Team 16 Final Report

Dongmin Kim, Limbeom Choi, Sunbin Kwon, and kyenghwan kim

1. Abstract

Coding MBTI is a web service from which users can get a chance to inspect their code, get insight on their coding habit and level compared to others and construct their personalized strategy to empower coding skills. Coding MBTI gives users the exact coordinates of their coding habits among many other coders around the world and next coordinates to improve their code. To achieve that goal, our software will give 3 main experiences to our users.

1. Coding MBTI test, which tells what kind of coding style each user has using ML techniques based on 4 core criteria we've made.
2. Personalized Report, which gives thorough analysis based on the result of the MBTI test.
3. Matching System, which recommends coders who have different style, similar style, or better quality using the information from each user's Personalized Report.

Coding MBTI collects and saves every small data which is made from the user's MBTI test or just problem solving. And It analyzes those data and visualizes them by providing a Personalized Report to each user. Personalized Report includes better solutions to each problem. Users will be able to check their coding levels or styles among other users from Personalized Report. Based on that, Coding MBTI could also help people who want to make a team, find a tutor, mentor, hire an employee, or find a friend. It would give them a recommendation that is suitable for their needs.

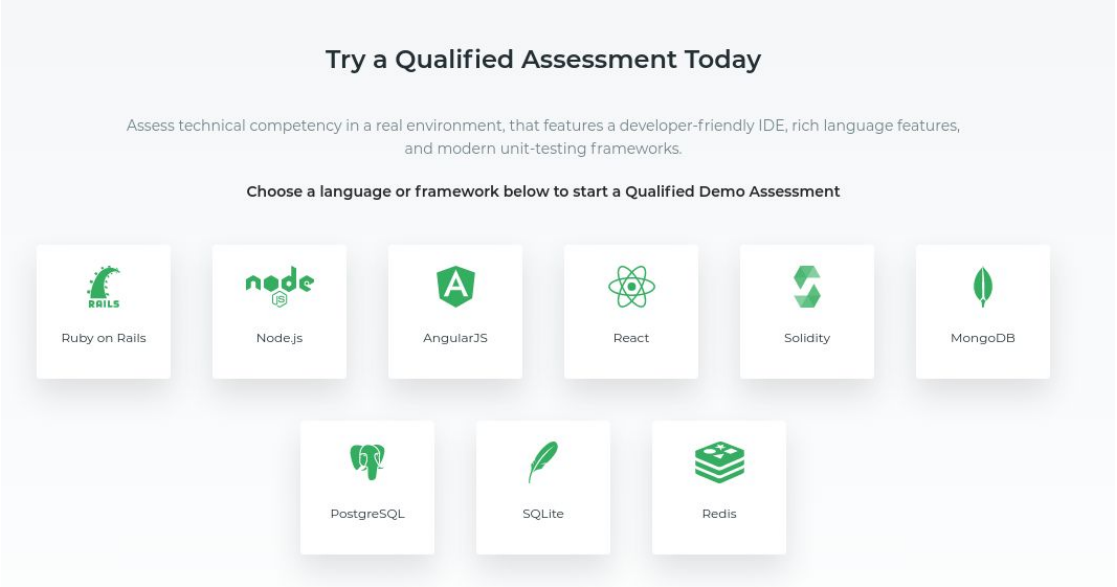
2. Motivation

Considering recent tendencies of everything being more personalized and being more specialized, Coding MBTI is a very attractive service for coders or people who work in related fields. Current problem-solving websites don't really provide anything personalized or specialized, they just provide problems and solutions. We've always wanted to make a service that can provide personalized reports on user's code writing habits and skills, so that they can actually compare their codes with others in a more statistical and visualized way. And also inspired by a MBTI test, which has been a bit hit recently, that classifies each person's personality with 4 alphabet letters, we thought mimicking this MBTI test could be an effective way to

