**Project documentation for**

**Technological Basics II WS 2018/19**

# ◊ **Shortly (re-)introduce your idea**.

In this project I will build an application named **Stint**, a fun based movie archive pack with digital art and fringe interests.

It is implemented base on a similar project done for a book library[[1]](#footnote-1) that provides functions of a CRUD application on Python and Flask.

Stint can be downloaded at https://github.com/chanmadeleine/Stint or https://chanmadeleine.github.io/Stint/

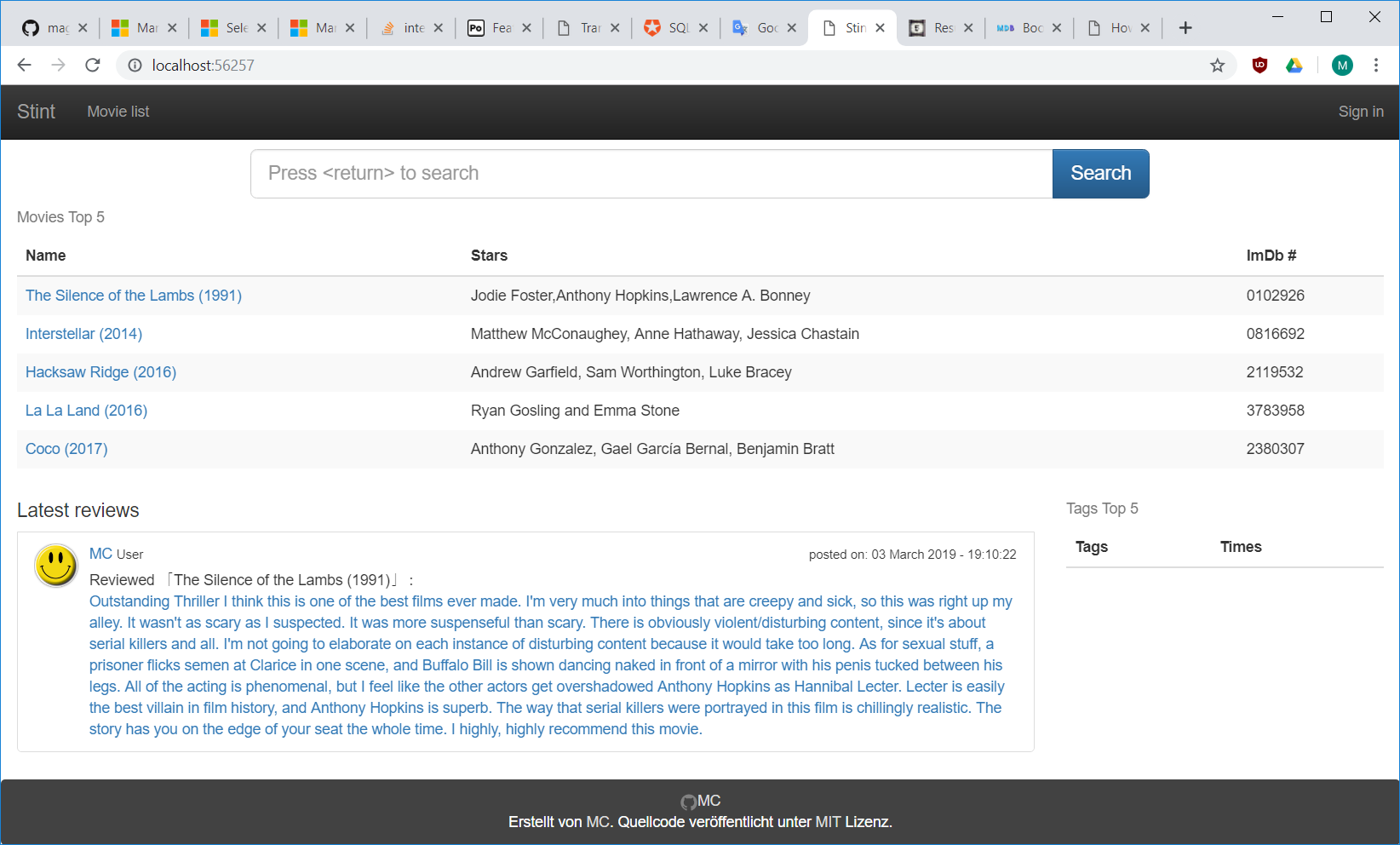
# ◊ **Describe the progress you made since your prototype**.

The project, at present, instead of just some non-functioning screens, has basic CRUD functions and decently layout movie screens. With the use of SQLAlchemy, data is stored in a local SQL Lite database and is now persistent across application restart. It accepts multiple users that can contribute to film reviews. A list of movies ranked by number of reviews is also created. Which makes it an **MVP** version of the project.

The database is initialized when none is present at application start up and populated with a few rows of sample records.

In the following paragraphs I will briefly introduce the main pages of Stint and their features.

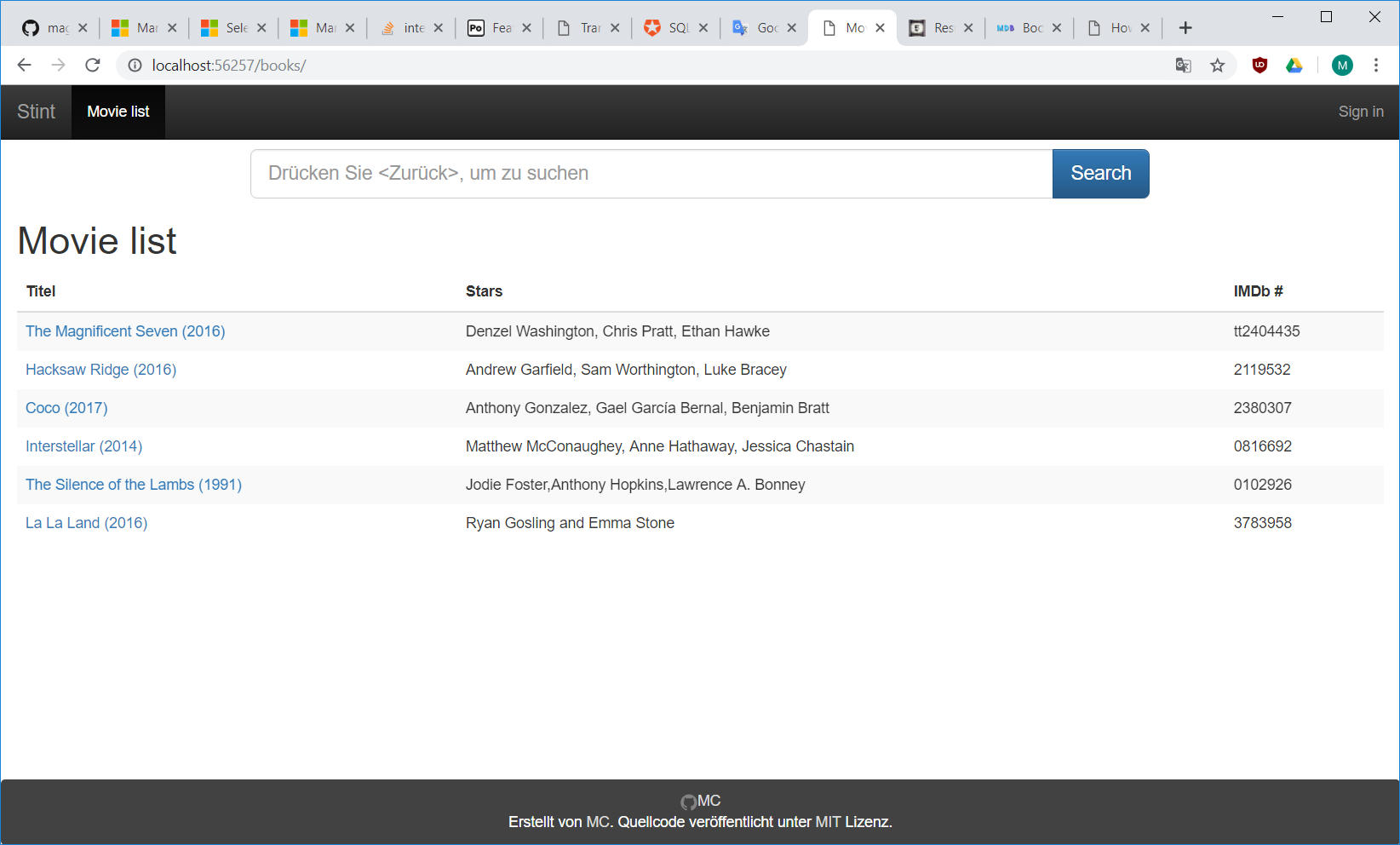
Stint’s main page:



The search box will match the entered substring to movie name, casts, and IMDb number.

