Jobs! Documentation
ITEC649
Author: Amith Raj and Steve Cassidy

The contents of the App are as follows:

- `database.py` - code to create the database and load sample data and reset the database to known values
- `main.py` - main application script
- `interface.py` – contains stubs for the front-end interface
- `users.py` - contains stubs for the login, log out and positions interface
- `positions.json` - sample job listings taken from the github jobs API.
- `static` - directory containing a sample CSS file and some HTML files
- `views` – directory containing templates of base and index html files

The `database.py` module contains code to create the database tables that you will need for the assignment and load in some sample data from the `positions.json` file. **You need to run this file to create the database when you start the project.** You can also re-run it if you want to reset your database to a known state.

The database contains the following tables:

- `users`: fields `nick` (user nickname), `password`, `avatar` (URL of an avatar image)
- `sessions`: fields `sessionid`, `usernick`, where usernick is a reference to the nick field in the users table
- `posts`: fields `id`, `timestamp`, `usernick`, `content`

There are also two other tables `votes` and `follows` that are not used in the project but you can use if you wish to implement further features.

Each user is identified by a nickname that is stored in the `nick` field in the `users` table. This field is used as a foreign key in other tables to refer to the user. To select data from both tables you will need to do a query with a join.

The `positions` table deserves further comment. The `id` field is an auto-incremented integer id for each post, you don't need to give this a value, it will automatically get a new unique value when you insert a row. The `timestamp` field will also default to the current time and date in the format '2015-02-20 01:45:06' if you don't provide a value when you add a row. So, to add a row to the `positions` table you just need:

```
sql = """INSERT INTO positions (owner, title, location, company, description)
VALUES (?, ?, ?, ?, ?)"""
cursor.execute(sql, [usernick, title, location, company, description])
db.commit()
```

Front-end

- **Welcome to Jobs**

As a visitor to the site, when I load the home page I see the title text *"Welcome to Jobs"*. This is implemented in the index.html and login_index files

- **About Page Link**

As a visitor to the site, when I load the home page I see a link to another page with the link text *"About"*.
This is implemented in the index.html and login_index files

- **About Page**

As a visitor to the site, when I click on the link *"About"* I am taken to a page that contains the site manifesto, including the words *"Jobs is a new, exciting, job posting service like nothing you've seen before!"*. This is implemented in the "about" file in views directory

- **Home Page List of positions**

  As a visitor to the site, when I load the home page (URL /) I see a list of up to 10 positions in order of their timestamps, most recent first. Each position must include the position timestamp, title, owner and the first 100 characters of the text of the position description. At the end of each listing should be a link to view the entire position with the text "Read More" and the url `/positions/DD` where `DD` is the position id.

  The list of positions that appears should be the result of a call to `position_list` which will extract the most recent 10 positions from the database. ***It is implemented in index.html using a loop and parsing the 'positions' value from main function. Using the parsed 'positions' value to display the most recent job details in the home page. Since the limit is 10, the for loop gets executed 10 times and 10 posts will be available for the user to see. Description is less than 100 words and has escaped html. The URL for readmore link is /positions/DD where DD is the jobid***

- **Position Page**

  As a visitor to the site, I can click on the "Read More" link after a position description on the home page, I see a page with the full description of the position at the URL `/positions/DD` where `DD` is the position id. This is implemented in descexpander.html file. ***Using the parsed 'description' value to display the job details and descriptions with escaped html from expand function from main.py***

  The position page should contain all of the database fields for the position, laid out in a readable way. It should also contain a link to return to the main page.

- **Login Form**

  As a visitor to the site, when I load the home page, I see a form with entry boxes for *nick* and *password* and a button labelled *Login*.

  - The login form will have the id 'loginform' and will use fields named 'nick' and 'password'.
  - The `action` of the login form will be `/login`.

- **Logging In**

  As a registered user, when I enter my user nickname (eg. Bobalooba) and password (bob) into the login form and click on the Login button, the response is a redirect to the main application page (/). When my browser loads that page, I see the normal home page with the login form replaced by a picture and the message *"Logged in as Bobalooba"* and a button labelled *Logout*.

  - The response generated by the successful login action is a redirect (302 Found) response that redirects the user to the home page.
  - The redirect response also includes a cookie with the name `sessionid` that contains the name of the user appended by “session” keyword
  - The logout button will be in a form with id `logoutform` and have an `input` submit field with the name `logout`.

- **Failed Login**

  As a registered user, when I enter my email address but get my password wrong and click on the Login button, the page I get in response contains a message "Login Failed, please try again". The page also includes another login form. This is implemented in login_error file in views.

- **Posting a Job**

  As a registered user, I can fill out a form on the main page to create a new job listing (position), when I submit the form I am redirected to the main page and my new position appears in the list.

  - The form to post a new position will have the id `postform`
  - The action attribute for the form will be the URL `/post`

- **Logout Button**

  As a registered user, once I have logged in, every page that I request contains my name and the logout button.