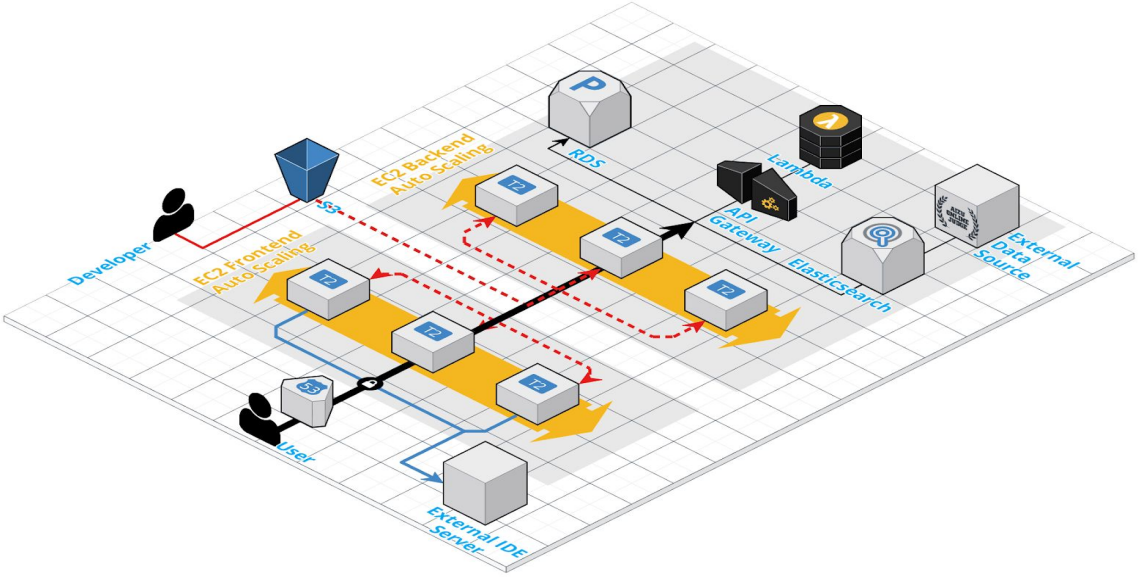
explain personalized information by classifying coders. And thought our coding MBTI could be a big hit among coders as MBTI tests did a while ago.

3. Related Work

Codewars is a manufacturer for a qualified assessment of code snippets for simple problems. So we set our project baselines to supply analysis results with solving problems. But we don’t judge a coder's code in a single criteria which tells who is a better coder and who is not, but classifies code with four different criteria. We believe like real MBTI, coding MBTI doesn’t mean some kinds of coding style is superior to others.



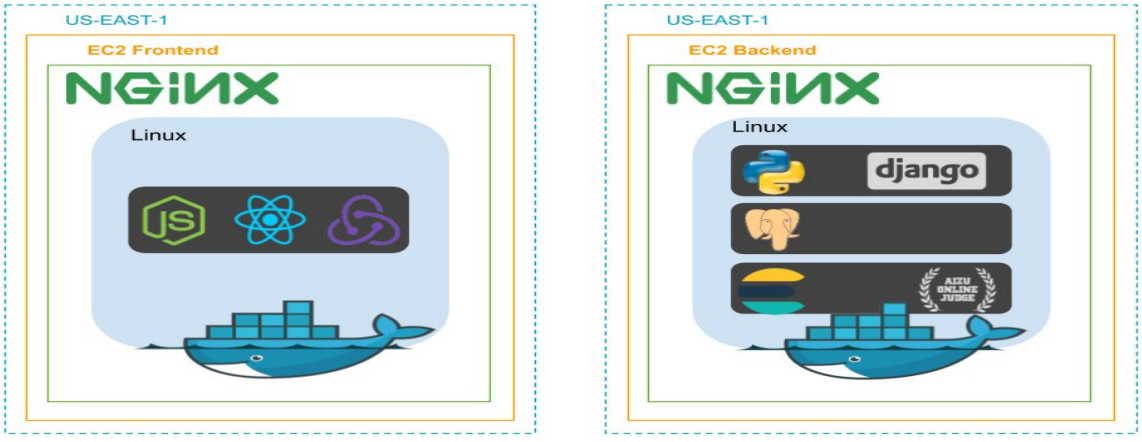
4. System Architecture



The system architecture of CodingMBTI contains two major components and one external minor component. The two major components are the frontend server and the backend server. Each is set up in an independent EC2 instance. The one minor component is External Web IDE Server. Because the minor component is exploited as an External API and not developed by ourselves, we are going to introduce only the two major components and its interactions in detail. The whole system architecture is configured as below.