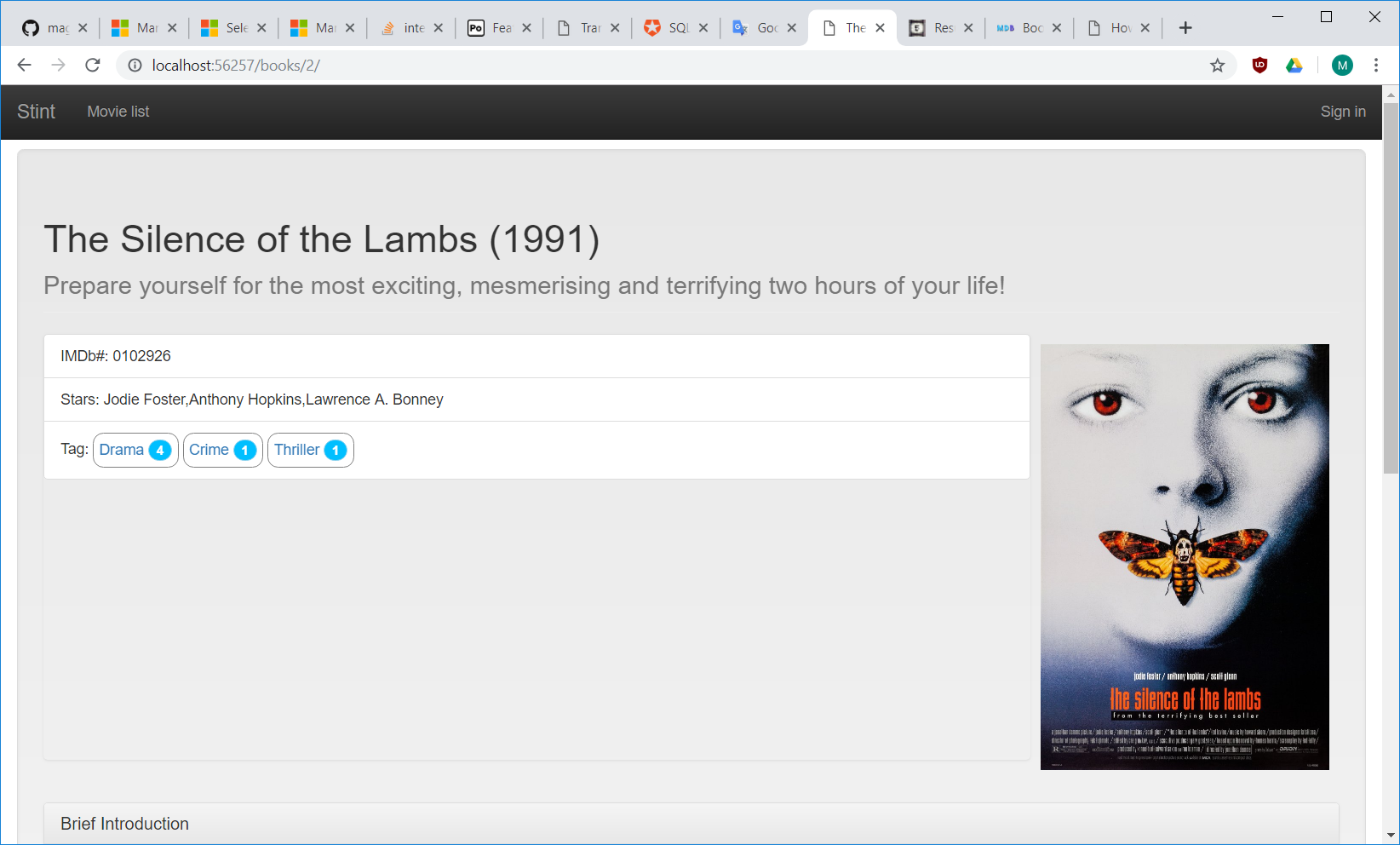
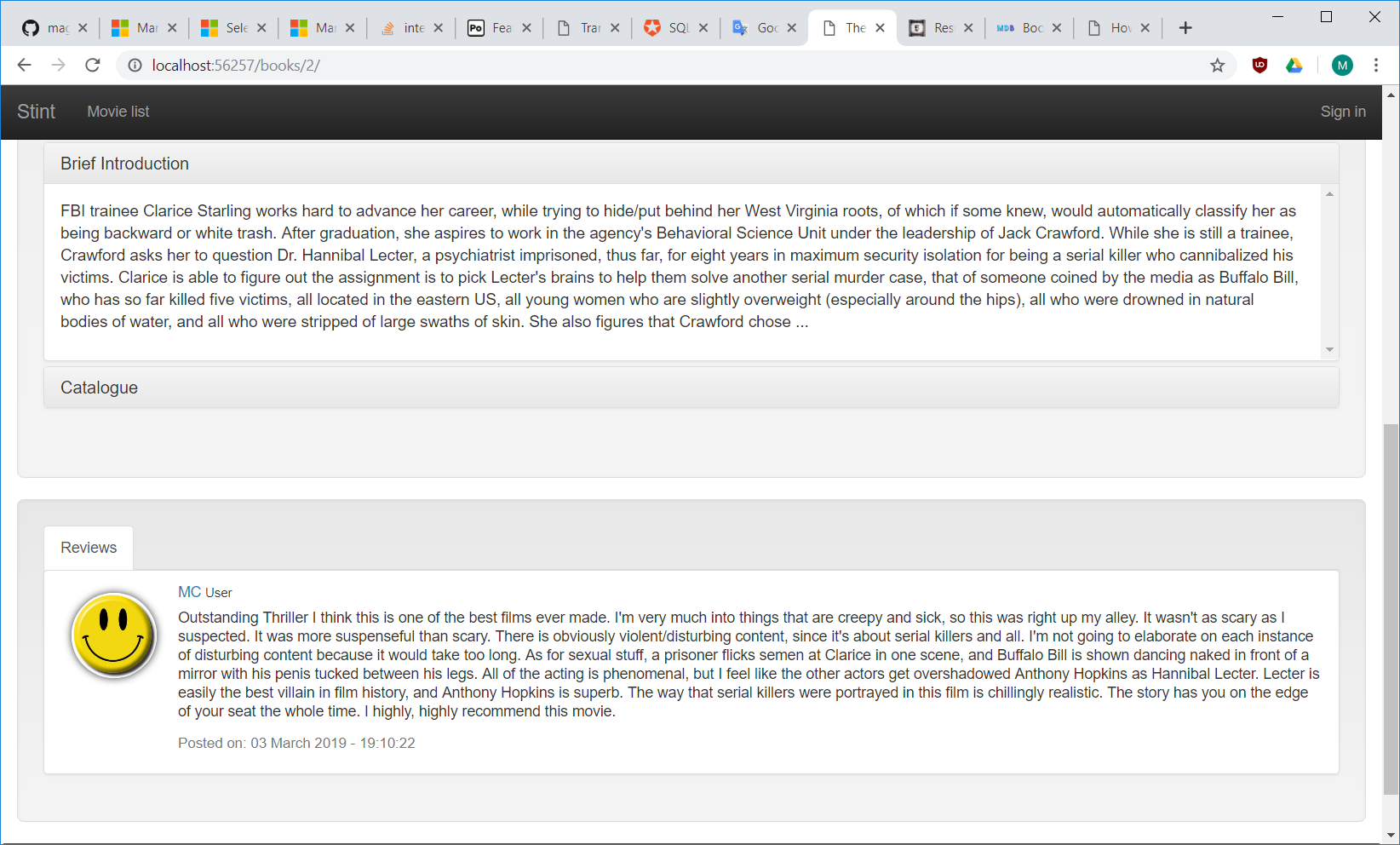
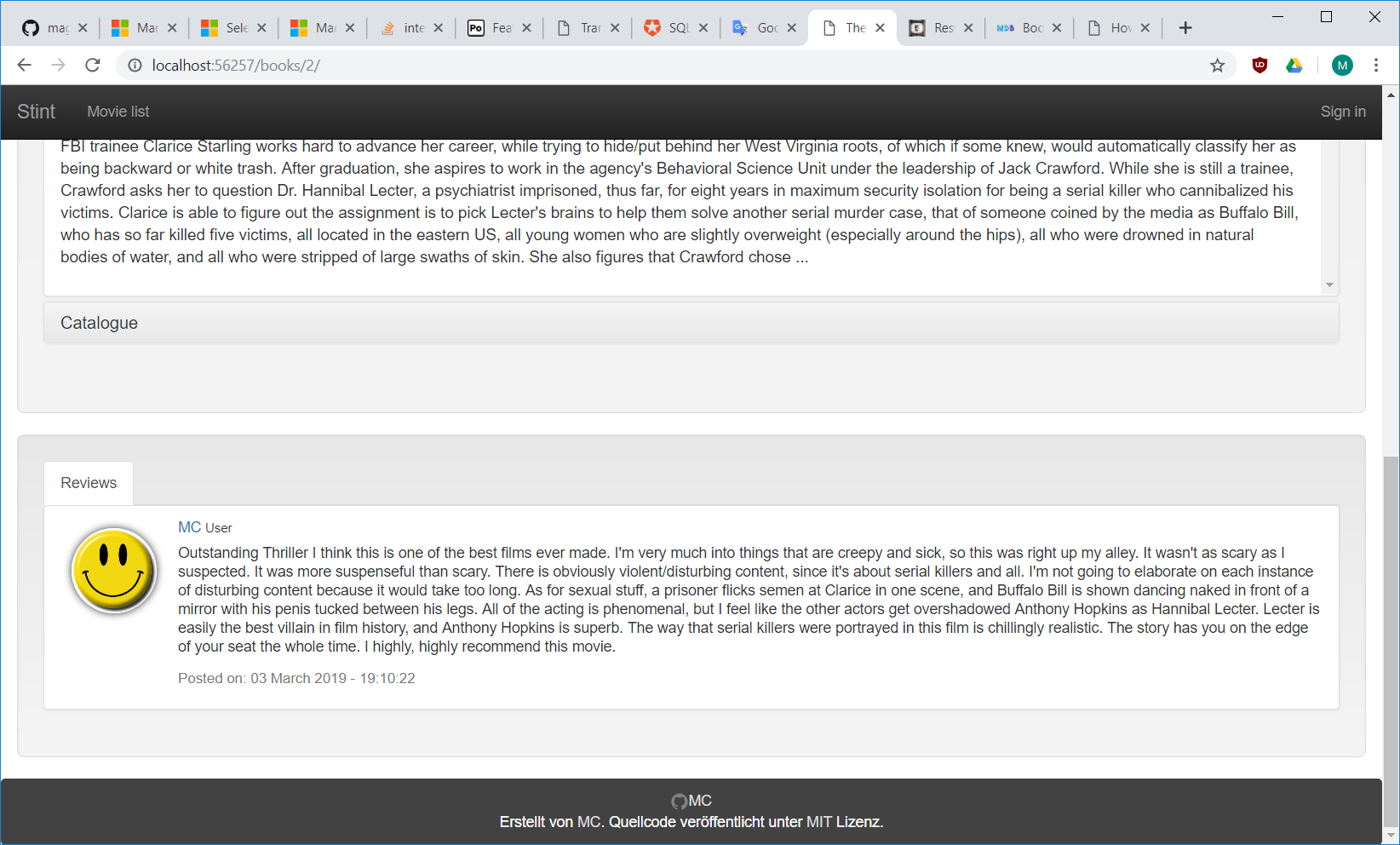
Click on any movie will bring up the film detail page.

