SWPP2022 Final Report

Team 18

oOo: Recommend Your Outfit

JaeYoung Jung

JunKyoung Kim

JunYoung Park

Pyojin Park

Contents

1. Introduction

1.1. Abstract

Recommend Your Outfit(oOo) is a service that analyzes what clothes users have and recommend possible outfits & fashion they can create. Unlike other online fashion services we focus not on clothes itself but on ourselves who actually wear the cloth, and try to help them get to know about current fashion trends and styles more easily.

Our service tries to help people follow fashion trends more easily and get to know about what would fit best. We provide services based on users' style and what clothes they have, and recommend the best clothes that match their style or they want to wear. We see this as a great chance for people who are new to fashion, especially men in their early 20-30s.

To implement this into our actual service, we have several key features our service provides. We provide a closet feature where users can take photos and upload their clothes to our service. Based on the clothes, we recommend the best fit according to outfits from online shopping malls and snaps. We're also going to provide purchase links to actually buy that clothes in the user's recommendation.

1.2. Target Customers

Our service's target customers are Korean men in their 20-30s, especially beginners to fashion, and having trouble following rapidly changing fashion trends.

Men's grooming has become a huge global trend and getting more and more demand. Resources that help men's grooming are still just in the beginning. Youtube stylists for men are getting more and more popular among young men in their early 20s, which shows their growing interests in their fashion styles - please find below for the specific market landscape and growing trend of market size of the online fashion industry.

1.3. Competitive Landscape

The fashion industry - especially online fashion markets - is growing rapidly. About 50 trillion won has been transacted in online shopping for fashion in 2021, showing 9.2% growth from 2020 according to Statistics Korea. This also makes up 25.8% of the total online shopping transaction amount in 2021.

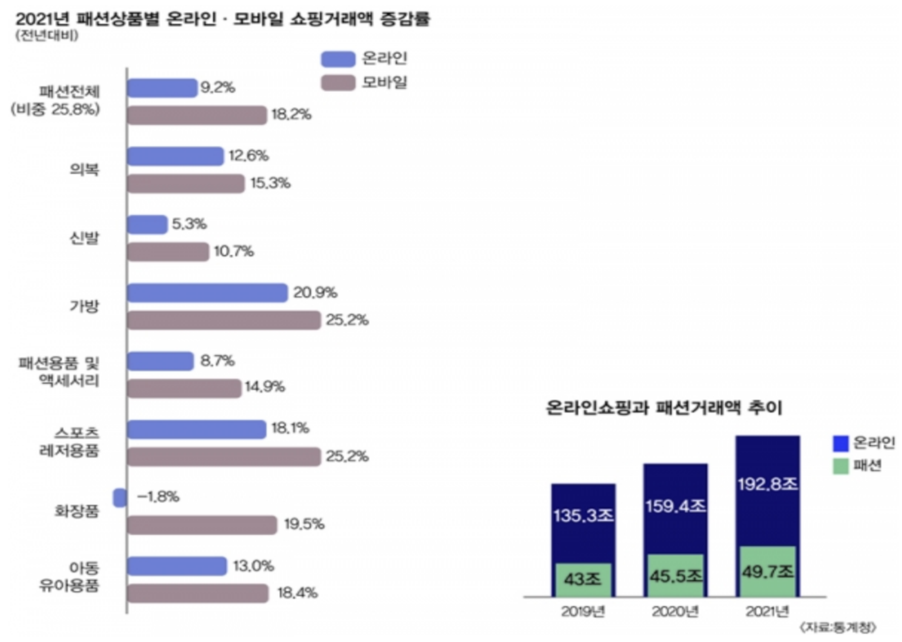
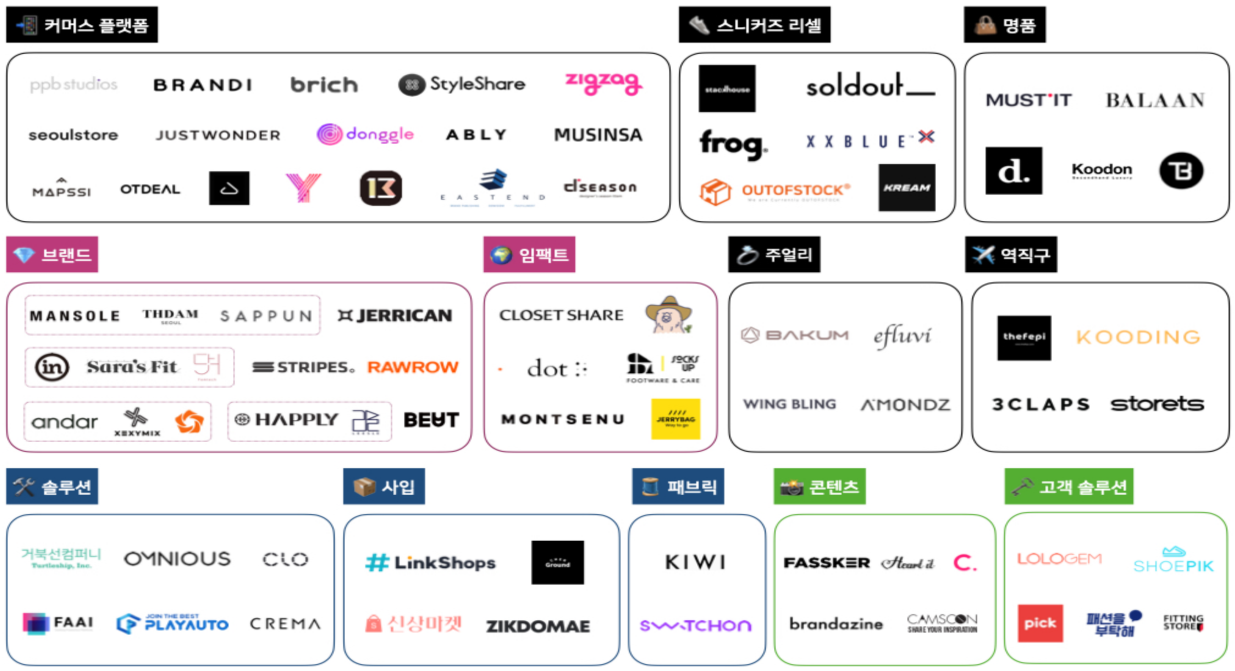


Figure 1. Increase or Decrease in shopping transaction amount

Below(Figure 2) is the landscape of the fashion-tech industry in Korea, where we can see most of the services are B2C, and online commerce platforms contain the most well-known services such as ‘Musinsa’, ‘Zigzag’, and ‘Ably’. The market size of these online commerce platforms reached 22.8 trillion won in 2021 which is over 30% of the market size of the fashion industry that contains both online and offline, according to the research NH Investments (Figure 2).