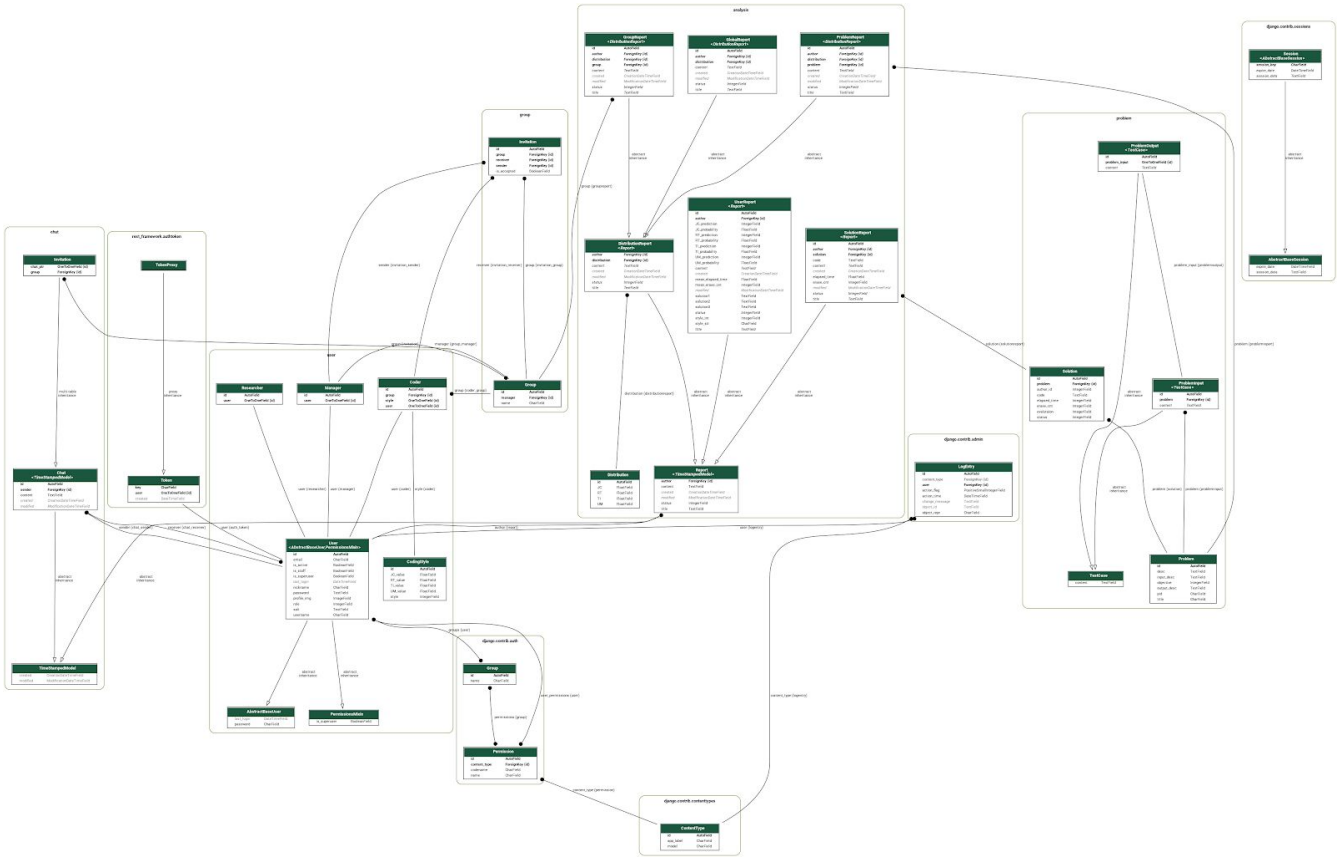
Let's look at the interactions between each component of architecture.

