**Chapter 0: Introduction and Setup**

### README: Setting Up mflix

[m220/mflix-pytziphon.](https://s3.amazonaws.com/edu-downloads.10gen.com/M220P/2022/September/static/handouts/m220/mflix-python.zip)

# Table of Contents

**Setting Up mflix:**

1. Project Structure
2. Local Development Environment Configuration
3. Python Library Dependencies
4. Running the Application
5. Running the Unit Tests

In order to run properly, the MFlix software project has some installation requirements and environmental dependencies.

These requirements and dependencies are defined in this lesson, and they can also be found in the **README.rst** file from the **mflix-python** project, which you will download shortly. This lesson serves as a guide for setting up these necessary tools. After following this README, you should be able to successfully run the MFlix application. First, you will need to download the **mflix-python** project, as described below.

# Download the mflix-python.zip file

You can download the **mflix-python.zip** file by clicking the link in the "Handouts" section of this page. Downloading this handout may take a few minutes. When the download is complete, unzip the file and cd into the project's root directory, **mflix-python**.

COPY

cd ~/Downloads

unzip mflix-python.zipcd mflix-python

# Project Structure

Everything you will implement is located in the mflix/db.py file, which contains all database interfacing methods. The API will make calls to db.py to interact with MongoDB.

The unit tests in tests will test these database access methods directly, without going through the API. The UI will run these methods in integration tests, and therefore requires the full application to be running.

The API layer is fully implemented, as is the UI. If you need to run on a port other than 5000, you can edit the **index.html** file in the build directory to modify the value of **window.host**.

Please do not modify the API layer in any way, movies.py and user.py under the **mflix/api** directory. Doing so will most likely result in the frontend application failing to validate some of the labs.

# Local Development Environment Configuration

## Anaconda

We're going to use [Anaconda](https://anaconda.org/) to install Python 3 and to manage our Python 3 environment.

**Installing Anaconda for Mac**

You can download Anaconda from their [MacOS download site](https://www.anaconda.com/download/#macos). The installer will give you the option to "Change Install Location", so you can choose the path where the anaconda3 folder will be placed. Remember this location, because you will need it to activate the environment.

Once installed, you will have to create and activate a conda environment:

COPY

# navigate to the mflix-python directorycd mflix-python

# enable the "conda" command in Terminalecho ". /anaconda3/etc/profile.d/conda.sh" >> ~/.bash\_profilesource ~/.bash\_profile

# create a new environment for MFlixconda create --name mflix

# activate the environmentconda activate mflix

You can deactivate the environment with the following command:

COPY

conda deactivate

**Installing Anaconda for Windows**

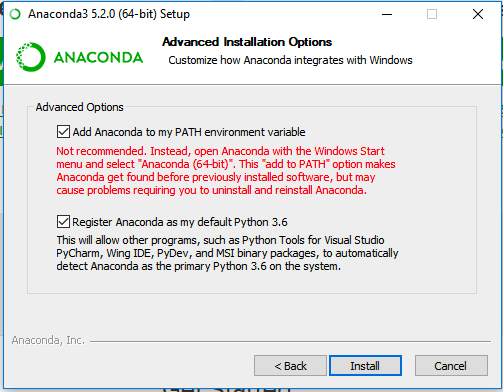
You can download Anaconda from their [Download site](https://www.anaconda.com/download/). Please be careful to select Windows Tab before downloading.

The Anaconda installer will prompt you for the following options:

* *Add Anaconda to my PATH environment variable*
* *Register Anaconda as my default Python 3.6*

Please select both of these options. The first option will allow you to use conda commands from the Command Prompt, and the second option will allow you to use Anaconda's Python 3.6 as your system's default.

You may see a red error message like the following:



This is expected. Please select both of the options above.

If you forget to select the *PATH* option before installing, no worries. The installer will let you choose an "Install Location" for Anaconda, which is the directory where the Anaconda3 folder will be placed.