Movie list

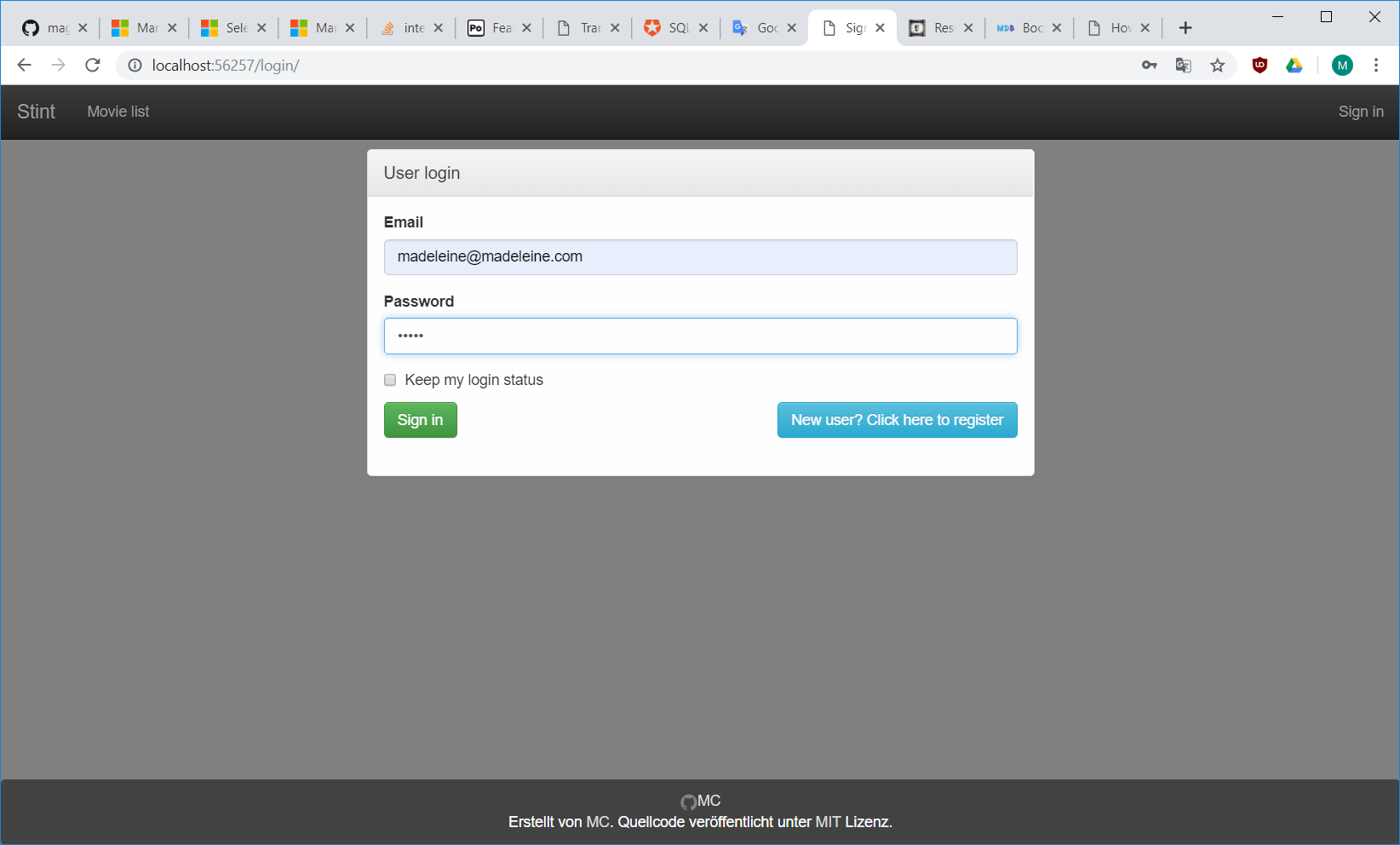


If no search string is given all films in the database will be shown.

Movie detail page

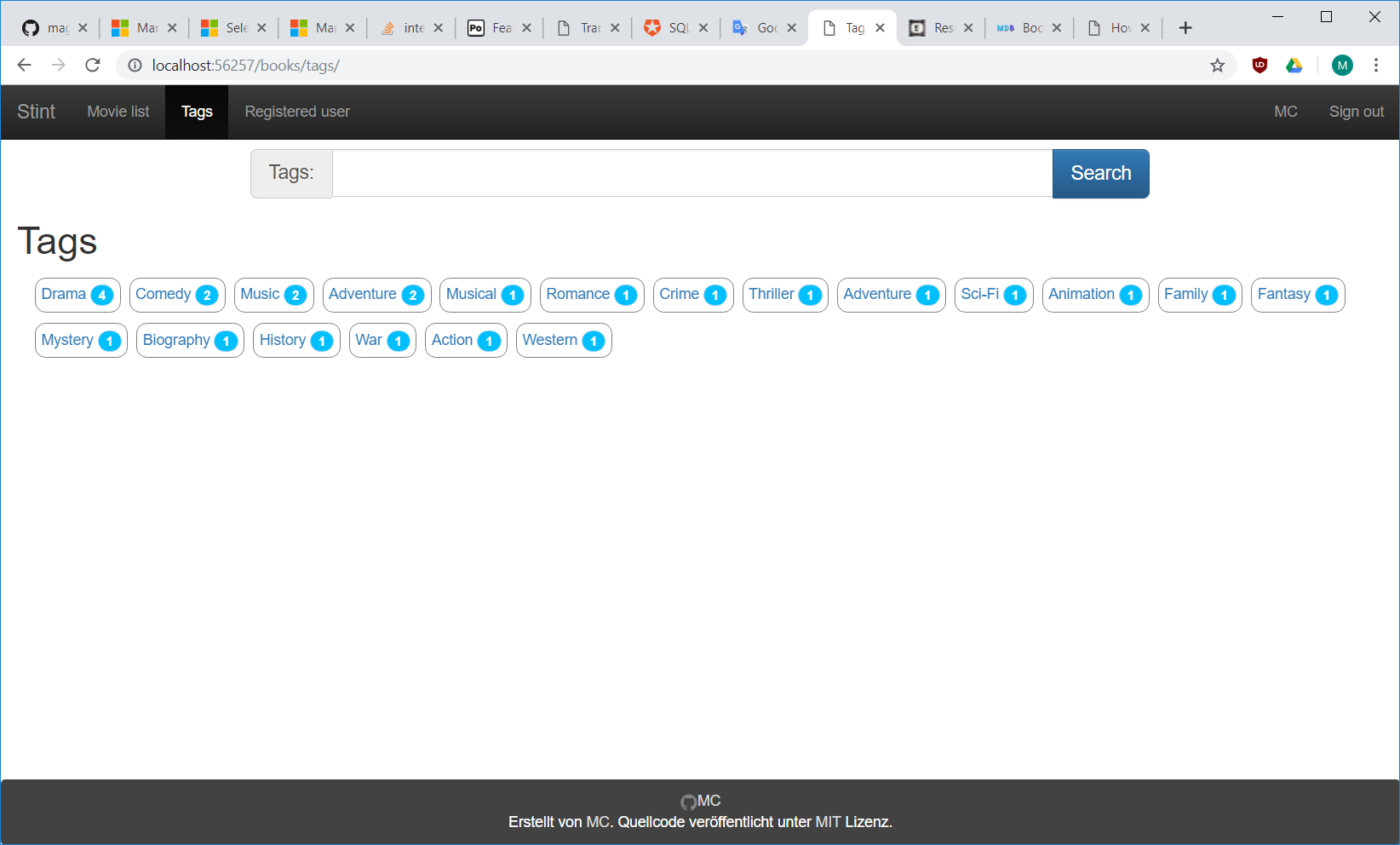
  

User login page



Logged in user can maintain tags and input reviews.

Tags



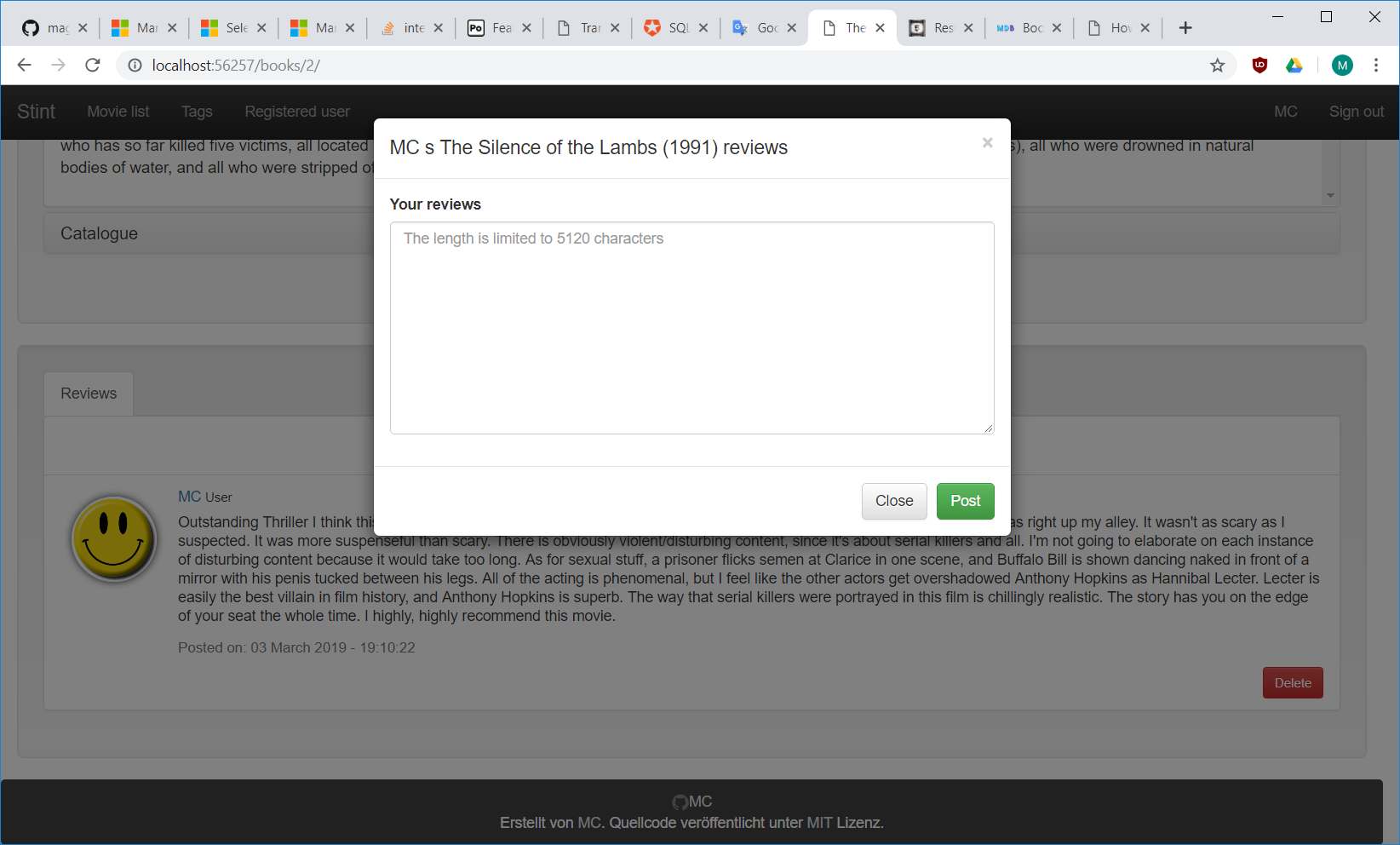
Tags are designed to have a many-to-many relation with movies, i.e. a movie can have many tags, and many movies can have the same tag. The database schema design is shown in the Implementation of static data through database section below.

