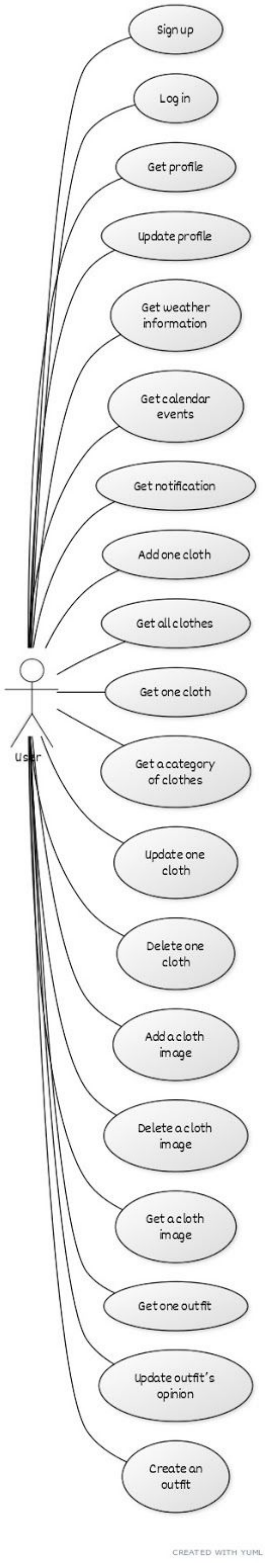
Requirements

1. All Use Cases

Use Case Name	**Description**
Sign up	Register a new user
Log in	User login
Get profile	Get user profile
Update profile	Update user profile information
Get weather information	Obtain today's and tomorrow's weather information **(External API: we used OpenWeather API in this use case)**
Get calendar events	Obtain events from Google Calendar **(External API: we used Google Calendar API in this use case)**
Get notification	User will get an welcome notification after logging in **(Push notification: we send user a notification in this use case)**
Add one cloth	User adds a new cloth into the closet
Get all clothes	User gets all clothes saved in the closet
Get one cloth	User gets a specific cloth
Get a category of clothes	User gets all clothes that are in a specific category
Update one cloth	User updates the information of a specific cloth
Delete one cloth	User delete a specific cloth
Add a cloth image	User adds an image for a specific cloth
Delete a cloth image	User delete the image of a specific cloth
Get a cloth image	User gets the image of a specific cloth

Get one outfit	User request the app for an outfit suggestion **(Non-trivial logic: we added complex logic when returning an outfit suggestion)**
Update outfit's opinion	Users can like or dislike the outfit we suggested. This opinion change will also affect the future outfit suggestions
Create an outfit	Users can create an outfit manually. (In the app, when the user dislikes all the outfits we suggested, we will let the user manually create an outfit by himself/herself)

2. Use Case Diagram



3. Non-functional Requirements

1. Users should not need more than 5 steps to a cloth item.
    - This requirement ensures the app is user friendly.
2. Users should wait at most 2 seconds when requesting for an outfit suggestion.
    - This requirement checks the app's performance

4. All necessary user accounts or similar details we need to test your app

You can create your own account, but for correctly testing the complex logic (recommend outfits to users based on their current clothes in the database), please use the account we have provided below.

UserAccount and password for testing:

UserAccount: closet@closet.com
UserPassword: 123123

Notes:

1. You need to have at least one cloth image in your emulator's gallery in .**jpg** format for testing the feature: add clothes into the closet.
2. You need to have a cloth image file in your emulator's Download folder named **test.jpg** for testing one of the UI tests: addClothesTest.

Design

5. A list of all backend components, with 1-2 sentences describing each component.

Components Name	Description
User Module	
User Router	A summary of user-related endpoints. It also validates the input data in request parameters and request body
User Controller	It contains all the handlers for user-related requests
User Model	Similar to User DB
Clothes Module	
Clothes Router	A summary of clothes-related endpoints. It also validates the input data in request parameters and request body
Clothes Controller	It contains all the handlers for clothes-related requests
Clothes Model	Similar to Clothes DB
Image Module	
Image Router	A summary of endpoints for clothes images. It also validates the input data in request parameters and request body
Image Controller	It contains all the handlers for requests of clothes images
Outfit Module	
Outfit Router	A summary of outfit-related endpoints. It also validates the input data in request parameters and request body
Outfit Controller	It contains all the handlers for outfit-related requests
Outfit Service	It services as a middle layer between outfit controller and outfit model to handle complex logic
Outfit Model	It connects to database and contains all the generated outfits

Today Outfit Model	It connects to the database and contains all the outfits returned on a specific day. The purpose of the model is to keep track the returned outfits so that we will not return duplicated outfits to the user on the same day
Weather Module	
Weather Router	A summary of weather-related endpoints. It also validates the input data in request parameters and request body
Weather Controller	It contains all the handlers for weather-related requests
Weather Service	It wraps up the OpenWeather API and obtain the weather data
Calendar Module	
Calendar Router	A summary of calendar-related endpoints. It also validates the input data in request parameters and request body
Calendar Controller	It contains all the handlers for calendar-related requests
Calendar Service	It wraps up Google Authentication API and Google Calendar API and obtains the calendar events data.
Notification Module	
Notification Router	A summary of notification-related endpoints. It also validates the input data in request parameters and request body
Notification Controller	It contains all the handlers for notification-related requests

Testing

6. Automated front-end UI testing

1. **RegisterWithExistingAccountTest**
   Location: frontend/app/java/com.example.frontend(android test)/RegisterTest
   Purpose: This UI test checks that the user cannot register with an already existing account by clicking the REGISTER button on the register screen.The expected results are verified by checking if a toast message “User exists already, please login instead” is displayed.

2. **LoginTest**
   Location: frontend/app/java/com.example.frontend(android test)/LoginTest

   2.1. **loginWithInvalidAccountTest**
   Purpose: This UI test checks that the user cannot login with an invalid account by clicking the LOGIN button on the login screen. The expected results are verified by checking if a toast message “Invalid credentials, could not log you in” is displayed.

   2.2. **loginSuccessTest**
   Purpose: This UI test checks the sequence of a success login use case. The expected results are verified by checking that a toast message “Login successfully” is displayed and a view on the main activity is displayed on the screen.

3. **OutfitTest**
   Location: frontend/app/java/com.example.frontend(android test)/GetOutfitTest
   Purpose: This UI test checks the sequence of a success get-an-recomended-outfit use case:getting a recommended outfit and being able to specify user’s like/dislike opinion about the outfit. The expected results are verified by checking that after clicking the button “GIVE ME SOME OUTFIT IDEA”, the outfit’s like and dislike buttons are enabled, and after clicking the dislike button, an undo button is displayed on the screen.