Using your machine's location of Anaconda3 as <path-to-Anaconda3>, run the following commands to activate conda commands from the Command Prompt:

COPY

set PATH=%PATH%;<path-to-Anaconda3>;<path-to-Anaconda3>\Scripts\

Once Anaconda is installed, you will have to create and enable a conda environment.

COPY

# enter mflix-python foldercd mflix-python

# create a new environment for MFlixconda create --name mflix

# activate the environmentactivate mflix

You can deactivate the environment with the following command:

COPY

deactivate

## Virtualenv

*Note: If you installed Anaconda instead, skip this step.*

As an alternative to Anaconda, you can also use virtualenv, to define your Python 3 environment. You are required to have a Python 3 installed in your workstation.

You can find the [virtualenv installation procedure](https://virtualenv.pypa.io/en/stable/installation/) on the PyPA website.

Once you've installed Python 3 and virtualenv, you will have to setup a virtualenv environment:

COPY

# navigate to the mflix-python directorycd mflix-python

# create the virtual environment for MFlixvirtualenv -p YOUR\_LOCAL\_PYTHON3\_PATH mflix\_venv

# activate the virtual environmentsource mflix\_venv/bin/activate

You can deactivate the virtual environment with the following command:

COPY

deactivate

## Python Library Dependencies

Once the Python 3 environment is activated, we need to install our python dependencies. These dependencies are defined in the requirements.txt file, and can be installed with the following command:

COPY

pip install -r requirements.txt

# Running the Application

In the mflix-python directory you can find a file called dotini.

Open this file and enter your Atlas SRV connection string as directed in the comment. This is the information the driver will use to connect. Make sure **not** to wrap your Atlas SRV connection between quotes:

COPY

MFLIX\_DB\_URI = mongodb+srv://...

Rename this file to .ini with the following command:

COPY

mv dotini\_unix .ini # on Unixren dotini\_win .ini # on Windows

*Note:* Once you rename this file to .ini, it will no longer be visible in Finder or File Explorer. However, it will be visible from Command Prompt or Terminal, so if you need to edit it again, you can open it from there:

COPY

vi .ini # on Unixnotepad .ini # on Windows

To start MFlix, run the following command:

COPY

python run.py

This will start the application. You can then access the MFlix application at <http://localhost:5000/>.

# Running the Unit Tests

To run the unit tests for this course, you will use pytest and needs to be run from mflix-python directory. Each course lab contains a module of unit tests that you can call individually with a command like the following:

COPY

pytest -m LAB\_UNIT\_TEST\_NAME

Each ticket will contain the command to run that ticket's specific unit tests. For example to run the Connection Ticket test your shell command will be:

COPY

pytest -m connection

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_0_Introduction_and_Setup/lesson/5a902a47563d6341ad0ce5cb/problem)

**Chapter 0: Introduction and Setup**

### Ticket: Connection

**Problem:**

**Task**

MFlix will use MongoDB as a storage layer, so for this ticket you'll be required to perform some application setup.

1. First, make sure you've created a user on your Atlas cluster with read and write access to the **sample\_mflix** database.

* The user name should be m220student and the password should be m220password
* Don't forget to whitelist your IP address!

Copy the connection string. Select that you'd like to connect with the mongo shell, version 3.6 or later - this will give you the **srv** connection string. Make sure this URI string contains your username and password!

Locate the file called **dotini\_win** or **dotini\_unix** (depending on your operating system) and replace the information within with your own **srv** connection string. The [TEST] URI will be used by the unit tests, while the integration tests will use the [PROD] URI:

[PROD]

SECRET\_KEY = super\_secret\_key\_you\_should\_change

MFLIX\_DB\_URI = mongodb+srv://m220student:m220password@<your-atlas-cluster-address>

MFLIX\_NS = sample\_mflix

[TEST]

SECRET\_KEY = super\_secret\_testing\_key

MFLIX\_DB\_URI = your\_testing\_db\_uri (can be the same as Atlas, or a local MongoDB database)

MFLIX\_NS = sample\_mflix

* It's highly suggested you also change the SECRET\_KEY to some very long, very random string. While this application is only meant for local use during this course, software has a strange habit of living a long time.