The number next to the tags indicates films has that tag, click on it to list them.

You can also search multiple tags to narrow down the search, simply enter all the tags in the text box separated by a space.

Tags are removed when no more films are associated with it.

Posting a review



The reviewer has the option to delete their own review.

# ◊ **Describe and discuss the challenges you encountered during the development process**.

## Getting started with Python and Flask

First, learning a computer language is not easier than learning a natural language. Creating and editing a project and reading other’s code with it is like communicating to many people at higher levels in a natural language environment; it was exciting, fruitful, and yet exhausting. My journey with Flask has been full of thrills and spills.

Creating a web-based app

Selecting a web-based approach is a challenge in itself. There are too many things to go through before I can even start to work that includes understand the development and dependences of the solution that I choose and test out combination of program and tools and whether they work together. It is a trail and error process and much time was spent unproductive before a working combination was found.

Github

Use Git and Github repository is a very good skill to learn but the vast functions and interfaces also are demanding to get the hang.

Choosing a dev env to work with

I love syntax highlighting because it makes reading code so much more comfortable. However, looking for an IDE that will do syntax highlighting, works with Github and Flask is such a task, I heard that Eclipse is good, but setting up Eclipse to have everything I need including a debugger is not easy. Some editors like Textmate, Sublime text or Atom do not integrate with Flask or Github, or are easy to organize. So, Visual studio is chosen for its seeming simplicity.

## Brief technical description of my Flask app: **Stint**

**Stint** has multiple views and templates.

First app folders are created, then inside those folders there is a file named \_\_init\_\_.py that creates the Flask instance and loads the app's view:

from flask import Flask

main = Flask(\_\_name\_\_)

import . . .

