Technical Documentation for Bee Aware Forum

Team QuadTree

This document is intended to help new developers quickly get started with this project.

1. Overview of Development Technologies

To set up the runtime environment, please refer to the ***Usage Guide***.
Development Language: Python
Development Framework: **Flask**
Frontend Technology: Jinja Templates
Database: MySQL

2. What is Flask?

Flask is a lightweight web development framework.
Routing, Rendering Templates, and Jinja are the three most important elements of this project.

Recommended Reference Documents:
Welcome to Flask — Flask Documentation (3.0.x) (palletsprojects.com)

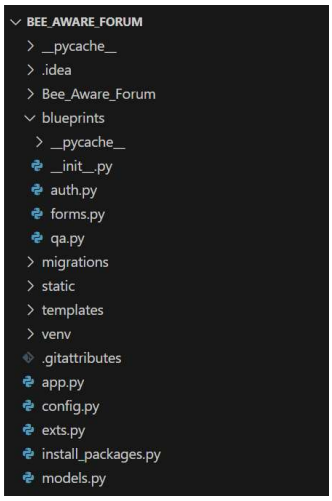
Routing: How the Backend process data
https://flask.palletsprojects.com/en/3.0.x/quickstart/#routing

Rendering Templates: How Processed Data is Rendered to the Frontend
https://flask.palletsprojects.com/en/3.0.x/quickstart/#rendering-templates

Jinja: The Frontend Template Used in this Project
https://flask.palletsprojects.com/en/3.0.x/templating/

3. Introduction to Non-functional Modules

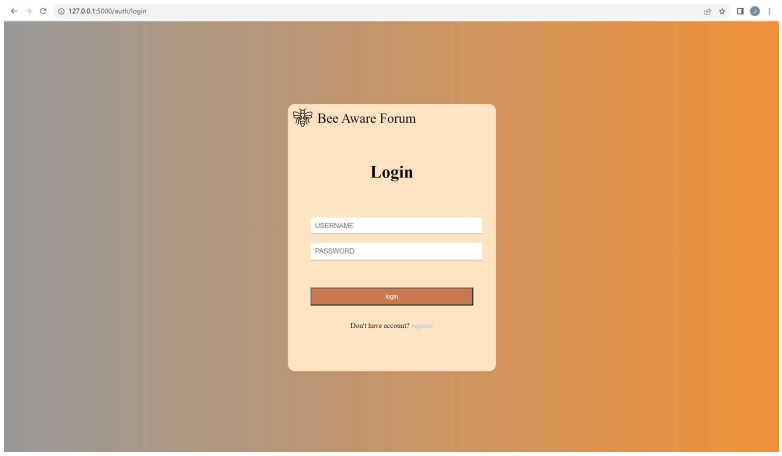
app.py: Launches the web server.
config.py: Defines database connection variables.
models.py: Defines database models (tables).
form.py: Defines form classes.
Static: Stores multimedia data such as images in the folder.



4. Introduction to FourTypical Functional Modules

4.1 User Login

When a user is on the login page, the URL is http://127.0.0.1:5000/auth/login.



After filling out the form information and clicking the "Login" button, the frontend page uses the POST method to send the form data to the web server.

```
<form method="POST">
    <div class="form-wrapper">
        <input type="text" name="username" placeholder="username" class="input-item">
        <input type="password" name="password" placeholder="password" class="input-item">
        <button class="btn btn-primary btn-lg" style="text-align: center">login</button>
    </div>
</form>
<div class="msg">
    Don't have account?
    <a href="{{ url_for('auth.register') }}">register</a>
</div>
```

Based on the current URL, the web server calls the following route function. It compares the provided user account and password with the data in the database for authentication. If the authentication is successful, it redirects to the main page; otherwise, it redirects back to the current page.

```
@bp.route("/login", methods = ['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template("login.html")
    else:
        form = LoginForm(request.form)
        if form.validate():
            username = form.username.data
            password = form.password.data
            user = UserModel.query.filter_by(username=username).first()
            if not user:
                print("No user")
                return redirect(url_for("auth.login"))
            if check_password_hash(user.password, password):
                session['user_id'] = user.id
                return redirect(url_for("qa.index"))
            else:
                print("Wrong password")
                return redirect(url_for("auth.login"))

        else:
            print(form.errors)
            return redirect(url_for("auth.login"))
```

4.2 How the main page displays multiple posts

When a user is on the main page, relevant post data is retrieved from the database using SQL and stored in some variables. The ***render_template*** method is then called to render the frontend template. For example, ***questions=questions*** means that the list of posts obtained through an SQL query (on the right-hand side, questions) is assigned to the variable ***questions*** (on the left-hand side, questions). Then Questions is a variable in the frontend template.