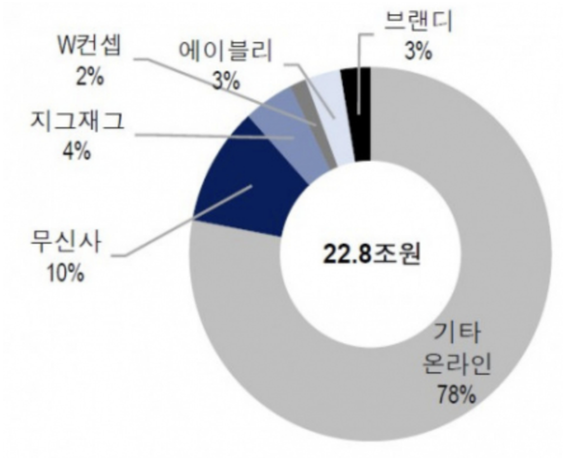


Figure 2.Landscape of the fashion-tech industry

We analyzed that one of the key success factors of these fashion commerce platforms is that these platforms provide various examples of trending outfits that customers can refer to when they purchase their clothes. We also analyzed that this phenomenon means people are getting more and more interested in how to dress up well, rather than just purchasing random clothes they need. Taking these into consideration, we thought changing the focus from the clothing itself to ourselves who dress up would make a new value. Below are the differences between our service to other existing fashion platforms.

1. Our service recommends various styles based on clothing we already have in our closet. Many online platforms provide outfit examples people can refer to, but still, people don't know well which style fits them best. This also helps people organize clothes they already have in their closets.
2. Our service saves time to find items or styles that match the clothing they have. There is an overload of information that makes people spend more time searching for a new fashion trend as well as thinking of what to wear. By recommending outfits according to the cloth user selects, we can shorten this unnecessary time thinking of what to wear.
3. Our service creates new fashion market segmentation - none of the above fashion industry landscapes matches our service. By having a feature that links users to online commerce platforms (where the market transaction size is largest among fashion segments) to buy new clothes they don't have, our service can penetrate the value chain of the fashion industry.

1.4. Key Features

Closet Settings

When users register their clothes, the ML model classifies the colors. Users can see their own clothes classified according to type, color, and pattern at a glance. In addition, users can record the date of each outfit, helping to manage their own clothes.

Outfit Recommendation

Users can be recommended to coordinate based on registered clothes, clothes users haven't worn recently, and specific clothes users want.

Popular Outfits Recommendation

If users have a desired coordination, they can access the clothing sales site and purchase it through the purchase link.

2. Architecture & Design

2.1. Design Details

2.1.1. Model

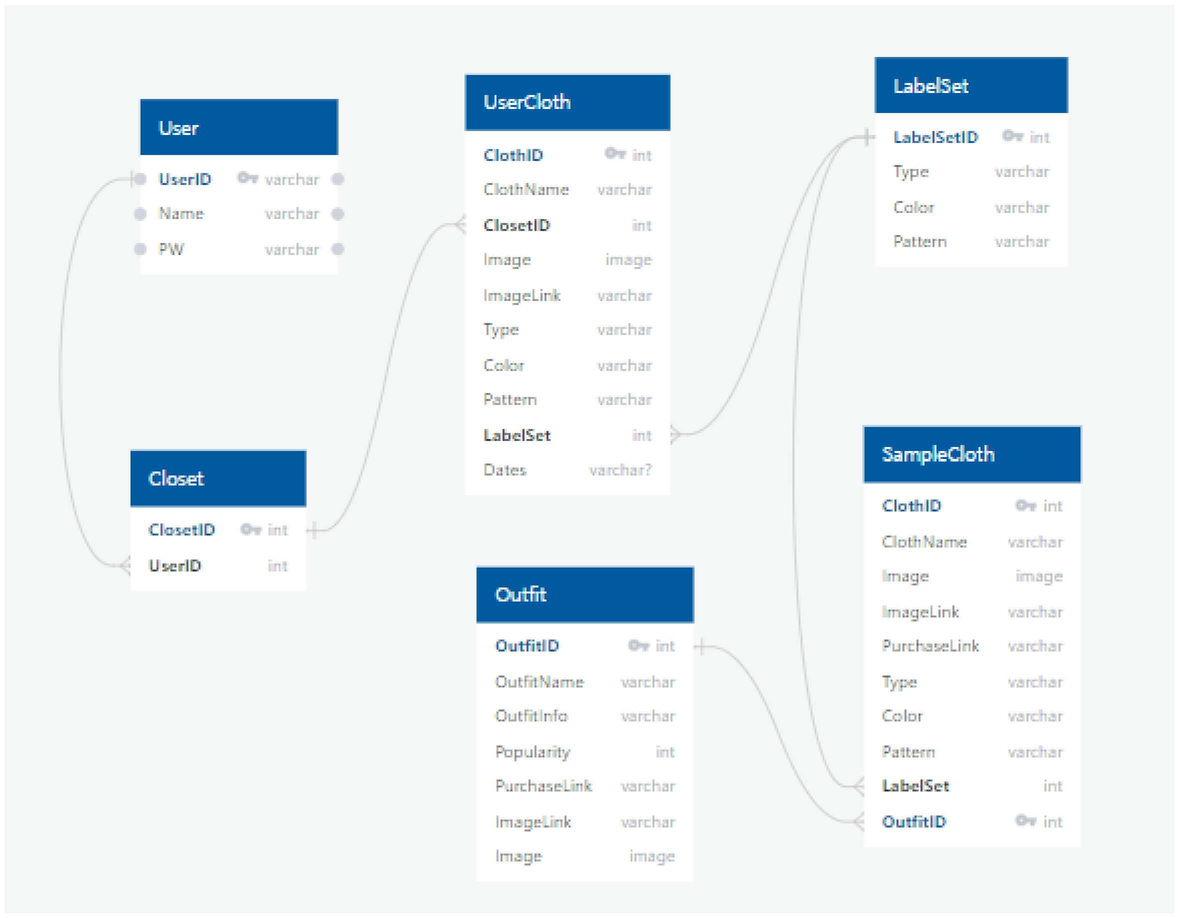


Figure 3. Backend database relation

Our service's database consists of User, LabelSet, UserCloth, SampleCloth, Outfit, and Closet. The User table stores information related to the user, and Closet is a table that connects the user with the clothes stored by the user. UserCloth stores information related to clothes stored by the user. SampleCloth and Outfit store clothes and coordination information to be used to connect the user's clothes and coordination. LabelSet is a table for storing labels used to compare whether the user's clothes match the coordination clothes.