## Templating with **Bootstrap**

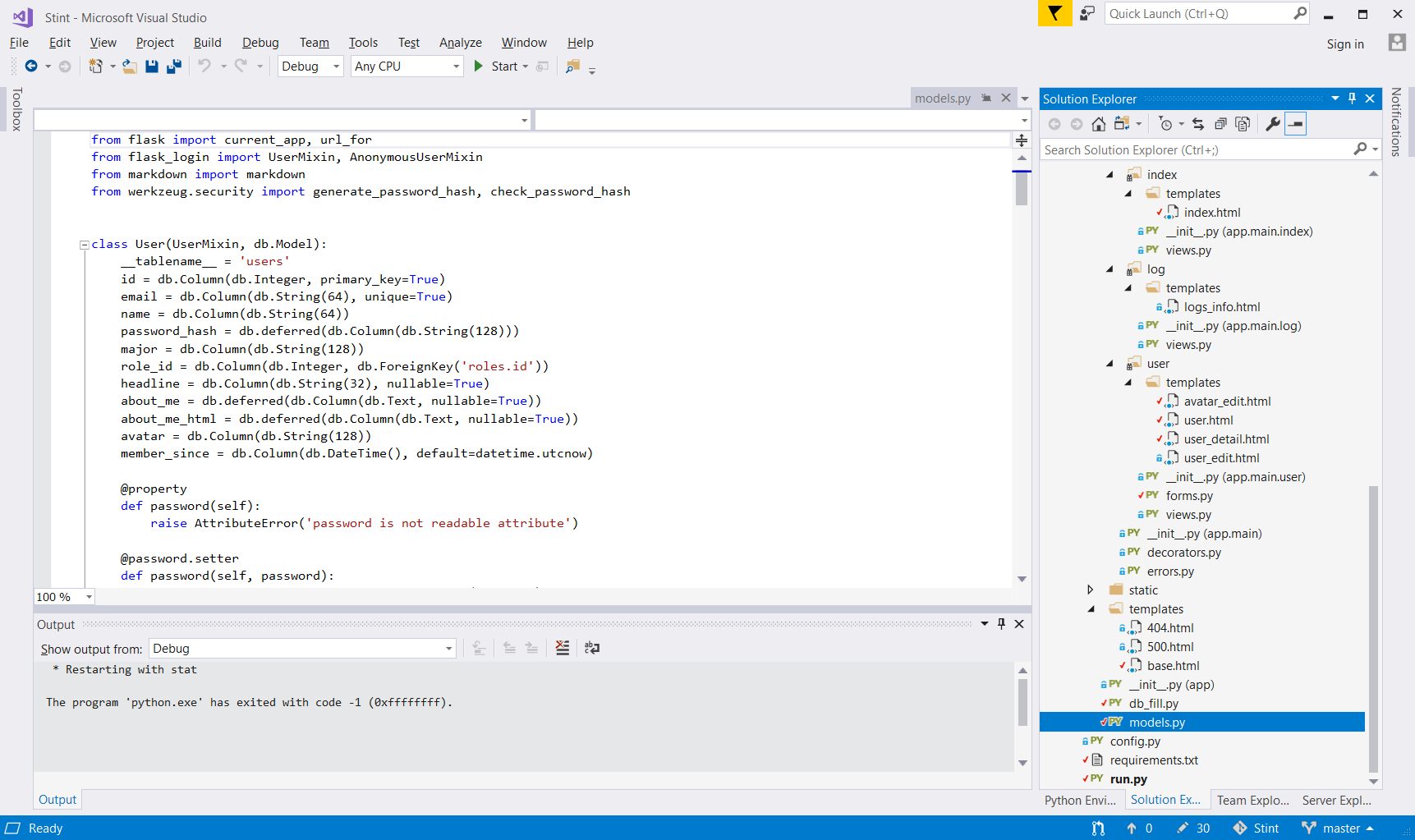
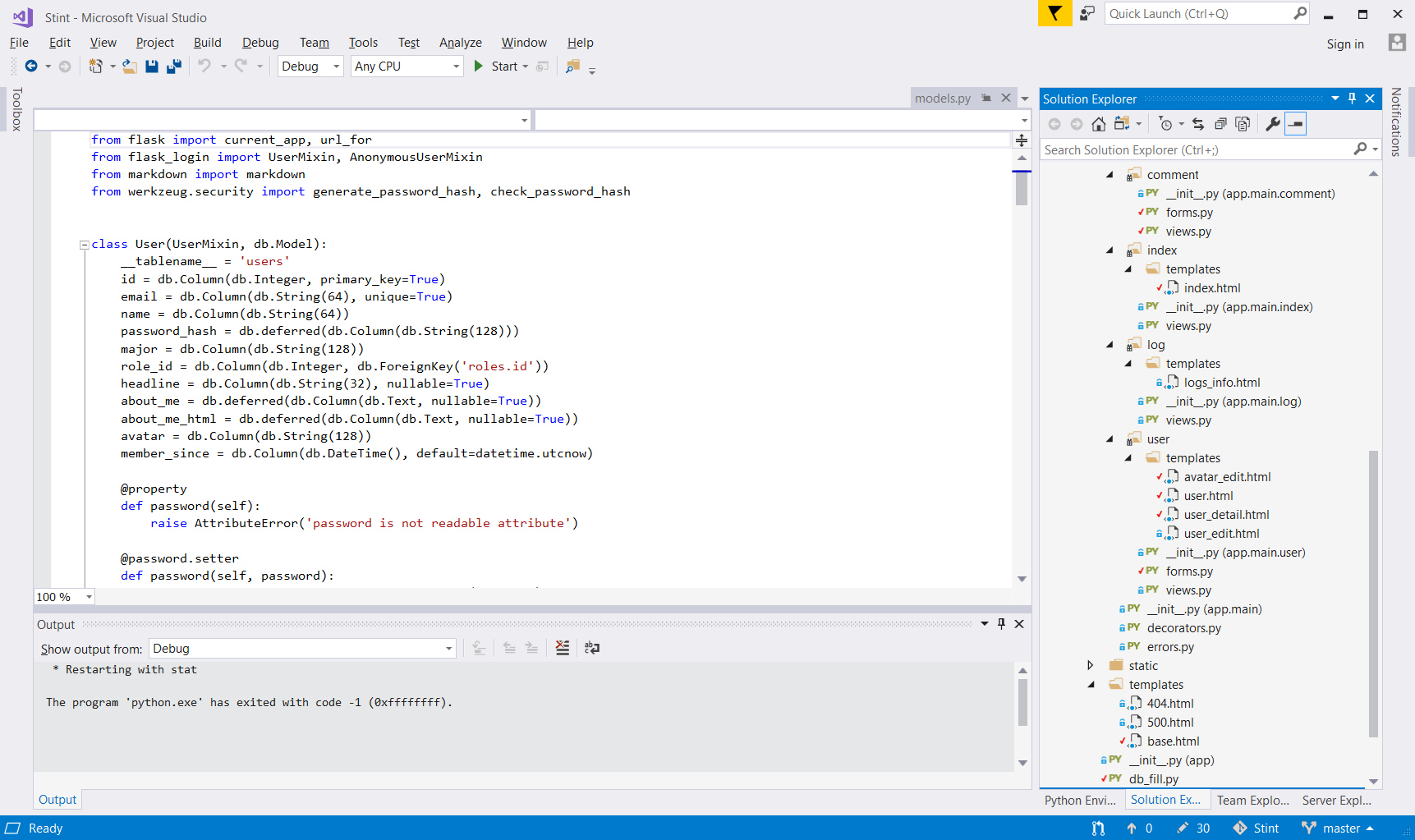
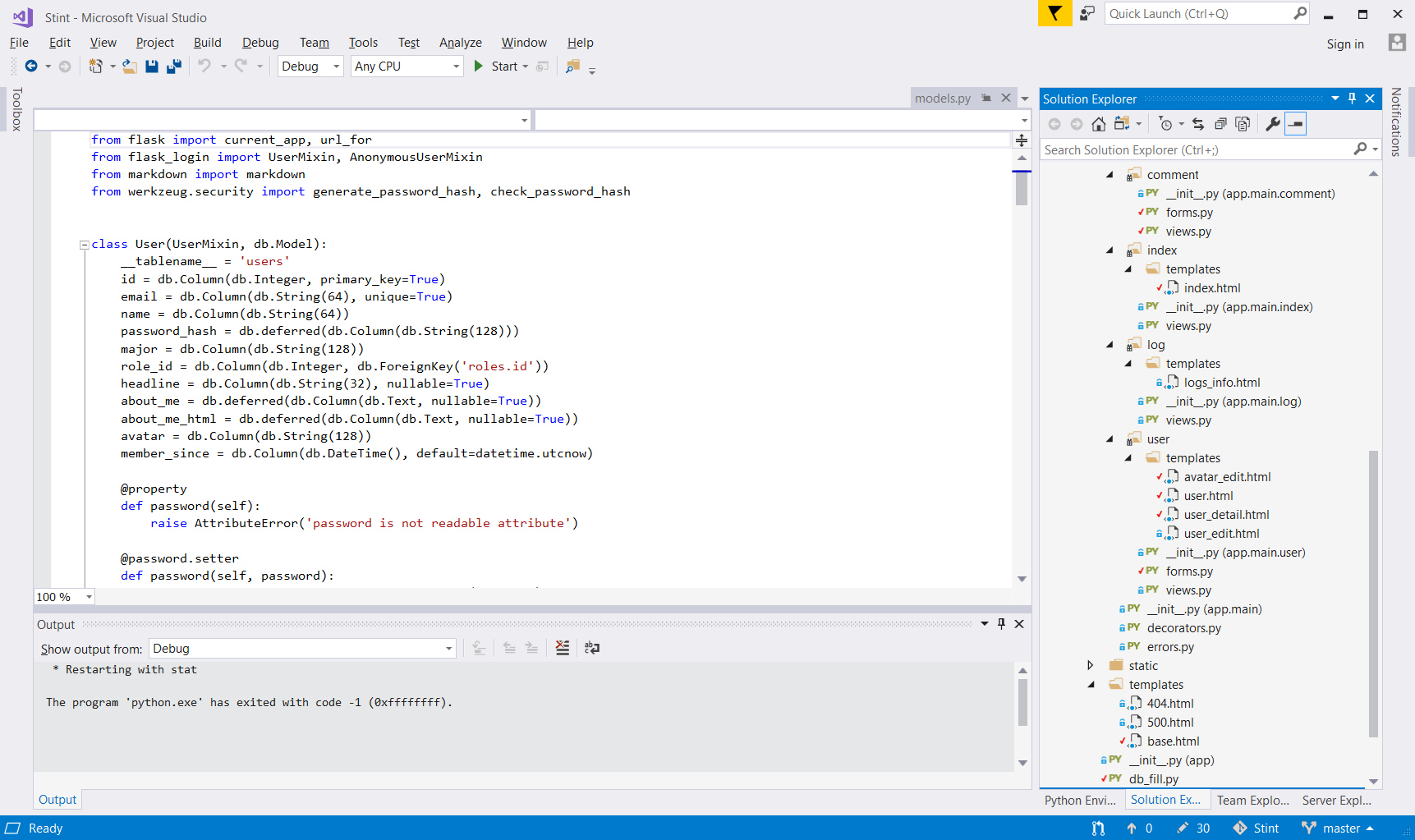
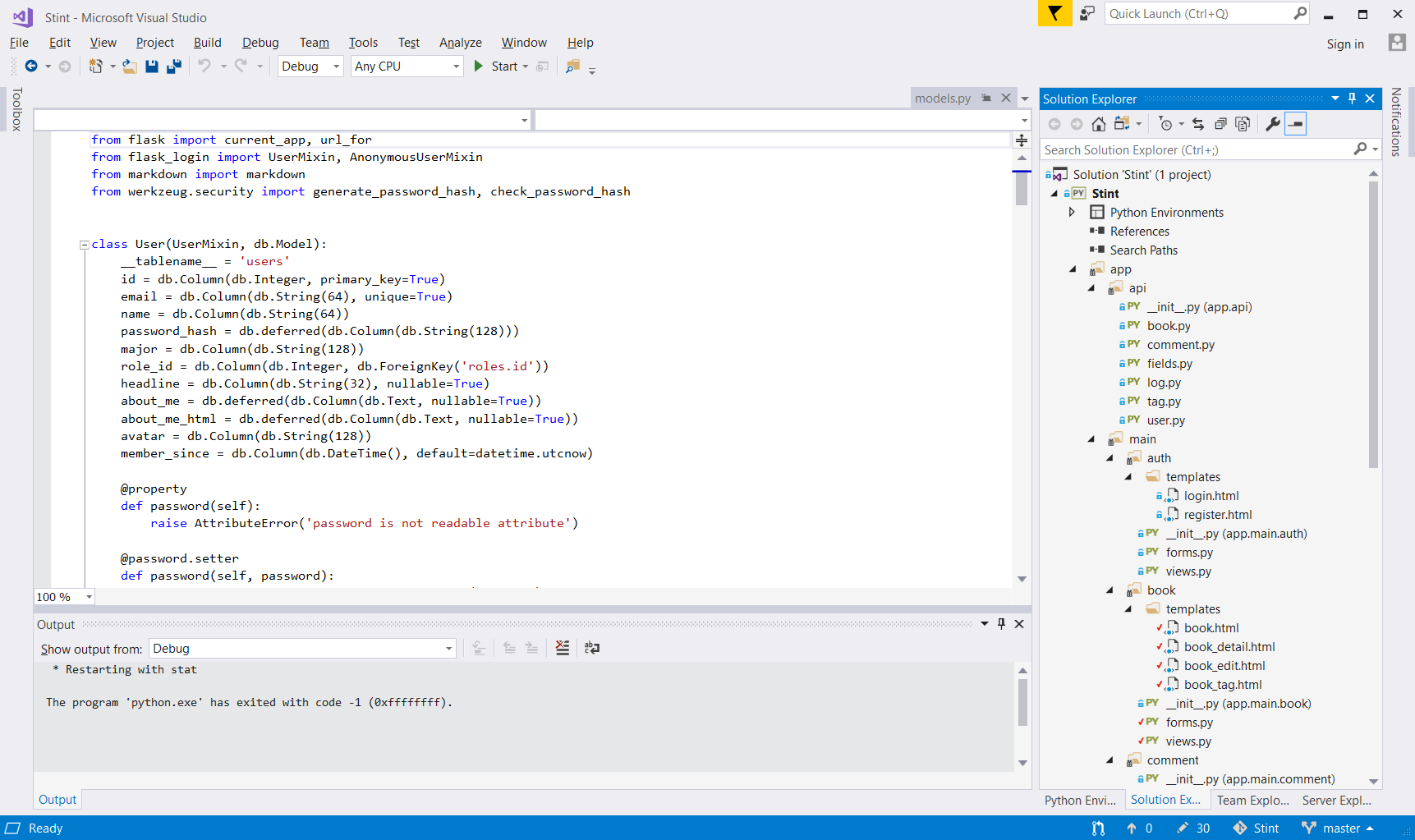
Bootstrap is used because of the beautiful page layouts it provides and it saves many coding efforts of taking care of CSS myself.

The core mechanics of Bootstrap is views.py that calls flask.render\_template

Templates are located in the apps’ templates folders that inherit from the base template.

## Project structure

The project folder structure is as follows:



■ CSS and Javascript

For the static folder, I used default folder created with Flask project template. I did not make any changes.

■ HTML (template)

In the templates folder, auth and film folders were added to define their respective screens. It seems better to divide the folders for each function.

The entire screen layout and menu bar etc. are defined in layout.html. I did not make much change from the project template either.

■ Stint is started with “run.py”, it is again taken from the template with almost no modification. Therefore, there is no need to change the project's startup file.

view.py is used to display the Home screen. There are separate files for most functions.

login function is implemented in auth.py.

movie management function is implemented the in film.py.

database handling is implemented in config.py; initialing database is in db\_fill.py

The source code of Stint is organized using **Flask blueprint**, detail will be described in the following paragraphs.

## Use of **Flask Blueprint**

The source code of Stint is organized using Flask Blueprint. It is a way to organize flask application into smaller and re-usable applications.

A blueprint defines a collection of views, templates and static assets.

To de-couple Stint into smaller re-usable components. Firstly, I moved my api calls into a blueprint.

This makes the code more maintainable and easier to debug.

## Setting up Flask blueprints

I treat blueprints as separate python packages with everything they provide in their own folder. That way, if something goes wrong with my api for example, I will know where to start looking.

