

Exercises 2.10a

Installed CORS, bcrypt and express-validator via npm (git bash)

```
MINGW32:/c/Users/Jacqueline/Documents/GitHub/movie_api
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ npm install cors
+ cors@2.8.5
added 2 packages from 2 contributors and audited 411 packages in 3.185s

1 package is looking for funding
  run 'npm fund' for details

found 0 vulnerabilities

Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ npm install bcrypt
> bcrypt@4.0.0 install C:\Users\Jacqueline\Documents\GitHub\movie_api\node_modules\bcrypt
> node-pre-gyp install --fallback-to-build

node-pre-gyp WARN Using needle for node-pre-gyp https download
[bcrypt] Success: "C:\Users\Jacqueline\Documents\GitHub\movie_api\node_modules\bcrypt\lib\binding\napi-v3\bcrypt_lib.node" is installed via remote
+ bcrypt@4.0.0
added 63 packages from 90 contributors and audited 510 packages in 15.614s

2 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities

Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ npm install @types/cors
+ @types/cors@2.8.6
added 1 package from 1 contributor and audited 524 packages in 3.086s

2 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities

Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ npm install @types/bcrypt
+ @types/bcrypt@3.0.0
added 1 package from 3 contributors and audited 525 packages in 8s

2 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

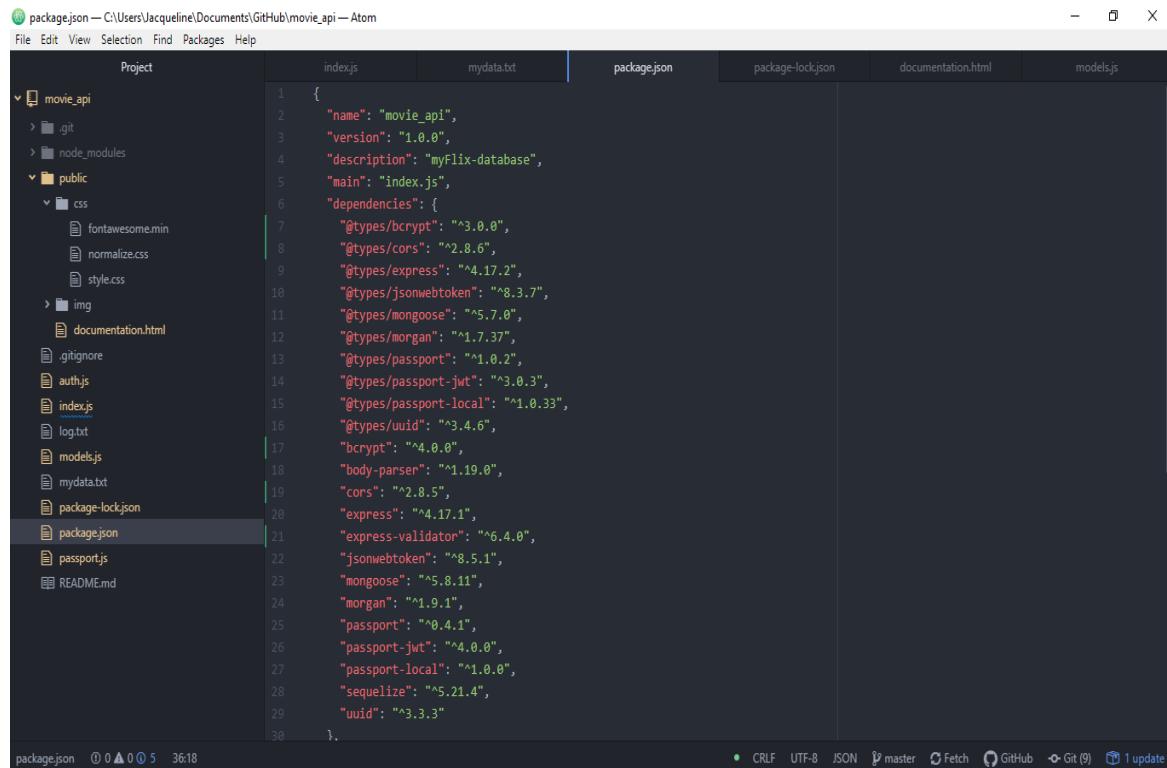


```
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ npm install express-validator
+ express-validator@6.4.0
added 2 packages from 5 contributors and audited 528 packages in 4.684s

2 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

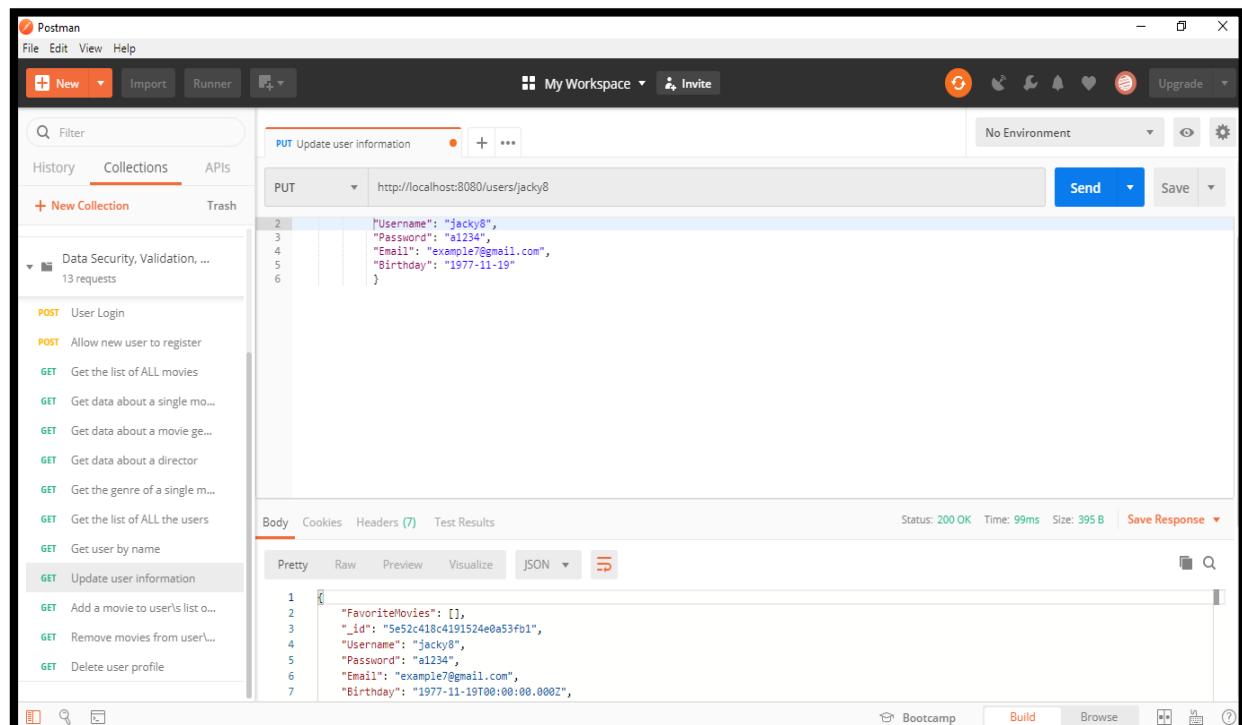
Updated package.json:



```
package.json — C:\Users\Jacqueline\Documents\GitHub\movie_api — Atom
File Edit View Selection Find Packages Help
Project index.js mydata.txt package.json package-lock.json documentation.html models.js
movie_api
  .git
  node_modules
  public
    css
      fontawesome.min
      normalize.css
      style.css
    img
    documentation.html
    .gitignore
    auth.js
    index.js
    log.txt
    models.js
    mydata.txt
    package-lock.json
    package.json
    passport.js
    README.md
index.js
{
  "name": "movie_api",
  "version": "1.0.0",
  "description": "myFlix-database",
  "main": "index.js",
  "dependencies": {
    "@types/bcrypt": "^3.0.0",
    "@types/cors": "^2.8.6",
    "@types/express": "4.17.2",
    "@types/jsonwebtoken": "8.3.7",
    "@types/mongoose": "^5.7.0",
    "@types/morgan": "1.7.37",
    "@types/passport": "^1.0.2",
    "@types/passport-jwt": "3.0.3",
    "@types/passport-local": "1.0.33",
    "@types/uuid": "^3.4.6",
    "bcrypt": "4.0.0",
    "body-parser": "^1.19.0",
    "cors": "2.8.5",
    "express": "4.17.1",
    "express-validator": "6.4.0",
    "jsonwebtoken": "8.5.1",
    "mongoose": "^5.8.11",
    "morgan": "1.9.1",
    "passport": "0.4.1",
    "passport-jwt": "4.0.0",
    "passport-local": "1.0.0",
    "sequelize": "5.21.4",
    "uuid": "3.3.3"
  }
}
```

Postman screenshots (testing code before deploying via Heroku and MongoDB Atlas)

New User Registration



The screenshot shows the Postman application interface. In the left sidebar, under 'Collections', there is a section for 'Data Security, Validation, ...' containing 13 requests. One of these requests is highlighted: 'PUT Update user information'. The main workspace shows a 'PUT' request to 'http://localhost:8080/users/jacky8'. The request body contains the following JSON payload:

```
{"Username": "jacky8", "Password": "a1234", "Email": "example7@gmail.com", "Birthday": "1977-11-19"}
```

Below the request, the 'Body' tab is selected, showing the JSON response received from the server. The response status is '200 OK'.

```
1 {
  "FavoriteMovies": [],
  "_id": "5e52c418c4191524e0a53fb1",
  "Username": "Jacky8",
  "Password": "a1234",
  "Email": "example7@gmail.com",
  "Birthday": "1977-11-19T00:00:00.000Z",
```

