CSCI 3428 - Installation and Maintenance Guide

Group 4

Wednesday 11th December, 2019

1 User Access

The project will be hosted on the undergraduate computer science student's server at Saint Mary's University, and is publicly accessible at: `ugdev.cs.smu.ca/~group4`. The project has been developed and tested on the latest desktop versions of Chrome (`v.78`), Firefox (`v.70`), and Safari (`v.13.0.1`), though should also be compatible with older versions and mobile platforms, provided there is support for HTML5, CSS3, and Javascript.

On access, existing users will be prompted to log into the messaging system with their given account. The information entered into the appropriate text-fields will be validated against the credentials stored in the system's database, and allow or reject access to the system as appropriate. By default, the user's credentials are the following:

```
Username: FIRST_NAME
Password: ——————————
```

Authenticating new users, changing passwords, and adjusting any other user-related settings will be done by the administrator using the user-management panel that is run on the server, and is accessible via the web.

2 Administration Panel

Most day-to-day management of the system will be performed by the administrator using the included user-management panel. This includes adding new users or removing existing ones from the system, and making connections between users in the form of new conversations. Currently, the administration panel is running on port 8000 of the undergraduate computer science student's server at Saint Mary's university. It can be accessed by navigating to: `ugdev.cs.smu.ca:8000/admin/`.

IMPORTANT: via the administration panel, the entire conversation history of all users of the system is visible. There further exists functionality to change or delete the contents of previously sent messages. Malicious use of this panel threatens the privacy of the system's users, and must be protected against accordingly. The default password given above must be changed to a more secure alternative, and access to the administration panel must be limited exclusively to the system administrator.

To change the host port for the panel, begin by killing any active tasks on the desired port (e.g. 4433):

```
$ fuser -k 4433/tcp
```

Then, from within `~/project_master/backend/smumessaging`, initiate the panel by running a new task on the desired port (e.g. 9000):

```
$ nohup ./redis-server --protected-mode no &
$ daphne -b 192.168.2.12 -p 9000 smumessaging.asgi:application
```

The panel can then be accessed by navigating to `ugdev.cs.smu.ca:9000/admin/`, using the following log-in credentials:

`Username:` ————————
`Password:` ————————

User

From within the administration panel, add a new user by navigating to the `Users` tab, and selecting `Add user`. Enter the desired log-in credentials, and click `Save` to confirm. The table of existing users is listed at the bottom of the page. Note that `Staff Status=True` indicates that the user is able to access the administration panel using their login credentials. As per above, the sole staff member should be the system administrator.

To delete a user, select the tick box next to their name in the table at the bottom of the `Users` page. Change the action drop-down to `Delete selected users`, and press `Go` to confirm.

To change a user's password and permissions, select the name of the user from the table. Fill out the form provided in the first box to change their password, and use the scroll-menu in the `Permissions` section to alter their permissions. Information concerning the date the user was created, and the last time they accessed the system is available at the bottom of this page.

Chats

From the administration panel, initiate a new conversation by navigating to the `Chats` tab. Select `Add chat`, and select the two (or more) desired participants from within the `Participants` scroll-menu. Click the `Save` button at the bottom of the page to confirm. Note that it is possible to populate this conversation with existing messages via the `Messages` menu. However, this is an artifact of the design of the administration pane, and is not intended to (and indeed should not) be used.

All active conversations are enumerated in the table at the bottom of the `Chats` page, which can further be used to delete a conversation.

Other

The design of the administration panel is such that it allows additional functionality, including changing and deleting messages, creating groups, changing a user's authorisation token, etc. Many of these features are not implemented in the system itself, and can therefore be ignored.

3 Server Access

The project's files, databases, and server configuration can be accessed via `SSH`:

```
Username: ——————————
Password: ——————————
```

Currently, all the tools and databases required for managing the project are installed on the server. This includes `v.3.6.8` of Python3, which is used to install and manage the Django framework, and the latest version of SQLite. With SQLite's integration in the Django user-management platform, direct access to the project's databases is not required, and any necessary modifications can again be performed by the system-administrator in the administration panel, as outlined above.