Rename **dotini\_win** or **dotini\_unix** to **.ini**. You can do this by running the following command from the mflix-python directory:

mv dotini\_unix .ini # on Unix

ren dotini\_win .ini # on Windows

Note: Once you rename this file to .ini, it will no longer be visible in Finder or File Explorer. However, it will be visible from Command Prompt or Terminal, so if you need to edit it again, you can open it from there:

vi .ini # on Unix

notepad .ini # on Windows

**Testing and Running the Application**

In order to reinforce good development practices, everything asked of you in this course is backed up by unit tests. Reading through the tests for a specific exercise will tell you exactly what is expected.

You can run the unit tests for this ticket by running:

pytest -m connection

Once the unit tests are passing, run the application with:

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Connection**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_1_Driver_Setup/lesson/5b7601694e136f572ec59623/lecture)

**Chapter 1: Driver Setup**

### Ticket: Projection

**Problem:**

**User Story**

"As a user, I'd like to be able to search movies by country and see a list of movie titles. I should be able to specify a comma-separated list of countries to search multiple countries."

**Task**

Implement the **get\_movies\_by\_country** method in db.py to search movies by country and use projection to return the title and \_id field. The \_id field will be returned by default.

You can find examples in notebooks/your\_first\_read.ipynb.

**MFlix Functionality**

Once you complete this ticket, the UI will allow movie searches by one or more countries.

**Testing and Running the Application**

Make sure to look at the tests in test\_projection.py to understand what is expected.

You can run the unit tests for this ticket by running:

COPY

pytest -m projection

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant unit tests, what is the validation code for **Projection**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_1_Driver_Setup/lesson/5a9479f79b94cf4717806184/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_1_Driver_Setup/lesson/5a96b99e32f68932b1f16870/problem)

窗体底端

**Chapter 1: Driver Setup**

### Ticket: Projection

**Problem:**

**User Story**

"As a user, I'd like to be able to search movies by country and see a list of movie titles. I should be able to specify a comma-separated list of countries to search multiple countries."

**Task**

Implement the **get\_movies\_by\_country** method in db.py to search movies by country and use projection to return the title and \_id field. The \_id field will be returned by default.

You can find examples in notebooks/your\_first\_read.ipynb.

**MFlix Functionality**

Once you complete this ticket, the UI will allow movie searches by one or more countries.

**Testing and Running the Application**

Make sure to look at the tests in test\_projection.py to understand what is expected.

You can run the unit tests for this ticket by running:

COPY

pytest -m projection

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant unit tests, what is the validation code for **Projection**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_1_Driver_Setup/lesson/5a9479f79b94cf4717806184/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_1_Driver_Setup/lesson/5a96b99e32f68932b1f16870/problem)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: Paging

**Problem:**

**User Story**

"As a user, I'd like to get the next page of results for my query by scrolling down in the main window of the application."

**Task**

Modify the method **get\_movies** in db.py to allow for paging. You can see how the page is parsed and sent in the **api\_search\_movies** method from movies.py.

You can find examples of using cursor methods in notebooks/cursor\_methods\_agg\_equivalents.ipynb.

**MFlix Functionality**

The UI is already asking for infinite scroll! You may have noticed a message stating "paging not implemented" when scrolling to the bottom of the page.

Once this ticket is completed, this message will go away, and scrolling to the bottom of the page will result in a new page of movies.

**Testing and Running the Application**

Make sure you look at the tests in test\_paging.py to look at what is expected.

