**APPLICATION DESIGN SPECIFICATION**

VERSION 2.0

October 26, 2024

**UP-TO-DATE (UTD)**

(News Aggregator Web Application)

**SUBMITTED TO**

Professor Gananand Kini

**SUBMITTED BY**

Sumiran Jaiswal

UMD Directory ID: sumi0309

UMD Email ID: [sumi0309@umd.edu](mailto:sumi0309@umd.edu)

Submitted in partial fulfillment of the requirements of ENPM680

(Introduction to Secure coding for Software Engineering)

**TABLE OF CONTENTS**

Table of Contents…………………………………………………...….…...i

List of Figures…………………………………………………...…….…...ii

Section 1 – Application Description……………………………........1

1.1. Project………………….…………………………………………….......1

1.2. Description………………………………………………………………1

1.3. Revision History…………….………………………………………...…1

Section 2 – Application Overview…………………………….…….…2

2.1 Purpose……………….…………………………………………………...2

2.2 Scope…………………….………………………………………………..2

2.3 Requirements…………………………………………………………….. 3

2.3.1 Estimates………………...……………………….……………….…..5

2.3.2 Traceability Matrix…………….……………………………………...5

Section 3 – Application System Architecture………………………6

3.1 System Architecture…………………..…………………………...……...6

3.2 MVC process flow …………………...…………………………...………7

3.3 Components and their interaction………………………………...………8

Section 4 – Application Software Component…………………...10

4.1 Software components…………….……………………………………...10

4.2 Testing……………...…………….……………………………………...12

Section 5 – Application Software UI Components……………..14

Section 6 – Application Data Component…………………….........17

Section 7 – Application Security Design..........................................19

Section 8 – References…………………………………………..………26

Section 9 – Index…….…………………………………………..………27

**LIST OF FIGURES**

Figure 1 – System Architecture………………..…........6

Figure 2 – MVC Architecture…………………..…......7 Figure 3 – System Interaction Diagram…………….…8

Figure 4 – Activity Diagram...……………...………..17

Figure 5 – Secure System Architecture........………..25

**SECTION 1 – APPLICATION DESCRIPTION**

**1.1 Project**

UP-TO-DATE

(UTD – a news aggregator web application)

**1.2 Description**

UTD (Up To Date) is a web application designed to keep users informed by aggregating the latest news in their region. Users can interact with the system in three roles:

1. Authors, who create and post news
2. Readers, who browse and search for news articles
3. Contributors, who suggest edits to improve content.

The application ensures users stay up to date by providing accurate and relevant news with features like search functionality and real-time updates.

|  |  |  |
| --- | --- | --- |
| Date | Comment | Author |
| 10-26-2024 | Version 2.0 – Updated to meet security requirements | Sumiran Jaiswal |
| 09-19-2024 | Version 1.0 completed | Sumiran Jaiswal |
|  |  |  |

**1.3 Revision History**

**SECTION 2 – APPLICATION OVERVIEW**

**2.1 Purpose**

The purpose of this document is to provide a detailed design overview of the UTD (Up To Date) web application. This document focuses on the design architecture, module breakdown, and interaction between the components of the system, including user interface components, class and entities, web pages, and database management. The intended audience includes developers, system architects, and stakeholders involved in the project, ensuring that the design is clearly understood for development and implementation purposes.

**2.2 Scope**

The scope of this document is to outline the design specifications for the web application. It details the key modules to be developed, including user interface components, class structures, application pages, and database management. The design will show how Authors, Readers, and Contributors interact with the system, as well as how data is managed. To illustrate these elements, UML diagrams will be used to represent system interactions, and wireframes will be included to show the layout and navigation of the user interface, where needed. This document ensures a clear understanding of the system architecture necessary for successful development and implementation.

**2.3 Requirements**

The UTD (Up To Date) web application will be developed using the following technology stack:

* Technology Stack: The application will be built using ASP.NET Core, a powerful framework for developing web applications. C# will be used as the primary programming language for developing the backend logic of the application. The frontend will be designed using razor pages in ASP.NET and will be coded in CSHTML.
* MVC Framework: The application will follow the Model-View-Controller (MVC) design pattern. This pattern separates the application into three main components:
  + Model: Manages the behavior and data.
  + View: Handles the presentation layer, displaying data to the user.
  + Controller: Handles page events and navigation between pages. Controllers will handle all the HTTP requests and deliver the response.

The MVC design pattern helps to enforce separation of concerns to help you avoid mixing presentation logic, business logic, and data access logic together.

* Database Management: Microsoft SQL Server will be used for managing the application's database. It will store and handle data related to news articles, suggestions, user information like username, password, email etc., all the Identity related data like Roles, Users etc. The application’s database is set up using an Object-Relational Mapper (ORM). For this project, we will be using Entity Framework, Microsoft’s ORM solution. Entity Framework allows for seamless interaction between the application’s objects (models) and the underlying database. It simplifies data management by automatically mapping objects in the code to the relational database, allowing the application to perform CRUD (Create, Read, Update, Delete) operations with ease. For user account management the application will utilize the Identity framework provided by ASP.NET.