4. **AddClothesTest**
   Location: frontend/app/java/com.example.frontend(android test)/AddClothesTest
   Purpose: This UI test checks the sequence of the use case adding a cloth to the closet. The expected results are verified by checking that after clicking the add clothes button,

we should be navigated to add clothes activity and after we set the cloth information and click the save button, we should be navigated back to the clothes navigation with a new clothes item displayed.

To run backend tests locally

- Setup and commands: https://github.com/CPEN321Closet/closet/blob/master/backend/README.md
- Command details: https://github.com/CPEN321Closet/closet/blob/master/backend/package.json

7. Backend Unit Tests

Our unit tests consist of not only unit tests for two main components, but also tests for other smaller components and utility functions.

1. List of the two components that you tested
    a. User component
    b. Outfit component
2. The location of the tests for each of the components in your Git repository
    a. User component
        - https://github.com/CPEN321Closet/closet/blob/master/backend/__tests__/controller/users-controller.test.js
        - https://github.com/CPEN321Closet/closet/blob/master/backend/__tests__/controller/users.test.js
    b. Outfit component
        - https://github.com/CPEN321Closet/closet/blob/master/backend/__tests__/controller/outfits-controller.test.js
        - https://github.com/CPEN321Closet/closet/blob/master/backend/__tests__/service/outfits-service.test.js
        - https://github.com/CPEN321Closet/closet/blob/master/backend/__tests__/controller/outfits.test.js
3. For each component, run the tests and provide the coverage report
    a. User component

All files controller

32.59% Statements 131/402 **9.4%** Branches 14/149 **16%** Functions 4/25 **32.91%** Lines 131/398

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
calendar-controller.js	19.35%	6/31	0%	0/11	0%	0/1	19.35%	6/31
clothes-controllers.js	11.76%	12/102	0%	0/76	0%	0/9	12%	12/100
image-controller.js	17.31%	9/52	0%	0/18	0%	0/2	17.31%	9/52
notifications-controller.js	41.18%	7/17	0%	0/4	0%	0/1	41.18%	7/17
outfits-controller.js	14.68%	16/109	0%	0/24	0%	0/7	14.95%	16/107
users-controllers.js	100%	75/75	100%	14/14	100%	4/4	100%	75/75
weather-controller.js	37.5%	6/16	0%	0/2	0%	0/1	37.5%	6/16

- The only file that user component has is user-controller.js
- There are two test files for user components: users-controller.test.js and users.test.js.
- For the first test file, we mocked all other components, and only used user-controller.js; however, testing only the controller is insufficient to understand whether the controller is actually functioning correctly - it is difficult to obtain outputs in some case and it does not give us an idea of how controller is connected to the router (the router files are merely declaration of how controllers are layed out).
- We also added the second test file to ensure better coverage. Some lines in other controllers are hit, but those lines are merely static declarations and do not contain actual useful logic - artifics of running the tests.
- We also mentioned this in our video presentation.

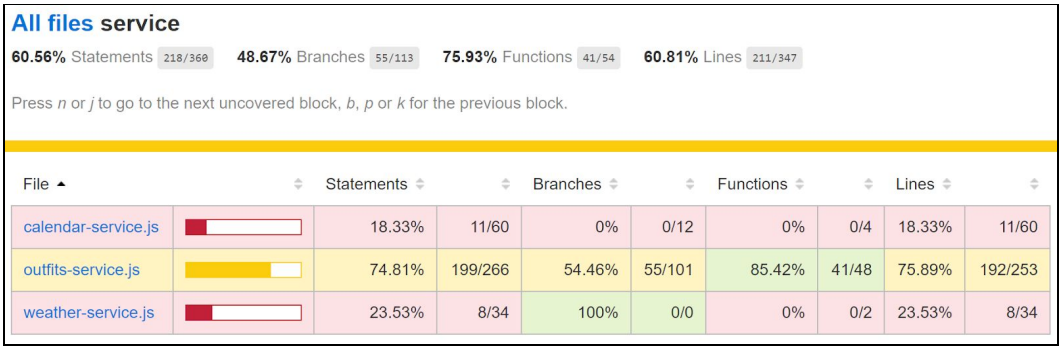
b. Outfit component

All files controller

44.28% Statements 178/402 **18.12%** Branches 27/149 **36%** Functions 9/25 **44.22%** Lines 176/398

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
calendar-controller.js	19.35%	6/31	0%	0/11	0%	0/1	19.35%	6/31
clothes-controllers.js	11.76%	12/102	0%	0/76	0%	0/9	12%	12/100
image-controller.js	17.31%	9/52	0%	0/18	0%	0/2	17.31%	9/52
notifications-controller.js	41.18%	7/17	0%	0/4	0%	0/1	41.18%	7/17
outfits-controller.js	93.58%	102/109	87.5%	21/24	100%	7/7	93.46%	100/107
users-controllers.js	48%	36/75	42.86%	6/14	50%	2/4	48%	36/75
weather-controller.js	37.5%	6/16	0%	0/2	0%	0/1	37.5%	6/16



All files service

60.56% Statements 218/360 **48.67%** Branches 55/113 **75.93%** Functions 41/54 **60.81%** Lines 211/347

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
calendar-service.js	18.33%	11/60	0%	0/12	0%	0/4	18.33%	11/60
outfits-service.js	74.81%	199/266	54.46%	55/101	85.42%	41/48	75.89%	192/253
weather-service.js	23.53%	8/34	100%	0/0	0%	0/2	23.53%	8/34

- There are three test files that are used for testing outfit component: outfits-controller.test.js, outfits-service.test.js, and outfits.test.js
- Similar reasons for testing user components also apply here.
- The first two test files includes full mocking and partial mocking, because we hope to test this component in all circumstances - outfit component has a lot of code that it is very hard to grasp all segments completely
- This is one reason we are missing some coverages
- As well, there are usages of "random" generation in outfit components (we have a simple wrapper for random generator called randomInt() inside), so some code segments are sometimes hit, sometimes miss.
- We also mentioned these issues in our video presentation.

