Table of Contents

[Introduction 2](#_Toc132380427)

[Running The Project 2](#_Toc132380428)

[Project Contents 2](#_Toc132380429)

[Features & Design Decisions 4](#_Toc132380430)

[Front-end 4](#_Toc132380431)

[Mobile-Responsiveness 4](#_Toc132380432)

[Navigation Bar 4](#_Toc132380433)

[Infinity Scroll 4](#_Toc132380434)

[Toasts 5](#_Toc132380435)

[Asynchronous Programming 5](#_Toc132380436)

[Search & Pagination 5](#_Toc132380437)

[Back-end 5](#_Toc132380438)

[APIs 5](#_Toc132380439)

[Authentication 6](#_Toc132380440)

[Redirection 6](#_Toc132380441)

[Appendix: Login Credentials 7](#_Toc132380442)

# Introduction

The web application built is a fictitious news site. It is a dynamic web application containing features such as authentication, comments, reading lists, and search. The technologies used on the front-end are Vanilla HTML, Bootstrap 5, and JavaScript, whereas the technology used on the back-end is Django.

# Running The Project

One external library is used in this project – markdown2. The library can be installed using the command pip install markdown2 in the terminal.

To start the server, navigate to the project directory and enter the following command: python manage.py runserver. The server will then be hosted on localhost:8000.

# Project Contents

In this section, the information about the project as well as its structure and contents are described.

The project is named “Moonway”, which is the name for the fictional news company. Within the project, an app named “news” is created. This is the one and only application the project consists of. In the course of interacting with the news site, the users will be interacting with this application alone (apart from built-in apps such as authentication). All the pertinent files are therefore stored within the “news” directory under the project directory.

The pertinent directories and files under the “news” directory are described below:

1. **File**

This is a directory to store the files that are uploaded by users and admins when new objects are created from the models. Currently, there are 3 subdirectories in this directory – articles, authors, and users, which store the pictures associated with the respective models.

1. **Static/news**

This directory is used to store static resources to be used in the templates. The types of files that are stored within this directory include JS scripts, CSS stylesheets, and images. The noteworthy files are described below.

* 1. **articlePageModifyReadingList.js**

This script contains the processes that are carried out when adding or removing an article from the reading list in the article page itself. When an article is added to or removed from the reading list, the Fetch API is used to submit the request to the back-end, a confirmation status is shown to the user in the form of a toast, and the reading list icon on the article page is modified to reflect the current state of the article. Further explanations and justifications on the asynchronous aspect of the script are provided in the [Asynchronous Programming](#_Asynchronous_Programming) section below.

* 1. ***pageName*IS.js**

The scripts ending with “IS.js” describe the behaviour of the infinity scroll found in various parts of the web application. On a high level, when the infinity scroll is triggered by scrolling to the bottom of the page, a spinner will be added at the bottom of the page, the Fetch API is used to fetch the articles, the articles are rendered, and the spinner is removed. Further explanations and justifications on the asynchronous aspect of the script are provided in the [Asynchronous Programming](#_Asynchronous_Programming) section below.

* 1. **readingList.js**

This script describes the behaviour of removing an article from the reading list in the profile page. The Fetch API is used to send the request to the server to remove the article from the reading list of the user, and a deletion animation is played upon successful deletion on the server’s side. Further explanations and justifications on the asynchronous aspect of the script are provided in the [Asynchronous Programming](#_Asynchronous_Programming) section below.

* 1. **script.js**

This script contains the general behaviour across all parts of the web application. Currently, it contains 2 functions – linkToArticlePage and linkToAuthorPage. They are used as part of an event listener that is triggered when a div is clicked on and the user has to be redirected to an article or author page.

* 1. **search.js**

This script contains the behaviour of the search page. An event listener is attached to the document object to generate the page numbers. A second event listener is attached to the advanced search toggler to display and hide the advanced search section upon clicking on the toggler. Another event listener is attached to the advanced search form to send the request to the server via the Fetch API, and render the results when they are ready. Several utility functions that are used in the event listeners are included at the bottom of the script.