- For the frontend server,
  - We configured,
    - auto-scaling-enabled ec2 instances to run the frontend app.
    - Our current ec2 instance is t2.2xlarge
  - It interacts with,
    - (1) the EC2 backend server,
      - to enable saving/loading clients' data.
    - (2) the S3,
      - to enable the EC2's AWS CodeDeploy service to receive newly pushed source codes.
- For the backend server,
  - We configured,
    - (1) auto-scaling-enabled ec2 instances to run the backend app.
    - (2) the RDS Service for PostgreSQL database.
    - (3) Elasticsearch Service for preprocessing data from our external data source(the AIZU code problem-solving website).
  - It interacts with,
    - (1) the EC2 frontend server,
      - to enable saving/loading clients' data.
    - (2) the S3,
      - to enable the EC2's AWS CodeDeploy service to receive newly pushed source codes.



5. Backend Model

Django specific er relation models for our projects. We use only django based postgres connection with simple manipulation.



We listed all the apis as an image for convenience. We managed our interface between backend and frontend with a tool called swagger. You can visit url to see more information in the link here.

Coding mbti

Overview

Coding mbti api documentation

Find out more about Swagger

Tags

User
User authentication and profile management

Group
Group for manage coders

Problem
Everything about problem and solution

Chat
Everything about chat within users

Report
Everything about report and statistics

Paths

POST /api/user/login Login with given username and password

Responses

Code	Description	Links
204	Login Success	No Links
401	Authorization fail	No Links

POST /api/user/signup Login with given username and password

Responses

Code	Description	Links
204	SignUp Success	No Links
409	Duplicated *Content*	No Links

POST /api/user/withdraw Withdraw service

Responses

Code	Description	Links
204	Withdraw Success	No Links
405	Invalid argument	No Links

POST /api/user/change_password Change password

Responses

Code	Description	Links
204	Password Change Success	No Links
405	Invalid argument	No Links

POST /api/user/authenticate Additional authentication for researcher and manager

Responses

Code	Description	Links
405	Invalid input	No Links

POST /api/user/upload_profile_image Upload profile image

Responses

Code	Description	Links
204	Image upload success	No Links
400	Too large file size	No Links
405	Invalid input	No Links

GET /api/user/group/ get group by user_id of manager

Responses

Code	Description	Links
200	Return matching group json *Content*	No Links
404	Matching group not found	No Links

GET /api/user/style/ get coding style of user

Responses

Code	Description	Links
200	Return coding style of coder *Content*	No Links
404	No coding style found	No Links

GET /api/user/find_match/ get user list of similar coding style of current coder

Responses

Code	Description	Links
200	Successful matching user found *Content*	No Links
404	No User of same coding style found	No Links

POST /api/group/ Create group with group name

Responses

Code	Description	Links
200	Successful creation *Content*	No Links
401	Authorization fail	No Links
405	Already has a group	No Links

Type	Name	Scopes
oauth2	group_auth	write:group, read:group

POST /api/group/invite/ Invite coder to group with user id

Responses

Code	Description	Links
204	Invitation Success	No Links
403	Group max limit exceeded	No Links
404	User id not found	No Links

Type	Name	Scopes
oauth2	group_auth	write:group, read:group

POST /api/group/invite/accept/ Accept invitation to join group

Responses

Code	Description	Links
204	Invitaion Success	No Links
404	Group not found	No Links

Code	Description	Links
405	Token not valid	No Links

Type	Name	Scopes
oauth2	group_auth	write:group, read:group

GET `/api/group/find_relation/` Find coding style relation in group

application/json

Responses

Code	Description	Links
200	Successful retrieving relations *Content*	No Links
404	No relation in group	No Links
405	Someone hasn't complete solving	No Links

Type	Name	Scopes
oauth2	group_auth	read:group

GET `/api/problem/{style_id}` Get problem by style parameter

Parameters

Type	Name	Description	Schema
path	**style_id** *required*	Integer choice for corresponding query	integer

application/json

Responses

Code	Description	Links
200	successful operation *Content*	No Links

Code	Description	Links
405	Invalid input	No Links

Type	Name	Scopes
oauth2	problem_auth	read:problem

GET `/api/problem/{problem_id}/input` Get problem by style parameter

Parameters

Type	Name	Description	Schema
path	**problem_id** *required*	problem_id to find matching problemInput	integer

