WARDOCX

Release 0.6

Peter Asjes, Henry Frye, Caleb Koutrakos, Will Robinson,and Bob

Apr 15, 2025

CONTENTS:

WARDOCX is a undergraduate capstone project part of a larger initiative to modernize wargaming using AI.

Note

This project is under active development.

CHAPTER

ONE

USAGE

1.1 Installation

1.1.1 Download the Code

The code for this project can be downloaded via GitHub.

```
$ git clone https://github.com/sideoffryes/WARDOCX.git
```

1.1.2 Python Management

To use this project, the are several prerequisites that are necessary. The easiest way to manage these dependencies is using pip and a virtual environment.

If you do not have python already, download and install a release of python 3 for your platform.

To ensure that pip is available on your system, follow these instructions for your platform.

All of the required python packages can be easily installed via the provided configuration files and setup script.

Setup Script

```
$ ./setup.sh
```

The script will prompt you several times for different actions to take during the setup process. If you respond yes to each of the prompts, the project will be fully completed. If you choose not to generate the vector embeddings of the documents during the setup process, you must do it before attempting to generate any documents.

Once the setup is complete, make sure to activate the virtual environment.

```
$ source .venv/bin/activate
```

1.1.3 Hugging Face

The project accesses the Llama 3.1, 3.2, and 3.3 families from Meta via Hugging Face. Running this project requires a Hugging Face account and access to those families.

1. Visit the page for the model on Hugging Face. For example, Llama-3.2 (1B)
2. Create a free account and login.
3. Return to the Llama webpage (if not already there).
4. You should see a Community License Agreement at the top. Click the “Expand to review” button:
5. If you agree with the terms, fill out the form

6. Check email later.

Once you have received access to the models, visit your tokens page and click "Create new token". Choose the "Read" token type at the very top. Then click "Create token". Copy the generated string.

In the terminal, run the following command and paste in your access token when prompted:

```
(.venv) $ huggingface-cli login
```

1.2 Running the Project

Before attempting to run any of the scripts, make sure that you have the correct virtual environment activated.

```
$ source .venv/bin/activate
```

All of the code that runs the webserver and actually generated the documents can be found inside of the *capstone* directory. *app.py* is the webserver and *docgen.py* is the script that accesses the LLM to generate documents. Running the entire project can be accomplished with the following:

```
(.venv) $ cd capstone
(.venv) $ python3 app.py
```

The webserver can be reached from your browser by using one of the ip addresses printed out in the terminal when the server is created.

1.3 Interacting with the Webserver in the Browser

The form presented to you when the website is first loaded can be used to generate a document. Use the *selection options* dropdown menu to select the type of document that you would like to create. You can specify your requirements and any additional specifications in the *additional parameters* textbox.

Depending on the size of the model used to generate the document, the server may load for a few minutes before the final output is produced.

1.4 Generating the Docs

The repository is shipped with a precompiled PDF version of the documentation for the entire project for both users and developers.

The HTML documentation that can be viewed from the browser when running the webserver can be created by cding into the docs directory and using the make file.

```
(.venv) $ cd docs
(.venv) $ make html
```

The generated documentation will appear in the docs/build/html directory.

CHAPTER TWO

FUNCTION CALL API DOCUMENTATION

docGen
faissSetup
pdfToText

2.1 docGen

Functions

`find_most_rel`(query, index)	Returns the indices of the most related documents based on the user's query
`gen`(model_num, type_num, prompt[, save])	Generates a specificed document using a specified LLM and returns the result.
`load_examples`(type, prompt)	Returns real life examples of the requested document type.
`save_response`(response, prompt, model_name, ...)	Writes the response and information about how it was generated to the disk.
`select_doc`(num)	Translates between the numeric representation of a document type and its full name.
`select_model`(num)	Translates between the numeric representation of a model to the full string of its name.

