Project Goals can include but not limited to complete all proposed (essential) features, deploy the software successfully, the software has no known bugs, maintain high quality, etc )

Here is a high-level overview of our project objectives and priorities:

* Produce an MVP (minimal viable product) with the following features:
  + User account creation
  + User creation of watch lists
  + User creation of reviews
  + User sharing of reviews
  + Search functionality
* Deploy the application to Amazon ECS
  + Create Terraform files for IAC
  + Create/configure pipelines via GitHub Actions
  + Successfully deploy the application
* Address any security concerns.
  + Scan for security vulnerabilities
  + Implement KMS for data resources on AWS
  + Ensure connection is using TLS on 443
* Address any bugs that arise through the development process.
  + Create testing protocol
  + Design procedures for remediation of any bugs or security concerns

## Risk Management (need to be updated constantly)

One risk we identified was loss of team members, which has already occurred, so our team, which once had six members, is now down to five. Given that the timeframe for the project is very short, any loss of members can be catastrophic. On the topic of personnel, another risk we face is that team members are in different timezones, so scheduling meetings is a challenge. Another risk we identified is technical competence in that some team members may have to get up to speed on certain tools that will be used extensively, such as git.

Link to our risk management sheet:

<https://docs.google.com/spreadsheets/d/1mKJ1JkKP03J9T1klAxRonZBbWTAKLQQy/edit#gid=274500621>

**Risk Management Sheet Link:**

## Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration)

| **Iteration** | **Functional Requirements (Essential/Desirable/Optional)** | **Tasks (Cross requirements tasks)** | **Estimated/real person hours** |
| --- | --- | --- | --- |
| **1** | User Watch List (in progress) | A user should be able to track shows they have watched through a watch list and see their historical information | 6-8 hours |
| User Account Creation (completed) | A user should be able to create an account, so that they can login and use BUMTV. | 8 hours |
| Website Account Security (in progress) | As a website, someone should be able to access their account only, so all information stays secured. | 8-10 hours |
| **2** | Review Access (not started) | A should be able to access reviews, so they can collect information on shows and movies. | 6 hours |
| Data Updates (in progress) | A website administrator should be able to add data, so that the site stays relevant and useful. | 8 hours |
| User Search (not started) | A user should be able to search over all shows/ movies, so that they can find ones to watch. | 10-12 hours |
| **3** | Where to Watch (in progress) | A user should be able to find where to watch a show or movie, so that they know where they can watch it. | 6 hours |
| Notifications (not started) | Notifications on change of movie or TV Show information like streaming service, movie theater showings, etc. | 8-10 hours |
| User Rating Sharing (not started) | A user should be able to rate shows/ movies and share those with friends, so they can see their thoughts. | 10-12 hours |

# Configuration Management Plan

## Tools

* GitHub will be used as the code and documentation repository.
* Git will be used for version control.
* Individual team members will use their IDE of choice:
  + Trevor Cardoza - Visual Studio Code
  + Mahim Choudhury - Visual Studio Code
  + Zhe Huang - Visual Studio Code
  + David Mulvihill - Visual Studio Code
  + Delaney Sullivan - Visual Studio Code
* Please also specify any other tools to be used, e.g. IDE tools, CI/CD tools, container tools, SAST or DAST tools, and any other DevOps tools)
  + SAST: BANDIT
  + Github actions for CICD
  + Docker for containerization
  + AWS for deployment
  1. Code Commit Guideline and Git Branching Strategy
     1. For our branching strategy, we will be adopting a feature-branch workflow in which development will take place on feature branches off of the main branch.
     2. When a feature branch is ready to be merged with the main branch, another team member will review the work, and if approved, a pull/merge will be completed.
     3. One shortcoming to this strategy is that our version of GitHub does not allow us to enforce approvals of pulls/merges.