The ReactJS framework was used to create the source files for the project's front-end, and while it is not required for the basic day-to-day running and administration of the system, any long-term changes to the project (including bug-fixes and adding functionality) will require that the server has a suitable JavaScript development environment available. See the installation procedure below for setting up a new environment.

3.1 Project Directories

The high-level directory structure for the server is outlined below. The web-facing `public_html` directory contains an optimised build version of the project, that condenses the various source and package files. While fully functional, these files are not (easily) readable, and are not intended to be modified directly. Changes should first be made to the source-code within the project directory, tested appropriately, and lastly built and included in `public_html`.

```
~
├── project_master
│   ├── backend
│   └── messagingSystem
└── public_html
    ├── index.css
    ├── index.html
    └── index.js
```

The system files are split into the front- and back-end components within the `project_master` directory. The `backend` directory is responsible for initialising the database and populating its tables with user input, handling user communication, and specifying any dependencies for the Django project. Some files and directories of note:

- `/chat` Configures the system's server
    - `/admin.py` Initialises tables for display in administration panel
    - `/api/serializers.py` Formats data for entry into database tables
    - `/consumers.py` Handles message sync. on load + sending/receiving messages
    - `/models.py` Database table initialisation and structure
    - `/routing.py` Defines URL and connection to websocket
- `/manage.py` Initialises Django's CLI (auto-generated)

- `/smumessaging` Django configuration (auto-generated)

```
project_master
└── backend
    ├── chat
    │   ├── __init__.py
    │   ├── __pycache__
    │   ├── admin.py
    │   ├── api
    │   │   ├── __init__.py
    │   │   ├── __pycache__
    │   │   ├── serializers.py
    │   │   ├── urls.py
    │   │   └── views.py
    │   ├── apps.py
    │   ├── consumers.py
    │   ├── migrations
    │   ├── models.py
    │   ├── routing.py
    │   ├── tests.py
    │   ├── urls.py
    │   └── views.py
    ├── db.sqlite3
    ├── manage.py
    ├── media
    ├── package-lock.json
    ├── package.json
    ├── requirements.txt
    ├── runtime.txt
    └── smumessaging
        ├── __init__.py
        ├── __pycache__
        ├── asgi.py
        ├── routing.py
        ├── settings.py
        ├── urls.py
        └── wsgi.py
```

The `messagingSystem` directory contains the source-files for the system's front-end, which specifies the structure and appearance of the web-page, and reads input from the users. Some files and directories of note include:

- `/public` Base files for system structure and appearance (linked to by other files)
- `/src` Main system directory
    - `/components` Contains CSS and ReactJS files for general + user-specific pages
        - `/Communication.css` Specifies appearance of conversation panel
        - `/Communication.js` Specifies structure of conversation panel
        - `/Login.css` Specifies appearance of login page
        - `/Login.js` Specifies structure of login page
        - `/UserInterface.css` Base appearance for interface

    - `/UserInterface.js` Base structure for interface
  - `/index.js` Sub-initialisation
  - `/models` Handles communication with backend and websocket integration.
  - `/router.js` Handles navigation between pages
  - `/routes/IndexPage.js` Handles user login and authentication
  - `/services/global.js` Handles requests from server
  - `utils/request.js` Handles return data from server

```
project_master
└── messagingSystem
    ├── package-lock.json
    ├── package.json
    ├── public
    │   ├── index.css
    │   └── index.html
    └── src
        ├── assets
        │   └── bg2.jpg
        ├── components
        │   ├── BobCommunication.css
        │   ├── BobUserInterface.css
        │   ├── Communication.css
        │   ├── Communication.js
        │   ├── Login.css
        │   ├── Login.j
        │   ├── MayCommunication.css
        │   ├── MayUserInterface.css
        │   ├── UserInterface.css
        │   ├── UserInterface.js
        │   ├── ValerieCommunication.css
        │   └── ValerieUserInterface.css
        ├── index.css
        ├── index.js
        ├── models
        │   ├── IndexPage.js
        │   └── UserInterface.js
        ├── router.js
        ├── routes
        │   └── IndexPage.js
        ├── services
        │   └── global.js
        ├── settings.js
        ├── utils
        │   └── request.js
        └── websocket.js
```