2.1.2. Frontend Design

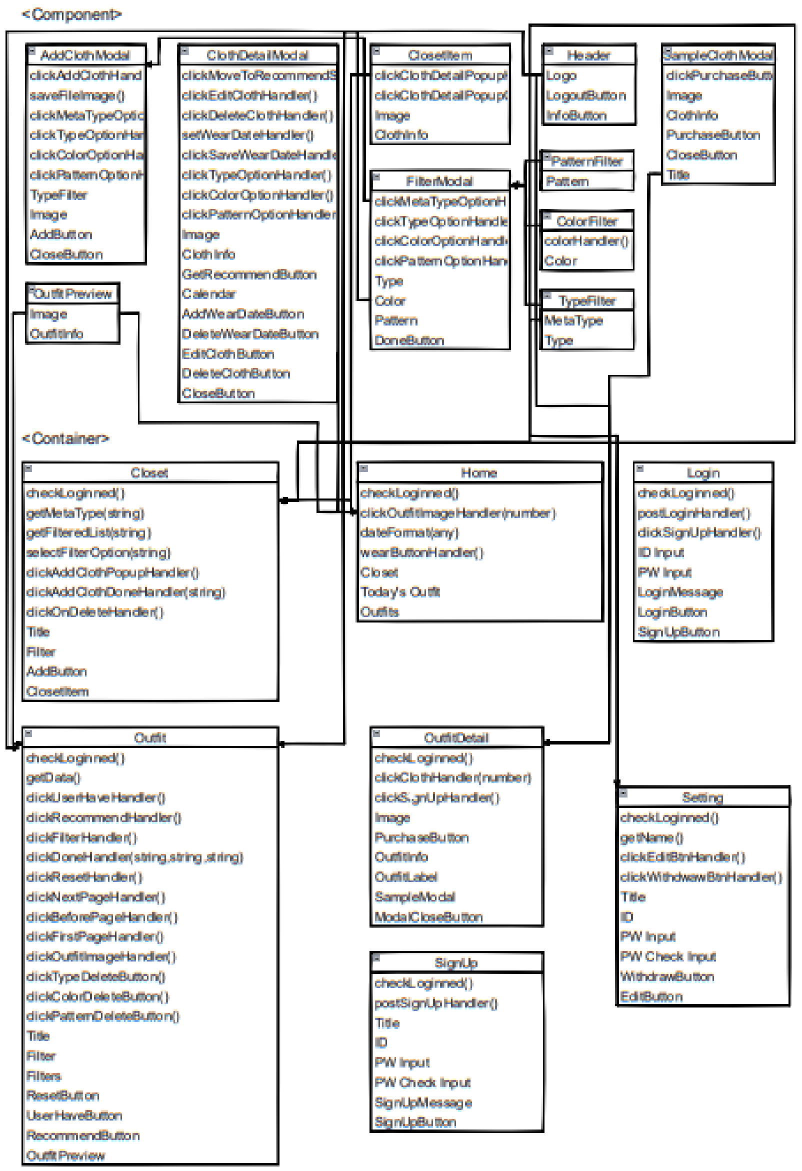


Figure 4. Frontend relation

1. Closet Page (/closet)

- Display all the clothes and labels user uploaded
- By selecting the filter of cloth class, users can see only the top, bottom or outer clothes.
- By clicking the ‘Add’ button, a modal page that users can upload their new clothes pops up.
    - Move to 2. Add Cloth Pop-up Page (/closet/add)
- By clicking the cloth image, a modal page that shows the details of the cloth pops up.
    - Move to Cloth Detail Pop-up Page (/closet/:id)
- Header is displayed.
    - By clicking My information button, move to **11. Setting page (/setting).
    - By clicking Logout button, move to **8.Login page (/login) and logout.

2. Add Cloth Pop-up

- User can add new cloth and label into ‘Closet’ page
- By clicking ‘Upload’ button, user can upload a cloth image and see the preview image
    - For the color label, color label of the image is set either by ML model that classifies color or user can select the color label on their own
    - For the type and pattern label, user can select cloth's labels
    - When both the cloth image and label is set -> ‘Add’ button is enabled
- By clicking ‘Add’ button, pop-up window is closed and closet page is refreshed, and user can see the outfit list filtered to class of the cloth he/she uploaded
- By clicking 'Cancel' button -> pop-up window closed

3. Cloth Detail Pop-up

- Cloth image and types are displayed
- Users can see the ‘get recommendation’ button, edit button, delete button and cancel button.
    - By clicking the ‘Delete’ button, the user can delete the cloth he/she chose.
    - By clicking the ‘Recommended Styles’ button, the user is moved to ‘Outfit Page (/outfit)’. Label information is sent so that the filter in ‘Outfit Page’ can be automatically set.
    - By clicking 'Cancel' button, pop-up window closed
- Calendar is displayed
- If the user selects the date in the calendar, 'add date' button or 'delete worn date' button is displayed.
    - The user can record the date when they wear the cloth by selecting the date and clicking the 'add date' button.
    - The user can delete the date in the worn date log by selecting the date and clicking the 'delete worn date' button.

4. Outfit Page (/outfit)

- Display all the outfits sorted by popularity
- Users can see the 'userHave', 'recommend', 'set Filter' and 'reset' buttons.
- By clicking the image of outfit in list
    - move to 6. Outfit Detail page
- By clicking 'Add Label Filter' button
    - move to 5. The modal 'Outfit Filter Setting' pops up.
    - If some labels are set, labels and erase buttons for each label are displayed.
    - If the user clicks the erase button for the label, the selected label is removed from the current filter.
- By clicking 'userHave' button
    - 'userHave' filter set and page refreshed.
- By clicking 'recommend' button
    - 'recommend' filter set and page refreshed.
- By clicking 'reset' button
    - All set filter removes
- Header is displayed.
    - By clicking My information button, move to **11. Setting page (/setting).
    - By clicking Logout button, move to **8.Login page (/login) and logout.

5. Outfit Filter Setting Pop-up

- Display 'MetaType', 'Type', 'Color' and 'Pattern' options.
    - Users can select labels in each category.
    - By setting the 'MetaType' category, 'Type' category's data changed to the specific type of class in that meta type.
- User can see 'Confirm' and 'Cancel' Button
    - If user chooses labels and clicks 'Confirm' button, pop-up window is closed, filter applied
    - If user clicks 'Cancel' button, pop-up window is closed

6. Outfit Detail Page (/outfit/:id)

- Display data of selected outfit and the list of samplecloth images that are included in outfit
    - By clicking the image of clothes, move to 7. A modal for Sample Cloth pops up.
- 'Purchase Link' button is displayed.
    - By clicking the 'Purchase Link' button, redirect to the shopping mall page for outfit outside our service.
- Header is displayed.
    - By clicking My information button, move to **11. Setting page (/setting).
    - By clicking Logout button, move to **8.Login page (/login) and logout.

7. Sample Cloth Pop-up

- Display image and label of selected sample cloth

- 'Purchase button' and 'close' button are displayed.
    - By clicking the 'Close 'button', the pop-up window is closed.
    - By clicking the 'Purchase Link' button, redirect to the shopping mall page for clothes outside our service.
- If user have userCloth with same LabelSet, display image of that userCloth

8. Login Page (/login)

- Users can log in and access the Sign Up Page (/signup).
    - By clicking the Sign Up button, move to 9. SingUp Page (/signup)
    - When the user enters the name and password and presses the Sign In button, the user is moved to the Home page (/home).
    - If there is no user whose name and password match, the letter "name or password does not match" appears under the password input area.

9. SignUp Page (/signup)

- If the user enters name, password, and password confirmation and presses Sign Up button, he or she can sign up.
    - If the name already exists, the letter "The name already exists" appears under the PW confirmation area, and the user cannot sign up.
    - If the PW input and PW confirmation input are not the same, the letter 'Password does not match' appears at the bottom of the PW confirmation area and cannot sign up.
    - If the subscription is successful, the user is moved to the Login page (/login) with a pop-up stating that the subscription was successful.