Responses

Code	Description	Links
200	successful operation *Content*	No Links
405	Invalid problem_id	No Links

Type	Name	Scopes
oauth2	problem_auth	read:problem

GET `/api/problem/{problem_input_id}/output` Get problemOutput by given problemInput

Parameters

Type	Name	Description	Schema
path	**problem_input_id** *required*	problem_input_id to find matching problemOutput	integer

Responses

Code	Description	Links
200	successful operation *Content*	No Links
405	Invalid problem_input_id	No Links

Type	Name	Scopes
oauth2	problem_auth	read:problem

GET `/api/problem/{problem_id}/solution` Get Solutions by given problem_id and user information in request

Parameters

Type	Name	Description	Schema
path	**problem_id** *required*	problem_id to find matching solutions	integer

Responses

Code	Description	Links
200	successful operation *Content*	No Links
405	Invalid problem_input_id	No Links

Type	Name	Scopes
oauth2	problem_auth	read:problem

POST `/api/problem/{problem_id}/solution` Submit solution to given problem_id with user information in request

Parameters

Type	Name	Scopes
oauth2	chat_auth	write:chat

DELETE `/api/chat/{chat_id}/` Delete chats

Parameters

Type	Name	Description	Schema
path	**chat_id** *required*	chat_id to resolve chat	integer

Responses

Code	Description	Links
204	Successful delete	No Links
404	Chat not found	No Links

Type	Name	Scopes
oauth2	chat_auth	write:chat

application/json

GET `/api/report/` get list of available reports

Responses

Code	Description	Links
200	Reports of given users *Content*	No Links
404	No report found	No Links

Type	Name	Scopes
oauth2	report_auth	read:report

POST `/api/report/` Create report

Responses

Code	Description	Links
200	Successful Update *Content*	No Links
405	Invalid argument	No Links

Type	Name	Scopes
oauth2	report_auth	write:report

PUT `/api/report/{report_id}/` update Report of given report_id

Parameters

Type	Name	Description	Schema
path	**report_id** *required*	report_id to resolve report	integer

Responses

Code	Description	Links
204	Successful update	No Links
404	No such report found	No Links

application/json

Type	Name	Scopes
oauth2	report_auth	write:report

DELETE `/api/report/{report_id}/` delete report of given report_id

Parameters

Type	Name	Description	Schema
path	**report_id** *required*	report_id to resolve report	integer

Responses

Code	Description	Links
204	Successful delete	No Links
404	No such report found	No Links

Type	Name	Scopes
oauth2	report_auth	write:report

Components

Schemas

Login

Properties

Name	Description	Schema
username *optional*		string
password *optional*		string

Signup

Properties

Name	Description	Schema
username *optional*		string
password *optional*		string
email *optional*		string
role *optional*		integer

Withdraw

Properties

Name	Description	Schema
username *optional*		string
password *optional*		string

ChangePassword

Properties

Name	Description	Schema
username *optional*		string
password *optional*		string
new_password *optional*		string

Authenticate

Properties

Name	Description	Schema
email *optional*		string
role *optional*		integer
affliation *optional*		string
reason *optional*		string

CodingStyle

Properties

Name	Description	Schema
id *optional*		integer
style *optional*		integer

Group

Properties

Name	Description	Schema
id *optional*		integer
manager *optional*		integer
name *optional*		string

CreateGroup

Properties

Name	Description	Schema
name *optional*		string

CreateGroupResponse

Properties

Name	Description	Schema
id *optional*		integer

InviteGroup

Properties

Name	Description	Schema
group_id *optional*		integer
receiver *optional*		integer

AcceptGroup

Properties

Name	Description	Schema
group_id *optional*		integer

Name	Description	Schema
token *optional*		integer

Problem

Properties

Name	Description	Schema
id *optional*		integer
content *optional*		string
style *optional*		integer