2.2 faissSetup

Functions

`cache_faiss`(chunks, fname)	Converts each of the given text chunks into a vectory representation, adds them to a FAISS index, and writes the FAISS index to the disk.
`gen_embeds`(text)	Converts the given string to its vector embedding
`mar`()	Creates vector embeddings for all documents in the data/MARADMINS/ directory and adds them to FAISS index in same directory.

continues on next page

Table 3 – continued from previous page

`nav()`	Creates vector embeddings for all documents in the data/NAVADMINS/ directory and adds them to FAISS index in same directory.
`opord()`	Creates vector embeddings for all documents in the data/OpOrds directory and adds them to FAISS index in the same directory.
`rtw()`	Creates vector embeddings for all documents in the data/RTW/ directory and adds them to FAISS index in the same directory.

2.3 pdfToText

Functions

`pdf_to_text`(pdf_path, output_txt)	

CHAPTER

THREE

WEBSERVER API DOCUMENTATION

Describes the API endpoints for the Flask web server

3.1 User Endpoints

GET /userprofile

Displays the user profile page.

This endpoint renders the user profile page, allowing logged-in users to view their username and change their password. If the user is not logged in (no *user_id* in the session), it displays a default view indicating they are not logged in.

Session Requirements:

- Requires the *user_id* to be present in the session to identify the logged-in user.

Response Body:

- **Content-Type:** `text/html`
- **Renders the *userprofile.html* template. The template will receive the following context variables:**
  - `username`: The username of the logged-in user, or "Not logged in" if no user is logged in.
  - `errorMsg`: An empty string by default for GET requests. May contain an error message after a failed POST request.
  - `successMsg`: An empty string by default for GET requests. May contain a success message after a successful POST request.

Example Response (User logged in):

```
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Username: testuser</p>
    <form method="POST" action="/userprofile">
        <label for="curpwd">Current Password:</label><br>
        <input type="password" id="curpwd" name="curpwd"><br>
        <label for="newpwd">New Password:</label><br>
        <input type="password" id="newpwd" name="newpwd"><br>
```

(continues on next page)

(continued from previous page)

```
        <label for="conpwd">Confirm New Password:</label><br>
        <input type="password" id="conpwd" name="conpwd"><br><br>
        <input type="submit" value="Change Password">
    </form>
    </body>
</html>
```

Example Response (User not logged in):

```
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Username: Not logged in</p>
    <p></p>
    <p></p>
    <form method="POST" action="/userprofile">
        <label for="curpwd">Current Password:</label><br>
        <input type="password" id="curpwd" name="curpwd"><br>
        <label for="newpwd">New Password:</label><br>
        <input type="password" id="newpwd" name="newpwd"><br>
        <label for="conpwd">Confirm New Password:</label><br>
        <input type="password" id="conpwd" name="conpwd"><br><br>
        <input type="submit" value="Change Password">
    </form>
    </body>
</html>
```

`POST /userprofile`

Handles user password change requests.

This endpoint processes the form submission from the user profile page to change the user's password. It requires the user to be logged in (a valid *user_id* in the session) and validates the provided current password against the stored hash before updating the password.

Session Requirements:

- Requires the *user_id* to be present in the session to identify the logged-in user.

Request Body:

- **Content-Type:** `application/x-www-form-urlencoded`
- **The request body should contain the following form parameters:**
  - `curpwd`: The user's current password.
  - `newpwd`: The desired new password.
  - `conpwd`: Confirmation of the new password.

Response Codes:

- **200 OK (on successful password change):**
  - **Response Body:**

    * **Content-Type:** `text/html`
    * Renders the *userprofile.html* template with the logged-in `username` and a `successMsg` indicating that the password was successfully changed. The `errorMsg` will be an empty string.

- **200 OK (on errors):**
  - **Response Body:**
    * **Content-Type:** `text/html`
    * **Renders the *userprofile.html* template with the logged-in `username` and an `errorMsg` explaining the issue. The `successMsg` will be an empty string. Possible error messages include:**
      · "ERROR: Please fill out all fields." (if any of the password fields are missing)
      · "ERROR: Current password is incorrect." (if the provided current password does not match)
      · "ERROR: New passwords do not match." (if the new password and confirmation do not match)

Example Request (using curl):

```
curl -X POST -d "curpwd=oldpassword&newpwd=newsecurepassword&
↪conpwd=newsecurepassword" http://yourdomain.com/userprofile -c cookies.txt -b␣
↪cookies.txt

# Note: You might need to handle session cookies appropriately for a real-world␣
↪scenario.
```