ASP.NET Core Identity will:

* Supports user interface (UI) login functionality using its API.
* Manage users, passwords, profile data, roles, claims, tokens, email confirmation, and more.
* Programming Languages: The development will involve several programming languages and technologies:
  + C#: For server-side logic and application development.
  + JavaScript: For client-side interactions and dynamic content.
  + HTML/CSS: For structuring and styling the web pages.
  + CSHTML: For creating dynamic web pages that combine HTML and C# code.

Functional requirements, including features such as the login page, news feed component, news adding interface, and search functionality, are specified in the updated SRS document. This ensures that the design aligns with the application’s intended functionality and user needs.

**2.3.1 Estimates**

|  |  |  |
| --- | --- | --- |
| # | Description | Hrs. Est. |
| 1 | Login page | 3 |
| 2 | News feed component | 6 |
| 3 | News adding interface | 4 |
| 4 | News Search component | 3 |
| 5 | Database designing | 1 |
| 6 | Miscellaneous | 5 |
| 7 | Suggestion Component | 5 |
| 8 | Articles Page | 6 |
| 9 | Total | 33 |

**2.3.2 Traceability Matrix**

|  |  |
| --- | --- |
| **SRS Requirement** | **SDD Module** |
| |  | | --- | | FR1 |  |  | | --- | |  | | |  | | --- | | Add news view |  |  | | --- | |  | |
| |  | | --- | | FR2 |  |  | | --- | |  | | |  | | --- | | News Controller |  |  | | --- | |  | |
| |  | | --- | | FR3 |  |  | | --- | |  | | |  | | --- | |  |   News Controller   |  | | --- | |  | |
| |  | | --- | | FR4 |  |  | | --- | |  | | News Controller   |  | | --- | | News Controller | |
| |  | | --- | | FR5 |  |  | | --- | |  | | News Feed View |
| FR6 | News Controller |
| FR7 | News Feed View |
| FR8 | News Controller |
| FR9 | News Controller |
| FR10 | Suggestions Controller |
| FR11 | Articles Controller |