10. Home Page (/home)

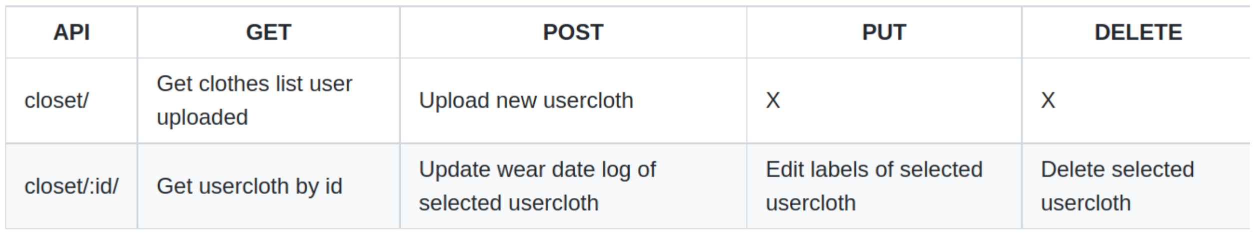
- Closet preview window is displayed.
    - Users can see three photos of clothes the user has uploaded in the closet.
    - User can access **1. 'Closet' page (/closet) by pressing the More button in the closet preview window.
- Today's outfit window is displayed.
    - Users can see a picture of recommended outfit and detailed information of outfit.
    - Press the Wear button in today's outfit window, the data of worn date is recorded.
- Popular Outfit window is displayed.
    - Users can see the list of images and info of popular outfits.
    - By clicking the image of outfit, move to **6. Outfit Detail Page (/outfit/:id).
    - My clicking More button, move to **4.Outfit Page (/outfit).
- Header is displayed.
    - By clicking My information button, move to **11. Setting page (/setting).
    - By clicking Logout button, move to **8.Login page (/login) and logout.

11. Setting Page (/setting)

- The user can modify his/her name, email, and password.
    - When modifying a password, the password and password confirmation input values must be the same.
    - The user can withdraw through the Withdrawal button.
    - The modified information is saved through Save button and moved to **10. Home page (/home).

2.1.3. Backend Design

Closet

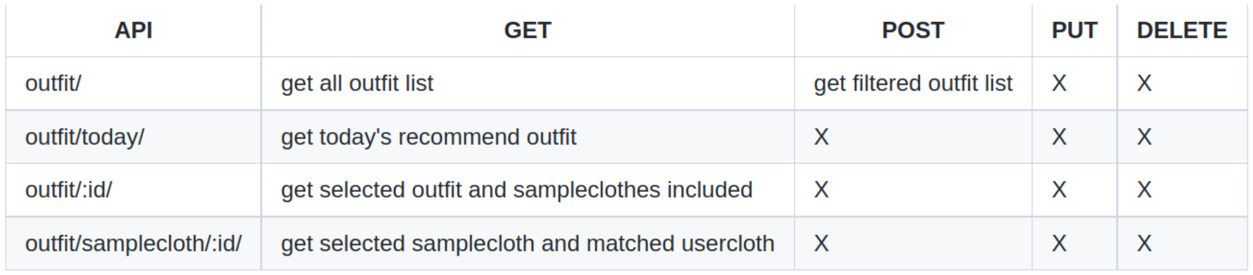


API	GET	POST	PUT	DELETE
closet/	Get clothes list user uploaded	Upload new usercloth	X	X
closet/:id/	Get usercloth by id	Update wear date log of selected usercloth	Edit labels of selected usercloth	Delete selected usercloth

Figure 5. Backend design - closet

These APIs are used in Closet Page and Cloth Detail Pop-up. In closet/:id/, we separated the update function into POST api and PUT api. Wear date updating only adds or deletes data in the last index of dates log in DB, so we implemented this function as POST API. Label updating changes the label data of DB, so we implemented this function as PUT API.

Outfit



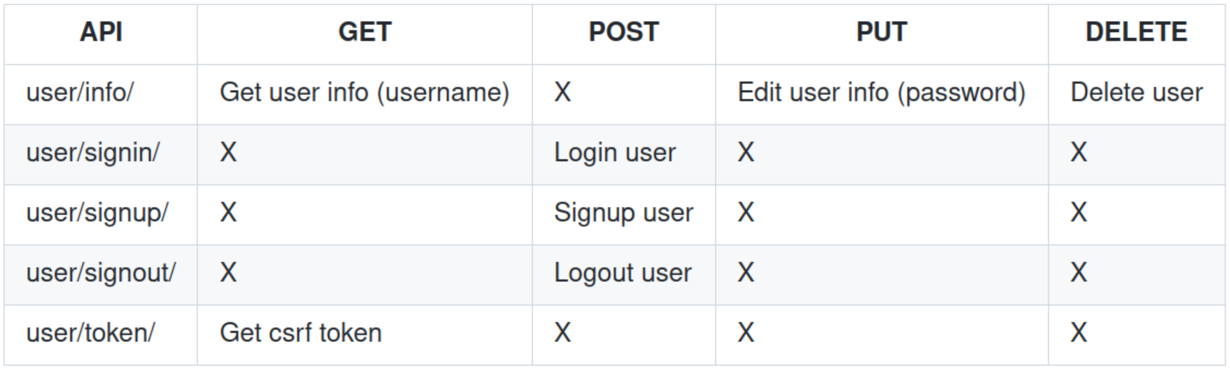
API	GET	POST	PUT	DELETE
outfit/	get all outfit list	get filtered outfit list	X	X
outfit/today/	get today's recommend outfit	X	X	X
outfit/:id/	get selected outfit and sampleclothes included	X	X	X
outfit/samplecloth/:id/	get selected samplecloth and matched usercloth	X	X	X

Figure 6. Backend design - outfit

outfit/today/ API is used in the Home Page. outfit/ API is used in Outfit Page and Home Page, outfit/:id/ API is used in Outfit Detail Page, and outfit/samplecloth/:id/ API is used in Sample Cloth Pop-up.

In outfit/ API, there are 2 methods, GET and POST method. We implemented GET API as the function to get an outfit list without filter and POST API as the function to get with the filter.

User

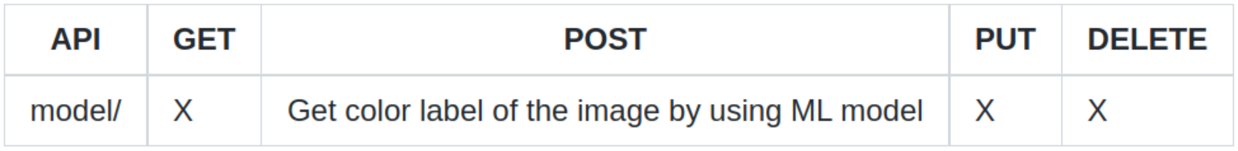


API	GET	POST	PUT	DELETE
user/info/	Get user info (username)	X	Edit user info (password)	Delete user
user/signin/	X	Login user	X	X
user/signup/	X	Signup user	X	X
user/signout/	X	Logout user	X	X
user/token/	Get csrf token	X	X	X

Figure 7. Backend design - user

user/info/ API is used in Setting Page. Other APIs are used to handle the user's authentication state.

Model



API	GET	POST	PUT	DELETE
model/	X	Get color label of the image by using ML model	X	X

Figure 8. Backend design - model

This model/ API is about the ML feature. This API is used in Add Cloth Pop-up. If this API is called with image data, our ML classifier in server classifies the cloth image's color label and response with this label.

