ABOUT BLOGLITE

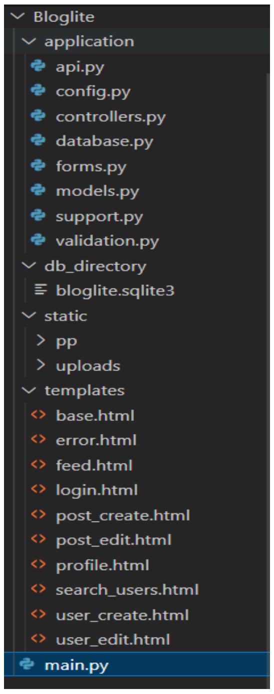
BlogLite is a hosted web application, and has the following features.

- Signup in the app. While signing up, user must provide a profile picture. (Core)
- Login to the app with the right combination of username/password. (Core)
- Proper login system, powered by flask-login. Multiple users can work on the app simultaneously (Optional)
- Allow user to create and edit (only text) posts/blogs* about anything, including an image. (Core)
- Upon logging in, the user will be shown a list of posts created by users (s)he follows. (Core)
- Basic search capability on username or a part thereof. Latter might result in multiple users being listed. (Core)
- Ability to follow/unfollow one or more users, at a time. (Core)
- View the profile of currently logged-in user with basic statistics of the user. (Core)
- Ability to comment on and like/unlike posts (Recommended)
- APIs for users allows user to add, view, update or delete users (Recommended)
- APIs for posts allows user to add, view, update or delete posts (Recommended)
- API for posts allows user to view user feed from the database. (Recommended)

BLOGLITE APPLICATION DESIGN

BlogLite GUI is built using *Flask*, and has an SQLite backend powered by *Flask-SQLAlchemy*.

File organization of the project is as follows.



```
Bloglite
  application
    api.py
    config.py
    controllers.py
    database.py
    forms.py
    models.py
    support.py
    validation.py
  db_directory
    bloglite.sqlite3
  static
    > pp
    > uploads
  templates
    base.html
    error.html
    feed.html
    login.html
    post_create.html
    post_edit.html
    profile.html
    search_users.html
    user_create.html
    user_edit.html
  main.py
```

- *main.py* is the web server. Uses flask.
- configuration/setup changes are made using *config.py*
- *controllers.py* define all the endpoints of the web application, and contains most of the back-end code. Rest of it are distributed between
    - *models.py* (defines the tables in *SQLAlchemy* classes),

- *database.py* (hooks up the database to models.py),
- *support.py* (helper functions and messages),
- *validation.py* (handles exceptions while using APIs)

- *forms.py* contains classes that represents the various front-end forms. Uses *flask-wtf*.
- *bloglite.sqlite3*, contained in the *db_directory* folder is the SQLite database used in the application.
- *Templates* folder contains the HTML templates used by all endpoints in the application. Uses Jinja extensively. Specifically, all templates have been "extended" from *base.html* and hence ensures the same look and feel for all pages in the application.

Find the list of tables present in the database and the endpoint URLs, below.

Table	Purpose (stores)	Primary	Foreign	Unique
user	users in the system.	user_id.	-	name
post	posts made by users.		user_id	-
follows	user follower info	(user_id, follow_user_id)	user_id follow_user_id	-
post_like	likes of posts	(user_id, post_id)	user_id post_id	-
post_comment	comments on posts	comment_id	user_id post_id	-

Endpoint	API	Purpose
/, /feed	/api/feed/<user_id>	Home page. Displays user feed.
/login		Login page. Flask-session.
/logout		Logs out. Removes session.
/post/create/	/api/post/create/<user_id> (POST)	Add post. Picture upload.
/post/edit/<post_id>	/api/post/edit/<post_id> (PUT)	Edit post. Add comments.
	/api/post/<post_id> (DELETE)	Delete post.
	/api/post/<post_id> (GET)	Get information about posts.
/user/create	/api/user (POST)	Signup. Requires PP of the user.
	/api/user/<user_id> (PUT)	Edit user. Used only by API.
	/api/user/<user_id> (DELETE)	Delete user.
	/api/user/<user_id> (GET)	Get information about users.
/profile/<user_id>		Profile. List user posts.
/search_users/<term>		Search by name(full or part). Follow one or more users.
/post/<post_id>/like		Like/unlike posts.

KEY-POINTS ABOUT THE APP USAGE.

- During signup, uploading a profile picture is mandatory. If the picture is not uploaded, error is displayed. If the uploaded file is not a picture, it uses default image.
- Upon successful upload, file is copied to the *static/pp* folder and filename of the image is stored in in the database. */profile* endpoint retrieves the filename and reconstructs the file-path and display the image in the front-end. Once uploaded, the profile picture can be only changed through an API call.
- User is automatically routed to the login page, if not already logged in. Once logged in, they're redirected to the requested (next) page.
- The password is stored in hashed form (*flask-security*) in database. Sessions are maintained by *Flask*. Multiple users can work from different browsers. User sessions expire after 5 minutes of inactivity.
- User authorization to edit profile/posts is decided based on the session information.
- APIs support CRUD operations for users and posts. There also exists an API to get the user feed.
- In the case of any DB operation failure, a common error gets displayed.
- Validation of user-provided data is done at the front-end and back-end (server/database) layer.
- During creation of a post, if an image is not provided or if uploaded file is not a picture, it uses default image.

More details about the project in this video.

Submitted by: Anand K.Iyer, Email: 21f1001185@student.onlinedegree.iitm.ac.in