4. Test execution status

Running all of our tests, including unit tests and integration test for two main components

```
Test Suites: 20 passed, 20 total
Tests:       144 passed, 144 total
Snapshots:   0 total
Time:        54.497 s, estimated 58 s
Ran all test suites.
```

a. User component

```
closet@closet:~/closet/backend$ npm run test:users

> closet-backend@0.9.0 test:users /home/closet/closet/backend
> cross-env NODE_ENV=test jest --verbose --runInBand --detectOpenHandles --coverage
"users-controller.test.js" "users.test.js"

(node:20916) DeprecationWarning: Mongoose: `findOneAndUpdate()` and `findOneAndDelet
e()` without the `useFindAndModify` option set to false are deprecated. See: https:/
/mongoosejs.com/docs/deprecations.html#findandmodify
(Use `node --trace-deprecation ...` to show where the warning was created)
 PASS  __tests__/controller/users.test.js (7.721 s)
  Authentication Tests
    ✓ 422 should fail to sign up if inputs are invalid (109 ms)
    ✓ 201 should succeed to sign up if every input is correct (441 ms)
    ✓ 422 should fail to sign up if the user has existed (399 ms)
    ✓ 200 should succeed to login for an existing user (381 ms)
    ✓ 401 should fail to login if the user does not exist (13 ms)
    ✓ 401 missing login info (9 ms)
    ✓ 401 should fail to login if the password is invalid (361 ms)
    ✓ 200 should get user profile (18 ms)
    ✓ 200 should update user profile if inputs are all correct (756 ms)
    ✓ 422 should fail to update user profile if inputs are not valid (20 ms)
    ✓ 500 should fail to update user profile if city is not found (525 ms)
    ✓ 422 should fail to update user profile if the email has been registered alread
y (1392 ms)

 PASS  __tests__/controller/users-controller.test.js
  User controller signup with mocking
    ✓ signup expect bcrypt exception (8 ms)
    ✓ signup expect database exception (2 ms)
    ✓ signup expect jwt exception (1 ms)
  User controller login with mocking
    ✓ login expect database exception (1 ms)
    ✓ login expect database return null (1 ms)
    ✓ login expect bcrypt exception (1 ms)
    ✓ login expect jwt exception (1 ms)
  User controller user profile with mocking
    ✓ getUserProfile expect database exception (1 ms)
    ✓ updateUserProfile expect getGeoCode exception (1 ms)
    ✓ updateUserProfile expect getGeoCode failed (1 ms)
    ✓ updateUserProfile expect database exception (1 ms)
```

```
Test Suites: 2 passed, 2 total
Tests:       23 passed, 23 total
Snapshots:   0 total
Time:        11.235 s
Ran all test suites matching /users-controller.test.js|users.test.js/i.
```

b. Outfit component

```
closet@closet:~/closet/backend$ npm run test:outfits

> closet-backend@0.9.0 test:outfits /home/closet/closet/backend
> cross-env NODE_ENV=test jest --verbose --runInBand --detectOpenHandles --coverage
"outfits-controller.test.js" "outfits-service.test.js" "outfits.test.js"

 PASS  __tests__/controller/outfits.test.js
  Outfit controller tests
    ✓ get token (936 ms)
    ✓ 500 internal generateOutfit failed (28 ms)
    ✓ 400 one outfit generateOutfit warning (10 ms)
    ✓ 200 one outfit generateOutfit success (17 ms)
    ✓ 200 multiple outfit generateOutfit success true (14 ms)
    ✓ 200 multiple outfit generateOutfit success false (9 ms)
    ✓ 422 update user opinion missing parameter (16 ms)
    ✓ 500 update user opinion database exception (10 ms)
    ✓ 200 update user opinion (19 ms)
    ✓ 200 delete all outfits (15 ms)

(node:22296) UnhandledPromiseRejectionWarning: TypeError: Cannot read property 'stat
us' of null
(Use `node --trace-warnings ...` to show where the warning was created)
(node:22296) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This err
or originated either by throwing inside of an async function without a catch block,
or by rejecting a promise which was not handled with .catch(). To terminate the node
 process on unhandled promise rejection, use the CLI flag `--unhandled-rejections=st
rict` (see https://nodejs.org/api/cli.html#cli_unhandled_rejections_mode). (rejectio
n id: 2)
(node:22296) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecat
ed. In the future, promise rejections that are not handled will terminate the Node.j
s process with a non-zero exit code.
 PASS  __tests__/controller/outfits-controller.test.js
  Outfits controller with database mocking
    ✓ createOneOutfit expect database exception: Outfit found (5 ms)
    ✓ createOneOutfit expect database exception: Outfit save (2 ms)
    ✓ createOneOutfit expect database exception: Outfit findOneAndUpdate (2 ms)
    ✓ createOneOutfit expect database exception: Clothes findById (2 ms)

 PASS  __tests__/service/outfits-service.test.js
  Outfit service tests
    ✓ failed to initialize (21 ms)
    ✓ failed to initialize: database exceptions (4 ms)
    ✓ failed generate outfit: need more clothes (8 ms)
    ✓ generate normal outfit (5 ms)
    ✓ generate formal outfit (8 ms)
```

```
Test Suites: 3 passed, 3 total
Tests:       19 passed, 19 total
Snapshots:   0 total
Time:        7.47 s
Ran all test suites matching /outfits-controller.test.js|outfits-service.test.js|out
fits.test.js/i.
```