3. Implementation, Testing & Deployment

3.1. Key Implementations

3.1.1 Data Crawling

Data crawling was conducted using a library called **Selenium**. Because selenium supports dynamic web crawling, we chose selenium as the reason for accessing and crawling data through direct clicks. Since the purpose is to collect data to match the user's clothes, information such as color, color, type, and popularity was crawled. We crawled at an online shopping mall called ‘Musinsa’ and collected about 20,000 clothes and coordination information. Data has been crawled through css selector or className in HTML.

3.1.2 Libraries and Frameworks for Key Functions

A. Add Cloth → Select Color Function (**React Color**)

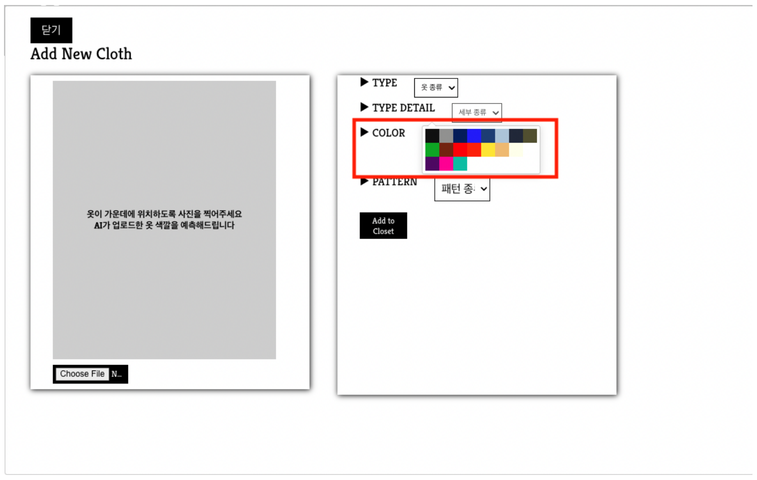


Figure 9. Add cloth modal

When a user uploads their cloth into the ‘Closet’ page, one has to choose the color of the cloth. Instead of implementing this as a ‘select’ dropdown like other features, we used **‘React Color’ library** to show users a color palette for their color of the cloth, which would be much more intuitive to users. We customized colors in the color palette to match the color we had in our database.

B. Add Cloth → Machine-learning Color Classifier (**Tensorflow/Keras**)

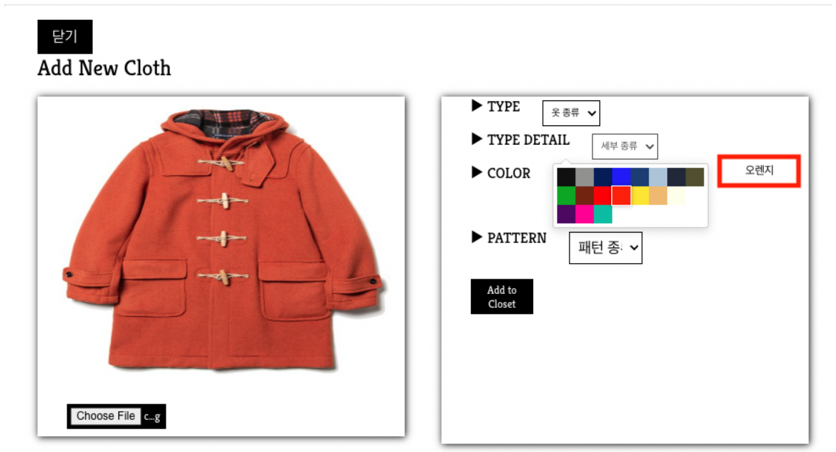


Figure 10. Add Cloth Modal

As previously mentioned, the user has to fill in the color of the cloth they want to upload. However, it's quite burdensome for users to choose and click the color of the cloth whenever they upload their clothes. To reduce the number of clicks of users to finish uploading, we built a machine-learning model that classifies the color of the cloth. We built a MLP model which classifies target cloth colors given R/G/B integers as X values. Our model has 5 fully connected layers and is trained with Cross Entropy loss function, which shows an accuracy of 89%. Unlike other large-scale services that have separate servers for training and saving the result of the model regularly, our model didn't need to be re-trained as our X values are not time-series and always the same. Thus for simplicity we put our model in our service's backend and implemented a backend API that gets an image of the cloth user uploaded and returns the color the model classifies. The result(color) is shown up as in the above screen capture.

C. Cloth Detail → '입은 날짜 추가하기' Function (**React Datepicker**)

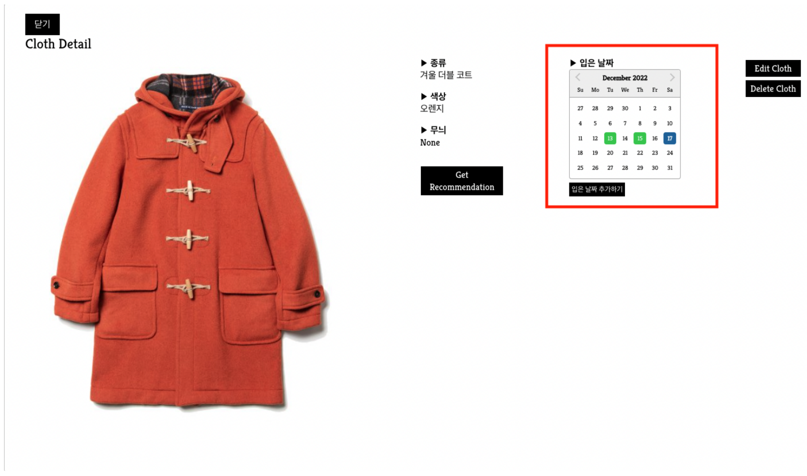


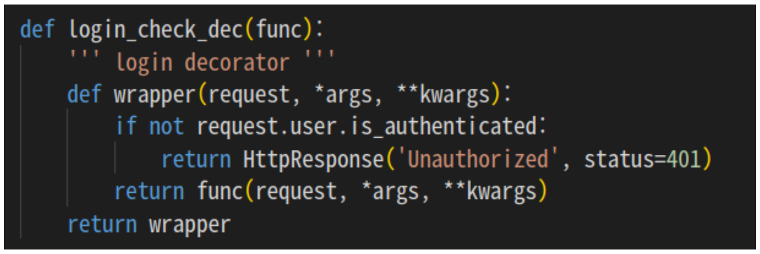
Figure 11. Cloth detail modal

One of our service's key functions is to recommend 'today's outfit' based on the clothes users already have in their closet. One of the logics when we recommend 'today's outfit' is only to recommend an outfit that contains the user's clothing which the user wore at least 3 days before or more. In order to implement this rule, we need a function for the user to put in dates he wore the cloth. Instead of implementing this as a 'select' dropdown like other features, we used **'React Datepicker' library** to show users a calendar to put in or delete the date he/she wore the cloth, which would be much more intuitive to users. Dates the cloth had worn are marked as green on the calendar.

3.1.3 Recommendation System

Users select and upload their clothing image and label. The color classification model provides a hint about the color of users' image. Store the label set that contains type, color, and pattern label with clothing information. Today's outfit is recommended considering whether the label sets on the clothes users have matched the label sets on all the clothes in the coordination and whether users wore it for the past week.