You can run the unit tests for this ticket by running:

COPY

pytest -m paging

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Paging**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5a98373a1ae56732100b3c52/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5b770bf64e136f572ec5962b/lecture)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: Faceted Search

**Problem:**

**User Story**

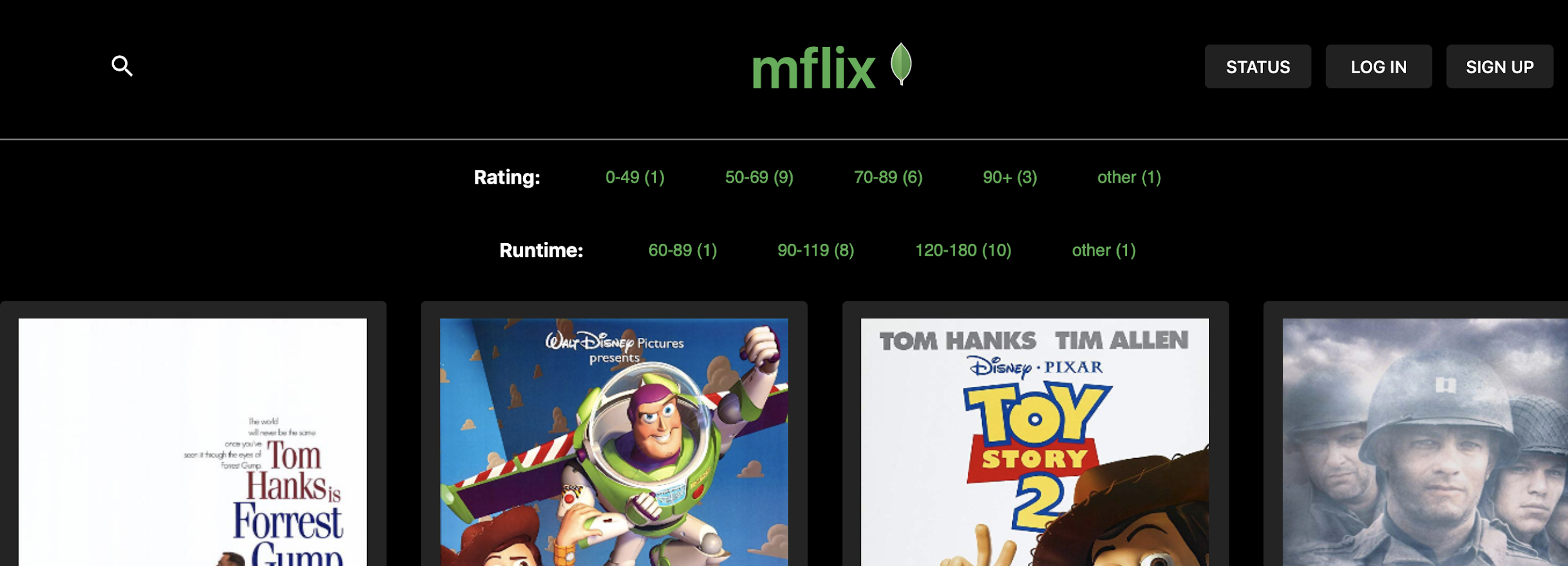
"As a user, I want to be able to filter cast search results by one facet, **metacritic** rating."

**Task**

For this Ticket, you'll be required to implement one method in db.py, **get\_movies\_faceted**, so the MFlix application can perform faceted searches.

**MFlix Functionality**

Once the change is implemented for this ticket, the user interface will reflect this change when you search for cast (e.g. "Tom Hanks"), then additional search parameters will be added as shown below:



What is a Faceted Search?

Faceted search is a way of narrowing down search results as search parameters are added. For example, let's say MFlix allows users to filter movies by a rating from 1 to 10, but Kate Winslet has only acted in movies that have a rating of 6 or higher.

If we didn't specify any other search parameters, MFlix would allow us to choose a rating between 1 and 10. But if we first search for Kate Winslet, the application would only let us choose a rating between 6 and 10, because none of the movie documents in the result set have a rating below 6.

If you're curious, you can read more about Faceted Search [here](https://en.wikipedia.org/wiki/Faceted_search).

