Open-Source Report

Proof of knowing your stuff in CSE312

Guidelines

Provided below is a template you must use to write your reports for your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository**: Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we need to see the code you're referring to as well.
- **License Type**: Three letter acronym is fine.
- **License Description**: No need for the entire license here, just what separates it from the rest.
- **License Restrictions**: What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

FLASK

General Information & Licensing

Code Repository	https://github.com/pallets/flask
License Type	BSD 3-Clause License
License Description	-The BSD 3-Clause License has three main restrictions: Redistributions of the source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

License Restrictions	• Liability • Warranty


Magic

The Flask framework automatically handles the parsing of HTTP headers in incoming requests. The request object provides a convenient and easy-to-use interface for accessing these headers, although the provided code does not explicitly do so. The primary focus of this code is on managing routes, handling form data, and interacting with a database to manage users, courses, and questions. It can intake data in multiple formats such as form data, JSON data, and HTTP headers.

https://github.com/hchou3/CabinFever/blob/main/app.py

app = Flask(__name__)
The name parameter passed to the Flask class enables it to locate resources within the file system. Flask utilizes folders like templates and statics to store and access HTML and CSS files. By providing name as a parameter, the class can effectively locate files within the directory relative to where the class is instantiated.

In the app.py file, the Flask class is instantiated on the 7th line in app.py, as the route function within the class will be employed to direct website directories.

The Flask class is available in the app.py file of the Flask repository, located at https://github.com/pallets/flask/blob/main/src/flask/app.py, starting from line 92 until the end of the file. The class can accept parameters, following the format of the Scaffold class, which can be found at https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/scaffold.py#L751, spanning from line 62 to the end of the file. When only a single parameter, name, is passed, the class utilizes the provided file directory to locate other folders, such as templates and statics, as mentioned previously.

It is important to note that the class itself does not possess any specific functionality until it is invoked.

flask.render_template():
The render_template function, invoked from the Flask class, is utilized to display an HTML template. This function searches for the HTML file within the templates folder, leveraging the directory where the class was initialized. This function is called whenever there is a need to render a pre-designed HTML file.

When using Flask, after establishing a TCP socket and starting the Flask application, a client sends an HTTP request to the server. Upon receiving the request, Flask processes it, and when the associated route handling function is executed, it can invoke the render_template() function https://github.com/pallets/flask/blob/main/src/flask/templating.py at line 139. Render_template() searches for the specified HTML template file within the application's "templates" folder. Utilizing various internal mechanisms, including the Jinja templating engine at line 150, render_template() dynamically fills out the HTML template with the provided data at line 151. Once the rendering process is complete, the resulting HTML content is returned as the response to the client's request. Render_template is called on lines 65, 80, 98, 113, 124, 135, 145, 168, 177, 187, 195, and 256 to send the html template to the server in https://github.com/hchou3/CabinFever/blob/main/app.py.

flask.session():
The session refers to the period between a client logging in and logging out of the server. Any essential data that needs to be preserved during this session is securely stored in a temporary directory on the server. Its primary purpose is to enable access to the stored session data whenever it is required. In summary, flask.session serves as a vital tool for

handling session-related data throughout your codebase.

Flask serializes the data and securely signs it using a secret key. This serialized and signed data is then stored in a cookie. The cookie is sent back to the client's browser, where it is stored locally. In line 12 of our app.py https://github.com/hchou3/CabinFever/blob/main/app.py.

The https://github.com/pallets/flask/blob/main/src/flask/sessions.py flask.session implementation initializes sessions by creating cookies to store user data. This determines whether the user is a current existing user or if the cookies should be null if the user leaves or signs out. Line 48 creates the cookie session if the user requests the data from the server and line 90 is when the cookie gets set to null, where the user left or signed out of their account. Line 276 is where the cookies are being salted for security purposes. Flask.session is used on lines 75, 76, 90, 91, 102, 106, 118, 128, 139, 157, 163, 164, 181, 199, 207, 215, 223, 244, 245, 249, 250, 290, 317, 318, 325, and 327 in https://github.com/hchou3/CabinFever/blob/main/app.py. Session is called whenever data is being accessed or created with cookies. This ensures security for users and it allows us to access their data privately.

flask.redirect():
The flask.redirect() function is used to redirect the user to a different endpoint on the web. It's a way to redirect the client to a different URL with a specific HTTP response status, typically 301 for permanent redirection, or 302 for temporary redirection.

The redirect function is called from the werkzeug library(https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/utils.py) on lines 221. This function takes a url as a parameter and creates a response object(line 255) using the function in https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/wrappers/response.py#L66(line 66).