The Blueprint class takes a few basic arguments:

The first argument is the blueprints name

The second argument is very important it’s the import\_name. This name has to be set to the name of my packages as Flask uses the import\_name for some internal operations such as locating the template folder of the blueprint and locating various files and objects of the main application from the blueprint. (This ensures that methods like render\_template and send\_static\_files work properly and give the actual files that I want)

The third argument is the url prefix of the blueprint. With this I was able to remove the redundancy of prefixing all my api urls with /api.

Also, there is the template\_folder argument which ‘registers’ the blueprint’s template path so I can save prefixing template references.

## Flask Blueprints used in Stint

These six blueprints are defined:

main, auth, user, film, review, api\_bp

api\_bp = Blueprint('api', \_\_name\_\_, url\_prefix='/api')

auth = Blueprint('auth', \_\_name\_\_, template\_folder='templates')

film = Blueprint('film', \_\_name\_\_, url\_prefix='/films',template\_folder='templates')

review = Blueprint('review', \_\_name\_\_, url\_prefix='/reviews')

main\index main = Blueprint('main', \_\_name\_\_, template\_folder='templates')

user = Blueprint('user', \_\_name\_\_, url\_prefix='/users', template\_folder='templates')

blueprints:

|  |  |
| --- | --- |
| blueprint names | Blueprint main function |
| main: | Defines the home screen of Stint |
| auth: | Login and register users |
| user: | Display and maintain user information |
| film: | Display and maintain movie detail and tag |
| review: | Input reviews |
| api\_bp: | * Set up Flask-RESTPlus fields of data structures used for Response marshalling * Defines the urls and their underlying object classes of Stint, including film, review, tag and user |

# ◊ **Describe the decisions you took while developing your application and explain why you decided for a particular option against other possibilities**.

## Selection of Flask over Django

Django is a full framework and enforce more standard structure. I wish to create Stint in Django to learn these standards. However, since more Flask source codes and projects are available on this subject matter, so Flask was chosen for the convenience of picking up.

## Ditching of cosmetic, nice to have features

These are the little pieces that I wish to pick up when I have the time. So, this section is dedicated to acknowledging my earlier researches.

pointillism

Because of my love of **pointillism**, I was intended to add pointillism https://github.com/tpoulsen/PseudoPointillism function to Stint but again it is not functioning yet.

## IMDbPY

IMDbPY https://github.com/alberanid/imdbpy was intended to aid the creation of new movie records in Stint, however due to time constraints it is not fully functioning yet.

Digital art

I explored some interesting possibilities in my TBI work e.g. ASCII art, Knowlton and Schwartz, but only to find out my selected workload was too intensive for me to implement them.

# ◊ **Describe and discuss specific problems**.

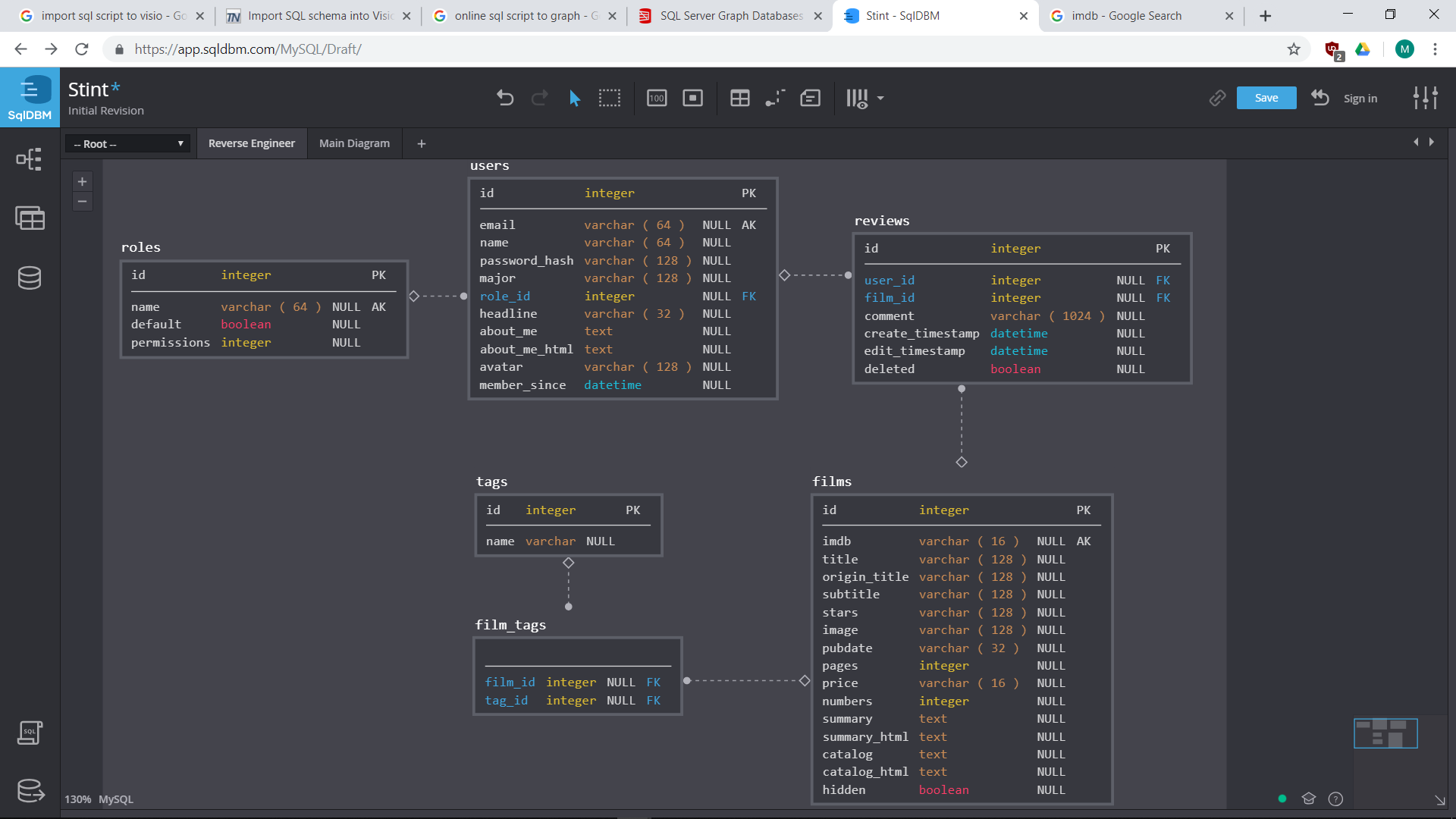
## Implementation of static data through database

When I first started with Stint in TBI, I started with a sample implementation that store all data in memory, so when the app restarts, data are gone.

In the current implementation I changed to using SQL Lite.

When Stint starts it will check for existence of local database, if not found, one will be created and initialized.

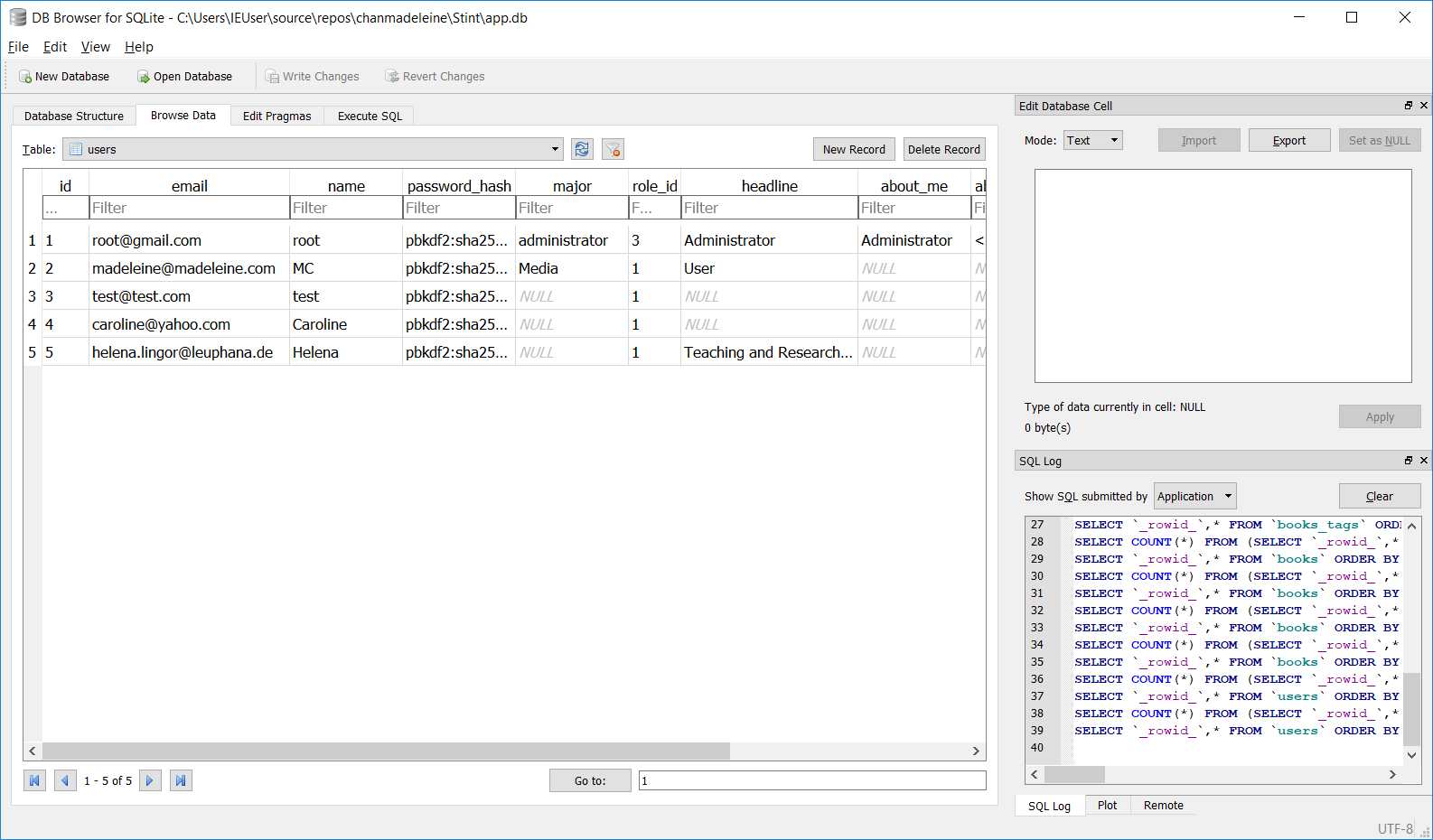
Database schema:



Database tables:

|  |  |
| --- | --- |
| Table names | Table function |
| users: | contents users of Stint and their information |
| roles: | functions that a user can perform |
| reviews: | film reviews given by users |
| films: | film records |
| tags: | tags associated with films |
| film\_tags: | links tags to films |

DB Browser for SQLLite and used for initial database maintenance and debugging.