3.1.4 Design Patterns

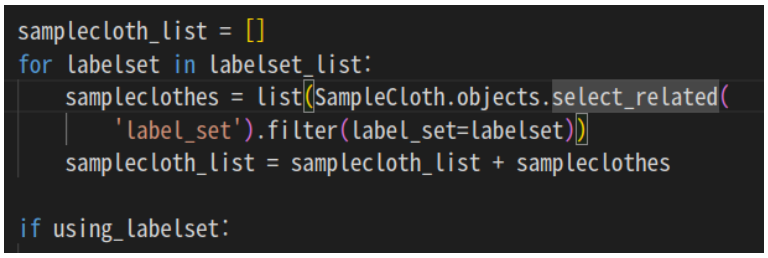


```python
def login_check_dec(func):
    ''' login decorator '''
    def wrapper(request, *args, **kwargs):
        if not request.user.is_authenticated:
            return HttpResponse('Unauthorized', status=401)
        return func(request, *args, **kwargs)
    return wrapper
```

Figure 12. login_check_dec

We used the **Decorator design pattern** in our Backend Implementation. In most of our API views, user authentication checking is needed before the core operation starts. Thus we made the 'login_check_dec' decorator to check the user's authentication state, and added this decorator in front of APIs that needed authentication checking.

3.1.5 DB Optimization



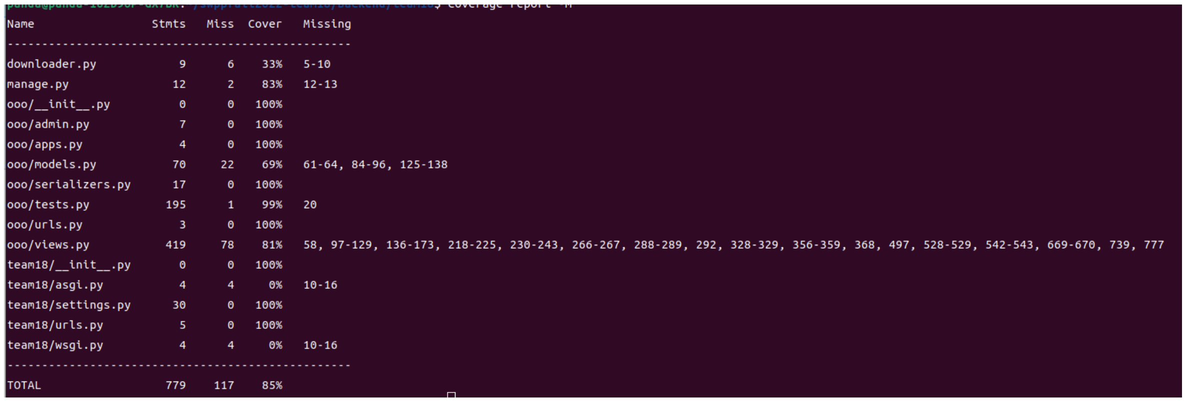
```python
samplecloth_list = []
for labelset in labelset_list:
    sampleclothes = list(SampleCloth.objects.select_related(
        'label_set').filter(label_set=labelset))
    samplecloth_list = samplecloth_list + sampleclothes

if using_labelset:
```

Figure 13. select_related function

We optimized DB by using ‘**select_related**’. In outfit/today/ API and outfit/ POST API, there are a chain of queries to find the appropriate outfit object to fit the filter through userCloth objects, sampleCloth objects and LabelSet objects. Thus we used the ‘select_related’ parameter to solve the ‘n+1 queries problem’ in our Backend Implementation.

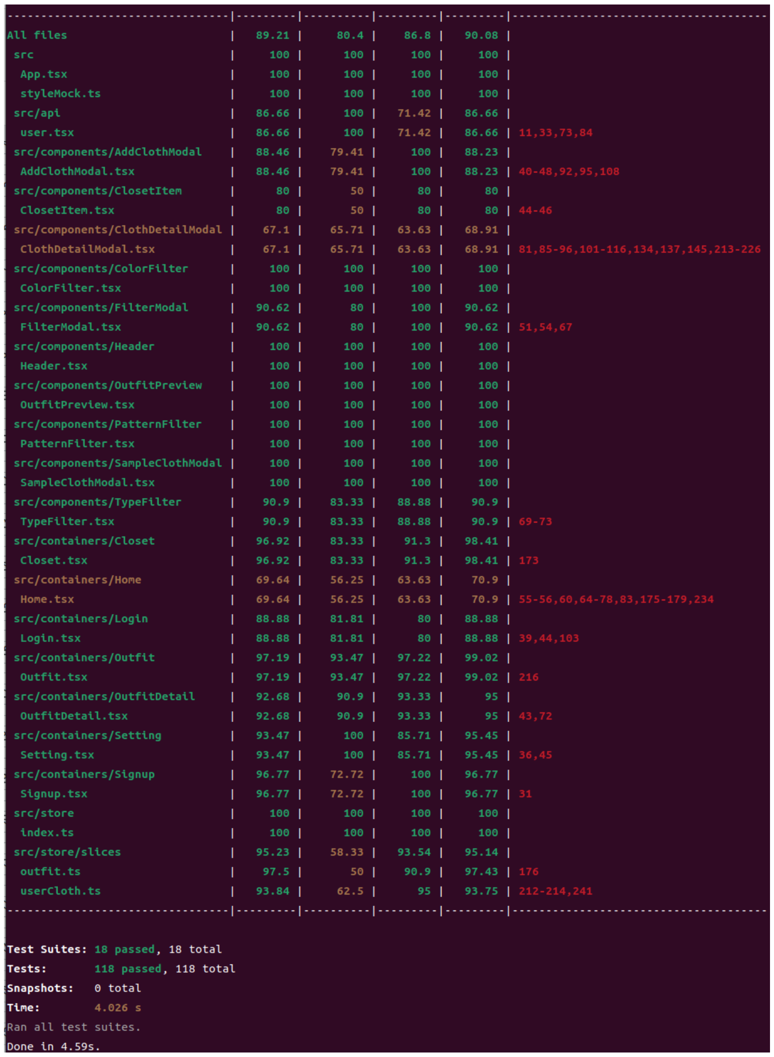
3.2. Testing Result

The overall test coverage was approximately 90% for the frontend, and 85% for the backend.



```
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
downloader.py            9      6    33%   5-10
manage.py               12      2    83%   12-13
ooo/__init__.py          0      0   100%
ooo/admin.py             7      0   100%
ooo/apps.py              4      0   100%
ooo/models.py           70     22    69%   61-64, 84-96, 125-138
ooo/serializers.py      17      0   100%
ooo/tests.py           195      1    99%   20
ooo/urls.py              3      0   100%
ooo/views.py           419     78    81%   58, 97-129, 136-173, 218-225, 230-243, 266-267, 288-289, 292, 328-329, 356-359, 368, 497, 528-529, 542-543, 669-670, 739, 777
team18/__init__.py       0      0   100%
team18/asgi.py           4      4     0%   10-16
team18/settings.py      30      0   100%
team18/urls.py           5      0   100%
team18/wsgi.py           4      4     0%   10-16
--------------------------------------------------
TOTAL                  779    117    85%
```