Redirect is used in lines 56, 72, 82, 133, 144, 186, 195 to redirect users in app.py https://github.com/hchou3/CabinFever/blob/main/app.py.

flask.request():
Once the socket connection is established, the web server, such as Werkzeug, receives the HTTP request from the client. The server then parses the raw HTTP request data and extracts relevant information, such as the request method, headers, URL, and any submitted form data. The request module in Flask interacts with the Werkzeug library, which in turn utilizes the BaseRequest class to parse the HTTP request data. The BaseRequest class, found in the base.py file of the Werkzeug repository, handles the core logic of parsing the request data and populating attributes like headers, cookies, and form data.

Request is used in lines 69, 70, 71, 84, 85, 86, 150, 151, 231, 232, 234, 235, 236, 237, and 241 to request data from the server and cookie data in app.py https://github.com/hchou3/CabinFever/blob/main/app.py. To parse the request data for necessary data, these function are used from the Werkzeug library https://github.com/pallets/werkzeug/blob/main/src/werkzeug/formparser.py.

flask.flash():
flask.flash() is a function that allows you to send temporary messages between views. These messages get stored, and once they are accessed, they are removed from the

message queue. Flash.flash is called from https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/helpers.py. The function stores the message in a special part of the user's session, which is a kind of secure, server-side cookie. When the server sends a response to the client, it includes this flashed message in the response.

It is used in lines 74, 132, 178, 180, 184, 194, 208 in app.py https://github.com/hchou3/CabinFever/blob/main/app.py.

flask.abort():
The flask.abort() function is used to abort a request and return an HTTP error code to the client immediately. The function takes in the HTTP status code to return. After the TCP socket is created and the HTTP request is received, the Flask application uses a routing mechanism to match the URL of the request to a corresponding view function. The abort() function in Flask is responsible for generating an HTTP error response

app.route():
App.route is a decorator function used in the Flask class and is used to route the user to the directed path. It binds specific functions to a specific route, so when the route is accessed the function is called. The code is located at https://github.com/pallets/flask/blob/main/src/flask/app.py

Route is used in https://github.com/hchou3/CabinFever/blob/main/app.py for every path created. It is called in lines 41,45, 60,78, 93, 103, 115, 126, 149, 156, 165, 172, 188. It intakes a string which represents the path as well as an optional array input which represents what methods are used (GET, POST), If there is no input for the array it is defaulted to only GET.

Within route these functions are called:

The getattr() function is a Python built-in function that returns the value of a named attribute of an object, if it exists. getattr() is used to parse data and check for the existence of certain attributes.

The process of upper casing the elements of the GET method and finding the required_methods in view_func is related to HTTP method handling. This means that the function is making sure the methods being used are valid and that they meet any specific conditions or requirements needed for the view function to operate correctly.

url_rule_class is used to build an instance of Rule or a Rule subclass, which is an object describing a URL rule. This includes details such as the string of the URL rule, the methods allowed, and the endpoint. The URL rule is then added to the URL map, which is a list of all the URL rules in the Flask application.

Once the URL rule is defined and added to the URL map, the corresponding endpoint is identified. The endpoint function parses the returned HttpRequest, which will then be used to check with the .route() parameter to determine if the appropriate function should be run.

The import from Werkzeug (https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py) provides additional functionality to Flask, including HTTP request handling and server capabilities. Werkzeug is a comprehensive WSGI web application library; it can handle things like requests, responses, cookies, and other HTTP-related actions.

SQLAlchemy:
This process begins after establishing a TCP socket connection to the database server. From that point on, SQLAlchemy handles various tasks, such as executing queries, managing transactions, and mapping Python objects to database tables.

Werkzeug.security.generate_password_hash:
Once a TCP socket is created and the necessary communication channels are established, the process of generating a password hash begins. It utilizes cryptographic algorithms and techniques to ensure the strength and integrity of the resulting hash.

In the werkzeug library https://github.com/pallets/werkzeug/blob/main/src/werkzeug/security.py in line 83 is where the function is defined. In this function the password is salted and then hashed to ensure security. Our code utilizes this function on line 72 in our app.py https://github.com/hchou3/CabinFever/blob/main/app.py to salt and hash the user's password.

Werkzeug.security.check_password_hash:
The user's password is initially hashed and stored securely in a database during user registration. This can be accomplished using a hash function like bcrypt or SHA-256. When a user attempts to log in, their provided password needs to be compared with the stored hashed password in the database. This is where check_password_hash comes into play. The check_password_hash function takes two arguments: the stored hashed password from the database and the user's inputted password during login. It then performs the necessary operations to verify whether the inputted password matches the stored hashed password.