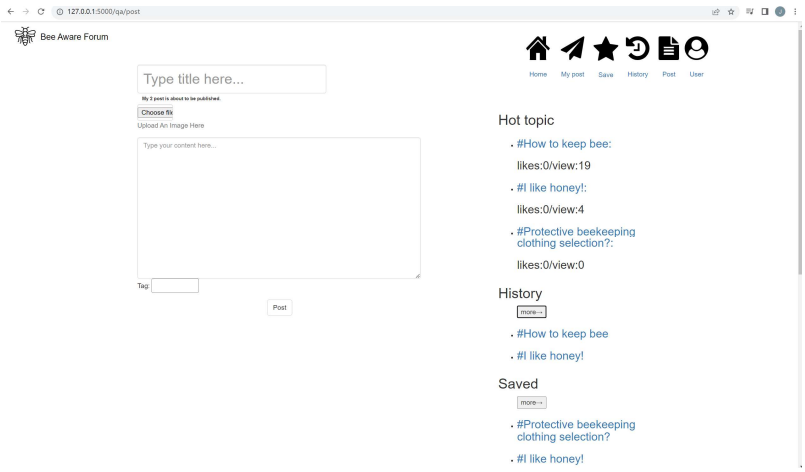
```python
@bp.route("/", methods=['POST', 'GET'])
def index():
    per_page = 8
    now = datetime.now()
    days_ago = now - timedelta(days=5)
    page = request.args.get('page', 1, type=int)
    questions = QuestionModel.query.order_by(QuestionModel.create_Time.desc()).paginate(page=page, per_page=per_page)
    hot_posts = QuestionModel.query.filter(QuestionModel.create_Time > days_ago).order_by(
        QuestionModel.NumOfLikes.desc()).all()
    if not g.user:
        histories = " "
        saves = " "
        my_posts= " "
    else:

        histories = History.query.filter_by(user_id=g.user.id).order_by(History.created_time.desc()).all()
        saves = SaveModel.query.filter_by(user_id=g.user.id).order_by(SaveModel.created_time.desc()).all()
        my_posts=QuestionModel.query.filter_by(author_id=g.user.id).order_by(QuestionModel.create_Time.desc()).all()
    return render_template("home_page.html", questions=questions, hot_posts=hot_posts, my_posts=my_posts,now=now, days_ago=days_ago,histories=histories, saves=saves, author=g.user)
```

The questions in ***{% for question in questions %}*** is a variable passed to this page by the ***'/' route function***. Through the loop, it embeds information for each post into the HTML template one by one. This way, it achieves arranging multiple posts sequentially.

```jinja
{% for question in questions %}
    <li>
        <table width="100%" border="0%" style="border: 2px solid black;">
            <td colspan="0">
                <div style="font-size: larger; float: right;">
                    <p style="margin-top: 10px;margin-bottom: 5px;">Date: {{ question.create_Time }}</p>
                    <p>view: {{ question.NumOfView }}</p>
                </div>
                <div style="float: left; margin-left: 10px; margin-top: 10px;">
                    <a href="{{ url_for('qa.qa_like', qa_id=question.id) }}"
                       style=" margin-bottom: 0px; vertical-align: middle;">
                        <img src="/static/ulike.svg" alt="Like"/>
                    </a>
                    <p style="font-size:larger; margin-bottom: 0px; margin-left: 4px; vertical-align: middle;">{{ question.NumOfLikes }}</p>
                    <a href="{{ url_for('qa.qa_dislike', qa_id=question.id) }}"
                       style="vertical-align: middle;">
                        <img src="/static/udislike.svg" alt="Dislike"/>
                    </a>
                </div>
                <h1 style="margin-left: 50px; font-weight: bold">{{ question.title }}</h1>
                <!-- 插入图片-->
                <div class="postimgs">
                    {% if question.image_filename %}
```

4.3 Post publishing

On the Post Publishing page, the current URL is http://127.0.0.1:5000/qa/post. After filling out the form on the previous page, when you click "post," it uses the POST method to send the form data to the web server.