* 1. **styles.css**

This stylesheet contains the styles that are unable to be implemented using Bootstrap.

1. **Templates/news**

This directory contains all the templates that are used in this web application. Among the templates, the layout.html file is the base template from which all the other templates extend. It contains the navigation bar and the footer, which are common to all pages in the web application.

1. **Forms.py**

This python script contains the Django forms that are used in the web application. In this case, there is one form – UserForm. It is a form that is customized based on the default UserCreationForm provided in the Django authentication module. This form is used in the sign up page to collect data from the user to sign them up.

1. **Models.py**

This python script contains the models that are used within this web application.

1. **Utils.py**

This script contains utility functions that are used in the views. In this web application, there is one function in the utils.py script – article\_to\_dict. This function is used to extract the pertinent information from an Article object to be put in a python dictionary to be sent through the network in JSON format.

1. **Views.py**

This script contains the logic of processing the user requests to the server and sending the response back to the clients.

# Features & Design Decisions

In this section, the features of the web application and the rationale behind the decision on the design of the web application are described. The front-end and back-end will be described separately.

## Front-end

The design of the front-end is driven by core UI/UX principles of usability since a news site is frequently patronized by all kinds of users. Throughout the site, various mechanisms have been implemented to ensure that the user has a smooth experience in using the site. The mechanisms and features are further described in their respective sections.

### Mobile-Responsiveness

With the use of Bootstrap, a mobile-first approach is taken in the course of building the front-end of this web application. This feature is most salient in the navigation bar, where all navigation links will be contained in a hamburger menu rather than the navigation bar itself when the width of the viewport is below a certain dimension. The content of all pages will also adjust themselves based on the size of the screen to ensure that users have a more optimized experienced. A mobile-first approach is preferred in this scenario as more and more users are sourcing their news with a mobile phone.

### Navigation Bar

A navigation bar is included in all pages to allow for easy navigation around the web application at all times. The components of the navigation bar are as follows:

1. **Brand Logo**

Clicking on the brand logo brings the user back to the home page (index.html)

1. **Navigation Links (Categories)**

The navigation links are the categories of the articles. Clicking on them brings the user to a page displaying the articles of the corresponding category.

1. **Search**

A search bar is included for users to easily search for articles. Entering a search term and hitting the “Enter” key or clicking on the “Search” button brings the user to the search page with the results of the search. The search feature is details in the [Search & Pagination](#_Search_&_Pagination) section below.

1. **User Section**

Depending on whether or not the user is authenticated, users may see different things in this part of the navigation bar. If the user is **NOT** authenticated, the user will see 2 links to either sign up or log in to the site; if the user is authenticated, he will see his own profile picture instead. Clicking on the profile picture of the user will produce a dropdown containing links to go to the profile page or logout from the site.

### Infinity Scroll

The infinity scroll feature is also included in pages where users are meant to explore various articles rather than search for a specific one. Infinity scrolls facilitate exploration since new articles will be shown to the user without any extra interaction upon scrolling to the bottom of the page, which is akin to modern social media sites. This feature is included in the index, category, and author pages.

### Toasts

Toasts are used in parts of the application where the state of the application has to be reflected to the user. The reason of using toasts is that they are salient yet non-intrusive since they pop up near the top of the page and disappear automatically within a set period of time. For example, upon successfully logging out of the application, a toast is shown to the user indicating that the user has been logged out. Toasts appear in the article page (modifying reading list and adding comments), index page (upon logging out), login page (entering the wrong credentials or successfully signing up), and signup page (entering an invalid input).

### Asynchronous Programming

Asynchronous programming with the Fetch API is used in various instances. This allows requests to be made asynchronously to the server, the response to be received, and the DOM updated, all without refreshing the page or redirecting users to another page. This improves the user experience as their browsing is not interrupted while the requests are being made and processed in the background. This is used in the [infinity scroll](#_Infinity_Scroll) feature, adding and removing from the reading list (article and profile page), and the [search & pagination](#_Search_&_Pagination) feature.