- **Logging Out**

  As a registered user, once I have logged in, if I click on the Logout button in a page, the page that I get in response is the site home page which now doesn't have my name and again shows the login form.

  - The response to a logout request is again a redirect (302 Found) response that redirects the user to the home page.
  - When I now request the home page, I see the login form again because the session has been deleted.

Back-end:

Interface.py

All procedures will be implemented in the module `interface.py`.

When we refer to a database connection below we mean the connection returned by the `sqlite3.connect` function. In all cases this will be passed into your functions. The connection is created either by your main application or the test code.

- **position_list**

  There is a function `position_list(db, limit=10)`. `db` is a database connection and the optional argument limit is an integer. The function returns a list of tuples representing the positions stored in the database, each tuple contains: `(id, timestamp, owner, title, location, company, description)`. Messages are returned in reverse order of the timestamp, most recent first.

- **position_get**

  There is a function `position_get(db, id)`. `db` is a database connection and `id` is an integer position identifier. The function returns a tuple representing the position with the given id, the tuple contains: `(id, timestamp, owner, title, location, company, description)`.

  If the `id` does not match a record in the database, the function should return `None`.

- **position_add**

  There is a function `position_add(db, usernick, title, location, company, description)`. `db` is a database connection, the argument `usernick` is a user name and `title, location, company` and `description` are the values of the respective fields in the positions table. The function adds the new position to the database.

If `usernick` does not a user in the users table, no new record should be created and the function should return `False`

If the record is added successfully, the function should return `True`.

Users.py

This file adds four procedures in a new `users` that deal with authenticating users and managing user sessions. They act as an interface to the `users` and `sessions` tables in the database. These procedures are implemented in the module `users.py`; a version of this file with just the procedure stubs is provided for you.

1. **check_login**

   There is a procedure `check_login` in the `users` module that takes three arguments, a database connection, a user nick and a password, and returns True if the password is correct for this user and False otherwise. Note that the password is stored in the database as a one-way hash. You can use the `password_hash(text)` to generate a one-way hash from a password (imported from the database module).

2. **generate_session**

   There is a procedure `generate_session` in the `users` module that takes two arguments, a database connection and a user nick. If the nick doesn't correspond to an existing user, then it returns None. If this user doesn't already have an active session (an entry in the sessions table) then a new entry is created. If there is an existing entry, then the existing session id is retrieved. The procedure then creates a cookie in the Bottle `response` with the name `sessionid` and a value of the session id for this user. The procedure returns the `sessionid`.

3. **delete_session**

   There is a procedure `delete_sessions` in the `users` module that takes two arguments, a database connection and a user nick. The procedure removes all entries for this user in the sessions table. It does not return a value.

4. **session_user**

   There is a procedure `session_user` in the `users` module that takes one arguments, a database connection, and returns the name of the logged in user if one can be identified or None if not. This is done by finding the session id from the cookie in the Bottle `request` if present and using it to look up the user in the sessions table.

5. **insert_session**

   There is a procedure `insert_session` in the `users` module that takes two arguments, a database connection and a user nick. The procedure inserts session for this user in the session table if session doesn't exist and returns the sessionid inserted.

6. **exist_user**

   There is a procedure `exist_user` in the users module that takes two arguments, a database connection and a user nick. The procedure checks for this user in the user_table. Returns True if user exists, else returns False

7. **get_avatar**

   There is a procedure `get_avatar` in the users module that takes two arguments, a database connection and a user nick and returns the `avatar` of the logged in user if one can be identified or None if not. This is done by finding the session id from the cookie in the Bottle `request` if present and using it to look up the user in the sessions table.

Main.py

1. **routing "/"**

   index : There is a procedure `index` in the main module that takes one argument, a database connection Passing positions data to the index.html page by calling the position_list function from interface.py sending that info to the index.html where it is separated and displayed accordingly. Returns the template along with the most recent 10 positions to be displayed in the homepage. If the user has logged in, then render "login_index" template. Pass the username and avatar of the user to display in the template.

2. **Routing "/positions/<jobid>"**

   expand: There is a procedure `expand` in the main module that takes two arguments, a database connection and jobid. Passing position data to the descexpander.html page by calling the position_get function from interface.py sending that info to the descexpander.html where it is separated and displayed accordingly. Returns the template along with the data of the given job id. Page with the full description of the position at the URL /positions/jobid where jobid is the position id.

3. **Routing "('/login', method='POST')"**

   check_login: There is a procedure `check_login` in the main module that takes one argument, a database connection. Checking the Username and Password against the user's database. Generate a session for the user and display index.html, else create a session and redirect to "/" if it fails returns a fail page template

4. **Routing "/post"**

   add_database: There is a procedure `add_database` in the main module that takes one argument, a database connection. Logged in user can add new positions using position_add() from interface.py to add positions. After adding redirects to "/".

5. **Routing “/logout”**

logout: There is a procedure `logout` in the main module that takes one argument, a database connection. Username is retrieved from the session and session will be terminated using the delete_session from users.py After deleting redirects to “/”.