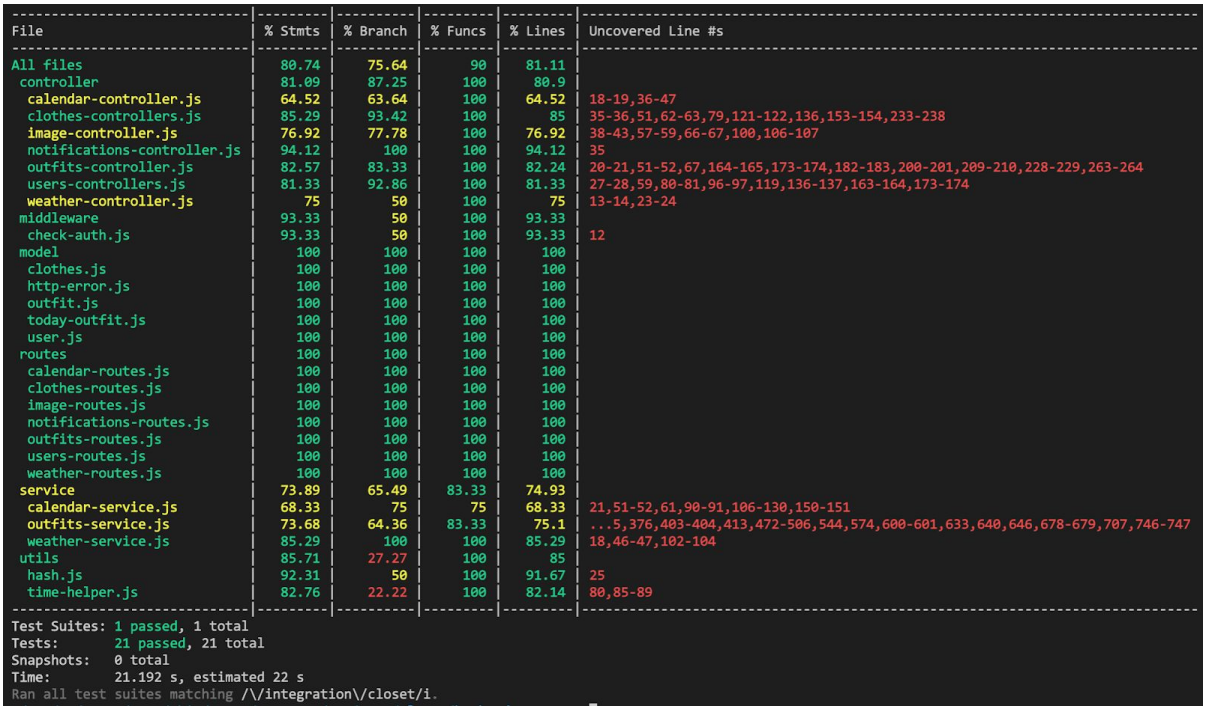
8. For backend integration test

1. A list of backend APIs that you tested (all APIs that the backend exposes to the frontend).

All Categories (**6** in total)	Endpoints (**21** in total)
User Authentication	POST /api/users/signup POST /api/users/login
User Profile	GET /api/users/me PUT /api/users/me
Weather & Calendar & Notification	GET /api/weather/ POST /api/calendar/:date POST /api/notifications/
Clothes	POST /api/clothes/:userId GET /api/clothes/:userId GET /api/clothes/:userId/:clothesId GET /api/clothes/:userId?category=category PUT /api/clothes/:userId/:clothesId DELETE /api/clothes/:userId/:clothesId
Clothes Image	POST /api/images/:userId/:clothesId GET /UserClothingImages/:userId/imageName.imageExt DELETE /api/images/:userId/:clothesId
Outfit	GET /api/outfits/one GET /api/outfits/multiple PUT /api/outfits/:outfitId POST /api/outfits/one DELETE /api/outfits/all

2. The location of the integration tests in your Git repository.
    a. You can find the tests in the following url: https://github.com/CPEN321Closet/closet/blob/master/backend/__tests__/integration/closet.test.js
3. Run the integration tests and provide the coverage report. We expect to see a very high (close to 100%) coverage, as all/most of the backend code should be triggered from the frontend. If the coverage is not as expected, specify the reasons for such behavior.

a. Here is an image of the jest coverage summary for the integration tests



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	80.74	75.64	90	81.11	
controller	81.09	87.25	100	80.9	
calendar-controller.js	64.52	63.64	100	64.52	18-19,36-47
clothes-controllers.js	85.29	93.42	100	85	35-36,51,62-63,79,121-122,136,153-154,233-238
image-controller.js	76.92	77.78	100	76.92	38-43,57-59,66-67,100,106-107
notifications-controller.js	94.12	100	100	94.12	35
outfits-controller.js	82.57	83.33	100	82.24	20-21,51-52,67,164-165,173-174,182-183,200-201,209-210,228-229,263-264
users-controllers.js	81.33	92.86	100	81.33	27-28,59,80-81,96-97,119,136-137,163-164,173-174
weather-controller.js	75	50	100	75	13-14,23-24
middleware	93.33	50	100	93.33	
check-auth.js	93.33	50	100	93.33	12
model	100	100	100	100	
clothes.js	100	100	100	100	
http-error.js	100	100	100	100	
outfit.js	100	100	100	100	
today-outfit.js	100	100	100	100	
user.js	100	100	100	100	
routes	100	100	100	100	
calendar-routes.js	100	100	100	100	
clothes-routes.js	100	100	100	100	
image-routes.js	100	100	100	100	
notifications-routes.js	100	100	100	100	
outfits-routes.js	100	100	100	100	
users-routes.js	100	100	100	100	
weather-routes.js	100	100	100	100	
service	73.89	65.49	83.33	74.93	
calendar-service.js	68.33	75	75	68.33	21,51-52,61,90-91,106-130,150-151
outfits-service.js	73.68	64.36	83.33	75.1	...5,376,403-404,413,472-506,544,574,600-601,633,640,646,678-679,707,746-747
weather-service.js	85.29	100	100	85.29	18,46-47,102-104
utils	85.71	27.27	100	85	
hash.js	92.31	50	100	91.67	25
time-helper.js	82.76	22.22	100	82.14	80,85-89

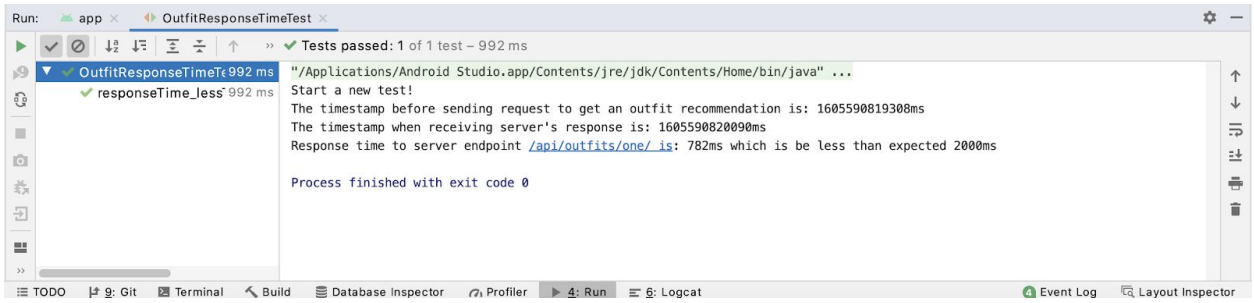
Test Suites: 1 passed, 1 total
Tests: 21 passed, 21 total
Snapshots: 0 total
Time: 21.192 s, estimated 22 s
Ran all test suites matching /\/integration\/closet/i.

b. Reasons for files that do not achieve 100% coverage
   i. For "**calendar-controller.js**", "**calendar-service.js**", "**weather-controller**", we do not achieve 100% coverage since we have many error handling code for external APIs. We use Google Authentication, Google Calendar API, and OpenWeather API in the calendar and weather modules. It is very hard to test the cases that those external APIs are failing.
   ii. For "**image-controller.js**", we do not have 100% coverage since we have many error handling code for the file system (fs in Node Js). This is similar to the external API, and it is also very hard to test when the file system is failing.
   iii. For "**outfits-service.js**", it is the file that contains the most complicated code and logic in this project since we put all "complex-logic" code in this file. **We have many error handling code for database operation (save, find) and corner case check**. Thus, it is hard to test all of them.
   iv. For "**time-helper.js**", the branch percentage is quite low because of the "getDaysInMonth" function (you can find it in this url https://github.com/CPEN321Closet/closet/blob/d02f9295986d9c25854984da067b7553ebf9b467/backend/utils/time-helper.js#L76). We want to return the number of days in a specific month, and "february" is a special case. To handle it, we added checks for a leap year, but those checks are very hard to be included in the integration tests.