New User login and authorization token

The screenshot shows the Postman application interface. In the center, there is a request panel for a 'User Login' POST request to 'http://localhost:8080/login?Username=jacky&Password=password1000'. The 'query params' section contains 'Username' (jacky) and 'Password' (password1000). Below the request, the response body is displayed in JSON format:

```
1 "user": {
2     "favoriteMovies": [],
3     "_id": "5e52c418c4191524e0a53fb1",
4     "username": "jacky",
5     "password": "$2b$10$F1xi647000/TigWzTbWeK0tX05Bw4KkrJdaZ8RjOrc4bDbkaAwu",
6     "email": "example@gmail.com",
7     "birthday": "1990-06-27T00:00:00.000Z",
8     "_v": 0
9 },
10 "token":
11   "v": "eJhbGciOiJIUzI1NiIsInR5cCI6IkpXCVJ9.eyJGXVcm10ZU1vdml1cyI6W10sI19pZC16IjV1NTjJNDE4yQxOTE1MjRlMGF1M2ZiMSiI1VzZXJuYh11joiamFja3k4IiwiUGFzc3dvcnQiOjIwMmIkMTAkRnoxeGk2NDpRE8vVG1HV1plVGjXUteFhPNLU3NtrckpkQVo4UmPcmWYkR1a2FBd3U1LcJFbwFpbCI6ImV4Yh1wbgU3QGdtYhlsLmNvbSisIk3pcnR0ZGF5IjoiMTk5MC0wNi0yN1QwDowIC4wM0BaTiwiX192IjowLCjpxXQ1oje100100MwNThsImV4cCI6MTU4HtA4Nzg1Mywic3VijoiamFja3k4In0.PjHfp7IAVCojeFuasnjl_adyVb4VLbgtxALbpSTvQ"
12 }
```

Postman screenshots with authorization token (11 requests)

One: Return a list of ALL movies to the user:

The screenshot shows the Postman application interface. In the center, there is a request panel for a 'GET User Login' GET request to 'http://localhost:8080/movies'. The response body is displayed in JSON format:

```
6 },
7     "Director": {
8         "Name": "Ridley Scott",
9         "Bio": "Sir Ridley Scott is an English filmmaker",
10        "Birth": "1937"
11    },
12     "Actors": [
13         "Russell Crowe",
14         "Joaquin Phoenix",
15         "Connie Nielsen"
16     ],
17     "_id": "5e4063921733a4ac0b01c8e0",
18     "Title": "Gladiator",
19     "Year": "2000",
20     "Description": "A former Roman General sets out to exact vengeance against the corrupt emperor who murdered his family and sent him into slavery.",
21     "ImageURL": "Gladiator.png",
22     "Featured": true
23 },
24     {
25         "Genre": {
26             "Name": "Drama",
27             "Description": "In film and television, drama is a genre of narrative fiction or semi-fiction intended to be more serious than humorous in tone. All forms of cinema or television that involve fictional stories are forms of drama in the broader sense if their storytelling is achieved by means of actors who represent (mimesis) characters."
28         }
29 }
```

Two: Return data (description, genre, director, image URL, whether it's featured or not) about a single movie by title to the user:

The screenshot shows the Postman application interface. In the left sidebar, under 'Collections', there is a list of API endpoints. One endpoint, 'Get data about a single movie...', is highlighted. The main workspace shows a 'GET' request for 'http://localhost:8080/movies/Gladiator'. The 'Authorization' dropdown is set to 'Bearer Token', and a token value is displayed in the 'Token' field. The 'Preview Request' button is visible. Below the request, the 'Body' tab is selected, showing a JSON response for the movie 'Gladiator'. The response includes the genre ('Action'), description ('Action film is a film genre in which the protagonist or protagonists are thrust into a series of events that typically include violence, extended fighting, physical feats, and frantic chases. Action films tend to feature a resourceful hero struggling against incredible odds, which include life-threatening situations, a villain, or a pursuit which usually concludes in victory for the hero.'), and the director ('Ridley Scott' with bio and birth year information).

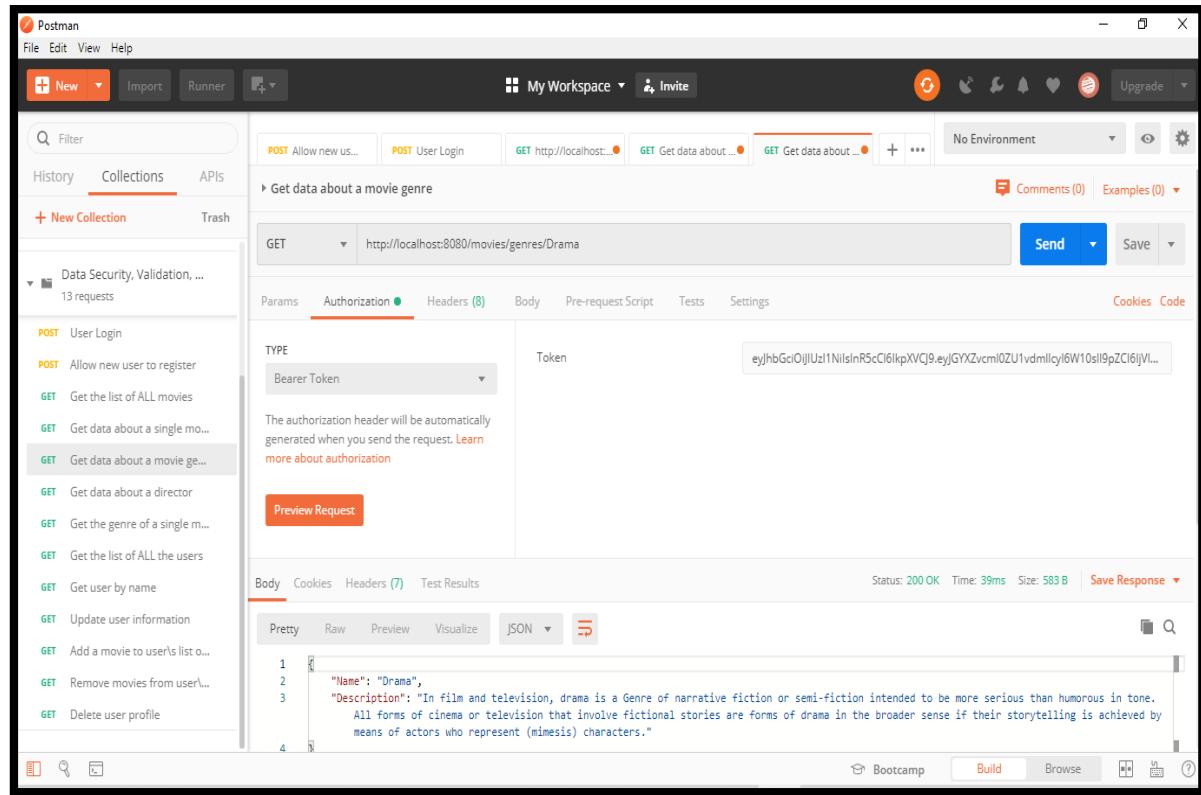
```
1 "Genre": {  
2   "Name": "Action",  
3   "Description": "Action film is a film genre in which the protagonist or protagonists are thrust into a series of events that typically include violence, extended fighting, physical feats, and frantic chases. Action films tend to feature a resourceful hero struggling against incredible odds, which include life-threatening situations, a villain, or a pursuit which usually concludes in victory for the hero."  
4 },  
5 "Director": {  
6   "Name": "Ridley Scott",  
7   "Bio": "Sir Ridley Scott is an English filmmaker",  
8   "Birth": "1937"  
9 }  
10
```

Three: Return data about a genre by the movie title (e.g., “Gladiator”):

The screenshot shows the Postman application interface. In the left sidebar, under 'Collections', there is a list of API endpoints. One endpoint, 'Get the genre of a single movie by title', is highlighted. The main workspace shows a 'GET' request for 'http://localhost:8080/movies/genres/movies/Gladiator'. The 'Authorization' dropdown is set to 'Bearer Token', and a token value is displayed in the 'Token' field. The 'Preview Request' button is visible. Below the request, the 'Body' tab is selected, showing a JSON response. The response contains a single key-value pair: 'Movie with the title : Gladiator is the Action movie.'

```
1 Movie with the title : Gladiator is the Action movie.
```

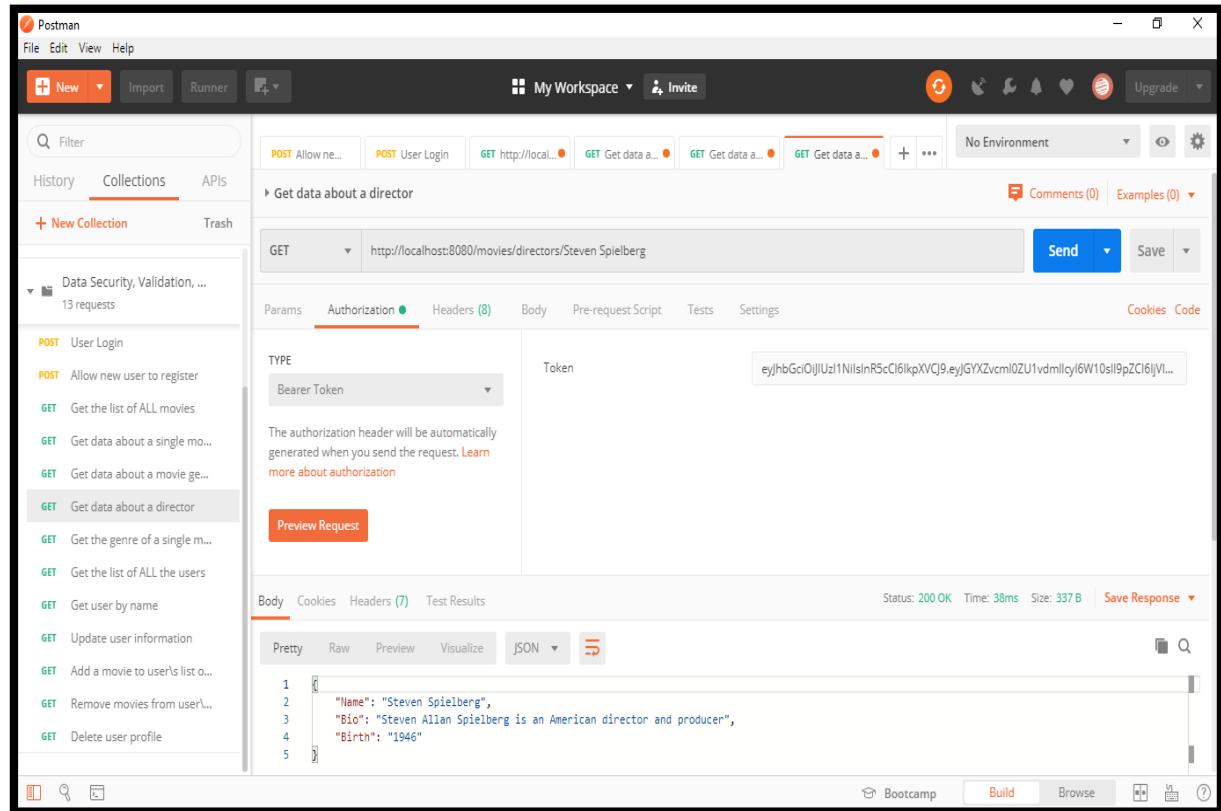
Four: Return data about a genre (description) by the genre Name (e.g., "Drama"):