### 1. Preparing to use SQLite and DB Browser for SQLite

I followed this manual for SQLite installation:

・[SQLite installation | DB Online](https://www.dbonline.jp/sqlite/install/)

For installation of DB Browser for SQLite, I referred to DBOnline as well.

・[Download and install DB Browser for SQLite | DB Online](https://www.dbonline.jp/sqlite-db-browser/install/index1.html)

### 2. Using DB Browser for SQLite

Open the DB Browser for SQLite and then click on the 'Execute SQL' tab.

SQL is executed by pressing the "F5" key.

When finished, click "Write Change". in the menu or "Ctrl + S" to reflect the change.

### 3. Create tables for use with Stint

Click the project file "Stint" from the top of Solution Explorer.

I will build a database here.

First, create the following SQL file as schema.sql and put it in the folder.

In Solution Explorer, right-click the "Stint" folder, select "Add" → "New Item" and "Text File", and name the file "schema.sql".

There is no need to save the SQL file, but it is useful for recreating the database. Because the table is often recreated, I dropped the existing table with "DROP TABLE" and re-created it in advance. In fact, I rewrite this code in the db\_fill.py, so it is done from within Stint.

DROP TABLE IF EXISTS films;

DROP TABLE IF EXISTS users;

CREATE TABLE films (

id INTEGER PRIMARY KEY AUTOINCREMENT,

imdb VARCHAR(16),

title VARCHAR(128),

star VARCHAR(128),

image VARCHAR(128),

summary TEXT,

summary\_html TEXT,

hidden BOOLEAN

);

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

email VARCHAR(64),

name VARCHAR(64),

password\_hash VARCHAR(128),

major VARCHAR(128),

role\_id INTEGER,

headline VARCHAR(32),

avatar VARCHAR(128)

);

Database creation procedure with DB Browser

First, drop the existing table with DROP TABLE.

Then, CREATE TABLE Create table with .... I will create the film table "films" and the user table "users".

The primary key (PRIMARY KEY) that uniquely identifies a record is the column name id. By setting "AUTOINCREMENT", the number will increase by one each time you add a record.

I specified "UNIQUE" to prevent duplicate usernames (username) in the user table.

When you right-click the instance folder in Solution Explorer and "Open the folder in Explorer", the folder is opened in Explorer.

Make a copy of the folder path.

Start DB Browser for SQLite and click "New DataBase" from the menu. A dialog for selecting a folder is displayed, so specify the folder earlier.

The database file is named "app.db". now press "Save".

The window "edit table definition" is opened, but I did not use this.

Making a table in the GUI seems to be convenient, but you have to push the button from scratch every time you make it, so it is actually more troublesome here. Close the edit table definition by pressing the x button.

Then click the "Execute SQL" tab, issue the above SQL and commit.

Click on the DataBase Structure tab and you should see that the table is ready.

The file "app.db" has been added to the folder.

However, if you did not add a file with Visual Studio, you will not see it within Solution Explorer.

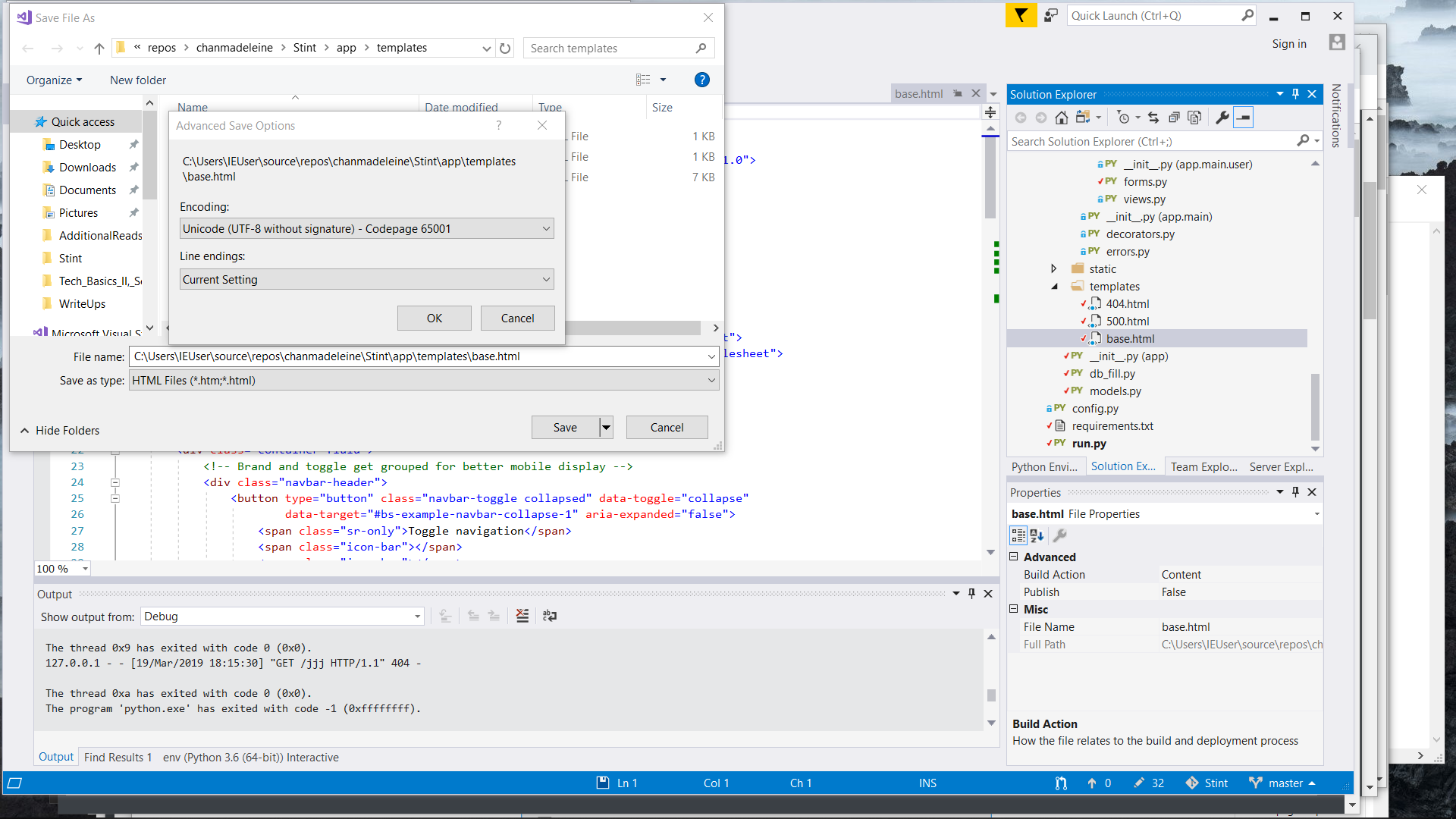
Then you can inventory it from Solution Explorer by right-click on the folder from Solution Explorer and select “app.db” file then “Add” → “Existing Item”.

## Unicode safeness

In the middle of my implementation I was suddenly hit by this error without clue.

UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 3: ordinal not in range(128

After much googling, including mess around with Flask rendering and encoded without any luck, I finally found out my template file has some characters that break the renderer at run time, and all I need to do is to save the file with proper encoding e.g. UTF-8. Problem solved.