4. Test execution status. We expect all tests to pass at this point.
    a. Here is an image of the test execution status

9. Testing Non-functional Requirements

1. **OutfitResponseTimeTest**
   Location: frontend/app/src/test/java/com/example/frontend/OutfitResponseTimeTest
   Requirement: Users should wait less than 2 seconds when requesting for an outfit suggestion.
   Status: Passed



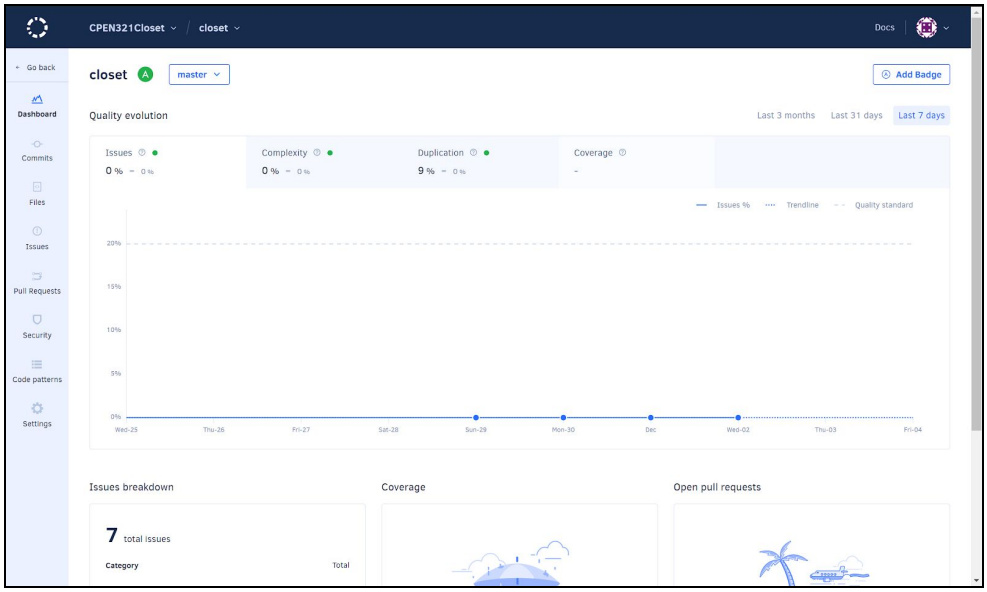
2. **AddClothesStepTest**
   Location: frontend/app/src/androidTest/java/com/example/frontend/AddClothesStepTest
   Requirement: Users do not need more than 5 steps to add a cloth item.
   Status: Passed

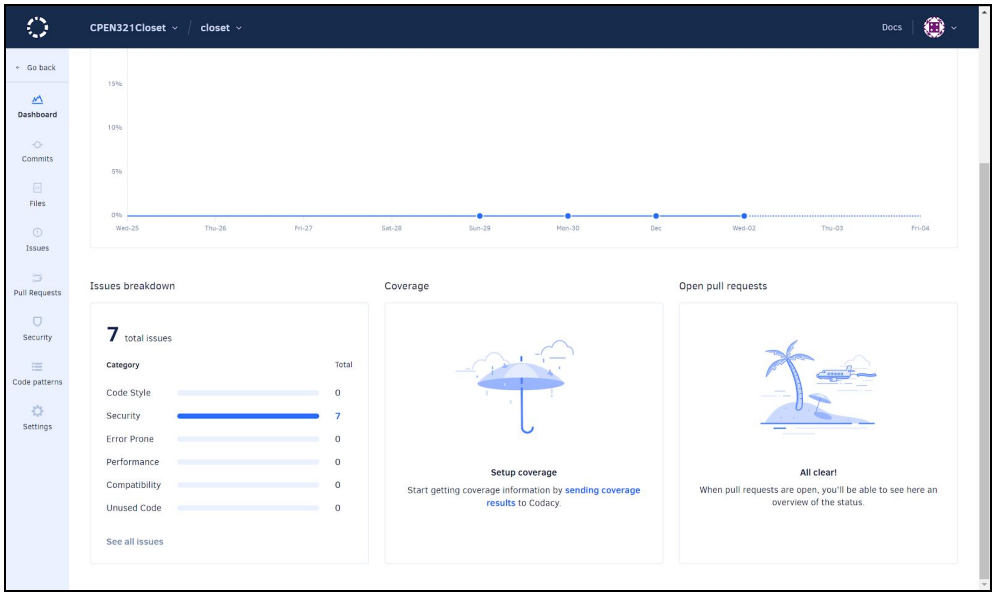
Code Review

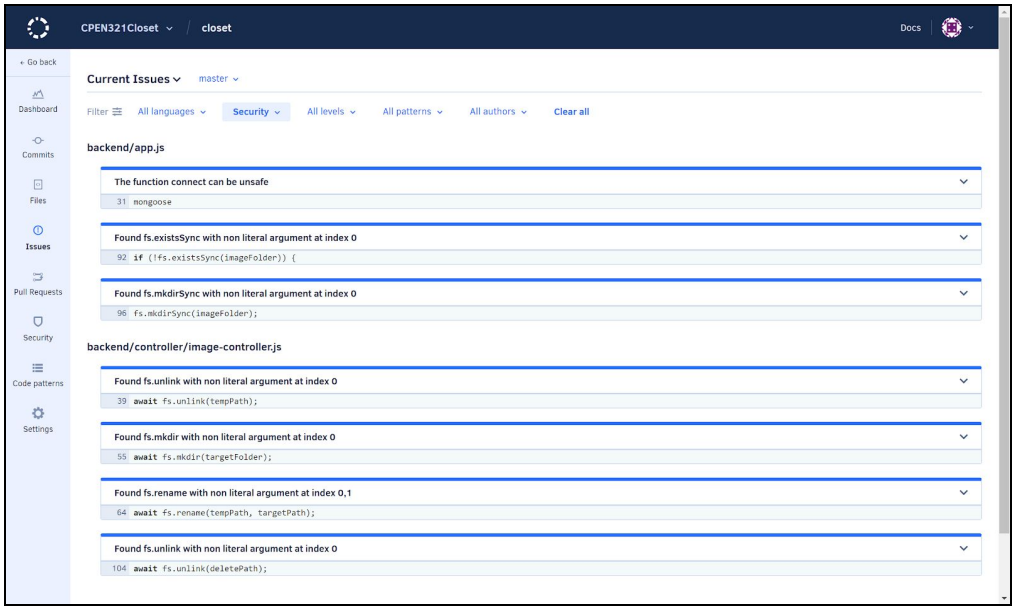
10. Codacy Report

Codacy dashboard page: https://app.codacy.com/gh/CPEN321Closet/closet/dashboard

Coday is setup to run on the latest commit on master branch, at the time of writing this report, the latest Codacy is ran on commit "**c37d292**" , and can be found at https://app.codacy.com/gh/CPEN321Closet/closet/commit?bid=20909386&cid=514742478







Here we have the same issues in M8, which we provided explanations for.

