Tutorial Part 4: Automated Testing

This tutorial begins where Tutorial 3 left off. We've built a simple chat server and now we'll create some automated tests for it.

Testing the views

To ensure that the chat server keeps working, we will write some tests.

We will write a suite of end-to-end tests using Selenium to control a Chrome web browser. These tests will ensure that:

- when a chat message is posted then it is seen by everyone in the same room
- when a chat message is posted then it is not seen by anyone in a different room

Install the Chrome web browser, if you do not already have it.

Install chromedriver.

Install Selenium. Run the following command:

```
$ python3 -m pip install selenium
```

Create a new file `chat/tests.py`. Your app directory should now look like:

```
chat/
    __init__.py
    consumers.py
    routing.py
    templates/
        chat/
            index.html
            room.html
    tests.py
    urls.py
    views.py
```

Put the following code in `chat/tests.py`:

```python
# chat/tests.py
from channels.testing import ChannelsLiveServerTestCase
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.wait import WebDriverWait


class ChatTests(ChannelsLiveServerTestCase):
    serve_static = True  # emulate StaticLiveServerTestCase

    @classmethod
    def setUpClass(cls):
        super().setUpClass()
        try:
            # NOTE: Requires "chromedriver" binary to be installed in $PATH
            cls.driver = webdriver.Chrome()
        except:
            super().tearDownClass()
            raise

    @classmethod
    def tearDownClass(cls):
        cls.driver.quit()
        super().tearDownClass()

    def test_when_chat_message_posted_then_seen_by_everyone_in_same_room(self):
        try:
            self._enter_chat_room("room_1")

            self._open_new_window()
            self._enter_chat_room("room_1")

            self._switch_to_window(0)
            self._post_message("hello")
            WebDriverWait(self.driver, 2).until(
                lambda _: "hello" in self._chat_log_value,
                "Message was not received by window 1 from window 1",
            )
            self._switch_to_window(1)
            WebDriverWait(self.driver, 2).until(
                lambda _: "hello" in self._chat_log_value,
                "Message was not received by window 2 from window 1",
            )
        finally:
            self._close_all_new_windows()

    def test_when_chat_message_posted_then_not_seen_by_anyone_in_different_room(self):
        try:
            self._enter_chat_room("room_1")

            self._open_new_window()
            self._enter_chat_room("room_2")

            self._switch_to_window(0)
            self._post_message("hello")
            WebDriverWait(self.driver, 2).until(
                lambda _: "hello" in self._chat_log_value,
```

```
                "Message was not received by window 1 from window 1",
            )

            self._switch_to_window(1)
            self._post_message("world")
            WebDriverWait(self.driver, 2).until(
                lambda _: "world" in self._chat_log_value,
                "Message was not received by window 2 from window 2",
            )
            self.assertTrue(
                "hello" not in self._chat_log_value,
                "Message was improperly received by window 2 from window 1",
            )
        finally:
            self._close_all_new_windows()

    # === Utility ===

    def _enter_chat_room(self, room_name):
        self.driver.get(self.live_server_url + "/chat/")
        ActionChains(self.driver).send_keys(room_name, Keys.ENTER).perform()
        WebDriverWait(self.driver, 2).until(
            lambda _: room_name in self.driver.current_url
        )

    def _open_new_window(self):
        self.driver.execute_script('window.open("about:blank", "_blank");')
        self._switch_to_window(-1)

    def _close_all_new_windows(self):
        while len(self.driver.window_handles) > 1:
            self._switch_to_window(-1)
            self.driver.execute_script("window.close();")
        if len(self.driver.window_handles) == 1:
            self._switch_to_window(0)

    def _switch_to_window(self, window_index):
        self.driver.switch_to.window(self.driver.window_handles[window_index])

    def _post_message(self, message):
        ActionChains(self.driver).send_keys(message, Keys.ENTER).perform()

    @property
    def _chat_log_value(self):
        return self.driver.find_element(
            by=By.CSS_SELECTOR, value="#chat-log"
        ).get_property("value")
```

Our test suite extends `ChannelsLiveServerTestCase` rather than Django's usual suites for end-to-end tests (`StaticLiveServerTestCase` or `LiveServerTestCase`) so that URLs inside the Channels routing configuration like `/ws/room/ROOM_NAME/` will work inside the suite.

We are using `sqlite3`, which for testing, is run as an in-memory database, and therefore, the tests will not run correctly. We need to tell our project that the `sqlite3` database need not to be in memory for run the tests. Edit the `mysite/settings.py` file and add the `TEST` argument to the **DATABASES** setting:

```
# mysite/settings.py
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
        "TEST": {
            "NAME": BASE_DIR / "db.sqlite3",
        },
    }
}
```

To run the tests, run the following command:

```
$ python3 manage.py test chat.tests
```

You should see output that looks like:

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 5.014s

OK
Destroying test database for alias 'default'...
```

You now have a tested chat server!

What's next?

Congratulations! You've fully implemented a chat server, made it performant by writing it in asynchronous style, and written automated tests to ensure it won't break.

This is the end of the tutorial. At this point you should know enough to start an app of your own that uses Channels and start fooling around. As you need to learn new tricks, come back to rest of the documentation.