Faceted Search in MFlix

Faceted searches on the MFlix site cannot be supported with the basic search method **get\_movies**, because that method uses the Mongo query language. For faceted searches, the application must use the Aggregation Framework.

The method **get\_movies\_faceted** uses the Aggregation Framework, and the individual stages in the pipeline have already been completed. Follow instructions in the db.py file to append the required stages to the pipeline object.

**Testing and Running the Application**

Look in the test\_facets.py file in your **tests** directory to view the unit tests for this ticket.

You can run the unit tests for this ticket by running:

COPY

pytest -m facets

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Faceted Search**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5c9565550618cee85faf2232/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5b72fdc98df4183dbda16180/lecture)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: User Management

**Problem:**

**User Story**

"As a user, I should be able to register for an account, log in, and logout."

**Task**

For this Ticket, you'll be required to implement all the methods in db.py that are called by the API endpoints in user.py. Specifically, you'll implement:

* **get\_user**
* **add\_user**
* **login\_user**
* **logout\_user**
* **get\_user\_session**
* **delete\_user**

For this ticket, you will need to use the find\_one(), update\_one() and delete\_one() methods. You can find examples in the following notebooks:

* notebooks/deletes.ipynb
* notebooks/your\_first\_write.ipynb

**MFlix Functionality**

Once this ticket is completed, users will be able to register for a new account, log in, logout, and delete their account.

Registering should create an account and log the user in, ensuring an entry is made in the **sessions** collection. There is a [unique index](https://docs.mongodb.com/manual/core/index-unique/?jmp=university) on the user\_id field in **sessions**, so we can efficiently query on this field.

**Testing and Running the Application**

Look within the test\_user\_management.py file in your **tests** directory to view the unit tests for this ticket.

You can run the unit tests for this ticket by running:

COPY

pytest -m user\_management

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **User Management**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5a8dd589ad47df77afa14f0d/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5b7301fd8df4183dbda16184/lecture)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: Durable Writes

**Problem:**

**Task**

For this ticket, you'll be required to increase the durability of the **add\_user** method from the default write concern of w: 1.

When a new user registers for MFlix, their information must be added to the database before they can do anything else on the site. For this reason, we want to make sure that the data written by the **add\_user** method will not be rolled back.

We can completely eliminate the chances of a rollback by increasing the write durability of the **add\_user** method. To use a non-default write concern with a database operation, use Pymongo's [with\_options](https://dfproj.readthedocs.io/en/latest/api/pymongo/collection.html#pymongo.collection.Collection.with_options) flag when issuing the query.

You can find examples of write concerns in notebooks/write\_concerns.ipynb.

**Testing and Running the Application**

There are no unit or integration tests for this lab.

Please complete the multiple choice question below, and then implement the correct write concern in the **add\_user** method.

The implementation of this task will not be tested, but using the default of w: 1 might result in a rollback of your users' account data!

Which of the following write concerns are more durable than the default?

**Attempts Remaining:3 Attempts left**

窗体顶端

**Check all answers that apply:**

w: 0

w: 2

w: "majority"

w: 1

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5b770ccd4e136f572ec5962d/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5b7303f08df4183dbda16188/lecture)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: User Preferences

**Problem:**

**User Story**

"As a user, I want to be able to store preferences such as my favorite cast member and preferred language."

**Task**

For this Ticket, you'll be required to implement one method in db.py, **update\_prefs**. This method allows updates to be made to the "preferences" field in the users collection.

**MFlix Functionality**

Once this ticket is completed, users will be able to save preferences in their account information.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m user\_preferences

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **User Preferences**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5aac015e25a7bff8d2002291/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5b7719f64e136f572ec59631/lecture)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: Get Comments

**Problem:**

**User Story**

"As a user, I want to be able to view comments for a movie when I look at the movie detail page."

**Task**

For this ticket, you'll be required to extend the **get\_movie** method in db.py so that it also fetches the comments for a given movie.