The screenshot shows the Postman application interface. In the left sidebar, under the 'Collections' tab, there is a list of API endpoints. One endpoint, 'Get data about a movie genre', is selected. The main workspace shows a 'GET' request to the URL `http://localhost:8080/movies/genres/Drama`. The 'Authorization' tab is selected, showing a 'Bearer Token' type with a token value. The 'Headers' tab shows '(8)' headers. The 'Body' tab shows the response body in JSON format:

```
1 "Name": "Drama",
2 "Description": "In film and television, drama is a Genre of narrative fiction or semi-fiction intended to be more serious than humorous in tone.
All forms of cinema or television that involve fictional stories are forms of drama in the broader sense if their storytelling is achieved by
means of actors who represent (mimesis) characters."
```

Five: Return data about a director (bio, birth year, death year) by name:



The screenshot shows the Postman application interface. In the left sidebar, under the 'Collections' tab, there is a list of API endpoints. One endpoint, 'Get data about a director', is selected. The main workspace shows a 'GET' request to the URL `http://localhost:8080/movies/directors/Steven Spielberg`. The 'Authorization' tab is selected, showing a 'Bearer Token' type with a token value. The 'Headers' tab shows '(8)' headers. The 'Body' tab shows the response body in JSON format:

```
1 "Name": "Steven Spielberg",
2 "Bio": "Steven Allan Spielberg is an American director and producer",
3 "Birth": "1946"
```

Six: Return a list of ALL users:

The screenshot shows the Postman application interface. In the top navigation bar, 'File', 'Edit', 'View', and 'Help' are visible. Below the bar, there are buttons for 'New', 'Import', 'Runner', and workspace selection ('My Workspace'). The main area shows a list of requests under the 'Collections' tab. A specific GET request to 'http://localhost:8080/users' is selected. The response status is '201 Created', time is '59ms', size is '2.5 KB', and the 'Save Response' button is present. The response body is displayed in JSON format, showing two user objects. Each user has properties like 'FavoriteMovies', '_id', 'username', 'password', 'email', and 'Birthday'. The JSON is formatted with line numbers and collapsible sections.

```
1 {
  "FavoriteMovies": [
    "5e4063921733a4ac0b01c8e0",
    "5e4063921733a4ac0b01c8e1",
    "5e4063921733a4ac0b01c8e2"
  ],
  "_id": "5e44a1afc8733e0a91a9dab7",
  "username": "jondoe123",
  "password": "password123",
  "email": "example1@gmail.com",
  "Birthday": "1977-02-19T00:00:00.000Z"
},
{
  "FavoriteMovies": [
    "5e4063921733a4ac0b01c8e3",
    "5e4063921733a4ac0b01c8e4",
    "5e4063921733a4ac0b01c8e5"
  ],
  "_id": "5e44a1fc8733e0a91a9dab8",
  "username": "jack1",
  "password": "password1234",
  "email": "example2@gmail.com",
  "Birthday": "1988-04-20T00:00:00.000Z"
}
```

Seven: Allow users to update their user info (username, password, email, date of birth):

The screenshot shows the Postman application interface. In the top navigation bar, 'File', 'Edit', 'View', and 'Help' are visible. Below the bar, there are buttons for 'New', 'Import', 'Runner', and workspace selection ('My Workspace'). The main area shows a list of requests under the 'Collections' tab. A specific PUT request to 'http://localhost:8080/users/jacky8' is selected. The response status is '200 OK', time is '99ms', size is '395 B', and the 'Save Response' button is present. The response body is displayed in JSON format, showing the updated user object with the new information.

```
2 {"Username": "jacky8",
  "Password": "a1234",
  "Email": "example7@gmail.com",
  "Birthday": "1977-11-19"
}
```

```
1 {
  "FavoriteMovies": [],
  "_id": "5e52c418c4191524e0a53fb1",
  "Username": "jacky8",
  "Password": "a1234",
  "Email": "example7@gmail.com",
  "Birthday": "1977-11-19T00:00:00.000Z",
}
```

Eight: Allow users to add a movie to their list of favorites:

The screenshot shows the Postman application interface. In the top bar, there are tabs for File, Edit, View, Help, New, Import, Runner, and a workspace dropdown set to "My Workspace". Below the workspace, there are icons for Refresh, Home, Invite, and Upgrade.

The main area displays a POST request titled "Add a movie to user's list of fa...". The URL is `http://localhost:8080/users/jacky8/movies/5e4063921733a4ac0b01c8eb...`. The "Type" dropdown is set to "Bearer Token". A token value is shown in a text input field: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJXvcmI0ZU1vdmlcy6W10sII9pZC16ljlVl...`.

The "Body" tab is selected, showing a JSON payload:

```
1  {
2      "FavoriteMovies": [
3          "5e4063921733a4ac0b01c8eb",
4          "5e4063921733a4ac0b01c8eb0",
5          "5e4063921733a4ac0b01c8eb"
6      ],
7      "_id": "5e52c418c4191524e0a53fb1",
8      "Username": "jacky8",
9      "Password": "a1234",
10     "Email": "example7@gmail.com",
11     "Birthday": "1977-11-19T00:00:00.000Z",
12     "__v": 0
13 }
```

The status bar at the bottom indicates: Status: 200 OK, Time: 27ms, Size: 475 B, Save Response.

Nine: Get user by Name:

The screenshot shows the Postman application interface. The top bar and workspace are identical to the previous screenshot.

The main area displays a GET request titled "Get user by name". The URL is `http://localhost:8080/users/jacky8`. The "Type" dropdown is set to "Bearer Token". A token value is shown in a text input field: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJXvcmI0ZU1vdmlcy6W10sII9pZC16ljlVl...`.

The "Body" tab is selected, showing a JSON response:

```
1  {
2      "FavoriteMovies": [
3          "5e4063921733a4ac0b01c8eb",
4          "5e4063921733a4ac0b01c8eb",
5          "5e4063921733a4ac0b01c8eb7",
6          "5e4063921733a4ac0b01c8eb0"
7      ],
8      "_id": "5e52c418c4191524e0a53fb1",
9      "Username": "jacky8",
10     "Password": "a1234",
11     "Email": "example7@gmail.com",
12     "Birthday": "1977-11-19T00:00:00.000Z",
13     "__v": 0
14 }
```

The status bar at the bottom indicates: Status: 200 OK, Time: 36ms, Size: 503 B, Save Response.

Ten: Allow users to remove a movie from their list of favorites:

The Postman interface shows a collection named "Data Security, Validation, ...". A specific request titled "DEL Remove movies from user's list..." is selected. The method is set to "DELETE" and the URL is "http://localhost:8080/users/jacky8/movies/5e4063921733a4ac0b01c8e0". The "Authorization" tab is active, showing "Bearer Token" selected. The token field contains a long string of characters: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJGXZvcmI0ZU1vdmlcyI6W10sI9pZC16jVl... . The "Body" tab shows a JSON payload with the key "FavoriteMovies" containing two movie IDs: "5e4063921733a4ac0b01c8e8" and "5e4063921733a4ac0b01c8eb". The "Headers" tab lists "Content-Type: application/json". The response status is 200 OK with a size of 448 B.

Eleven: Allow existing users to deregister with authorization token:

The Postman interface shows a collection named "Data Security, Validation, ...". A specific request titled "Delete user profile" is selected. The method is set to "DELETE" and the URL is "http://localhost:8080/users/jacky8". The "Authorization" tab is active, showing "Bearer Token" selected. The token field contains a long string of characters: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJGXZvcmI0ZU1vdmlcyI6W10sI9pZC16jVl... . The "Body" tab shows a JSON payload with the message "jacky8 was deleted.". The "Headers" tab lists "Content-Type: application/json". The response status is 200 OK with a size of 238 B.