Lucky, we changed some implementations, so now we have 7 issues instead of 10, even though essentially they are still the same issues. For details, please see previous M8.

Another note is that, although any of these issues could be fixed by perhaps upgrading our NPM packages, we did not do that since we do not know what upgrades could do to our current backend implementation. Considering these issues are not problems with our code or style, we decided it was best to maintain the NPM packages version.

- This issues is related to calling “connect” on mongoose
- Issues reported by **Scanjs rules: Call_connect** ESLint_scanjs-rules_call__connect
    - The rule implemented (https://github.com/mozfreddyb/eslint-plugin-scanjs-rules/blob/master/lib/rules/call_connect.js) does overall enforces not to call “connect” on any object
- We did follow official documentation provided by Mongoose to call connect to the database (https://mongoosejs.com/docs/connections.html)
- To mitigate this issue, we have already implemented Promise catching on error is mongoose.connect throws error
- If such error is thrown, we would also be able to see it during automatic deployment log or manually log on our Azure VM

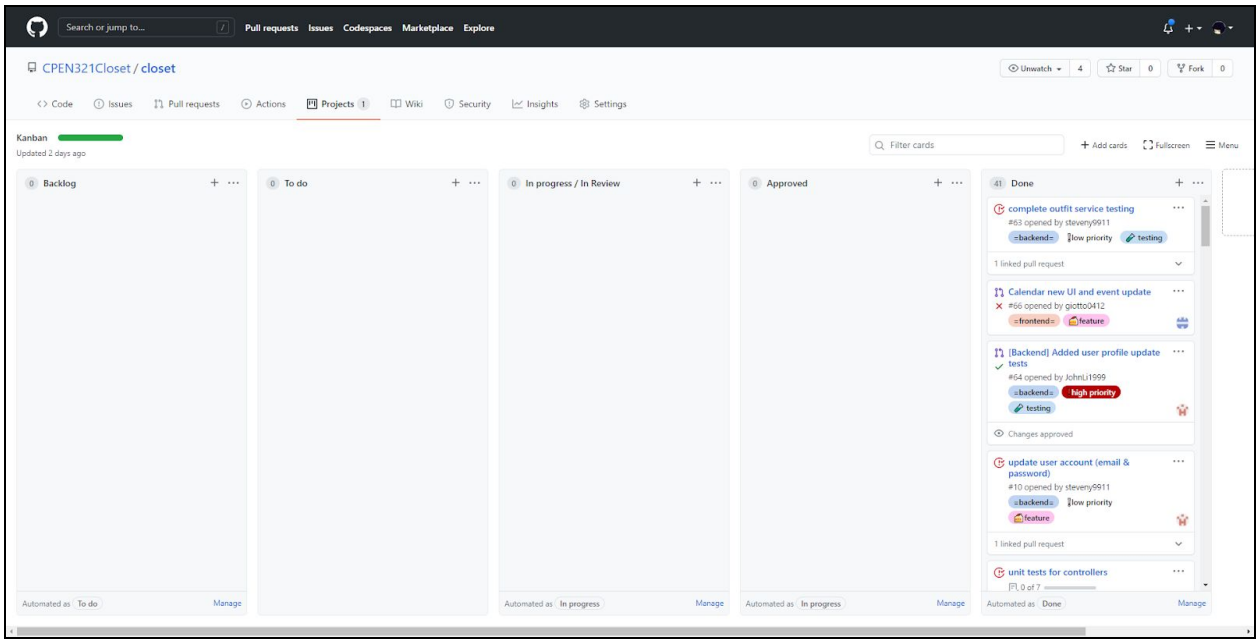
- These issues are related to how we are using a variable to call fs functions
- This seems to be a common issue that it has an open Github issue
    - https://stackoverflow.com/questions/63262683/how-to-fix-found-fs-readfile-with-non-literal-argument-at-index-0
    - https://github.com/nodesecurity/eslint-plugin-security/issues/65
- Currently there are no official response other than disabling ESLint on this pattern or using absolute string path to call fs function
    - We could temporarily disable the pattern --- since this is not a major issue and we have unit test around fs functions (in image-controller) for checking --- until an official response from the Github open issue
    - Using absolute path defeats the purpose of using variable and relatively path
    - Since this is related to our logic of finding user images, we need relatively path to manage files and folders, so we can not simply switch to another method
- We implemented few mitigations and our own error checking to ensure better safety
- We implemented try/catch to ensure our app.js does not completely crash, but instead send an error (although not needed, a good extra layer or safety)
- Wrapping file checking inside if-else ensure we can also see the result
- We applied our mitigation to similar issues of the same types

Humblebrag

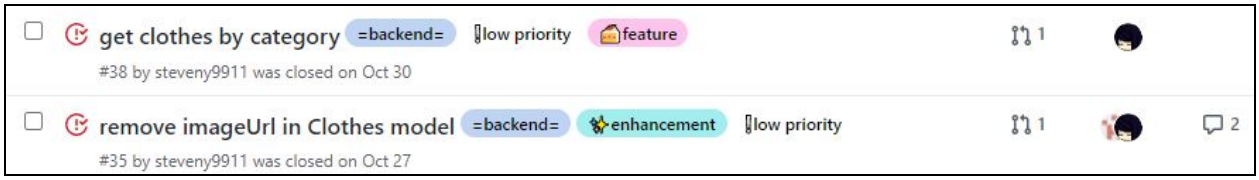
11. A description of any “extra” part of your project you are particularly proud of

1. **Backend Structure**
   We are proud of our backend structure. We use “route -> controller -> service -> model” layers to handle requests and interact with the database. This structure is simple and easy to understand (**KISS**). All the modules (User, Clothes, Outfit, etc.) can focus on their specific features and do not depend on other modules (**Single Responsibility Principle** & **Independence**). We put shared logic and complicated logic into the service layer, and different controllers can call the same service functions (**Separation of Concerns** & **Don’t Repeat Yourself**). By separating different requests into different routes which will be handled by different controllers, we make sure each module does not know the internal details of other modules (**Principle of Least Knowledge**). Therefore, our backend structure follows all the core architectural principles taught in class!