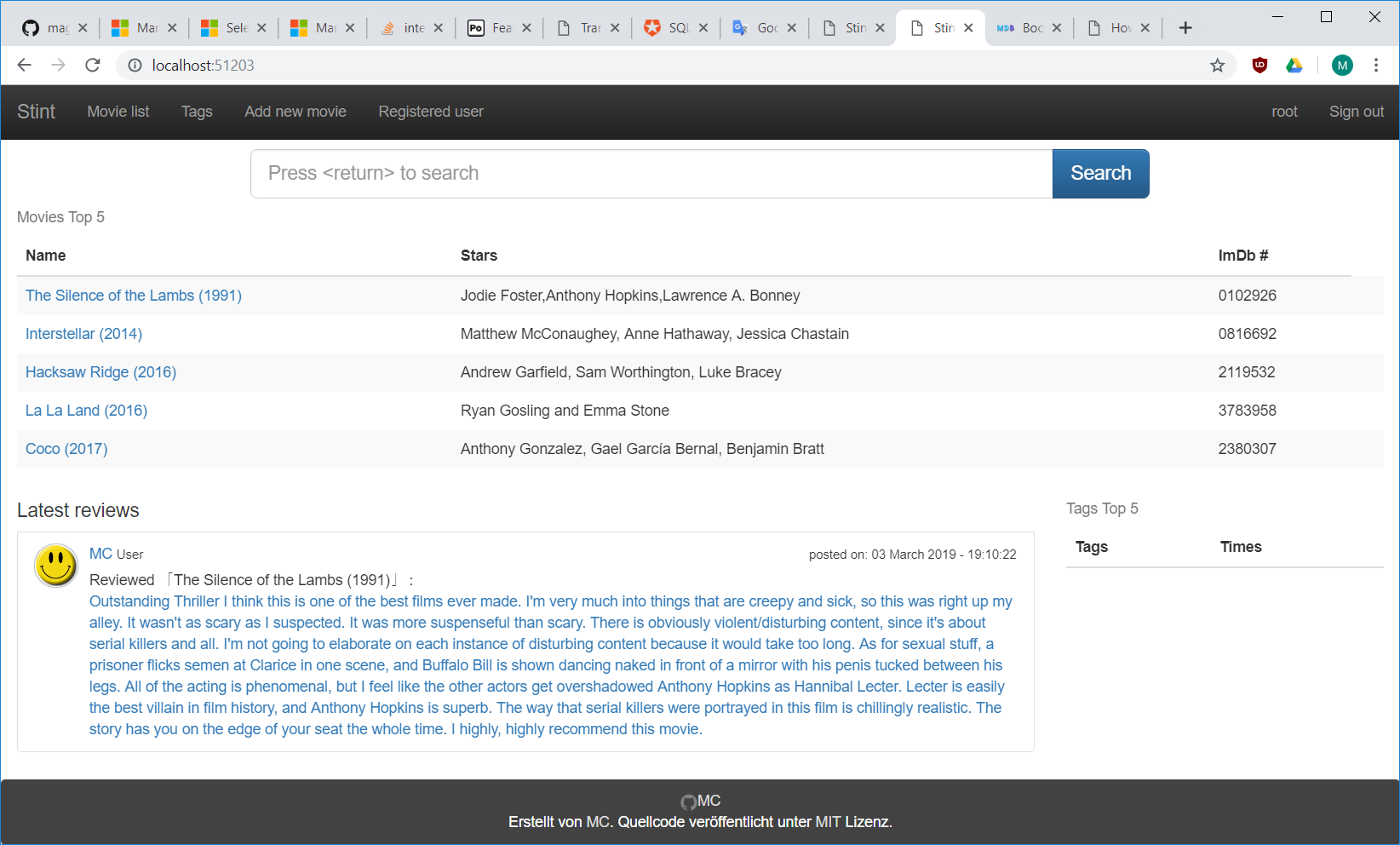
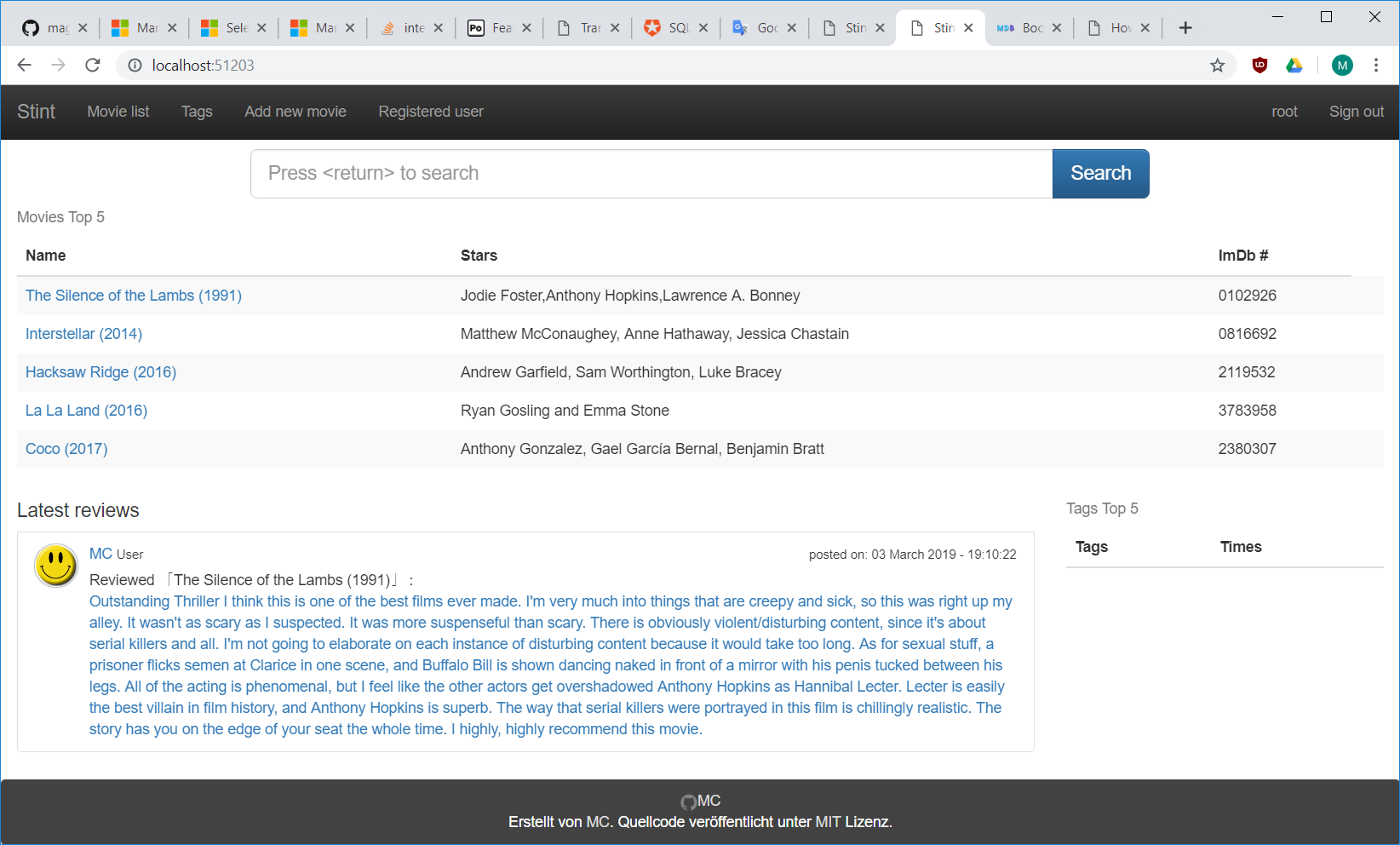
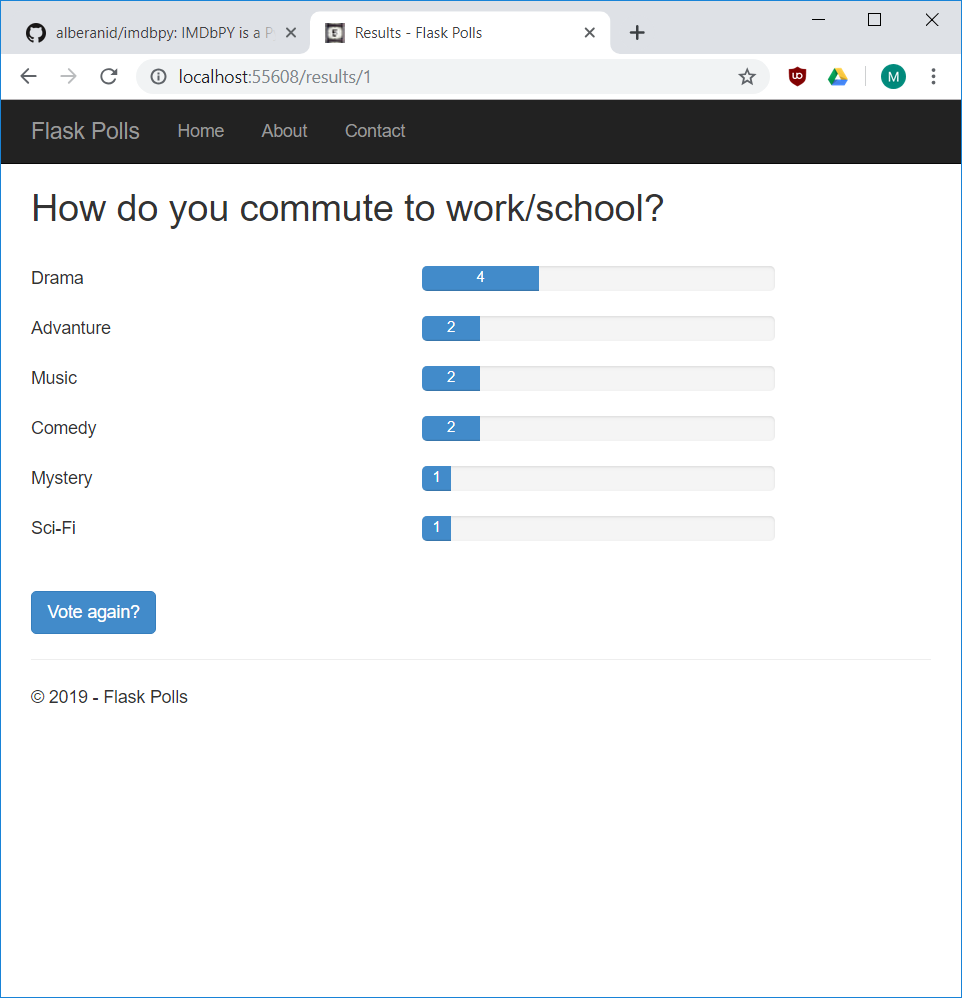


Digital arts and ASCII arts

I first designed Stint with digital art and ASCII components to enhance user experiences but they are not functionally implemented due to time constraints.

Graphs

The same goes to graphs, I planned to utilize MATPLOTLIB that I learnt from TBII to display some graphical statistics.

# ◊ Final reflection

By dictionary definition, a reflection is a thought, idea, or opinion formed, or a remark made as a result of meditation.

From my research, a reflection is also the ability of a [computer program](https://en.wikipedia.org/wiki/Computer_program) to examine, [introspect](https://en.wikipedia.org/wiki/Introspection_(computer_science)), and modify its own structure and behavior at [runtime](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)).

My refection on the project:

Implementation efforts vs documentation efforts

I did not gasp the resources management too well that I mis-estimated the effort I needed to put together everything into this documentation. In addition to the effort needed to write the code to include everything I wanted to play around with.

I was a present experience and I dig out many aspects that has rooms for improvements.

I learnt and improved how to look for technical resources to archive the results. I also learnt to make decisions faster so that I can work more efficient.

Features that were dropped

I listed those features that I fancy in Stint and was failed to put into this version in the decisions I took section above.

1. #### Credits

   [Akari, Simple Book library](https://github.com/magic-akari/BookLibrary)  
   [Shinya Baba, Book management](https://logics-of-blue.com/visual-studio-flask-book-management-crud/) [↑](#footnote-ref-1)