Example Successful Response:

```
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Username: testuser</p>
    <p style="color: green;">Password successfully changed.</p>
    <form method="POST" action="/userprofile">
        <label for="curpwd">Current Password:</label><br>
        <input type="password" id="curpwd" name="curpwd"><br>
        <label for="newpwd">New Password:</label><br>
        <input type="password" id="newpwd" name="newpwd"><br>
        <label for="conpwd">Confirm New Password:</label><br>
        <input type="password" id="conpwd" name="conpwd"><br><br>
        <input type="submit" value="Change Password">
    </form>
    </body>
</html>
```

Example Error Response (Incorrect current password):

```
<!DOCTYPE html>
<html>
```

(continues on next page)

(continued from previous page)

```
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Username: testuser</p>
    <p style="color: red;">ERROR: Current password is incorrect.</p>
    <form method="POST" action="/userprofile">
        <label for="curpwd">Current Password:</label><br>
        <input type="password" id="curpwd" name="curpwd"><br>
        <label for="newpwd">New Password:</label><br>
        <input type="password" id="newpwd" name="newpwd"><br>
        <label for="conpwd">Confirm New Password:</label><br>
        <input type="password" id="conpwd" name="conpwd"><br><br>
        <input type="submit" value="Change Password">
    </form>
    </body>
</html>.. http:post:: /userprofile
```

Handles user password change requests.

This endpoint processes the form submission from the user profile page to change the user's password. It requires the user to be logged in (a valid *user_id* in the session) and validates the provided current password against the stored hash before updating the password.

Session Requirements:

- Requires the *user_id* to be present in the session to identify the logged-in user.

Request Body:

- **Content-Type:** `application/x-www-form-urlencoded`
- **The request body should contain the following form parameters:**
  - `curpwd`: The user's current password.
  - `newpwd`: The desired new password.
  - `conpwd`: Confirmation of the new password.

Response Codes:

- **200 OK (on successful password change):**
  - **Response Body:**
    * **Content-Type:** `text/html`
    * Renders the *userprofile.html* template with the logged-in `username` and a `successMsg` indicating that the password was successfully changed. The `errorMsg` will be an empty string.
- **200 OK (on errors):**
  - **Response Body:**
    * **Content-Type:** `text/html`
    * **Renders the *userprofile.html* template with the logged-in `username` and an `errorMsg` explaining the issue. The `successMsg` will be an empty string. Possible error messages include:**
      · "ERROR: Please fill out all fields." (if any of the password fields are missing)

- “ERROR: Current password is incorrect.” (if the provided current password does not match)
- “ERROR: New passwords do not match.” (if the new password and confirmation do not match)

Example Request (using curl):

```
curl -X POST -d "curpwd=oldpassword&newpwd=newsecurepassword&
↪conpwd=newsecurepassword" http://yourdomain.com/userprofile -c cookies.txt -b␣
↪cookies.txt

# Note: You might need to handle session cookies appropriately for a real-world␣
↪scenario.
```

Example Successful Response:

```
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Username: testuser</p>
    <p style="color: green;">Password successfully changed.</p>
    <form method="POST" action="/userprofile">
        <label for="curpwd">Current Password:</label><br>
        <input type="password" id="curpwd" name="curpwd"><br>
        <label for="newpwd">New Password:</label><br>
        <input type="password" id="newpwd" name="newpwd"><br>
        <label for="conpwd">Confirm New Password:</label><br>
        <input type="password" id="conpwd" name="conpwd"><br><br>
        <input type="submit" value="Change Password">
    </form>
    </body>
</html>
```

Example Error Response (Incorrect current password):

```
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Username: testuser</p>
    <p style="color: red;">ERROR: Current password is incorrect.</p>
    <form method="POST" action="/userprofile">
        <label for="curpwd">Current Password:</label><br>
        <input type="password" id="curpwd" name="curpwd"><br>
        <label for="newpwd">New Password:</label><br>
        <input type="password" id="newpwd" name="newpwd"><br>
        <label for="conpwd">Confirm New Password:</label><br>
```

(continues on next page)

(continued from previous page)

```
        <input type="password" id="conpwd" name="conpwd"><br><br>
        <input type="submit" value="Change Password">
    </form>
    </body>
</html>
```

3.2 Authentication Endpoints

`GET /login`

Displays the login form.