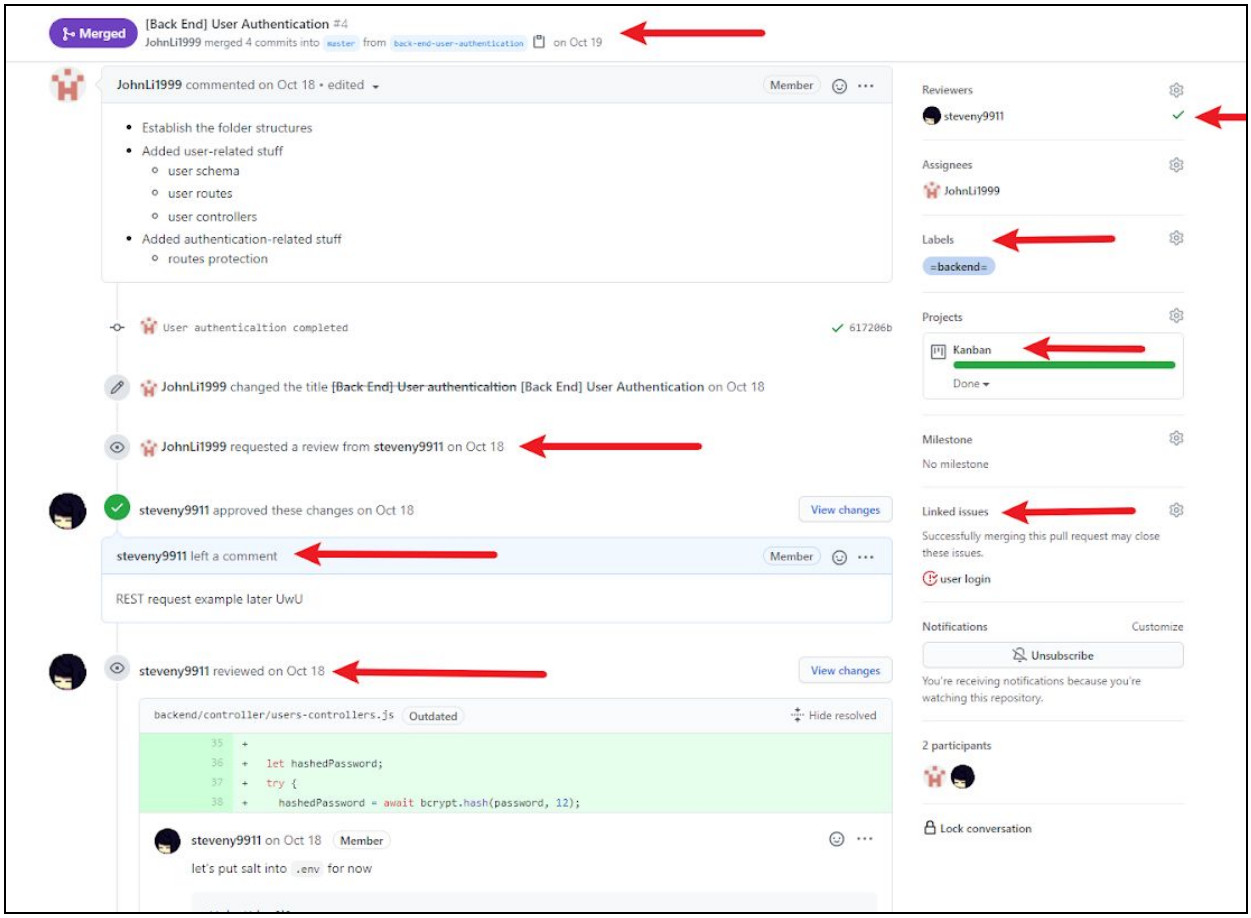
2. **Github Setup**
   We also enjoyed using some pretty Github features such as Project Kanban boards which helped us to track progress, especially at the early stage. These are also automated with pull requests and issues, so that the process is much easier to manage.



   And using the Github issues to track and manage our progress with labels, code reviews, feedback on pull requests for new features or bug fixes.



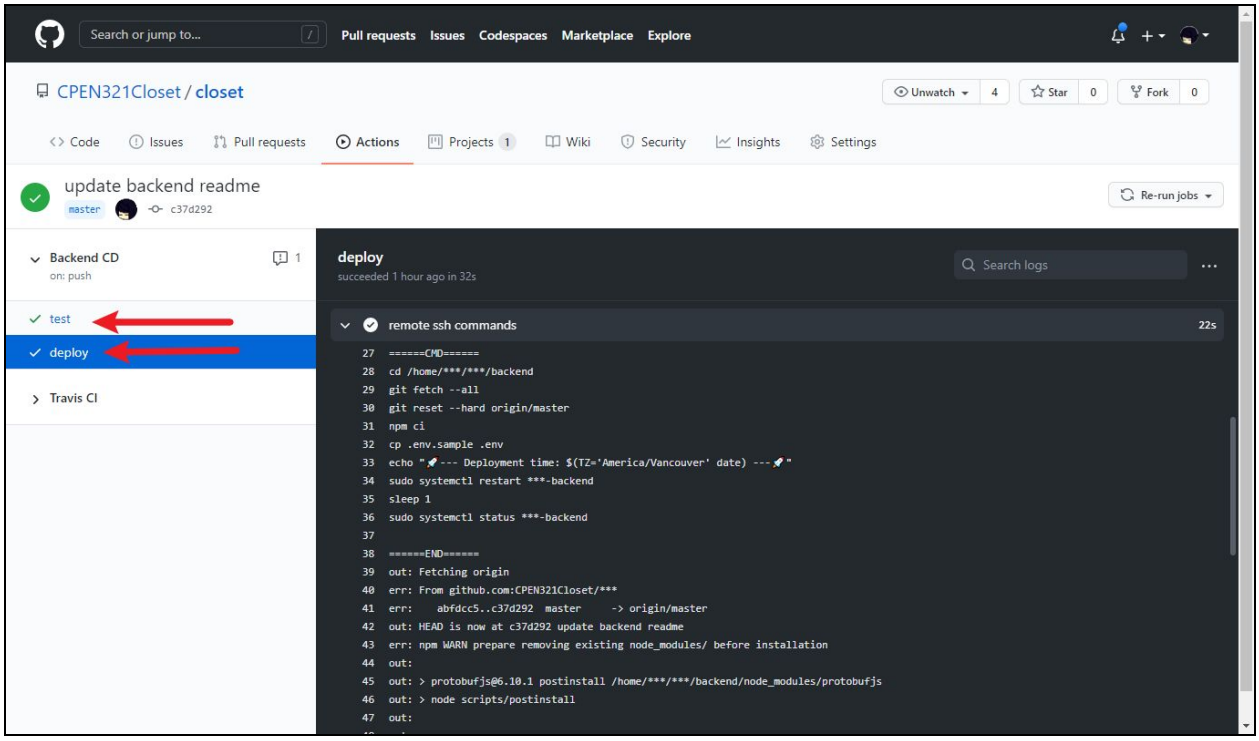
All of these come together in a pull request nicely.