**SECTION 3 – APPLICATION SYSTEM ARCHITECTURE**

**3.1 System architecture**

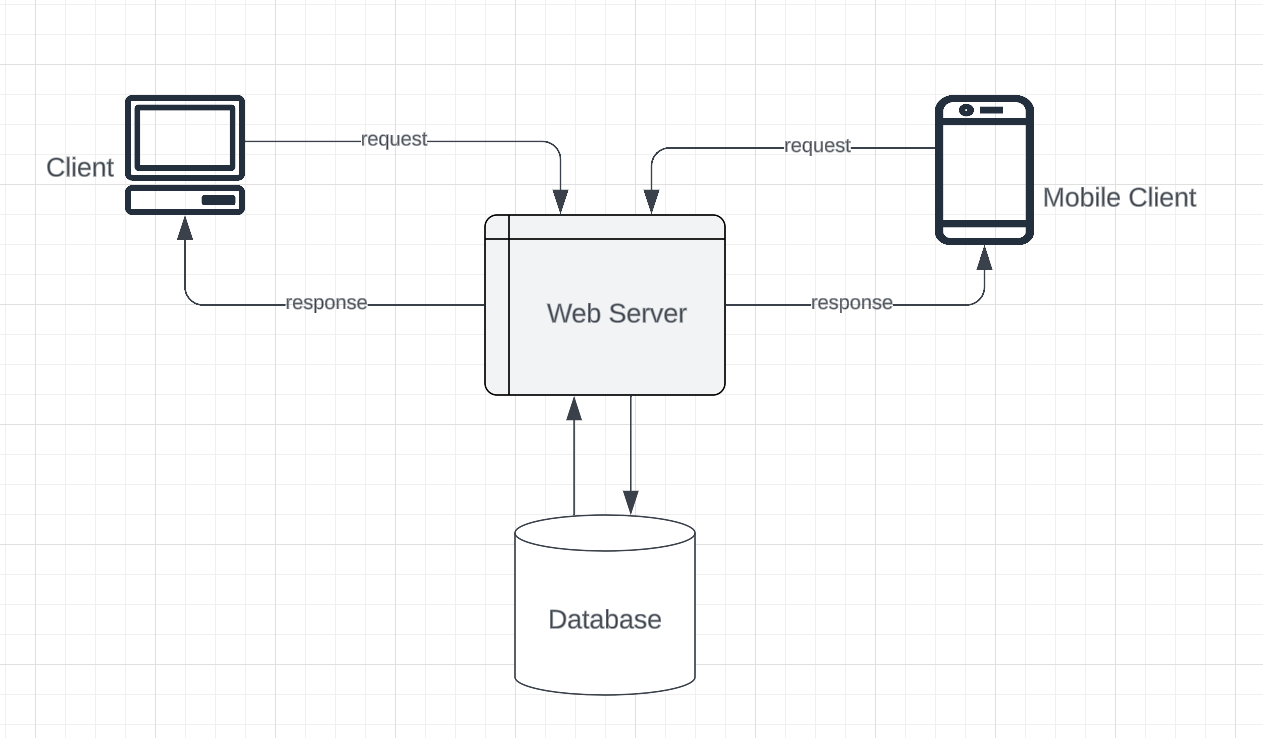
****

Fig. 1 System Architecture

**3.2 MVC process flow**

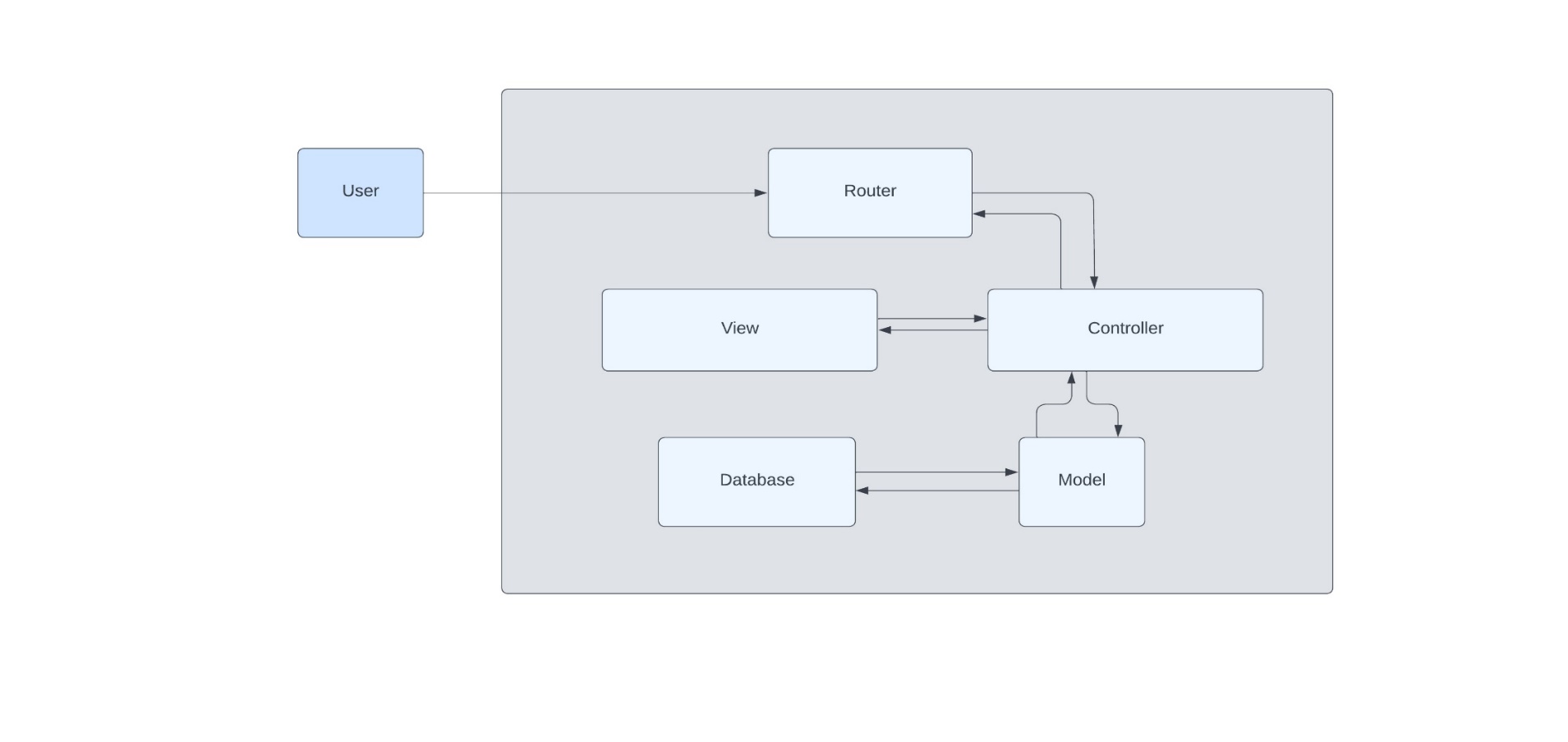
Below is the MVC system general architecture diagram, which represents how communication takes place and the relationships between all the components of the MVC architecture.

Fig. 2 MVC Architecture

Communication Flow:

1. User Interaction: The user interacts with the application through the View. This interaction triggers an input or request.
2. Router: The request is sent to the Router, which determines how to process the request.
3. Controller: The Router directs the request to the appropriate Controller. The Controller processes the request and interacts with the Model to retrieve or manipulate data.
4. Model: The Model performs the necessary operations on the data and retrieves information from the database.
5. Response: The data retrieved by the Model is sent back to the Controller.
6. View Creation: The Controller then uses this data to create or update the View.
7. Presentation: The updated View is sent back to the user, displaying the relevant information.

This flow ensures that user requests are handled efficiently, with data processed and presented in a structured manner. The separation of concerns provided by the MVC architecture improves the application’s maintainability and scalability, allowing for easier updates and modifications to individual components.

**3.3 Components and their interaction**

Below is a diagram that showcases how the components of the application interact with each other.