4 Migration and Maintenance

4.1 Django and Server

In order to migrate the project to a new server (or make and push any changes to the source-code), the host machine will require that Python3, MySQL, and a JavaScript development

environment is installed. Ensure first that the Client-URL (`curl`) command-line utility for Linux is installed:

```
$ curl -V
```

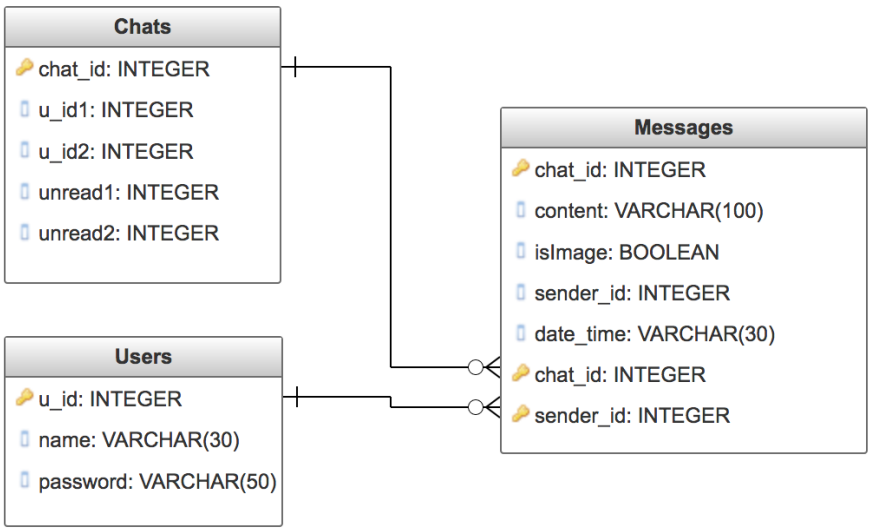
If not, install it using your system's appropriate package-installation tool. For Debian, Ubuntu, or related distributions, use:

```
$ sudo apt-get install curl
```

Most Linux distributions will include Python3 by default. Verify that it is installed, and install it if not. Preferably, select a distribution that includes the Python package manager pip, which will be used to install the Django framework. Otherwise, it can be obtained separately:

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
$ python get-pip.py
$ pip install Django
```

To configure the project's database, ensure that SQLite (`v.3.30.1` or later) is installed on the host-machine. The system's tables are structured as follows:



To run the server on the new host-machine, begin by ensuring that all required dependencies are first installed:

```
$ pip3 install -r
    requirements.txt
$ pip3 install django-allauth
$ pip3 install channels
$ pip3 install django-rest-auth
$ pip3 install channels_redis
$ pip3 install "whitenoise <4"
$ pip3 install service_identity
```

Lastly, run the server and backend:

```
$ redis-server
$ python manage.py runserver
```

4.2 React and Front-end

In order to maintain the project's source code, a JavaScript development environment will need to be present on the host machine (in particular, the React framework). Ensure that the newest version of the Node.js runtime environment is installed. If not, install Node.js and additional build tools:

```
$ sudo apt-get install nodejs
$ sudo apt-get install build-essentials
```

As of `v.8.10` of Node, and `v.5.6` of NPM (bundled in the Node installation), a new React project can be created using:

```
$ npx create-react-app my-app
$ cd my-app
$ npm start
```

A local instance of the project's front-end structure can be run locally to test changes to the structure or the appearance from within the `messagingSystem` directory. Use NPM to first install any missing dependencies, and then start the application:

```
$ npm install
$ npm start
```