This endpoint is responsible for rendering the HTML form that allows users to log in to the web application. No data is submitted or processed when this endpoint is accessed via a GET request.

Response Body:

- **Content-Type:** `text/html`
- The response body will contain the HTML content of the *login.html* template, which includes the login form with fields for username and password.

Example Response:

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form method="POST" action="/login">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Login">
    </form>
    </body>
</html>
```

`POST /login`

Handles user login submissions.

This endpoint processes the login form submitted by users. It expects a POST request with username and password data. It authenticates the user against stored credentials and either logs them in or displays an error message.

Request Body:

- **Content-Type:** `application/x-www-form-urlencoded`
- **The request body should contain the following form parameters:**
  - `username`: The username provided by the user.
  - `password`: The password provided by the user.

Response Codes:

- **200 OK (on successful login):**
  - **Response Body:**

      - **Content-Type:** `text/html`
      - Renders the *index.html* template with a success message indicating the logged-in username.
- **200 OK (on failed login or non-existent user):**
  - **Response Body:**
      - **Content-Type:** `text/html`
      - Renders the *login.html* template with an *errorMsg* parameter containing an error message (e.g., “ERROR: That username does not exist, please try again.” or “ERROR: Given login credentials were incorrect, please try again.”).

Example Request (using curl):

```
curl -X POST -d "username=testuser&password=securepassword" http://yourdomain.com/
↪login
```

Example Successful Response (renders index.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <p>Successfully logged into: testuser</p>
    </body>
</html>
```

Example Error Response (renders login.html with error):

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <p style="color: red;">ERROR: Given login credentials were incorrect, please␣
↪try again.</p>
    <form method="POST" action="/login">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Login">
    </form>
    </body>
</html>
```

`GET /register`

Displays the user registration form.

This endpoint renders the HTML form that allows new users to create an account on the web application. No data is submitted or processed when this endpoint is accessed via a GET request.

Response Body:

- **Content-Type:** `text/html`
- The response body will contain the HTML content of the *new_account.html* template, which includes the registration form with fields for username, password, and password confirmation.

Example Response:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
</head>
<body>
    <h1>Create New Account</h1>
    <form method="POST" action="/register">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="ogpassword">Password:</label><br>
        <input type="password" id="ogpassword" name="ogpassword"><br>
        <label for="repassword">Confirm Password:</label><br>
        <input type="password" id="repassword" name="repassword"><br><br>
        <input type="submit" value="Register">
    </form>
    </body>
</html>
```

`POST /register`

Handles user registration submissions.

This endpoint processes the registration form submitted by new users. It expects a POST request with username, password, and password confirmation data. It validates the input, checks for existing usernames, hashes the password, and creates a new user account in the database.

Request Body:

- **Content-Type:** `application/x-www-form-urlencoded`
- **The request body should contain the following form parameters:**
  - `username`: The desired username for the new account.
  - `ogpassword`: The desired password for the new account.
  - `repassword`: Confirmation of the desired password.

Response Codes:

- **200 OK (on successful registration):**
  - **Response Body:**
    - **Content-Type:** `text/html`
    - Renders the *login.html* template with a success message (*errorMsg*) instructing the user to log in.
- **200 OK (on errors):**
  - **Response Body:**
    - **Content-Type:** `text/html`
    - **Renders the *new_account.html* template with an *errorMsg* explaining the issue. Possible error messages include:**
      - "ERROR: This username already exists. Please use a different one."

- “ERROR: The passwords did not match.”