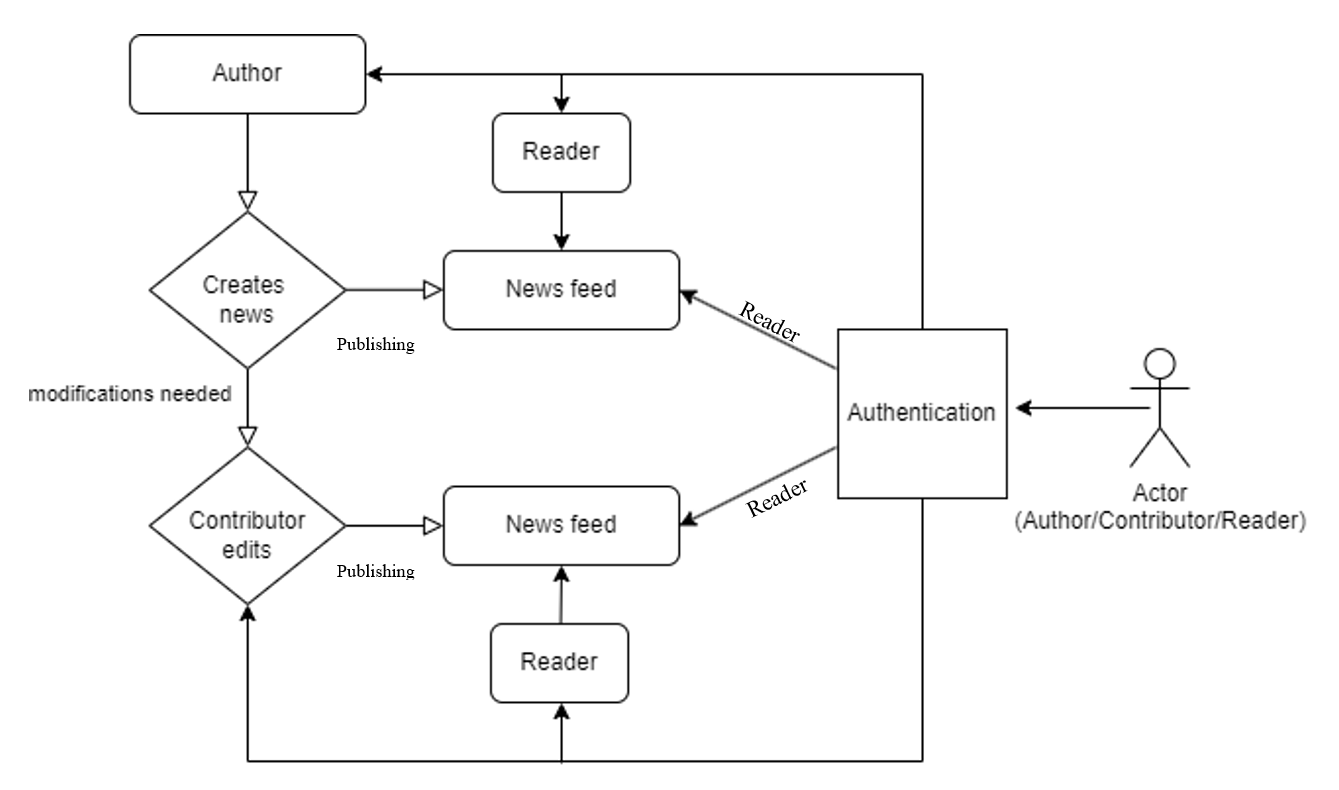


Fig. 3 System interaction diagram



The following describes how the components interact within the system:

1. Login:
   * The user, acting as either an Author, Reader, or Contributor, first logs into the system to access the news feed.
2. News Feed:
   * Once logged in, the user can view the news feed. The feed displays the latest news articles published by Authors.
   * The Reader can browse, search, and read news articles from the feed.
3. Author:
   * The Author publishes news by adding a title, description, time, and location. The news article is then added to the feed, making it available to all users.
   * When a new news item is added, the application interacts with the data model using the Entity Framework (EF). The new article is mapped to an entity in the data model (context), which represents the news object.
   * The Entity Framework manages the insertion of this new news object into the database, converting it into a structured format that can be stored and retrieved efficiently.
4. Contributor:
   * If any modifications or updates are needed, the Contributor can make changes to an existing news article. After editing, the Contributor publishes the updated version, which is then reflected in the feed.
5. Interaction Flow:
   * The Author adds and publishes news.
   * The Reader views the news on the feed.
   * If necessary, the Contributor edits the news and publishes updates.

**SECTION 4 – APPLICATION SOFTWARE COMPONENTS**

**4.1 Software Components**

The web application consists of several software components, each playing a crucial role in ensuring smooth functionality and interaction between the different modules. Below is an overview of the key software components:

1. Model Components: The Model represents the data structure of the application. The Entity Framework (EF) is used to map these objects to the database, ensuring that data is correctly stored and retrieved from the SQL Server. Following are the Models which will be used in the application:

* News Model - It will contain the News Id, News Heading and the News Description.
* Suggestions Model– It will contain the Suggestion Id, content and the date on which it was posted.
* Article Model – This model will contain the Article Id, Title, Content, and the FilePath for it in the database.
* ErrorViewModel – It will function for returning the view for any error with a RequestId.