To accomplish this, check_password_hash internally utilizes the same hash function used during password hashing. In the werkzeug library https://github.com/pallets/werkzeug/blob/main/src/werkzeug/security.py in line 120 the function is defined. It extracts the salt and the hash algorithm information from the stored hashed password. The function then applies the hash algorithm and salt to the user's inputted password, generating a new hash. Finally, check_password_hash compares the newly generated hash with the stored hashed password. If they match, it indicates that the inputted password is correct. We call this function in our app.py https://github.com/hchou3/CabinFever/blob/main/app.py in line 89. In this part of the code it checks if the user logging in matches the username and hashed password.

Html:
HTML (Hypertext Markup Language) prevents potential security vulnerabilities such as Cross-Site Scripting (XSS) attacks by encoding special characters that have a specific meaning in HTML. The function takes a string as input and returns a new string where special characters, such as '<', '>', and '&', are replaced with their corresponding HTML ascii entities ('<', '>', '&', etc.). This ensures that the HTML tags and entities within the user-generated content are not interpreted as actual HTML tags or code, but instead displayed as plain text.

Json:
JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for machines to parse. It is based on a subset of the JavaScript language and is often used to transmit data between a server and a web application as an alternative to XML. In our code, we use JSON to receive data for our questions and courses. For example, in line app.y, we pack the information for a newly made questionnaire in a JSON format string. Github: (https://github.com/python/cpython/blob/main/Lib/json/__init__.py)

```
question_data = {
        'text': question_text,
        'course_id': course_id,
        'question_id': question.id,
        'answers': answer_id_dict }
```

We also use a JSON file as a template for storing secrets for the tokens.
In line 9 to 13 in app.y, the program loads the config.json template as a string object, and we assign the ‘SECRET_KEY’ to the database.

```
config = json.load(open('config.json'))

app = Flask(__name__)
app.config['SECRET_KEY'] = config['secret']
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tophat.db'
db = SQLAlchemy(app)
socketio = SocketIO(app)
```

```
from flask_socketio import SocketIO, join_room, emit
```

SocketIO: A class that provides the core functionality for Flask-SocketIO. It is used to create a SocketIO server instance that can handle WebSocket connections and events.
Github: (https://github.com/miguelgrinberg/Flask-SocketIO.git)

We initialize a SocketIO object to allow users to post questions to the classroom.
Github:(https://github.com/miguelgrinberg/Flask-SocketIO/blob/288119a11664d887c47522509d010f502ff742e8/src/flask_socketio/__init__.py#L5)
“socketio = SocketIO(app) “ ←

join_room: A function that allows a client to join a particular room on the server. Rooms are used to group clients that are interested in the same events or data.
Github:(https://github.com/miguelgrinberg/Flask-SocketIO/blob/288119a11664d887c47522509d010f502ff742e8/src/flask_socketio/__init__.py#L1013)

emit: A function that allows the server to send a message to a client or a group of clients. It can be used to send various types of data (e.g., strings, integers, JSON objects) and to specify the event name and recipient(s). This is used to send questions to the server.
Github:(https://github.com/miguelgrinberg/Flask-SocketIO/blob/288119a11664d887c47522509d010f502ff742e8/src/flask_socketio/__init__.py#L401)

The function create_question, occurs when the user creates a short answer or multiple choice question. To send the newly made question to the client, we call emit:

```
socketio.emit('new_question', question_data, room=str(course_id))
```

The first input 'new_question', is the event, it is a label sent to the socket server
The second input is the “question_data”, which is a JSON object containing the text, course ID, question ID, and the answers.
The third input is the course_id, received as an input for create_question representing the ID of the course object.

from datetime import timedelta

Datetime is a python import that provides classes for dates and times. We use the ‘timedelta’ class to represent time durations for our users.
Github: (https://github.com/python/cpython/blob/main/Lib/datetime.py)

Timedelta is a function representing the amount of time between two different dates or clock times. This function is used to set time limits for login sessions and to update session times, checking for users that have been idle for >1 hour.
Github:
(https://github.com/python/cpython/blob/563c7dcba0ea1070698b77129628e9e1c86d34e2/Lib/xmlrpc/client.py#L15)

When a /login from the front GET or POST request is received, we set the app’s session to timedelta(hours=24). Meaning the user session has a maximum time of 24 hours.

```
if user and check_password_hash(user.password, password):
        session['user_id'] = user.id
        session.permanent = True

        app.permanent_session_lifetime = timedelta(hours=24)
```

Before receiving the request, our code checks if the time has been updated. This logs out users that have been inactive for over an hour.

```
@app.before_request
def before_request():
    session.permanent = True
    app.permanent_session_lifetime = timedelta(hours=1)
    session.modified = True
```