Example Request (using curl):

```
curl -X POST -d "username=newuser&ogpassword=secure123&repassword=secure123" http://
↪yourdomain.com/register
```

Example Successful Response (renders login.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <p style="color: green;">NOTICE: Please login using previously created username␣
↪and password.</p>
    <form method="POST" action="/login">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Login">
    </form>
    </body>
</html>
```

Example Error Response (Username already exists):

```
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
</head>
<body>
    <h1>Create New Account</h1>
    <p style="color: red;">ERROR: This username already exists. Please use a␣
↪different one.</p>
    <form method="POST" action="/register">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="ogpassword">Password:</label><br>
        <input type="password" id="ogpassword" name="ogpassword"><br>
        <label for="repassword">Confirm Password:</label><br>
        <input type="password" id="repassword" name="repassword"><br><br>
        <input type="submit" value="Register">
    </form>
    </body>
</html>
```

Example Error Response (Passwords do not match):

```
<!DOCTYPE html>
<html>
```

(continues on next page)

(continued from previous page)

```html
<head>
    <title>Register</title>
</head>
<body>
    <h1>Create New Account</h1>
    <p style="color: red;">ERROR: The passwords did not match.</p>
    <form method="POST" action="/register">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="ogpassword">Password:</label><br>
        <input type="password" id="ogpassword" name="ogpassword"><br>
        <label for="repassword">Confirm Password:</label><br>
        <input type="password" id="repassword" name="repassword"><br><br>
        <input type="submit" value="Register">
    </form>
    </body>
</html>
```

`GET /logout`

Logs the user out.

This endpoint handles user logout functionality. When accessed via a GET request, it removes the *user_id* from the session, effectively logging the user out of their profile. It then redirects the user to the main index page with a success message.

Session Modification:

- Clears the *user_id* key from the user's session.

Response Body:

- **Content-Type:** `text/html`
- Renders the *index.html* template with an *errorMsg* parameter indicating successful logout.

Example Response:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <p>Successfully logged out of profile</p>
    </body>
</html>
```

3.3 Artifact Endpoints

`GET /`

Handles GET requsts to the root path and serves the main HTML home page containing the document generation form.

Undocumented

`GET /index`

Handles GET requsts to the /index path and serves the main HTML home page containing the document generation form.

Undocumented

POST /index

Description: Handles the submission of the artifact generation form. Based on the form data, it either returns an error message or triggers the artifact generation process and redirects to the output page.

Request Body (Form Data):

The request body is sent as *application/x-www-form-urlencoded* and contains the following fields:

- *artifact_type* (integer, required): An integer representing the type of artifact to generate.
- *model_selection* (integer, required): An integer representing the chosen language model.
- *artifact_parameters* (string, required if *artifact_type* is not 4): Free-form text providing parameters for the artifact generation.
- *opord_orientation* (string, required if *artifact_type* is 4): Orientation field for OPORD generation.
- *opord_situation* (string, required if *artifact_type* is 4): Situation field for OPORD generation.
- *opord_mission* (string, required if *artifact_type* is 4): Mission field for OPORD generation.
- *opord_execution* (string, required if *artifact_type* is 4): Execution field for OPORD generation.
- *opord_admin* (string, required if *artifact_type* is 4): Administration field for OPORD generation.
- *opord_logistics* (string, required if *artifact_type* is 4): Logistics field for OPORD generation.
- *opord_command* (string, required if *artifact_type* is 4): Command and Signal field for OPORD generation.

Request Body Example (Generic Artifact):

```
POST /index HTTP/1.1
Content-Type: application/x-www-form-urlencoded

artifact_type=2&model_selection=1&artifact_parameters=Provide%20a%20brief%20summary
↪%20of%20the%20topic.
```

Request Body Example (OPORD Artifact):

```
POST /index HTTP/1.1
Content-Type: application/x-www-form-urlencoded

artifact_type=4&model_selection=2&opord_orientation=Terrain%20and%20Weather...&
↪opord_situation=Enemy%20forces...&opord_mission=Conduct%20an%20attack...&opord_
↪execution=Phase%201...&opord_admin=Supply%20point...&opord_
↪logistics=Transportation...&opord_command=Commander's%20intent...
```

Response (Redirect - 302 Found):

On successful form submission and artifact generation, the server typically redirects the user to the */output* route (not documented here) to display the generated artifact.

Status Codes

- 302 Found – Found :description: Redirects to the */output* page upon successful artifact generation.

Response (Error - 200 OK with HTML):

If the *artifact_type* or *model_selection* are missing in the form data, the server returns the *index.html* template with an error message.