The web server calls the following route function corresponding to the current URL. It first stores the information from the form in multiple variables and then saves these variables to the database.

```python
@bp.route("/qa/post", methods=['GET', 'POST'])
def public_qa():
    if not g.user:
        return redirect(url_for("auth.login"))
    else:
        post_records = QuestionModel.query.filter_by(author_id=g.user.id).count()+1
        now = datetime.now()
        days_ago = now - timedelta(days=5)
        hot_posts = QuestionModel.query.filter(QuestionModel.create_Time > days_ago).order_by(
            QuestionModel.NumOfLikes.desc()).all()
        histories = History.query.filter_by(user_id=g.user.id).order_by(History.created_time.desc()).all()
        saves = SaveModel.query.filter_by(user_id=g.user.id).order_by(SaveModel.created_time.desc()).all()
        my_posts = QuestionModel.query.filter_by(author_id=g.user.id).order_by(
            QuestionModel.create_Time.desc()).all()
        if request.method == 'GET':
            return render_template("post.html",post_records=post_records,hot_posts=hot_posts, now=now,days_ago=days_ago, histories=histories, my_posts=my_posts,saves=saves, author=g.user)
        else:
            form = QuestionForm(request.form)
            if form.validate():
                title = form.title.data
                content = form.content.data
                NumOfLikes = 0
                NumOfView = 0
                tag = request.form['tag']
                image = request.files.get('image')
                image_filename = save_image(image)
                if image_filename:
                    question = QuestionModel(
                        title=title, content=content, NumOfLikes=NumOfLikes, NumOfView=NumOfView,
                        author=g.user, tag=tag, image_filename=image_filename
                    )
                else:
                    question = QuestionModel(
                        title=title, content=content, NumOfLikes=NumOfLikes, NumOfView=NumOfView,
                        author=g.user, tag=tag
                    )
                db.session.add(question)
                db.session.commit()
                if g.user.exp+30>=150:
                    g.user.exp=g.user.exp+30-150
                    g.user.level=g.user.level+1
                else:
                    g.user.exp=g.user.exp+30
                db.session.commit()
                return redirect(url_for("qa.index"))
            else:
                print(form.errors)
                return redirect(url_for("qa.index"))
```

For example, the following code stores variables containing post-related information into the ***question table***. Images, as multimedia data, have their filenames stored in the database. The actual images themselves are saved in the static subfolder of the project. When calling them, they are accessed using relative paths from the project files.

```python
157                 question = QuestionModel(
158                     title=title, content=content, NumOfLikes=NumOfLikes, NumOfView=NumOfView,
159                     author=g.user, tag=tag, image_filename=image_filename
160                 )
```

4.4 Click to collect post into favorite

On the main page, clicking the "Save" button triggers the ***'/qa/clicksave/<qa_id>' route function***, and the current post's ID is passed through the URL.

```
    <a onclick="confirmSave()"
       href="{{ url_for('qa.qa_clicksave', qa_id=question.id) }}"
       style="float:left; margin-left: 20px;">
        <img src="/static/usave.svg" alt="Save Icon"
             style="vertical-align: middle; margin-right: 5px;">
        <p style="font-size: large; display: inline-block;">Save</p>
    </a>
</div>


<script>
    function confirmSave() {
        var result = confirm("Save the post");
        if (result) {
        }
    }
</script>
```

The relationship between a post and a user, such as when a user saves a post, is stored in the ***user_question_save*** table corresponding to the ***SaveModel***.

```
@bp.route("/qa/clicksave/<qa_id>")
def qa_clicksave(qa_id):
    if not g.user:
        return redirect(url_for("auth.login"))
    else:
        question = QuestionModel.query.get(qa_id)
        save_record = SaveModel.query.filter_by(user_id=g.user.id, question_id=question.id).first()
        if not save_record:
            save_record = SaveModel(user_id=g.user.id, question_id=question.id)
            db.session.add(save_record)
            db.session.commit()
        return redirect(url_for("qa.index", question=question))
```

user_question_save is a relationship table. The pair of a user's ID and the ID of the saved post serve as the primary key. Both of these variables are foreign keys, as shown in the diagram below.

```
class SaveModel(db.Model):
    __tablename__ = 'user_question_save'
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), primary_key=True)
    question_id = db.Column(db.Integer, db.ForeignKey('question.id'), primary_key=True)
    created_time = db.Column(db.DateTime, default=datetime.now)
    question = db.relationship('QuestionModel', backref=db.backref('user_save', cascade='all, delete-orphan'))
    user = db.relationship('UserModel', backref=db.backref('question_save', cascade='all, delete-orphan'))
```

5. Database ER Diagram