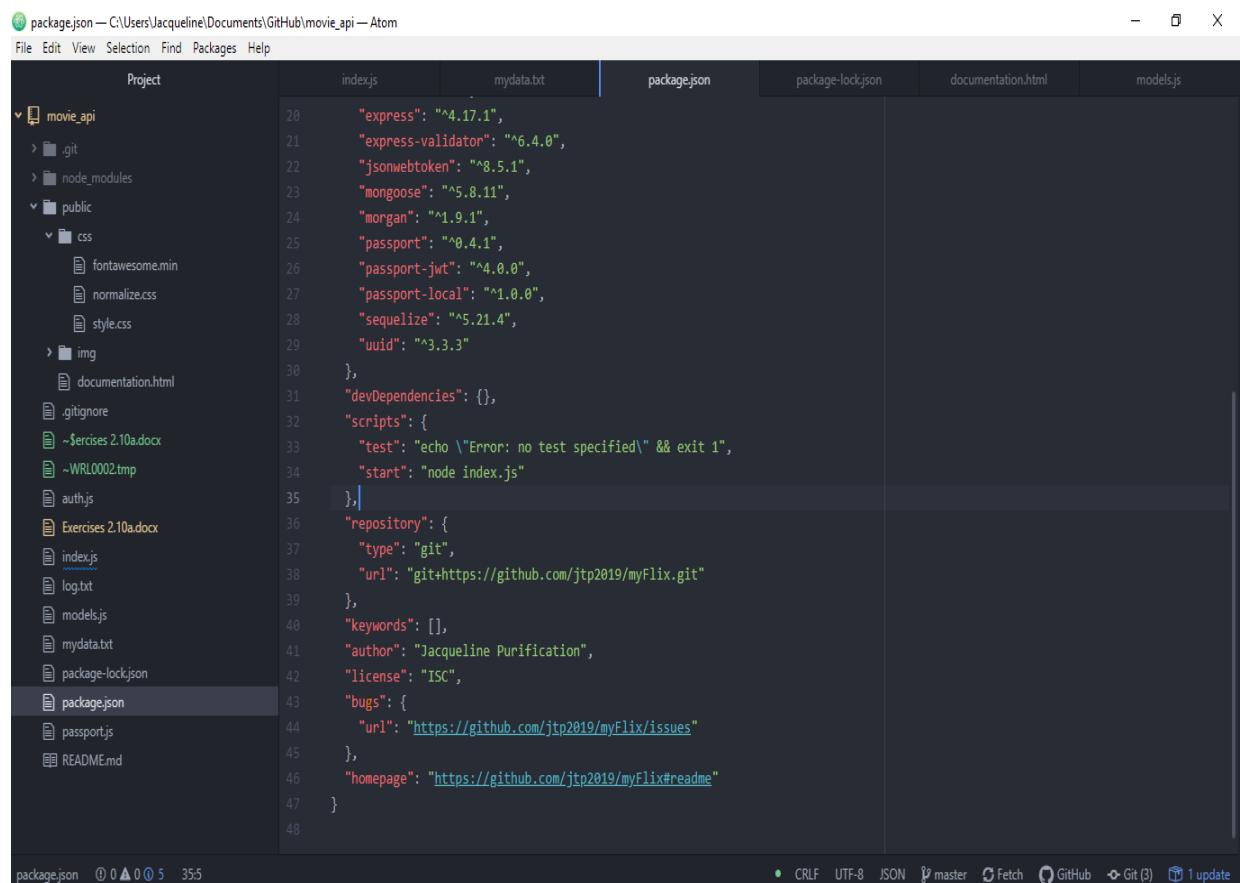
Exercises 2.10b

HEROKU

Installed Heroku via command prompt:

```
C:\ Select Command Prompt - heroku login -i  
Microsoft Windows [Version 10.0.18362.175]  
(c) 2019 Microsoft Corporation. All rights reserved.  
  
C:\Users\Jacqueline>heroku login  
heroku: Press any key to open up the browser to login or q to exit:  
Opening browser to https://cli-auth.herokuapp.com/auth/cli/browser/a98993cf-9fab-409d-b136-2c18c68258dc  
Logging in... done  
Logged in as jackpr77@gmail.com  
  
C:\Users\Jacqueline>heroku --version  
heroku/7.38.2 win32-x86 node-v12.13.0  
  
C:\Users\Jacqueline>heroku login  
heroku: Press any key to open up the browser to login or q to exit:  
Opening browser to https://cli-auth.herokuapp.com/auth/cli/browser/3a2f0d95-a70d-424f-b9f4-0b3847b57e68  
Logging in... done  
Logged in as jackpr77@gmail.com
```

Updated package.json added "start": "node index.js" :

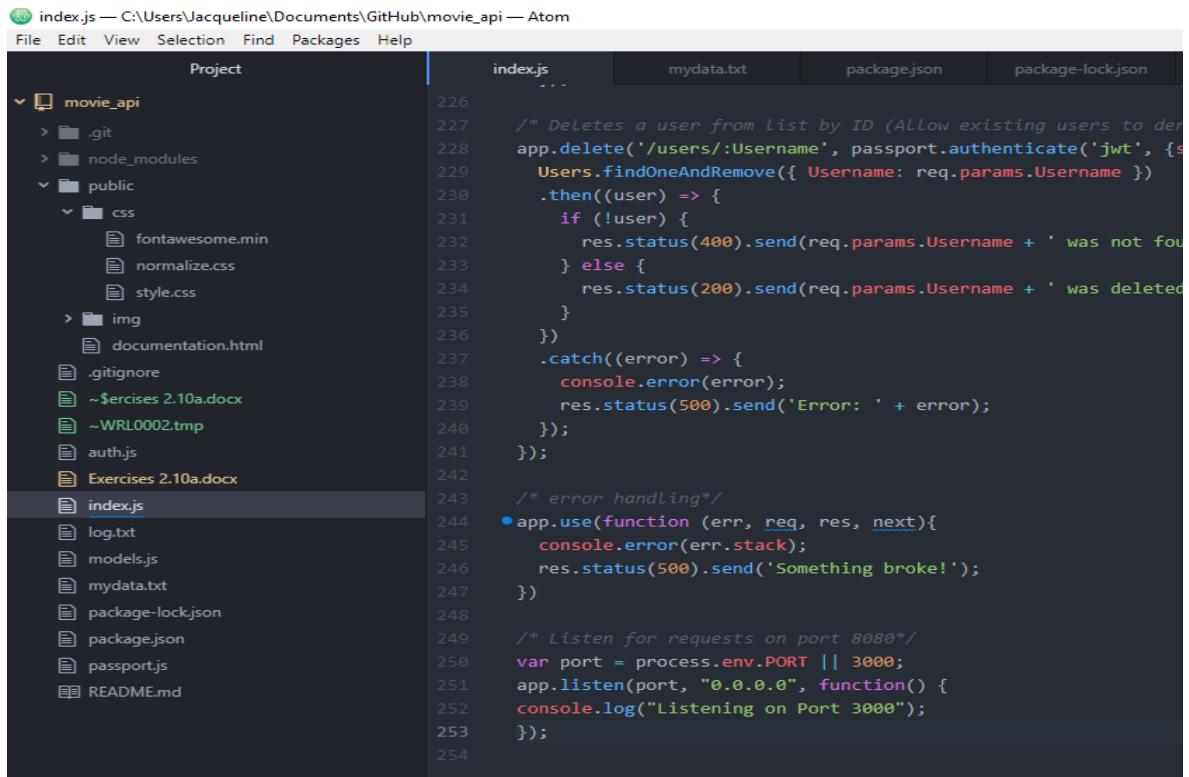


The screenshot shows the Atom code editor interface with the file 'package.json' selected in the sidebar. The code editor displays the following JSON content:

```
 20   "express": "^4.17.1",  
 21   "express-validator": "^6.4.0",  
 22   "jsonwebtoken": "^8.5.1",  
 23   "mongoose": "^5.8.11",  
 24   "morgan": "^1.9.1",  
 25   "passport": "^0.4.1",  
 26   "passport-jwt": "^4.0.0",  
 27   "passport-local": "^1.0.0",  
 28   "sequelize": "^5.21.4",  
 29   "uuid": "^3.3.3"  
 30 },  
 31 "devDependencies": {},  
 32 "scripts": {  
 33   "test": "echo \\\"Error: no test specified\\\" && exit 1",  
 34   "start": "node index.js"  
 35 },  
 36 "repository": {  
 37   "type": "git",  
 38   "url": "git+https://github.com/jtp2019/myFlix.git"  
 39 },  
 40 "keywords": [],  
 41 "author": "Jacqueline Purification",  
 42 "license": "ISC",  
 43 "bugs": {  
 44   "url": "https://github.com/jtp2019/myFlix/issues"  
 45 },  
 46 "homepage": "https://github.com/jtp2019/myFlix#readme"  
 47 }  
 48 }
```

The status bar at the bottom of the editor shows the file path 'C:\Users\Jacqueline\Documents\GitHub\movie_api', the file name 'package.json', line count '0', character count '355', and various GitHub and Git-related icons.

Updated app.listen();



The screenshot shows the Atom code editor interface. The left sidebar displays a project structure for a 'movie_api' folder containing '.git', 'node_modules', 'public' (with 'css' and 'img' subfolders), 'documentation.html', '.gitignore', 'Exercises 2.10a.docx', 'auth.js', 'index.js' (which is the active tab), 'log.txt', 'models.js', 'mydata.txt', 'package-lock.json', 'package.json', 'passport.js', and 'README.md'. The right pane shows the content of 'index.js' with line numbers 226 through 254. The line 'app.listen(3000, function() { console.log("Listening on Port 3000"); })'; is highlighted with a blue dot and a red rectangle.

```
/* Deletes a user from list by ID (Allow existing users to delete themselves) */
app.delete('/users/:Username', passport.authenticate('jwt', {session: false}), (req, res) => {
  User.findOneAndRemove({ Username: req.params.Username })
    .then((user) => {
      if (!user) {
        res.status(400).send(req.params.Username + ' was not found');
      } else {
        res.status(200).send(req.params.Username + ' was deleted');
      }
    })
    .catch((error) => {
      console.error(error);
      res.status(500).send('Error: ' + error);
    });
});

/* error handling */
app.use(function (err, req, res, next){
  console.error(err.stack);
  res.status(500).send('Something broke!');
})

/* Listen for requests on port 8080*/
var port = process.env.PORT || 3000;
app.listen(port, "0.0.0.0", function() {
  console.log("Listening on Port 3000");
});
```

Push changes to Git and heroku master:

```
MINGW32/c/Users/Jacqueline/Documents/GitHub/movie_api
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:  Exercises 2.9.pdf
    modified: auth.js
    modified: index.js
    modified: models.js
    modified: package-lock.json
    modified: package.json
    modified: passport.js
    modified: public/documentation.html
    deleted:  response.json

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Exercises 2.10a.docx

no changes added to commit (use "git add" and/or "git commit -a")

Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git add .
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  Exercises 2.10a.docx
    deleted:  Exercises 2.9.pdf
    modified: auth.js
    modified: index.js
    modified: models.js
    modified: package-lock.json
    modified: package.json
    modified: passport.js
    modified: public/documentation.html
    deleted:  response.json

Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git commit -m "installed CORS, bcrypt and express-validator"
[master f7b84bc] installed CORS, bcrypt and express-validator
```