Status Codes

- 200 OK – OK :contenttype text/html: :example:

```
<!DOCTYPE html>
<html>
<head>
```

(continues on next page)

(continued from previous page)

```
    <title>Artifact Generator</title>
</head>
<body>
    <h1>Generate Artifact</h1>
    <p style="color: red;">ERROR: Please select an␣
↪artifact, model type, and give a prompt.</p>
    </body>
</html>
```

GET /output

Renders the output.html template.

This endpoint is responsible for displaying the main output of the web application. It fetches no external data but directly renders the content defined in the *output.html* template.

Response Body:

- **Content-Type:** `text/html`
- The response body will contain the HTML content of the *output.html* template. This template likely includes the structure, styling, and dynamic content to be displayed to the user.

Example Response:

```
<!DOCTYPE html>
<html>
<head>
    <title>Output</title>
</head>
<body>
    <h1>Here is the output!</h1>
    </body>
</html>
```

GET /my_artifacts

Displays the logged-in user's generated artifacts.

This endpoint retrieves and displays a list of artifacts that have been generated by the currently logged-in user. It requires the user to be authenticated (i.e., having a *user_id* in the session).

Session Requirements:

- Requires the *user_id* to be present in the session to identify the logged-in user.

Response Codes:

- **200 OK (User has artifacts):**
  - **Response Body:**
    - **Content-Type:** `text/html`
    - Renders the *my_artifacts.html* template, passing a list of the user's generated artifacts as the `artifacts` context variable. The structure and content of these artifacts will depend on your application's data model.
- **200 OK (User has no artifacts):**
  - **Response Body:**
    - **Content-Type:** `text/html`
    - Renders the *index.html* template with an *errorMsg* indicating that there are no artifacts associated with the user's account.

- **200 OK (User not logged in):**
  - **Response Body:**
    - **Content-Type:** `text/html`
    - Renders the *login.html* template with an *errorMsg* prompting the user to log in.

Example Response (User logged in with artifacts - simplified example):

```
<!DOCTYPE html>
<html>
<head>
    <title>My Artifacts</title>
</head>
<body>
    <h1>My Generated Artifacts</h1>
    <ul>
        <li>Artifact 1: ... (details of artifact 1) ...</li>
        <li>Artifact 2: ... (details of artifact 2) ...</li>
        </ul>
    </body>
</html>
```

Example Response (User logged in, no artifacts):

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <p>NOTICE: There are no artifacts associated with this account.</p>
    </body>
</html>
```

Example Response (User not logged in):

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <p style="color: green;">NOTICE: Please login to see your generated artifacts.</
↪p>
    <form method="POST" action="/login">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Login">
    </form>
    </body>
</html>
```

3.4 Documentation Endpoint

`GET /docs/(path: ` *filename*`)`

Serves static documentation files.

This endpoint serves static files from the Sphinx-generated documentation build directory (*../docs/build/html*). The *filename* part of the URL path is dynamically used to locate and serve the requested file.

Path Parameters:

- **filename:** The path to the requested static file within the *../docs/build/html* directory. This can include subdirectories (e.g., *_static/style.css* or *index.html*).

Response Body:

- **Content-Type:** The Content-Type of the response will depend on the type of file being served (e.g., *text/html*, *text/css*, *image/png*, *application/javascript*).
- The response body will contain the content of the requested static file.

Example Request:

- `GET /docs/index.html`: Retrieves the main index page of the documentation.
- `GET /docs/_static/style.css`: Retrieves the stylesheet for the documentation.
- `GET /docs/api.html`: Retrieves the API documentation page.

Example Successful Response (for /docs/index.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>Your Project Documentation</title>
    <link rel="stylesheet" href="_static/style.css" type="text/css" />
</head>
<body>
    <div class="document">
      <div class="documentwrapper">
        <div class="bodywrapper">
          <div class="body" role="main">
            <h1>Welcome to Your Project's Documentation!</h1>
            <div class="toctree-wrapper compound">
              <ul>
                <li class="toctree-l1"><a class="reference internal" href="api.html
↪">API Reference</a></li>
              </ul>
            </div>
          </div>
        </div>
      </div>
      <div class="clearer"></div>
    </div>
</body>
</html>
```

Note: The actual response body will vary greatly depending on the specific *filename* requested. This example shows a typical Sphinx-generated *index.html* file.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

HTTP ROUTING TABLE

PYTHON MODULE INDEX