Figure 14. Backend test coverage



```
---------------------------------|---------|----------|---------|---------|-----------------------------------
All files                        |   89.21 |     80.4 |    86.8 |   90.08 |
 src                             |     100 |      100 |     100 |     100 |
  App.tsx                        |     100 |      100 |     100 |     100 |
  styleMock.ts                   |     100 |      100 |     100 |     100 |
 src/api                         |   86.66 |      100 |   71.42 |   86.66 |
  user.tsx                       |   86.66 |      100 |   71.42 |   86.66 | 11,33,73,84
 src/components/AddClothModal    |   88.46 |    79.41 |     100 |   88.23 |
  AddClothModal.tsx              |   88.46 |    79.41 |     100 |   88.23 | 40-48,92,95,108
 src/components/ClosetItem       |      80 |       50 |      80 |      80 |
  ClosetItem.tsx                 |      80 |       50 |      80 |      80 | 44-46
 src/components/ClothDetailModal |    67.1 |    65.71 |   63.63 |   68.91 |
  ClothDetailModal.tsx           |    67.1 |    65.71 |   63.63 |   68.91 | 81,85-96,101-116,134,137,145,213-226
 src/components/ColorFilter      |     100 |      100 |     100 |     100 |
  ColorFilter.tsx                |     100 |      100 |     100 |     100 |
 src/components/FilterModal      |   90.62 |       80 |     100 |   90.62 |
  FilterModal.tsx                |   90.62 |       80 |     100 |   90.62 | 51,54,67
 src/components/Header           |     100 |      100 |     100 |     100 |
  Header.tsx                     |     100 |      100 |     100 |     100 |
 src/components/OutfitPreview    |     100 |      100 |     100 |     100 |
  OutfitPreview.tsx              |     100 |      100 |     100 |     100 |
 src/components/PatternFilter    |     100 |      100 |     100 |     100 |
  PatternFilter.tsx              |     100 |      100 |     100 |     100 |
 src/components/SampleClothModal |     100 |      100 |     100 |     100 |
  SampleClothModal.tsx           |     100 |      100 |     100 |     100 |
 src/components/TypeFilter       |    90.9 |    83.33 |   88.88 |    90.9 |
  TypeFilter.tsx                 |    90.9 |    83.33 |   88.88 |    90.9 | 69-73
 src/containers/Closet           |   96.92 |    83.33 |    91.3 |   98.41 |
  Closet.tsx                     |   96.92 |    83.33 |    91.3 |   98.41 | 173
 src/containers/Home             |   69.64 |    56.25 |   63.63 |    70.9 |
  Home.tsx                       |   69.64 |    56.25 |   63.63 |    70.9 | 55-56,60,64-78,83,175-179,234
 src/containers/Login            |   88.88 |    81.81 |      80 |   88.88 |
  Login.tsx                      |   88.88 |    81.81 |      80 |   88.88 | 39,44,103
 src/containers/Outfit           |   97.19 |    93.47 |   97.22 |   99.02 |
  Outfit.tsx                     |   97.19 |    93.47 |   97.22 |   99.02 | 216
 src/containers/OutfitDetail     |   92.68 |     90.9 |   93.33 |      95 |
  OutfitDetail.tsx               |   92.68 |     90.9 |   93.33 |      95 | 43,72
 src/containers/Setting          |   93.47 |      100 |   85.71 |   95.45 |
  Setting.tsx                    |   93.47 |      100 |   85.71 |   95.45 | 36,45
 src/containers/Signup           |   96.77 |    72.72 |     100 |   96.77 |
  Signup.tsx                     |   96.77 |    72.72 |     100 |   96.77 | 31
 src/store                       |     100 |      100 |     100 |     100 |
  index.ts                       |     100 |      100 |     100 |     100 |
 src/store/slices                |   95.23 |    58.33 |   93.54 |   95.14 |
  outfit.ts                      |    97.5 |       50 |    90.9 |   97.43 | 176
  userCloth.ts                   |   93.84 |     62.5 |      95 |   93.75 | 212-214,241
---------------------------------|---------|----------|---------|---------|-----------------------------------

Test Suites: 18 passed, 18 total
Tests:       118 passed, 118 total
Snapshots:   0 total
Time:        4.026 s
Ran all test suites.
Done in 4.59s.
```

Figure 15. Frontend test coverage

3.3. Deployment

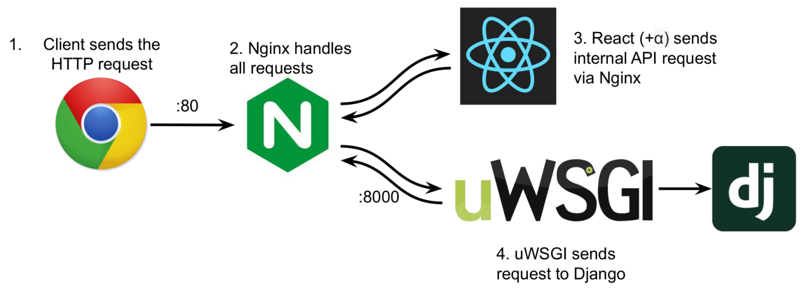
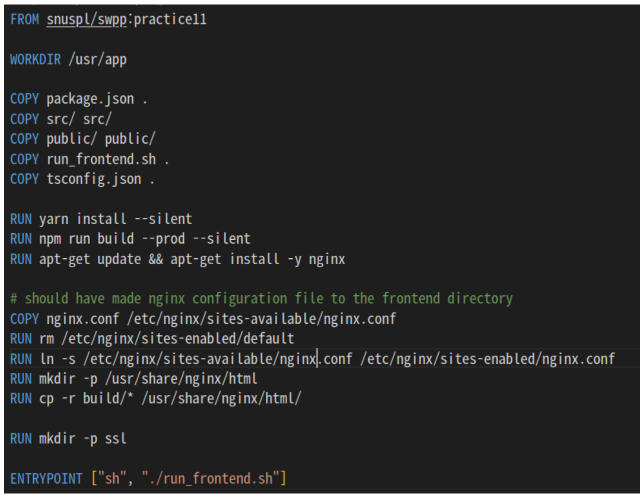

1. Client sends the HTTP request
:80
2. Nginx handles all requests
3. React (+α) sends internal API request via Nginx
:8000
uWSGI
4. uWSGI sends request to Django