1. Controller Components: Controllers manage the flow of information between the UI and the Model. They process user actions, perform necessary operations, and return data to be displayed. Following will be the controllers for the application:

* Home Controller – It will be responsible for delivering home page related requests.
* News Controller – This controller will take care of all requests for the News like creation, deletion, editing, getting News from database and posting them on feed, search functionality etc.
* Suggestions Controller – This is responsible for all requests for the suggestions page including the displaying of all suggestions on the page and storing the posted suggestions into the database.
* Articles Controller – This controller will handle the display of all articles on the Articles page along with storing the uploaded article files into the database.

1. View Components: Views represent the UI of the application. Views are responsible for what the user will se on the webpage. Following will be the View components in the application:

* Home View: Responsible for showing the content on home page. It includes the Privacy page as well.
* News View – It will display the news page to the user including the news feed, the interfaces for create-details-edit-delete and the search form interface.
* Suggestions View – It will display all the suggestions posted by the users.
* Articles View – It takes care of displaying the articles posted on the application along with providing the view for the create article interface.
* Error View – This view is for proving the respective error pages.

1. Routing Components:

* The application uses ASP.NET Core MVC's routing to guide user requests to the correct Controllers.
* Whether it is viewing the news feed or adding a new article, routing ensures that each request is properly handled.

**4.2 Testing**

To ensure the UTD(UpToDate) web application functions as intended, Integration and Unit testing will be performed throughout the development process. These tests will check different aspects of the application, from individual components to the entire system. Two separate Test projects will be made for Unit as well as the Integration testing of the components and the application respectively.

For doing so, the xUnit.net test framework will be used along with utilizing the Moq framework which is used in unit testing to isolate classes under test from their dependencies and ensuring that the proper methods on the dependent objects are being called.

Unittesting of the controllers will be done to verify the functionality in isolation for the following:

* Home controller – Test cases for returning Index and Privacy views.
* News Controller – Test cases for the create news interface, search news interface, delete news functionality.
* Article Controller – Test case for creating and uploading article on the application.
* Suggestions Controller – Test case to verify that posting a new suggestion adds it to the list.

To perform Integration testing which will identify and resolve any issues that may arise when components are combined, test classes will be written for the following components:

* Program.cs test - to test the role creation and assignment process.
* CustomWebApplicationFactory to use the in memory DB for testing
* Tests for ensuring the overall functionality of the following controllers:
* Home Controller - Test to verify that the Error action returns a valid view with an error model
* News Controller - Test to verify that ShowSearchResults returns news if it exists. Test to verify that the Delete action returns a view for an existing news item.
* Suggestions Controller – Creating a constructor to initialize the HTTP client for testing. Test to verify that the Index action returns a successful response.
* Articles Controller – Test for returning valid articles with file.

**SECTION 5 – APPLICATION SOFTWARE UI COMPONENTS**

UTD’s User Interface (UI) is designed to be simple and user-friendly. Each component is developed using C# with the ASP.NET Core MVC framework. Below are the key UI components:

1. Login Page:
   * Users can either register or log in from this page.
   * It provides fields for username and password, along with a link to register for new users.
   * After logging in, users are directed to the landing page.
2. Landing Page:
   * This page appears after a user log in and serves as the main navigation area.
   * It includes a navbar with the following options:
     + News Feed: Takes users to a page displaying all news articles.
     + Register/Login: Visible if the user is not logged in.
     + Logout: Allows users to log out.
     + Search News: Directs users to a page where they can search for news by keywords.
3. News Feed Page:
   * This is the main page where all news articles are displayed.
   * Each article shows the headline, author, and an option to view details such as the description, time, and place.
   * Users can also edit or delete news
   * There is an option for Authors to create a new news article.
4. Search Page:
   * Users can search for news articles by entering keywords in the search box.
   * The page displays a list of articles that match the search terms.
5. Edit Page:
   * Contributors can use this page to edit existing news articles.
   * They can change the title, description, time, and place before publishing the updated article.
6. Delete Page:
   * This page allows Authors to delete a news article from the system.
7. Suggestions Page:
   * This page allows users to post any suggestions for a particular news.
8. Articles page:
   * This page allows users to read the published articles and download them. It also allows users to upload article to be read by others.

Below is an activity diagram to demonstrate how the system operates:

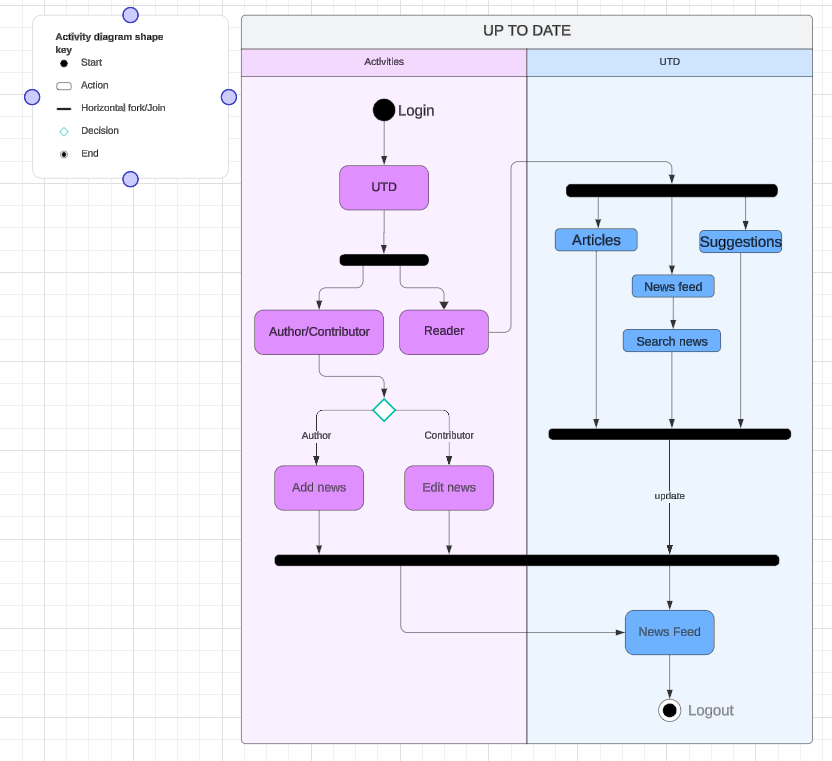


Fig. 4 Activity Diagram

**SECTION 6 – APPLICATION DATA COMPONENTS**

The application data component is designed to manage and store all relevant data for the UTD web application. As stated, the application will use SQL server as the database. The database model holds data related to users, news articles, and any interactions between these entities.

To manage the database efficiently, migrations are used to update and evolve the database schema. Migrations help in managing changes to the database over time, keeping track of each change and updating the database accordingly without losing data.

We are using EntityFramework, which is Microsoft'sObjectRelationalMapper **(**ORM**)**, to handle the database interactions. This framework allows us to work with the database using C#classes instead of writing complex SQL queries. The ORM maps these classes to database tables, making it easier to handle data through code.

Database Components: The application uses SQL Server to store and manage all user and news-related data. The Entity Framework simplifies database operations, allowing data to be managed through code without writing SQL queries. Tables in the database store information about users, news, articles, suggestions and their interactions. Migrations will be done for creating the tables for the following:

* ApplicationDBContext – It will create tables using the data structures as stated in the models. This includes the table for:
* News – This will contain the data related to news like the News Id, Title, Description, Date and Time posted etc.
* Identity Schema: It will create table for the Users including the details like name, email etc. It will also create:
* Roles Table
* UserRoles table
* Similar authorization related tables.
* Suggestions Table Migration: It will create tables for storing the suggestions into the database with their Id, Date and Time posted etc.
* Articles Table Migration: It will create the tables for storing the articles uploaded by the users. This includes the Id, Title, FilePath, Date and Time uploaded.

**SECTION 7 – Application Security Design**

This section deals with addressing the security requirements identified in the SRS document. The threats identified in the SRS documents will be addressed and a suitable design will be presented to meet the security requirements. This section will also include the mitigation strategies for the threats identified by the Microsoft Threat Modelling Tool (TMT).

Addressing the security requirements:

SR1: There are two spots to inject SQL queries in the application, suggestions input field and articles page. We will use parametrized queries to secure the application of SQL injection attacks.

SR2: XSS attack can also be done at the same input fields at the suggestions and articles page. To prevent them, we have to ensure encoding the input content before storing them into the database.

SR3: To prevent any DoS attack or upload of a malware, we have to ensure strict file upload functionality by allowing only PDF to be uploaded and that too of fixed size.

SR4: We have to ensure the use of a strong hashing algorithm for storing the passwords of users in the database.

SR5: We have to make sure about the HTTP connection, using TLS connection can prevent any vulnerability which may arise.

SR6: We have to make sure that cookies are securely generated. We also have to decide that to se user sessions or not, and if so using secure mechanism for them.

SR7: Including logging of events in the application will significantly enhance the security of the application.

SR8: Using secure connection protocol and verifying certificates will ensure that there is no interception of data in transit.

Following are the threats identified along with the possible mitigation strategies:

**1. An adversary may bypass critical steps or perform actions on behalf of other users (victims) due to improper validation logic.**

|  |  |
| --- | --- |
| **Stride Category:** | Elevation of Privileges |
| **Description:** | Failure to restrict the privileges and access rights to the application to individuals who require the privileges or access rights may result into unauthorized use of data due to inappropriate rights settings and validation. |
| **Possible Mitigation(s):** | Implement role-based authorization in ASP.NET. |

**2. An adversary can reverse weakly encrypted or hashed content**

|  |  |
| --- | --- |
| **Stride Category:** | Information Disclosure |
| **Description:** | An adversary can reverse weakly encrypted or hashed content |
| **Possible Mitigation(s):** | Using a strong hashing algorithm. |
|  |  |

**3. An adversary can gain access to sensitive data by sniffing traffic to Web Application**

|  |  |
| --- | --- |
| **Stride Category:** | Information Disclosure |
| **Description:** | An adversary may conduct man in the middle attack and downgrade TLS connection to clear text protocol, or forcing browser communication to pass through a proxy server that he controls. This may happen because the application may use mixed content or HTTP Strict Transport Security policy is not ensured. |
| **Possible Mitigation(s):** | Implementing using HTTPS connection. |
|  |  |