3. **Automated Continuous Deployment**

   We also implemented automated continuous deployment, so that all merged pull requests on master are deployed automatically onto our Azure cloud VM without any manual work.

   We first ensure we run all tests, and if they pass, we deploy whatever is on the latest on the master branch to the VM: pull the code, restart the service, log a timestamp. Although this is not perfect, since master is always deployed, but for the scope of our project, automating deployment saves a lot of time checking which version is up, is the service running, etc. We could still manually log-in to VM to check if everything is alright by reading previous logs.



4. **Backend Service Setup and Logging**

   We made our backend into a service running in the background with persistent logging. This made sure that if we ever restart our VM, the service is up and running automatically. And, this provided ease with logging, and enabled us to better deploy our backend.

   For logging, we formatted information so that we can track request and response easier: using arrows to indicate request or response, using request ID for tracking, using request body for information and bug fixes.

```
58  out: added 924 packages in 12.242s
59  out: 🚀 --- Deployment time: Wed Dec  2 20:20:08 PST 2020 --- 🚀
60  out: ● ***-backend.service - Closet Backend
61  out:    Loaded: loaded (/lib/systemd/system/***-backend.service; enabled; vendor preset: enabled)
62  out:    Active: active (running) since Thu 2020-12-03 04:20:13 UTC; 1s ago
63  out:  Main PID: 22124 (node)
64  out:     Tasks: 7 (limit: 2265)
65  out:    CGroup: /system.slice/***-backend.service
66  out:            └─22124 /usr/bin/node /home/***/***/backend/index.js
67  out:
68  out: Dec 03 04:20:13 *** node[22124]: :+:    :+: :+:        :+:    :+: :+:    :+: :+:            :+:
69  out: Dec 03 04:20:13 *** node[22124]: +:+        +:+        +:+    +:+ +:+        +:+            +:+
70  out: Dec 03 04:20:13 *** node[22124]: +#+        +#+        +#+    +:+ +#++:++#++ +#++:++#       +#+
71  out: Dec 03 04:20:13 *** node[22124]: +#+        +#+        +#+    +#+         +#+ +#+            +#+
72  out: Dec 03 04:20:13 *** node[22124]: #+#    #+# #+#        #+#    #+# #+#    #+# #+#            #+#
73  out: Dec 03 04:20:13 *** node[22124]:  ########  ##########  ########   ########  ##########     ###
74  out: Dec 03 04:20:13 *** node[22124]: INFO [2020-12-02, 8:20:13 p.m.] 🚀server startup time: 12/2/2020, 8:20:13 PM
75  out: Dec 03 04:20:13 *** node[22124]: INFO [2020-12-02, 8:20:13 p.m.] 🌲environment:production version:0.9.0
76  out: Dec 03 04:20:13 *** node[22124]: INFO [2020-12-02, 8:20:13 p.m.] 🔠PORT: 80
77  out: Dec 03 04:20:13 *** node[22124]: INFO [2020-12-02, 8:20:13 p.m.] 🔠MONGODB_URI: mongodb://localhost:27017/prod
78  ================================================
79  ✅ Successfully executed commands to all host.
80  ================================================
```

Run as service `closet-backend`

- Service is set to restart on fail or on reboot
- Partial environemnt variable is set already in the service file

```
sudo systemctl status closet-backend    # check status (partial log)
sudo systemctl start closet-backend     # start service
sudo systemctl stop closet-backend      # stop service
sudo systemctl restart closet-backend   # restart service
sudo systemctl enable                   # enable to run on boot
```

Service log

- Should also put into another location so we can have a clean log for each start of backend

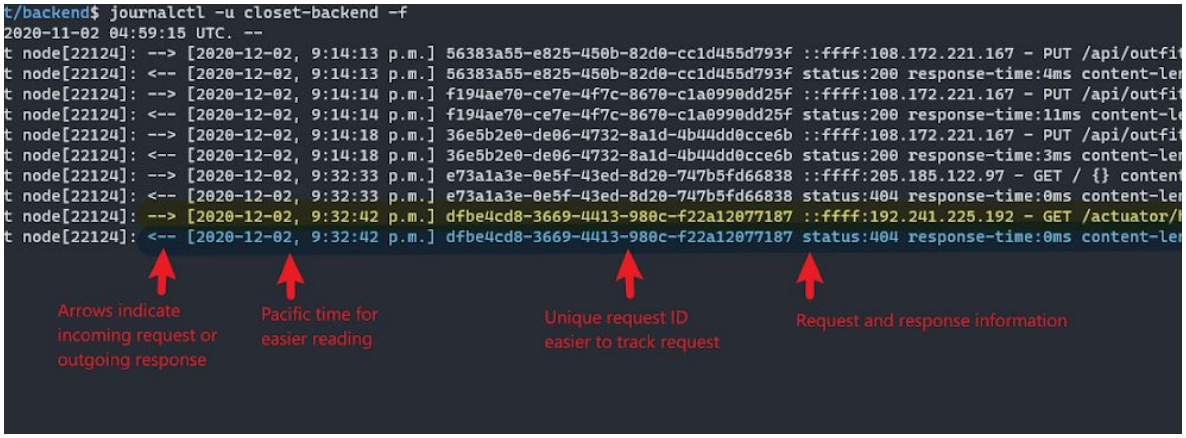
```
journalctl -u closet-backend          # all logs (use SHIFT-G to go to the bottom)
journalctl -u closet-backend -f       # follow low
```

Modify service

- A copy of service file is in this repository `closet/backend/closet-backend.service`

```
sudo vim /lib/systemd/system/closet-backend.service   # edit service file
sudo systemctl daemon-reload                          # reload service file
sudo systemctl start closet-backend                   # start service

sudo chmod +x /home/closet/closet/backend/index.js    # add exceutable permissions to express app
sudo chmod go+w /home/closet/closet/backend           # allows any users to write the app folder (for using fs
```



5. Pretty badges ❤️