Figure 16. deployment figure

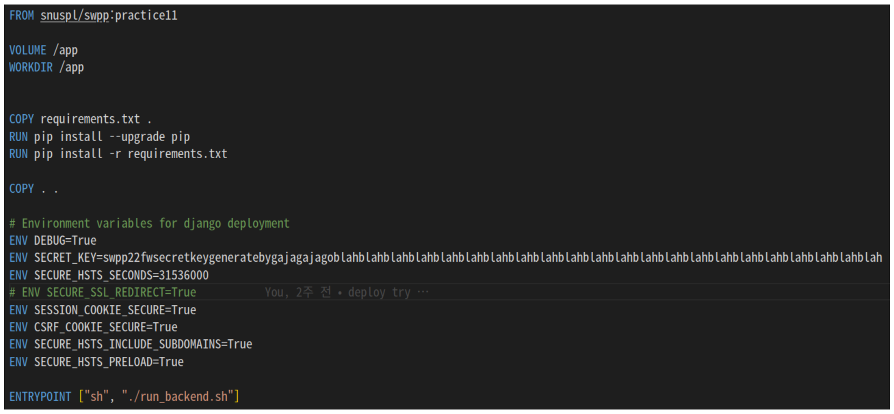
For each service, frontend and backend, we wrote Dockerfiles to build a container. Then, we deployed them to https://recommendyouroutfit.shop.



```
FROM snuspl/swpp:practice11

WORKDIR /usr/app

COPY package.json .
COPY src/ src/
COPY public/ public/
COPY run_frontend.sh .
COPY tsconfig.json .

RUN yarn install --silent
RUN npm run build --prod --silent
RUN apt-get update && apt-get install -y nginx

# should have made nginx configuration file to the frontend directory
COPY nginx.conf /etc/nginx/sites-available/nginx.conf
RUN rm /etc/nginx/sites-enabled/default
RUN ln -s /etc/nginx/sites-available/nginx.conf /etc/nginx/sites-enabled/nginx.conf
RUN mkdir -p /usr/share/nginx/html
RUN cp -r build/* /usr/share/nginx/html/

RUN mkdir -p ssl

ENTRYPOINT ["sh", "./run_frontend.sh"]
```

Figure 17. Frontend Dockerfile

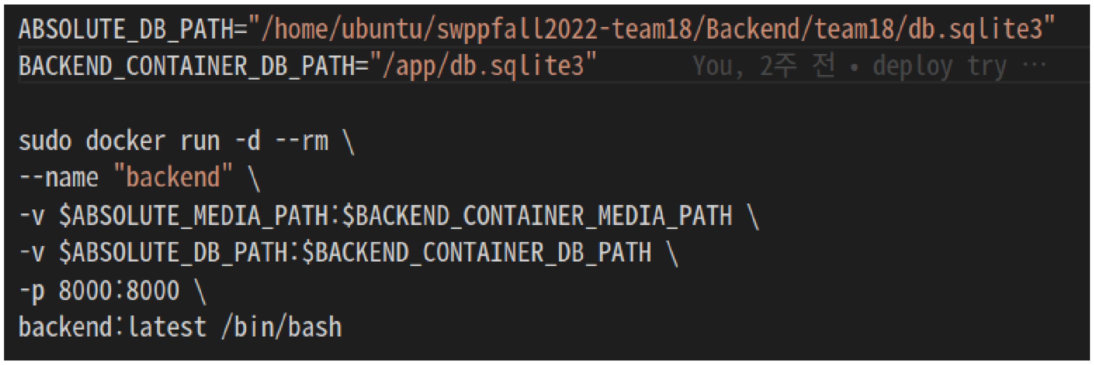
Dockerfile of frontend and backend sets up from ‘snuspl/swpp:practice11’ image that contains nginx, uWSGI and other useful things, build ooo app to bundle js file and copy it to public directory of nginx from which files are served.



```
FROM snuspl/swpp:practice11

VOLUME /app
WORKDIR /app


COPY requirements.txt .
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

COPY . .

# Environment variables for django deployment
ENV DEBUG=True
ENV SECRET_KEY=swpp22fwsecretkeygeneratebygajagajagoblahblahblahblahblahblahblahblahblahblahblahblahblahblahblahblahblahblahblahblahblahblah
ENV SECURE_HSTS_SECONDS=31536000
# ENV SECURE_SSL_REDIRECT=True          You, 2주 전 • deploy try ⋯
ENV SESSION_COOKIE_SECURE=True
ENV CSRF_COOKIE_SECURE=True
ENV SECURE_HSTS_INCLUDE_SUBDOMAINS=True
ENV SECURE_HSTS_PRELOAD=True

ENTRYPOINT ["sh", "./run_backend.sh"]
```

Figure 18. Backend Dockerfile

Dockerfile of backend gets all requirements requirements.txt, copy all and then run uWSGI server.



```
ABSOLUTE_DB_PATH="/home/ubuntu/swppfall2022-team18/Backend/team18/db.sqlite3"
BACKEND_CONTAINER_DB_PATH="/app/db.sqlite3"        You, 2주 전 • deploy try …

sudo docker run -d --rm \
--name "backend" \
-v $ABSOLUTE_MEDIA_PATH:$BACKEND_CONTAINER_MEDIA_PATH \
-v $ABSOLUTE_DB_PATH:$BACKEND_CONTAINER_DB_PATH \
-p 8000:8000 \
backend:latest /bin/bash
```

Figure 19. Docker run script

[Figure 19] is a script that runs backend docker. By giving -v command, save the db and media file on pre-mounted volume.

4. Conclusion

4.1. Significance of the project

Our service tried to analyze the clothes users have and recommend outfits to try to help them get to know about current fashion trends and styles more easily. By using our service, we expect users could organize their closet by bringing it from offline to online and get to know how to match their clothes to make a fashionable outfit.

Not only we built our service on the business aspect but also we made a tremendous effort to implement a web service which has robust design and architecture that is scalable, reliable and user-friendly. As discussed above, we implemented various design patterns and optimized DB to build faultless service and brought various libraries and built our own recommendation algorithms to make our service user-friendly.

4.2. Difficulties & How we resolved

Data Crawling and collecting metadata(labels)

We compare labels of the user's cloth user chose with sample clothes which outfit image contains. Labels for each cloth image are needed for comparison purposes, and we collect those by metadata of the images while crawling. As there are no labels for color for sample clothes images, we thought of 1) building a model to classify a color of clothes, 2) take RGB value of the cloth to decide color, or 3) crawl a cloth title that contains color value. In conclusion, we chose 2 because most titles contain color information and it is easy to match the labels used in the user image classification model.

Implementation difficulties

▪ Implementing each part of the code and merging it on git

Each of the members had to update the same file, which makes conflict among codes. People with the same implementation cross-checked and resolved the collision so that there were no missing parts or bugs.

▪ Implementing page as modal format

We thought implementing a page as a modal rather than one separate page would be easier to implement, but we found it quite tricky to implement. Building a program with a modal format requires many components to be newly created.

▪ Constructing the DB scheme was the most challenging problem in the Design & Planning process

Especially when designing an Outfit Page where outfit recommendation is provided to users, clothes that users have and the clothes from outfit recommendation should be distinguished. This is needed as our service should find out if a user has the cloth that the commendation system provides, and to implement this, the above problem should be considered. Not only should a new table (sample cloth – data which are needed for labels of outfit recommendation) be created, but we also have to consider the links between the cloth user has and sample clothes. It leads to a new problem that similar data are saved as duplicated in separate tables that would affect the speed of data processing.

▪ Our team had difficulties while deploying our service due to handling user image input.

Images user uploaded should be saved using Formdata and should be saved into the Backend folder as well as our database. All the processes worked fine when we tested these functions in our local computer, but the image didn't show up to the front (closet page) correctly when we deployed our service using Docker. The problem was that we didn't mount the frontend to our backend folder where we're saving user-input images, and made the Debug parameter as 'false'. We resolved this issue and now images that user uploaded are correctly shown up in our frontend page.