Create and deploy my app on Heroku:

```
MINGW32:/c/Users/Jacqueline/Documents/GitHub/movie_api
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git push heroku
Enumerating objects: 109, done.
Counting objects: 100% (109/109), done.
Delta compression using up to 4 threads
Compressing objects: 100% (109/109) done.
Writing objects: 100% (109/109) 8.93 KiB | 91.00 KiB/s, done.
Total 109 (delta 39), reused 0 (delta 0)
remote: Compressing source files...
remote: Building source:
remote:
remote: ----> Node.js app detected
remote: ----> Creating runtime environment
remote:
remote:   NPM_CONFIG_LOGLEVEL=error
remote:   NODE_ENV=production
remote:   NODE_MODULES_CACHE=true
remote:   NODE_VERBOSE=false
remote:
remote: ----> Installing binaries
remote:   engines.node (package.json): unspecified
remote:   engines.npm (package.json): unspecified (use default)
remote:
remote:   Resolving node version 12.x...
remote:   Downloading and installing node 12.16.1...
remote:   Using default npm version: 6.13.4
remote:
remote: ----> Installing dependencies
remote:   Installing node modules (package.json + package-lock)
remote:
remote: > bcrypt@4.0.0 install /tmp/build_52972d4e85379955ede1926c4b87e57a/node_modules/bcrypt
remote: > node-pre-gyp install --fallback-to-build
remote:
[bcrypt] Success: "/tmp/build_52972d4e85379955ede1926c4b87e57a/node_modules/bcrypt/lib/binding/napi-v3/bcrypt_lib.node" is installed via remote
remote: added 196 packages from 341 contributors and audited 528 packages in 6.086s
remote:
remote: 2 packages are looking for funding
remote:   run `npm fund` for details
remote:
remote: found 0 vulnerabilities
remote:
remote: ----> Build
remote:
remote: ----> Caching build
remote:   - node_modules
remote:
remote: ----> Pruning devDependencies
```

```
MINGW32:/c/Users/Jacqueline/Documents/GitHub/movie_api
remote: Resolving node version 12.x...
remote: Downloading and installing node 12.16.1...
remote: Using default npm version: 6.13.4
remote:
remote: ----> Installing dependencies
remote:   Installing node modules (package.json + package-lock)
remote:
remote: > bcrypt@4.0.0 install /tmp/build_52972d4e85379955ede1926c4b87e57a/node_modules/bcrypt
remote: > node-pre-gyp install --fallback-to-build
remote:
[bcrypt] Success: "/tmp/build_52972d4e85379955ede1926c4b87e57a/node_modules/bcrypt/lib/binding/napi-v3/bcrypt_lib.node" is installed via remote
remote: added 196 packages from 341 contributors and audited 528 packages in 6.086s
remote:
remote: 2 packages are looking for funding
remote:   run `npm fund` for details
remote:
remote: found 0 vulnerabilities
remote:
remote: ----> Build
remote:
remote: ----> Caching build
remote:   - node_modules
remote:
remote: ----> Pruning devDependencies
remote:   audited 528 packages in 1.73s
remote:
remote: 2 packages are looking for funding
remote:   run `npm fund` for details
remote:
remote: found 0 vulnerabilities
remote:
remote: ----> Build succeeded!
remote: ----> Discovering process types
remote:   Procfile declares types      -> (none)
remote:   Default types for buildpack -> web
remote:
remote: ----> Compressing...
remote:   Done: 28.7M
remote: ----> Launching...
remote:   Released v7
remote:   https://myflix-db1.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/myFlix-db1.git
 * [new branch]    master -> master
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
```

The screenshot shows the Heroku dashboard for the app 'rhubarb-crisp-92657'. At the top, there's a navigation bar with links for Overview, Resources, Deploy, Metrics, Activity, Access, and Settings. Below this, there's a section for 'Installed add-ons' which says '\$0.00/month' and a link to 'Configure Add-ons'. A message states 'There are no add-ons for this app' and suggests adding them. Another section for 'Dyno formation' also shows '\$0.00/month' and a link to 'Configure Dynos'. It indicates that the app is using free dynos. Under 'Collaborator activity', there's a link to 'Manage Access'. On the right side, there's a 'Latest activity' log with several entries from 'jackpr77@gmail.com' detailing deployments and build successes.

Upload database to MongoDB Atlas via Studio 3T

1st Step: Installed Studio 3T for 32 bit Win 10, Connect S3T with mongodb database (device) and added all collections to myFlixDB (localhost 27027) folder. After successfully getting the collections export & execute those data in Json format to my PC. Which will automatic created a folder in my PC named myFlixDB.

The screenshot shows the Studio 3T for MongoDB interface. The left sidebar lists connections: 'myFlixDB localhost:27017 [direct]' and 'myFlixDB-shard-0 [replica set]'. The main area features a 'Welcome to Studio 3T Enterprise' message with a note about the trial ending in 19 days. It includes a 'Recent Connections' list with items like 'myFlixDB-shard-0 (myBadmin@myflixdb-kow93.mongodb.net)' and 'myFlixDB (localhost:27017)'. To the right, there are sections for 'Tasks', 'Help and Learning' (with links to MongoDB 101 course, Getting started, Studio 3T features, Knowledge base, MongoDB tutorials, Studio 3T tips and tricks, and a survey), and 'Quick Options' with checkboxes for 'Show What's New tab after updating Studio 3T', 'Enable dark theme (requires restart)', and 'Automatically open Connection Manager at startup'.

2nd Step: Open account on MongoDB atlas, created a cluster named myFlixDB, and setup database & network access

The screenshot shows the MongoDB Atlas interface. On the left, a sidebar menu under 'ATLAS' has 'Clusters' selected. The main content area displays the 'Clusters' section for the 'myFlixDB' cluster. Key details shown include:

- Cluster Name:** myFlixDB, Version 4.2.3
- Cluster Tier:** Mo Sandbox (General)
- Region:** AWS / Frankfurt (eu-central-1)
- Type:** Replica Set - 3 nodes
- Linked Stitch App:** None Linked

Metrics and logs are displayed in a chart format for the last 6 hours and 30 days. A green banner at the top right encourages trying the MongoDB Atlas Search beta.

The screenshot shows the 'Database Access' section of the MongoDB Atlas interface. Under 'Database Users', there is one entry:

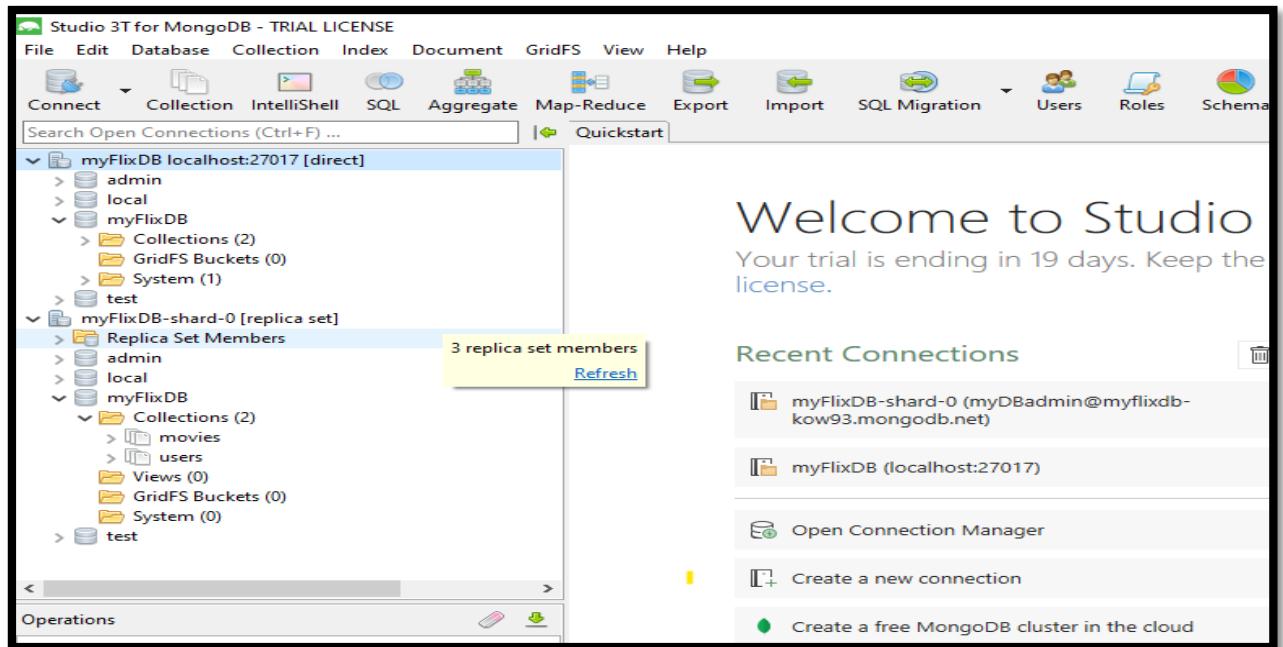
User Name	Authentication Method	MongoDB Roles	Actions
myDBAdmin	SCRAM	atlasAdmin@admin	<button>EDIT</button> <button>DELETE</button>

The screenshot shows the 'Network Access' section of the MongoDB Atlas interface. Under 'IP Whitelist', there is one entry:

IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		Active	<button>EDIT</button> <button>DELETE</button>

3rd Step: Connected Studio 3T with MonngoDB Cluster

Open a new connection named myFlixDB-shard-0 and export & executed all data in Json format from myFlixDB (localhost 27027) to my PC in different folder named myFlixDB and from that folder import data to the myFlixDB-shard-0 folder.



Now the collections are showing on the MongoDB Atlas

A screenshot of the MongoDB Atlas web interface. The left sidebar shows 'Clusters' selected under 'ATLAS'. The main pane shows the 'myFlixDB.movies' collection with a collection size of 11.49KB, 13 documents, and 20KB total index size. The 'Find' button is highlighted. Below it, the 'QUERY RESULTS 1-13 OF 13' section displays two document snippets. The first snippet is for 'Gladiator' and the second for 'Schindler's List'.