**4. An adversary can gain access to sensitive information through error messages**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Information Disclosure |
| **Description:** | An adversary can gain access to sensitive data such as the following, through verbose error messages - Server names - Connection strings - Usernames - Passwords - SQL procedures - Details of dynamic SQL failures - Stack trace and lines of code - Variables stored in memory - Drive and folder locations - Application install points - Host configuration settings - Other internal application details |
| **Possible Mitigation(s):** | Not exposing security details in error messages. |

**5. An adversary may gain access to sensitive data from uncleared browser cache**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Information Disclosure |
| **Description:** | An adversary may gain access to sensitive data from uncleared browser cache |
| **Possible Mitigation(s):** | Implementing secure methods for cache storing. |
|  |  |

**6. Attacker can deny the malicious act and remove the attack foot prints leading to repudiation issues**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Repudiation |
| **Description:** | Proper logging of all security events and user actions builds traceability in a system and denies any possible repudiation issues. In the absence of proper auditing and logging controls, it would become impossible to implement any accountability in a system |
| **Possible Mitigation(s):** | Ensure that auditing and logging is enforced on the application. |
|  |  |

**7. An adversary can get access to a user's session due to improper logout and timeout**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | The session cookies is the identifier by which the server knows the identity of current user for each incoming request. If the attacker is able to steal the user token he would be able to access all user data and perform all actions on behalf of user. |
| **Possible Mitigation(s):** | Set up session for inactivity lifetime. |

**8. An adversary can get access to a user's session due to insecure coding practices**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | The session cookies is the identifier by which the server knows the identity of current user for each incoming request. If the attacker is able to steal the user token he would be able to access all user data and perform all actions on behalf of user. |
| **Possible Mitigation(s):** | Enable ValidateRequest attribute on ASP.NET Pages. |
|  |  |

**9. An adversary can spoof the target web application due to insecure TLS certificate configuration**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | Ensure that TLS certificate parameters are configured with correct values |
| **Possible Mitigation(s):** | Verify X.509 certificates used to authenticate SSL, TLS, and DTLS connections. |
|  |  |

**10. An adversary can steal sensitive data like user credentials**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | Attackers can exploit weaknesses in system to steal user credentials. Downstream and upstream components are often accessed by using credentials stored in configuration stores. Attackers may steal the upstream or downstream component credentials. Attackers may steal credentials if, Credentials are stored and sent in clear text, Weak input validation coupled with dynamic sql queries, Password retrieval mechanism are poor, |
| **Possible Mitigation(s):** | Explicitly disable the autocomplete HTML attribute in sensitive forms and inputs. |

**11. Attackers can steal user session cookies due to insecure cookie attributes**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | The session cookies is the identifier by which the server knows the identity of current user for each incoming request. If the attacker is able to steal the user token he would be able to access all user data and perform all actions on behalf of user. |
| **Possible Mitigation(s):** | Applications available over HTTPS must use secure cookies. |

**12. An adversary can create a fake website and launch phishing attacks**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | Phishing is attempted to obtain sensitive information such as usernames, passwords, and credit card details (and sometimes, indirectly, money), often for malicious reasons, by masquerading as a Web Server which is a trustworthy entity in electronic communication |
| **Possible Mitigation(s):** | Verify X.509 certificates used to authenticate SSL, TLS, and DTLS connections. |
|  |  |

**13. An adversary may spoof Browser and gain access to Web Application**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | If proper authentication is not in place, an adversary can spoof a source process or external entity and gain unauthorized access to the Web Application |
| **Possible Mitigation(s):** | Consider using a standard authentication mechanism to authenticate to Web Application. |
|  |  |

**14. An adversary can deface the target web application by injecting malicious code or uploading dangerous files**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Tampering |
| **Description:** | Website defacement is an attack on a website where the attacker changes the visual appearance of the site or a webpage. |
| **Possible Mitigation(s):** | Implement Content Security Policy (CSP), and disable inline JavaScript. |
|  |  |

**15. An adversary can gain access to sensitive data by performing SQL injection through Web App**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Tampering |
| **Description:** | SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed. |
| **Possible Mitigation(s):** | Ensure that type-safe parameters are used in Web Application for data access. |
|  |  |

**16. An adversary can gain access to sensitive data by performing SQL injection through Web App**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Tampering |
| **Description:** | SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed. |
| **Possible Mitigation(s):** | Ensure that type-safe parameters are used in Web Application for data access. |

**17. An adversary may spoof Database and gain access to Web Application**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | If proper authentication is not in place, an adversary can spoof a source process or external entity and gain unauthorized access to the Web Application |
| **Possible Mitigation(s):** | Consider using a standard authentication mechanism to authenticate to Web Application. |
|  |  |

**18. An adversary can create a fake website and launch phishing attacks**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | Phishing is attempted to obtain sensitive information such as usernames, passwords, and credit card details (and sometimes, indirectly, money), often for malicious reasons, by masquerading as a Web Server which is a trustworthy entity in electronic communication |
| **Possible Mitigation(s):** | Verify X.509 certificates used to authenticate SSL, TLS, and DTLS connections. |
|  |  |

**19. An adversary can spoof the target web application due to insecure TLS certificate configuration**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Spoofing |
| **Description:** | Ensure that TLS certificate parameters are configured with correct values |
| **Possible Mitigation(s):** | Verify X.509 certificates used to authenticate SSL, TLS, and DTLS connections. |