## Deployment Plan

* Our application will be a web app, and will be deployed to Amazon Web Services.
* AWS Services to be used:
  + Elastic Container Service (ECS)
  + Elastic Container Registry (ECR)
  + Application Load Balancer (ALB)
  + Virtual Private Cloud (VPC)
  + RDS - Aurora Serverless V2, MySQL engine
  + Secrets Manager
  + Amazon Certificate Manager (ACM)
  + Key Management Service (KMS)
  + Route-53 for DNS Management
* AWS resources will be deployed via IAC using Terraform.
* CI/CD will be implemented via GitHub Actions.
  + We currently have two pipelines running:
    - Terraform Plan
    - Terraform Apply

# Quality Assurance Plan

## Metrics

| Metric Name | Description |
| --- | --- |
| LOC | We have no requirement of LOC |
| # of Files | No requirement of number of files |
| # of Classes | Quantify the total number of classes used |
| # of methods | Quantify the total number of methods/functions used. |
| cyclomatic complexity | Measure the complexity of the codebase, identifying areas that may need refactoring. |
| Defect rate (defects / KLOC) | The Defects per class |
| # of defects | Track the number of defects and how it resolved |
| # test cases | The test cases number per function/ class |
| Test case pass rate | The number that code runs well in that test case |
| Test coverage | The rate that the Test cases cover the Stories |
| Person hours used | No requirement on this |
| # of user stories completed | The number of the stories that the project can meet |

* 1. Coding Standard

1. **Naming Conventions**: Guidelines for naming variables, functions, classes, and other code elements. This helps make the code self-explanatory and easier to understand.
2. **Comments**: At least add a Comment for each Function, make a brief description of this function.
3. **Nesting:** To keep code readable, if three or more nesting features exist, please use a function to reduce nesting features.
4. **Test cases:** Add some test cases that you think errors will occur in this part of code.

## Code Review Process

1. Code writers should review their code before submitting.
2. peer review before a change will be merged to the main branch.
3. Checklist:
4. Code can implement level functions
5. Meet the minimum comments level.
6. Use words as variable names
7. No excessive nesting

## Testing

Types of testing:

1. **Unit Testing**: This involves testing individual components or units of code in isolation to ensure that they function correctly. It is usually the first level of testing and is automated.
2. **Integration Testing:** Integration testing checks how different units or components work together when integrated into larger portions of the software. It verifies that these components can communicate and function together as expected.
3. **Functional Testing:** This type of testing evaluates whether the software's functions and features work according to the specified requirements. Test cases are designed to validate the system's behavior against expected outcomes.
4. **Regression Testing:** After making changes or updates to the software, regression testing is performed to ensure that new code changes have not introduced new defects and that existing functionality remains intact.
5. **Manual Testing**: Manual testing is a type of software testing where testers execute test cases without the use of automation tools or scripts.

Testing tools and framework:

1. **Unit Testing:** use python “unittest” module
2. **Integration Testing:** test by QA leader
3. **Functional Testing:** test by Every Code programmers
4. **Regression Testing:** Pytest Framework.
5. **Manual testing:** using web browsers.

Personnel Responsibility:

1. **Developers:** Unit testing,Functional Testing
2. **QA Leader: Integration Testing,Regression Testing,Manual testing**

Testing objectives:

1. **Unit Testing:** Ensure that individual units or components of a software application perform correctly and meet their design specifications.
2. **Integration Testing:** ensure that individual software components, such as modules, functions, or objects, work correctly when combined or integrated with each other within a larger software system.
3. **Functional Testing:** Verify that a software application or system functions correctly according to its specified functional requirements.
4. **Regression Testing:** ensure that changes, updates, or enhancements made to a software application or system do not introduce new defects or break existing functionality that was previously working correctly.
5. **Manual testing:** ensure the quality and reliability of a software application by identifying defects, verifying that it meets its requirements, and assessing its overall usability.

## Defect Management

Using the word.

# References

(For more details, please refer to the encounter example in the book or the software version of the documents posted on blackboard. )

# Glossary

(Any acronym used in the document should be explained here)

* IAC - Infrastructure as code refers to the process of deploying cloud infrastructure via CloudFormation and/or Terraform.