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Clusters' selected under 'ATLAS'. The main area shows 'myFlixDB' with 'users' and 'movies' collections. The 'users' collection has 5 documents. One document is expanded to show fields: _id: ObjectId("5e44a1afc8733e0a91a9dab7"), username: "jondoe123", password: "password123", email: "example@gmail.com", Birthdate: 1977-02-19T00:00:00.000Z, and FavoriteMovies: Array.

Connected MongoDB atlas with Heroku (GitHub)

1st Step: From MongoDB atlas collected connection string

'mongodb+srv://myDBadmin:12345@myflixdb-kow93.mongodb.net/test?retryWrites=true&w=majority' changed the format to mongoose.connect(process.env.mongodb+srv://myDBadmin:12345@myflixdb-kow93.mongodb.net/myFlixDB?retryWrites=true&w=majority', { useNewUrlParser: true, useUnifiedTopology: true }); and added that URL to index.js and paste it into Heroku app Config Vars.

The screenshot shows the 'Connect to myFlixDB' wizard. Step 1: Select your driver and version (Node.js, 3.0 or later). Step 2: Add your connection string into your application code. The connection string is shown as: mongodb+srv://myDBadmin:<password>@myflixdb-kow93.mongodb.net/test?retryWrites=true&w=majority. A note says to replace <password> with the password for the user, myDBadmin, and ensure all special characters are URL encoded. A 'Copy' button is also present.

App Information

App Name: rhubarb-crisp-92657

Region: United States

Stack: heroku-18

Framework: Node.js

Slug size: 30.1 MiB of 500 MiB

Heroku git URL: <https://git.heroku.com/rhubarb-crisp-92657.git>

Config Vars

Config Vars

[Hide Config Vars](#)

CONNECTION_URI: mongoose.connect(process.env.mongodb+srv)

2nd Step: Push all changes to Heroku and GitHub master.

```
MINGW32:/c/Users/Jacqueline/Documents/GitHub/movie_api
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git add .
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git commit -am "Correction port connection"
[master e1c6b00] Correction port connection
 1 file changed, 1 insertion(+), 1 deletion(-)
Jacqueline@Jacqueline MINGW32 ~/Documents/GitHub/movie_api (master)
$ git push heroku master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 309 bytes | 103.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: ----> Node.js app detected
remote: ----> Creating runtime environment
remote:
remote:   NPM_CONFIG_LOGLEVEL=error
remote:   NODE_ENV=production
remote:   NODE_MODULES_CACHE=true
remote:   NODE_VERBOSE=false
remote:
remote: ----> Installing binaries
remote: engines.node (package.json): unspecified
remote: engines.npm (package.json): unspecified (use default)
remote:
remote:   Resolving node version 12.x...
remote:   Downloading and installing node 12.16.1...
remote:   Using default npm version: 6.13.4
remote:
remote: ----> Restoring cache
remote:   - node_modules
remote:
remote: ----> Installing dependencies
remote:   Installing node modules (package.json + package-lock)
remote:   audited 528 packages in 1.667s
remote:
remote:   2 packages are looking for funding
remote:     run 'npm fund' for details
remote:
remote:   found 0 vulnerabilities
remote:
```

New URL from Heroku

<https://rhubarb-crisp-92657.herokuapp.com/>

3rd Step: Checked Heroku logs

```
Admin@Jack MINGW64 ~/Documents/GitHub/movie_api (master)
$ heroku logs --app rhubarb-crisp-92657
2020-04-01T23:00:54.914748+00:00 app[api]: Initial release by user jackpr77@gmail.com
2020-04-01T23:00:55.297417+00:00 app[api]: Release v2 created by user jackpr77@gmail.com
2020-04-01T23:00:55.297417+00:00 app[api]: Enable Logplex by user jackpr77@gmail.com
2020-04-01T23:00:54.914748+00:00 app[api]: Release v1 created by user jackpr77@gmail.com
2020-04-01T23:19:50.000000+00:00 app[api]: Build started by user jackpr77@gmail.com
2020-04-01T23:20:11.251937+00:00 app[api]: Deploy 1e07e7c7 by user jackpr77@gmail.com
2020-04-01T23:20:11.251937+00:00 app[api]: Release v3 created by user jackpr77@gmail.com
2020-04-01T23:20:11.271106+00:00 app[api]: Scaled to web@1:Free by user jackpr77@gmail.com
2020-04-01T23:20:12.000000+00:00 app[api]: Build succeeded
2020-04-01T23:20:16.773156+00:00 app[web.1]:
2020-04-01T23:20:16.773179+00:00 app[web.1]: > movie_api@1.0.0 start /app
2020-04-01T23:20:16.773180+00:00 app[web.1]: > node index.js
2020-04-01T23:20:16.773180+00:00 app[web.1]:
2020-04-01T23:20:17.540293+00:00 app[web.1]: Listening on Port 3000
```

Postman screenshots (testing code before deploying via Heroku and MongoDB Atlas)

New User Registration

The screenshot shows the Postman application interface. On the left, the sidebar displays collections: 'Heroku app2' (8 requests), 'Mongoose CRUD Request' (12 requests), 'Movie REST API Request' (9 requests), and 'Heroku app' (8 requests). Under 'Heroku app', there are three requests listed: 'POST User login', 'POST User registration' (which is highlighted in grey), and 'PUT User Update'. The main workspace shows a POST request for 'User registration' to the URL <https://rhubarb-crisp-92657.herokuapp.com/users>. The request body is shown in JSON format:

```
1  "FavoriteMovies": [],
2  "_id": "5eb8545ee7dc7bd0001701b07d",
3  "Username": "jacky3",
4  "Password": "$2b$10$B0GpsizIZYdwqy0SPRTbuwe54iiJNfe30Bsg.IwGvvGX1KT0phU7A2",
5  "Email": "example@gmail.com",
6  "Birthday": "1957-06-27T00:00:00.000Z",
7  "__v": 0
```

New User login and authorization token

The screenshot shows the Postman application interface. In the center, there is a 'POST' request to the URL <https://rhubarb-crisp-92657.herokuapp.com/login?Username=jacky3&Password=password123>. The response status is 200 OK, time is 586ms, and size is 933 B. The response body is displayed in JSON format:

```
1 "user": {
2     "FavoriteMovies": [],
3     "_id": "5eb545ee7dc7bd001701b07d",
4     "Username": "jacky3",
5     "Password": "$2b$10$8GOpZsiIZYdwqy0SPRTbuwe54liJNfe30Bsg.IwGvVgXlKT0phu7A2",
6     "Email": "example7@gmail.com",
7     "Birthday": "1957-06-27T00:00:00Z",
8     "__v": 0
9 },
10 "token":
11     "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey3GYXzcmI0ZU1vdml1cyI6wI0sIl9pZC16IjVl0DU0MwA1n2rjN23kMDAxYjA32CisI1VzZjXuW11
12     Ijoiamrja3kZiiwiuUGFzC3dvcmcQloikwmIkwtAkqkdQenlpekLZHdxeu01UFJUyvN3ZTU0aw1KtMz1MzBCc0cuSvdhdnZhwexLVD8waFu3QT11Cj7bfpbC
13     I61mV4YwlwbGU3QGdtYwlsLmVbs1sk1pcnRoZGF5ijoiMTk1hy0Wl0yN1QwMDoWFC4wIDBa1iwi1921jowLCjpyXQiOjE100U30T12M0usimV4cc16
14     MTU4NjM5NzQwfSwic3Vi1joiamrja3kzIn0.Rq5ByFDegr0819c8wR2wD3UDjQEBTTfHmqJqB7ztya"
```

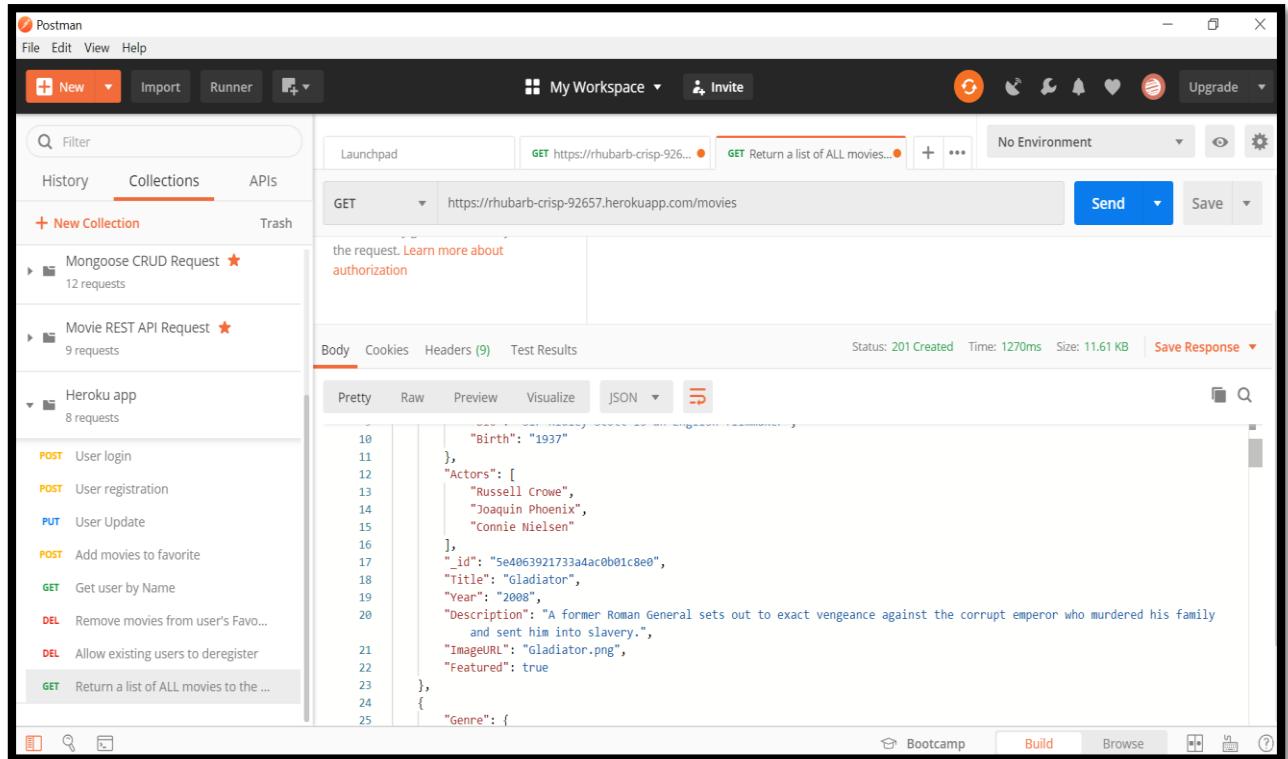
New users are successfully added with the MongoDB Atlas