The comments should be returned in order from most recent to least recent using the date key.

Movie comments are stored in the comments collection, so this task can be accomplished by performing a $lookup. Refer to the Aggregation [Quick Reference](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/?jmp=university#join-conditions-and-uncorrelated-sub-queries) for the specific syntax.

You can find examples of Aggregation with the Python driver in notebooks/basic\_aggregation.ipynb.

**MFlix Functionality**

Once this ticket is completed, each movie's comments will be displayed on that movie's detail page.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m get\_comments

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Get Comments**?

Hint: We need to sort the comments in the $lookup stage.

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5ab50722139bb8023d4037b6/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5aba954a31b11b851a7b87fc/problem)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: Create/Update Comments

**Problem:**

**User Story**

"As a user, I want to be able to post comments to a movie page as well as edit my own comments."

**Task**

For this ticket, you'll be required to implement two methods in db.py, **add\_comment** and **update\_comment**.

Ensure that **update\_comment** only allows users to update their own comments, and no one else's comments.

**MFlix Functionality**

Once this ticket is completed, users will be able to post comments on their favorite (and least favorite) movies, and edit comments they've posted.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m create\_update\_comments

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant unit tests, what is the validation code for **Create/Update Comments**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5aba954a31b11b851a7b87fc/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5b771a474e136f572ec59633/lecture)

窗体底端

**Chapter 2: User-Facing Backend**

### Ticket: Delete Comments

**Problem:**

**User Story**

"As a user, I want to be able to delete my own comments."

**Task**

For this ticket, you'll be required to modify one method in db.py, **delete\_comment**. Ensure the delete operation is limited so only the user can delete their own comments, but not anyone else's comments.

You can find examples of delete\_one() in notebooks/deletes.ipynb.

**MFlix Functionality**

Once this ticket is completed, users will be able to delete their own comments, but they won't be able to delete anyone else's comments.

**Testing and Running the Application**

You can run all the tests for this ticket by running:

COPY

pytest -m delete\_comments

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Delete Comments**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_2_User-Facing_Backend/lesson/5ac25d5e84fa396e29980e54/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_3_Admin_Backend/lesson/5b77271b4e136f572ec59637/lecture)

窗体底端

**Chapter 3: Admin Backend**

### Ticket: User Report

**Problem:**

**User Story**

"As an administrator, I want to be able to view the top 20 users by their number of comments."

**Task**

For this ticket, you'll be required to modify one method in db.py, **most\_active\_commenters**. This method produces a report of the 20 most frequent commenters on the MFlix site.

**Hint**

This report is meant to be run from the backend by a manager that is very particular about the accuracy of data. Ensure that the [read concern](https://docs.mongodb.com/manual/reference/read-concern/index.html) used in this read, avoids any potential document rollback.

Remember to add the necessary changes in the pipeline to meet the requirements. More information can be found in the comments of the method.

You can find examples of Aggregation with the Python driver in notebooks/basic\_aggregation.ipynb.

**MFlix Functionality**

Once this ticket is completed, users with database access can make users administrators. Administrators will be able to generate a user report listing top commenters.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m user\_report

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **User Report**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_3_Admin_Backend/lesson/5b7365e9c7506e1fa37bbc2d/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_3_Admin_Backend/lesson/5b74c32d4e136f572ec59611/lecture)

窗体底端

**Chapter 3: Admin Backend**

### Ticket: Migration

**Problem:**

**Task**

For this ticket, you'll be required to complete the command-line script located in the migrations directory of mflix called **movie\_last\_updated\_migration.py**.

Things always change, and a requirement has come down that the lastupdated value in each document of the movies collection needs to be stored as an **ISODate** rather than a **String**.

Complete the script so it updates the values using the [bulk API](https://pymongo.readthedocs.io/en/stable/api/pymongo/bulk.html).