**20. Attacker can deny the malicious act and remove the attack foot prints leading to repudiation issues**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Repudiation |
| **Description:** | Proper logging of all security events and user actions builds traceability in a system and denies any possible repudiation issues. In the absence of proper auditing and logging controls, it would become impossible to implement any accountability in a system |
| **Possible Mitigation(s):** | Ensure that auditing and logging is enforced on the application. |
|  |  |

**21. An adversary can gain access to sensitive information through error messages**

|  |  |
| --- | --- |
| **Stride Stride Category:** | Information Disclosure |
| **Description:** | An adversary can gain access to sensitive data such as the following, through verbose error messages - Server names - Connection strings - Usernames - Passwords - SQL procedures - Details of dynamic SQL failures - Stack trace and lines of code - Variables stored in memory - Drive and folder locations - Application install points - Host configuration settings - Other internal application details |
| **Possible Mitigation(s):** | Do not expose security details in error messages. |
|  |  |

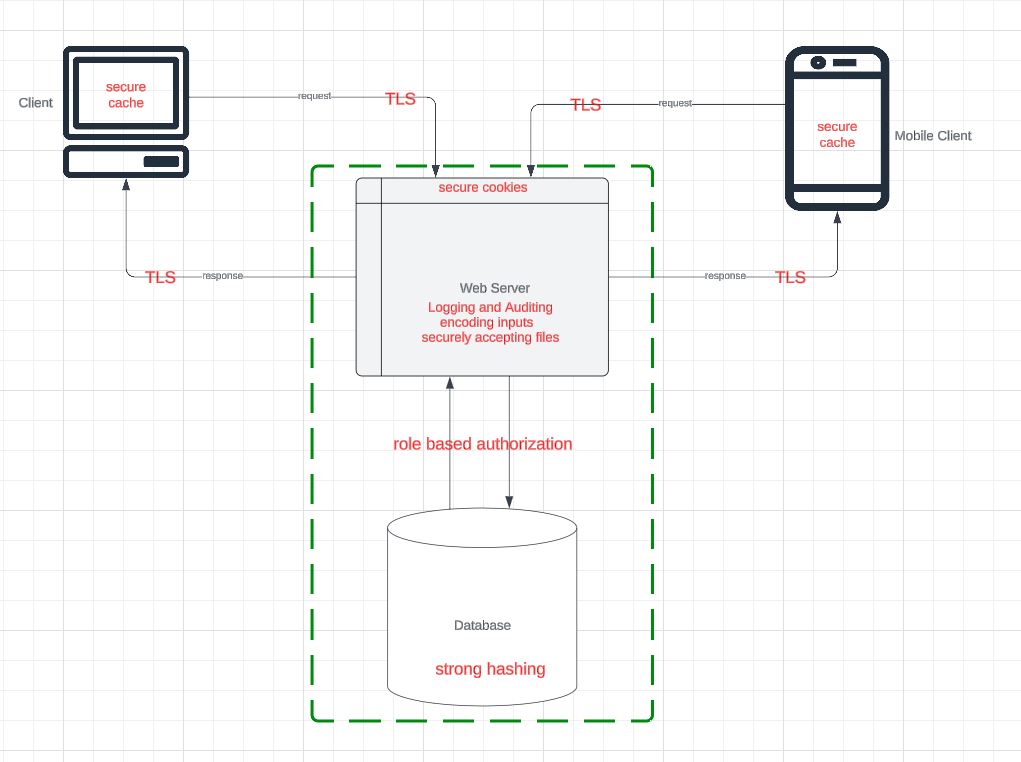
****

Fig. 5 Secure System Architecture

**SECTION 8 - REFERENCES**

1. Google Docs. "Sample SRS Document."

Accessed October 2024. <https://docs.google.com/document/d/1pgMutdDasJb6eN6yK6M95JM8gQ16IKacxxhPXgeL9WY/edit#heading=h.lomckg6w5y6>

1. Lucid Chart: "Online Diagram and Flowchart Software." Accessed October 2024.

<https://www.lucidchart.com>

1. Draw.io: "Diagramming Application for Creating Models."

Accessed September 2024.

<https://www.draw.io>

1. UTD Project: "Software Requirement Specification (SRS) Document."

Updated version (October 2024)

**SECTION 9 – INDEX**

Application Data Components………………………….................. 14

Application Software Components……………………................. 10

Application Software UI Components……………….…………. 12

Author……………………………………………………………….... 1, 2, 7

Components and their Interaction……………………..................... 7

Controller Components……………………………......................10, 11

Database Management……………………………………………10, 11

Design Overview……………………………………..…………………2

Data Flow………………………………………...…..…………………..14

Entity Framework………………………………………………….11, 14

Login Page…………………………………………….………………....12

Model Components……………………………………………………10

MVC Process Flow……………………………………..……………....6

News Feed Page………………………………………………………..12

Routing Components…………………………………….................... 11

Software Components………………………………………………...10

Software UI Components……………………………………………12

System Architecture……………………………………………………6

System Interaction……………………………………………………...7

Testing…………………………………………………………………….11

Traceability Matrix…………………………………………………..…5