### Search & Pagination

Two types of searches are provided in this web application – basic search and advanced search. Basic search is performed by using the search bar in the navigation bar at the top of each page, whereas advanced search is performed in the search page through the advanced search form. In terms of functionality, the basic search searches for articles based on whether or not the keyword is contained in either the article’s title or the author’s name. On the other hand, the advanced search searches based on 3 criteria – title, author, and category. Users can leave any of the 3 criteria empty should they not be needed. Additionally, users can choose the order in which the search results are shown based on the alphabetical order, date written, or popularity.

Instead of an infinity scroll, pagination is used in the search page. This is because users who interact with the search page are more likely to be searching for a specific type of article, or even one specific article. Combined with the sorting feature described above, pagination allows users to sift through articles quickly to find what they need. The current page that the user is on is also highlighted to allow users to keep track of the page that they are on.

## Back-end

The back-end is built with 2 guiding principles in mind – security, and maintainability. Ensuring the security of the web application of a news site is crucial since it stores user information and plays the role of disseminating information to the general public. On the other hand, maintainability is prioritized to ensure that future feature additions to the application and maintenance processes can be carried out smoothly. This is achieved mainly by modularizing the code, thus allowing developers to easily observe the moving parts of the application and swiftly identify the correct module to investigate. The components of the back-end are described in their respective sections below.

### APIs

In this web application, 3 simple APIs were implemented – articles, submit\_comment, and modify\_reading\_list.

The articles API is used throughout the web application to fetch for articles. This is built to cater for the infinity scroll and advanced search features, which use the Fetch API to send requests to this endpoint. This allows for the article-querying function in the web application to be consolidated in one place, thus improving the maintainability of the application since developers can focus on modifying and interacting with this API from the front-end.

The submit\_comment API is used when a user submits a comment for an article. The endpoint accepts only POST requests, and checks if the user is authenticated before adding a comment to the database. Users are then redirected to the page of the article which he commented on.

The modify\_reading\_list API is used to modify the reading list of a user. It takes a POST request containing the ID of the article. If the article is not currently in the reading list of the user, the article will be added to the reading list, otherwise it will be removed. The corresponding statuses are then sent back to the client, which the front-end can use to display appropriate toasts or animations to the user to notify them of the change in status.

In consolidating these core functionalities in one place and segregating them from all other parts of the system, the security of the system can also be improved since developers can focus on implementing robust security measures for these APIs.

### Authentication

An authentication system is also implemented to allow users to access features requiring the identification of users. For example, the reading list feature requires the server to identify which article is added to which users’ reading list.

The authentication feature consists of two parts – signup and login. The signup function is implemented using a Django UserCreationForm. The actual form used is inherited from and customized based on the UserCreationForm. When the user submits the request to create an account, the server performs validation on the form before saving the new user data to the database. If the validation fails, the user will be shown the errors in the corresponding fields, allowing him to make the necessary changes.

The login function is implemented using a basic HTML form. A POST request is sent to the server upon submission of the form, and the user is authenticated using the authenticate method provided by Django.

Depending on whether or not the user is authenticated, the server will render several pages slightly differently. For example, the icon to add an article to a reading list is not present for users that are not authenticated, and the users will not be able to make a comment. Since these are features that can only be provided if the user is authenticated, the server does not render these elements when sending the response to the client.

In addition, the APIs to modify the reading list and submit a comment is not made available to users who are not authenticated. This is a security measure to prevent malicious actors from tampering with other users’ reading list or submitting a comment their name.

### Redirection

Redirection on the back-end is also implemented in some pages to allow users to be redirected to the pertinent pages after performing an action on another. For example, if an unauthenticated user clicked on the prompt to login in the comments section of an article and proceeds to login, he will be redirected back to the article page where he was on. This improves the user experience as the user is brought back directly to the article that he wanted to make a comment on but failed due to not being authenticated.

This feature is implemented by adding a URL parameter at the end of the link to the login page. Upon logging in successfully, the server will check if the URL parameter exists, and redirects the user to the page in the parameter.

# Appendix: Login Credentials

The credentials to log in to the web application are as follows:

admin:admin (superuser)

jordanbpeterson:123456

mattwalsh:123abc123abc