The screenshot shows the MongoDB Atlas interface. On the left, the sidebar lists 'Clusters' under 'ATLAS'. The main area displays two user documents in a table:

Document	Username	Password	Email	Birthday
Object ID: 5eb545ee7dc7bd001701b07e	jacky3	\$1234	example7@gmail.com	1957-11-19T00:00:00.000+00:00
Object ID: 5eb553247dc7bd001701b07f	jacky77	\$2b\$10\$Qo517akuZIvGvvaOnuydoY..SxIBCKdgcU0cFH3B1Dp0PZT.UB6epy	example7@gmail.com	1957-06-27T00:00:00.000+00:00

Postman screenshots with authorization token (11 requests)

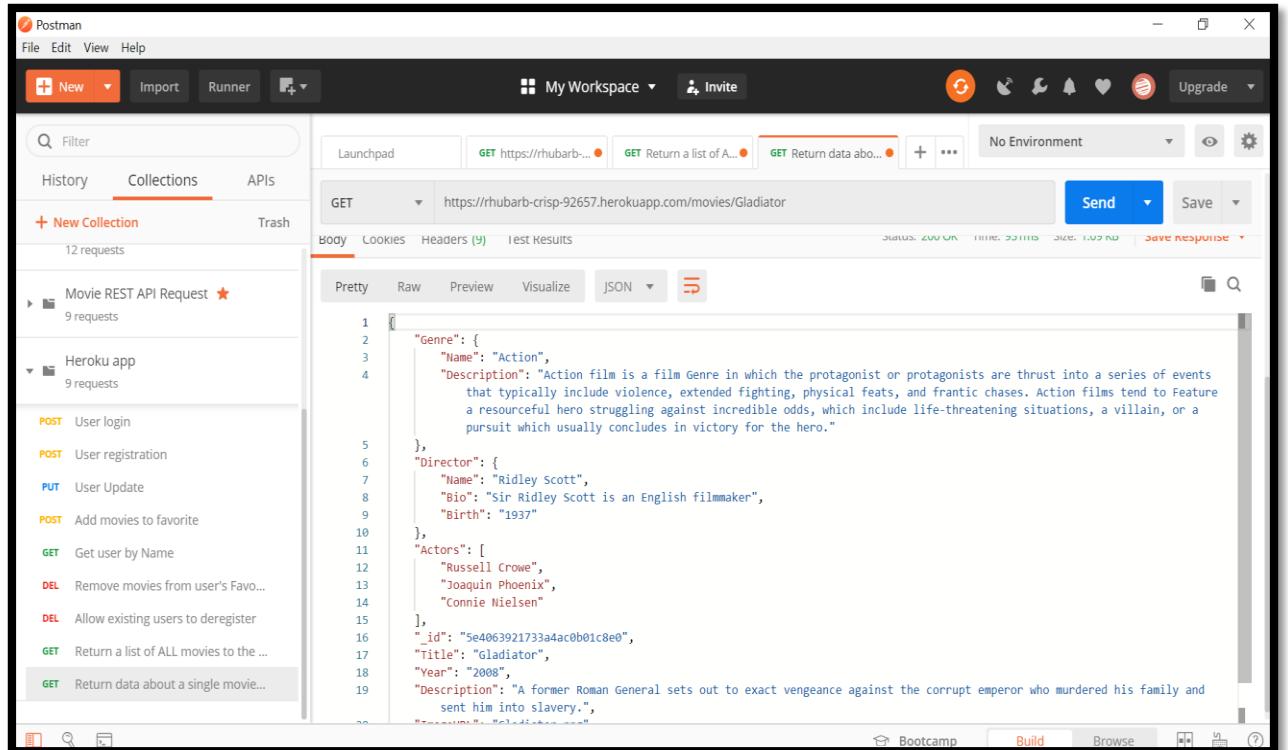
One: Return a list of ALL movies to the user:



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'History', 'Collections', and 'APIs'. Under 'Collections', there are two entries: 'Mongoose CRUD Request' (12 requests) and 'Movie REST API Request' (9 requests). The 'Movie REST API Request' collection is expanded, showing various methods: POST User login, POST User registration, PUT User Update, POST Add movies to favorite, GET Get user by Name, DEL Remove movies from user's Favo..., DEL Allow existing users to deregister, and GET Return a list of ALL movies to the ...

In the main workspace, there are three tabs: 'Launchpad', 'GET https://rhubarb-crisp-92657.herokuapp.com/movies', and 'GET Return a list of ALL movies...'. The second tab is selected. The 'Body' tab is active, showing the response content. The response is a JSON object with multiple movie entries. One entry is highlighted, showing details like 'Title': 'Gladiator', 'Year': '2008', 'Description': 'A former Roman General sets out to exact vengeance against the corrupt emperor who murdered his family and sent him into slavery.', and 'Genre': 'Action'.

Two: Return data (description, genre, director, image URL, whether it's featured or not) about a single movie by title to the user:



This screenshot shows the same Postman interface as the previous one. The 'Movie REST API Request' collection is still expanded, and the 'GET' request to 'https://rhubarb-crisp-92657.herokuapp.com/movies/Gladiator' is selected. The response body shows a JSON object for the movie 'Gladiator'. The object includes the 'Genre' (Action), 'Director' (Ridley Scott), 'Actors' (Russell Crowe, Joaquin Phoenix, Connie Nielsen), and a detailed 'Description' of the movie's plot.

Three: Return data about a genre by the movie title (e.g., “Gladiator”):

The screenshot shows the Postman application interface. In the left sidebar, under 'Collections', there is a 'Movie REST API Request' collection with 9 requests. One specific request is highlighted: a GET request to `https://rhubarb-crisp-92657.herokuapp.com/movies/genres/movies/Gladiator`. The 'Authorization' tab in the request settings is selected, showing 'Bearer Token' and a token value. The response body is displayed as:

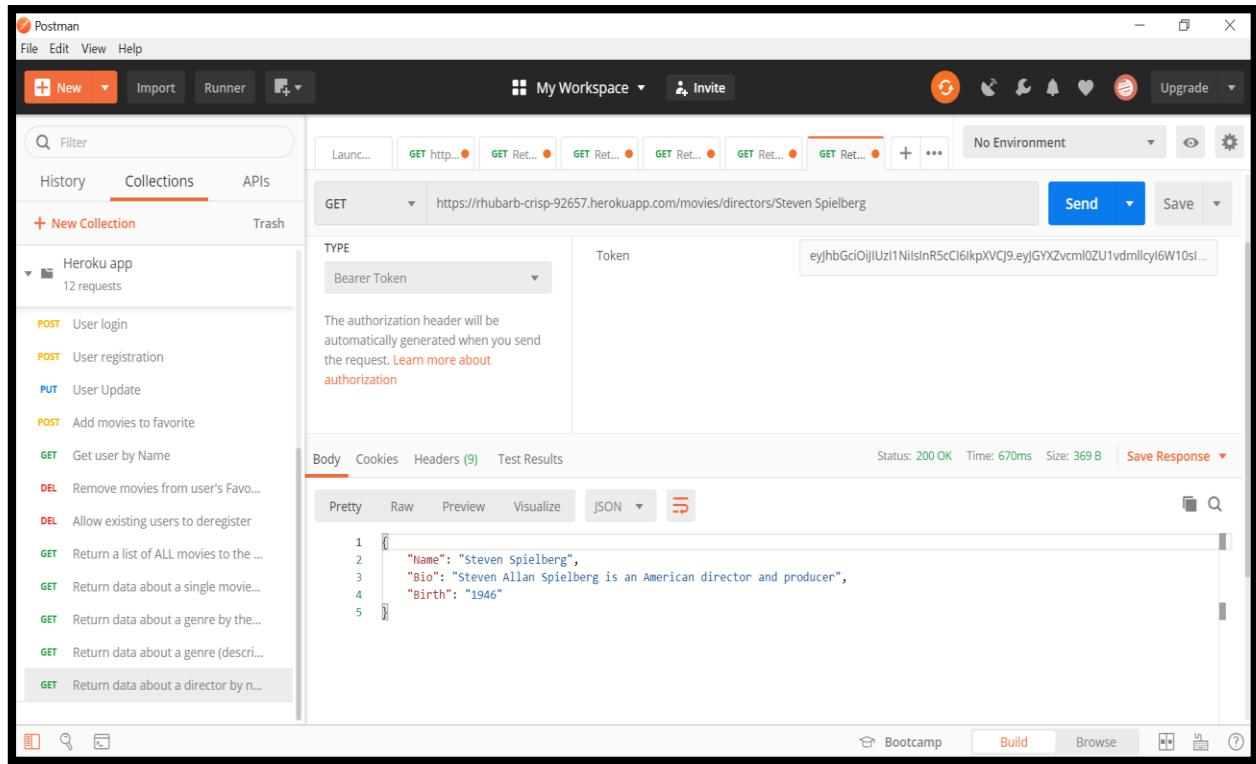
```
1 Movie with the title : Gladiator is the Action movie.
```

Four: Return data about a genre (description) by the genre Name (e.g., “Drama”):