To perform the migration, run the script:

COPY

python movie\_last\_updated\_migration.py

You can find examples of Bulk Operations in notebooks/bulk\_writes.ipynb.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m migration

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Migration**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_3_Admin_Backend/lesson/5accc225673215490e9f7886/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5b77275d4e136f572ec59639/lecture)

窗体底端

**Chapter 4: Resiliency**

### Ticket: Connection Pooling

**Problem:**

**Task**

For this ticket, you'll be required to modify the configuration of MongoClient to set the maximum size of the connection pool to 50 connections.

The MongoClient in db.py is initialized in the **get\_db** method. A link to the MongoClient documentation is included [here](https://pymongo.readthedocs.io/en/stable/api/pymongo/mongo_client.html) for your reference.

You can learn more about Connection Pooling in notebooks/connection\_pooling.ipynb.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m connection\_pooling

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Connection Pooling**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5ad4f6e14cc44babec65733d/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5b7733ce4e136f572ec59640/lecture)

窗体底端

**Chapter 4: Resiliency**

### Ticket: Timeouts

**Problem:**

**Task**

For this ticket, you'll be required to modify the connection information for MongoClient to set a write timeout of 2500 milliseconds.

The MongoClient in db.py is initialized in the **get\_db** method. A link to the relevant documentation is included [here](https://pymongo.readthedocs.io/en/stable/api/pymongo/mongo_client.html) for your reference.

You can learn more about timeouts in notebooks/robust\_applications.ipynb.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m timeouts

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Timeouts**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5addfc5eece427f29c1c5675/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5b7739634e136f572ec59644/lecture)

窗体底端

**Chapter 4: Resiliency**

### Ticket: Handling Errors

**Problem:**

**Task**

For this ticket, you'll be required to make the API more robust by handling exceptions. Specifically, what would happen should an incorrectly formatted **\_id** be passed to **get\_movie** in db.py?

To determine the exact exception that will be thrown in this case, please consult the documentation to see which exceptions pymongo and bson can raise:

* [pymongo exceptions](https://pymongo.readthedocs.io/en/stable/api/pymongo/errors.html)
* [bson exceptions](https://pymongo.readthedocs.io/en/stable/api/bson/errors.html)

Once you've determined the exception you need to handle, make sure to catch it in the **get\_movie** method.

Although this ticket will only test that you've handled exceptions for the get\_movie method, it's also a good idea to look over the entirety of **db.py** to look for other potential exceptions and handle them!

You can find examples of Error Handling in notebooks/error\_handling.ipynb.

**Testing and Running the Application**

You can run the unit tests for this ticket by running:

COPY

pytest -m error\_handling

Once the unit tests are passing, run the application with:

COPY

python run.py

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

After passing the relevant unit tests, what is the validation code for **Error Handling**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5ae9bb74bbcc75b11c43a61a/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5b773c194e136f572ec59648/lecture)

窗体底端

**Chapter 4: Resiliency**

### Ticket: Principle of Least Privilege

**Problem:**

**Task**

For this ticket, you'll be required to add a new user on your Atlas cluster for the MFlix application to connect with.

The user should follow credentials:

* username: **mflixAppUser**
* password: **mflixAppPwd**

This user should have the **readWrite** role on the **sample\_mflix** database. Use **Add Default Privileges** to assign the user this specific role.

After you have created this user, modify the SRV connection string in your configuration file so the application connects with the new username and password.

**Testing and Running the Application**

There are no unit tests associated with this ticket.

Once you have modified the connection string, stop and restart the application.

Now proceed to the [status page](http://localhost:5000/status) to run the full suite of integration tests and get your validation code.

What is the validation code for **Principle of Least Privilege**?

**Attempts Remaining:3 Attempts left**

窗体顶端

Enter answer here:

Show answer

[.](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5b61c3788ecc030e18f5ca48/answer)

[Proceed to next section](https://university.mongodb.com/mercury/M220P/2022_September_20/chapter/Chapter_4_Resiliency/lesson/5b773c6b4e136f572ec5964a/lecture)

窗体底端