The screenshot shows the Postman application interface. In the left sidebar, under 'Collections', there is a 'Movie REST API Request' collection with 11 requests. One specific request is highlighted: a GET request to `https://rhubarb-crisp-92657.herokuapp.com/movies/genres/Drama`. The 'Authorization' tab in the request settings is selected, showing 'Bearer Token' and a token value. The response body is displayed as JSON:

```
1 {
2   "Name": "Drama",
3   "Description": "In film and television, drama is a genre of narrative fiction or semi-fiction intended to be more serious than humorous in tone. All forms of cinema or television that involve fictional stories are forms of drama in the broader sense if their storytelling is achieved by means of actors who represent (mimesis) characters."
4 }
```

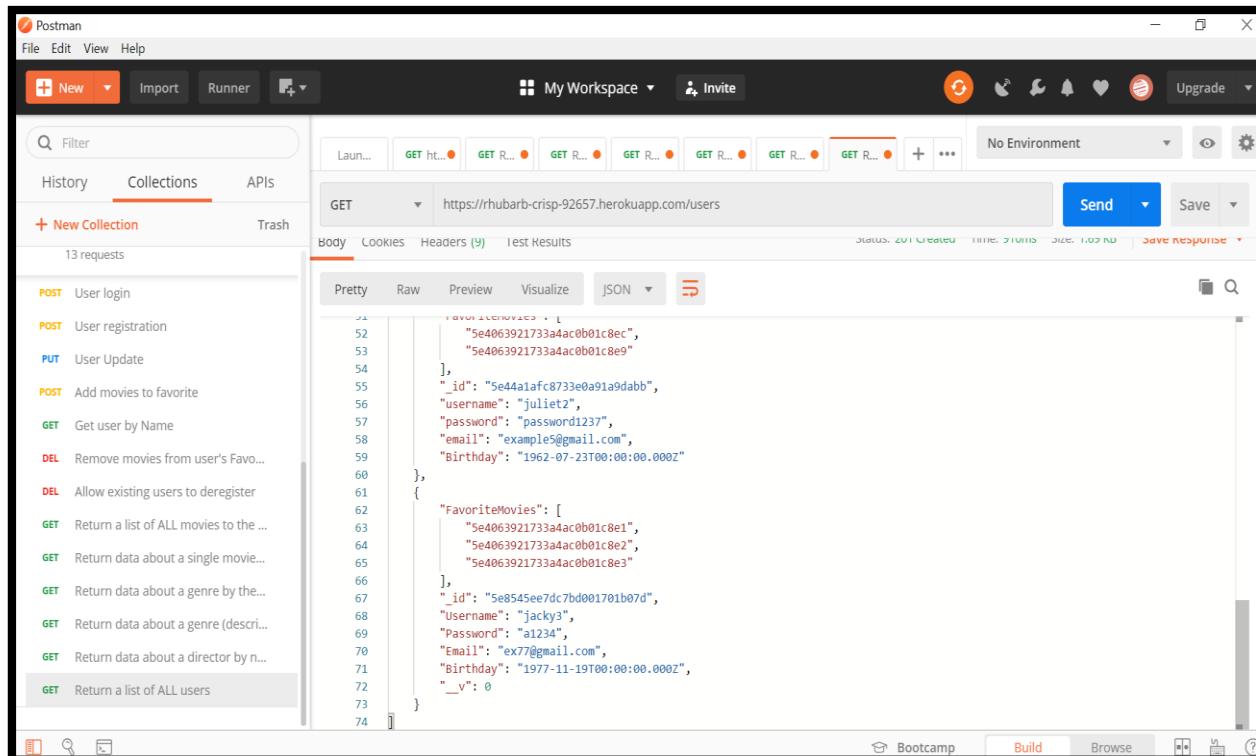
Five: Return data about a director (bio, birth year, death year) by name:



The screenshot shows the Postman application interface. In the left sidebar, under the 'Collections' tab, there is a section for 'Heroku app' with 12 requests listed. One of these requests is highlighted: 'GET Return data about a director by name'. The main workspace shows a GET request to the URL <https://rhubarb-crisp-92657.herokuapp.com/movies/directors/Steven Spielberg>. The 'Headers' tab shows a 'Bearer Token' header. The 'Body' tab displays the JSON response:

```
1 {
2   "Name": "Steven Spielberg",
3   "Bio": "Steven Allan Spielberg is an American director and producer",
4   "Birth": "1946"
5 }
```

Six: Return a list of ALL users:



The screenshot shows the Postman application interface. In the left sidebar, under the 'Collections' tab, there is a section for 'Heroku app' with 13 requests listed. One of these requests is highlighted: 'GET Return a list of ALL users'. The main workspace shows a GET request to the URL <https://rhubarb-crisp-92657.herokuapp.com/users>. The 'Headers' tab shows a 'Bearer Token' header. The 'Body' tab displays the JSON response, which is a list of user documents:

```
1 [
2   {
3     "_id": "5e4063921733a4ac0b01c8ec",
4     "username": "juliets",
5     "password": "password1237",
6     "email": "example5@gmail.com",
7     "Birthday": "1962-07-23T00:00:00.000Z"
8   },
9   {
10     "FavoriteMovies": [
11       "5e4063921733a4ac0b01c8e1",
12       "5e4063921733a4ac0b01c8e2",
13       "5e4063921733a4ac0b01c8e3"
14     ],
15     "_id": "5e8545ee7dc7bd001701b07d",
16     "username": "jacky3",
17     "password": "a1234",
18     "email": "ex77@gmail.com",
19     "Birthday": "1977-11-19T00:00:00.000Z",
20     "__v": 0
21   }
22 ]
```

Seven: Allow users to update their user info (username, password, email, date of birth):

The screenshot shows the Postman application interface. In the left sidebar, under 'APIs', the 'PUT User Update' request is selected. The main workspace shows a 'PUT' request to the URL <https://rhubarb-crisp-92657.herokuapp.com/users/jacky3>. The response status is 200 OK, time 681ms, size 423 B. The JSON response body is:

```
1  [
2    "FavoriteMovies": [],
3    "_id": "5e8545ee7dc7bd001701b07d",
4    "Username": "jacky3",
5    "Password": "a1234",
6    "Email": "ex77@gmail.com",
7    "Birthday": "1977-11-19T00:00:00.000Z",
8    "__v": 0
9 ]
```

Eight: Allow users to add a movie to their list of favourites:

The screenshot shows the Postman application interface. In the left sidebar, under 'APIs', the 'POST Add movies to favorite' request is selected. The main workspace shows a 'POST' request to the URL <https://rhubarb-crisp-92657.herokuapp.com/users/jacky3/movies/5e4063921733a4ac0b01c8e0>. The response status is 200 OK, time 681ms, size 449 B. The JSON response body is identical to the previous screenshot:

```
1  [
2    "FavoriteMovies": [
3      "5e4063921733a4ac0b01c8e0"
4    ],
5    "_id": "5e8545ee7dc7bd001701b07d",
6    "Username": "jacky3",
7    "Password": "a1234",
8    "Email": "ex77@gmail.com",
9    "Birthday": "1977-11-19T00:00:00.000Z",
10   "__v": 0
11 ]
```

Nine: Get user by Name:

The screenshot shows the Postman application interface. In the left sidebar, under the 'Collections' tab, there is a list of requests including 'Mongoose CRUD Request', 'Movie REST API Request', and 'Heroku app'. Under 'Heroku app', there are several requests: 'User login', 'User registration', 'User Update', 'Add movies to favorite', 'Get user by Name' (which is highlighted in grey), 'Remove movies from user's Favo...', and 'Allow existing users to deregister'. In the main workspace, a GET request is being viewed for the URL <https://rhubarb-crisp-92657.herokuapp.com/users/jacky3>. The response status is 200 OK, time is 651ms, and size is 531 B. The response body is displayed in JSON format:

```
1 "FavoriteMovies": [
2     "5e4063921733a4ac0b01c8e0",
3     "5e4063921733a4ac0b01c8e1",
4     "5e4063921733a4ac0b01c8e2",
5     "5e4063921733a4ac0b01c8e3"
6 ],
7 "_id": "5e8545ee7dc7bd001701b07d",
8 "Username": "jacky3",
9 "Password": "a1234",
10 "Email": "ex77@gmail.com",
11 "Birthday": "1977-11-19T00:00:00.000Z",
12 "__v": 0
13 }
```

Ten: Allow users to remove a movie from their list of favourites:

The screenshot shows the Postman application interface. The left sidebar is identical to the previous one, showing requests for 'Mongoose CRUD Request', 'Movie REST API Request', and 'Heroku app'. Under 'Heroku app', the 'Remove movies from user's Favo...' request is highlighted. In the main workspace, a DELETE request is being viewed for the URL <https://rhubarb-crisp-92657.herokuapp.com/users/jacky3/movies/5e4063921733a4ac0b01c8e0>. The response status is 200 OK, time is 662ms, and size is 503 B. The response body is displayed in JSON format:

```
1 "FavoriteMovies": [
2     "5e4063921733a4ac0b01c8e1",
3     "5e4063921733a4ac0b01c8e2",
4     "5e4063921733a4ac0b01c8e3"
5 ],
6 "_id": "5e8545ee7dc7bd001701b07d",
7 "Username": "jacky3",
8 "Password": "a1234",
9 "Email": "ex77@gmail.com",
10 "Birthday": "1977-11-19T00:00:00.000Z",
11 "__v": 0
12 }
```

Eleven: Allow existing users to deregister with authorization token:

The screenshot shows the Postman application interface. On the left, the sidebar lists collections: 'Movie REST API Request' (9 requests) and 'Heroku app' (13 requests). Under 'Heroku app', several methods are listed: POST User login, POST User registration, PUT User Update, POST Add movies to favorite, GET Get user by Name, DEL Remove movies from user's Fav..., and a selected 'DEL Allow existing users to deregister'. The main workspace shows a DELETE request to <https://rhubarb-crisp-92657.herokuapp.com/users/jacky3>. The 'Authorization' tab is selected, showing 'Bearer Token' with a dropdown menu. The 'Headers' tab shows 7 entries. The 'Body' tab contains the response: '1 jacky3 was deleted.' The status bar at the bottom indicates 'Status: 200 OK Time: 301ms Size: 270 B'. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'Import', 'Runner', 'My Workspace', 'Invite', 'Upgrade', and various icons.