ProblemInput

Properties

Name	Description	Schema
id *optional*		integer
problem *optional*		integer
content *optional*		string

ProblemOutput

Properties

Name	Description	Schema
id *optional*		integer
problemInput *optional*		integer
content *optional*		string

ProblemSolution

Properties

Name	Description	Schema
id *optional*		integer
evaluation *optional*		integer
status *optional*		integer
content *optional*		string

ProblemSolutionPost

Properties

Name	Description	Schema
content *optional*		string

Chat

Properties

Name	Description	Schema
id *optional*		integer
sender *optional*		integer
receiver *optional*		integer
content *optional*		string
created *optional*		string
modified *optional*		string

ChatWithoutTime

Properties

Name	Description	Schema
sender *optional*		integer
receiver *optional*		integer
content *optional*		string

Report

Properties

Name	Description	Schema
id *optional*		integer
author *optional*		integer
content *optional*		string
created *optional*		string
modified *optional*		string

ReportWithoutTime

Properties

Name	Description	Schema
content *optional*		string

inline_response_200

Properties

Name	Description	Schema
id *optional*		integer

inline_response_200_1

Properties

Name	Description	Schema
style *optional*		integer
member *optional*		< inline_response_200 > array

6. Frontend View

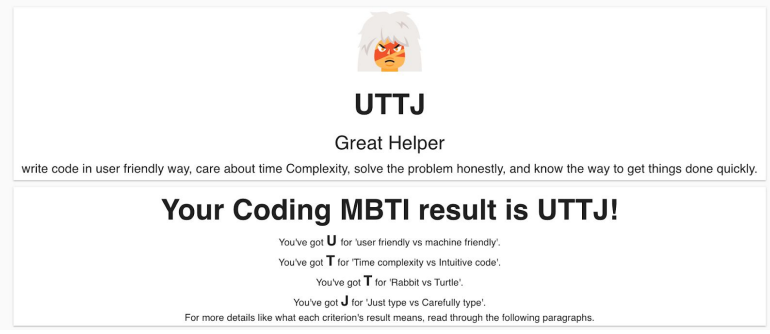
For the frontend view, we concentrated on 2 main pages.

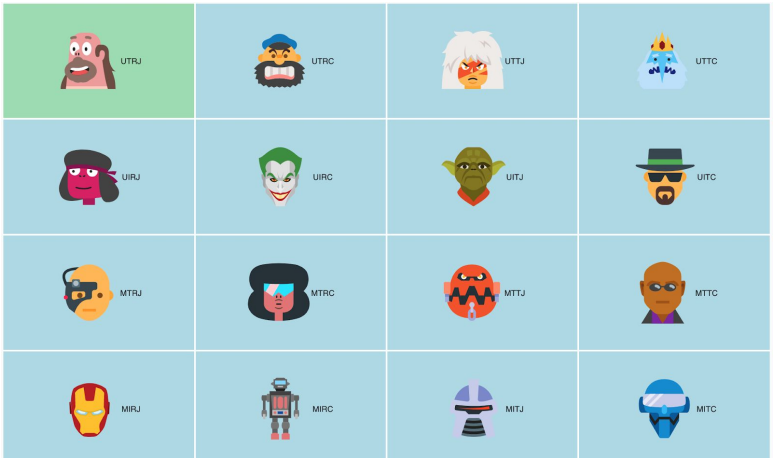
(1) Code IDE page

We tried to create pleasant UX design for coders to write down, run, test their code.

(2) Personal Report page

We tried to give a friendly and pleasant experience to users using images and detailed explanations on each type.





7. Implementation Details

7.1. Brief Development Process

Coding MBTI is implemented with python3.7 for backend, react and redux for frontend. Django is used as a web framework. For front-end testing, Jest&Enzyme is utilized as it is the default option for it. Like we have exercised in practice sessions and the final exam, we adopted the pytest framework.

All front-end, back-end files were managed on a docker container to prevent any possible bugs coming from different environments, and mis-communication. Finally, the whole web application is deployed to AWS ec2. We've tested our server by conducting a locust test, and optimized it by finding bottlenecks and upgrading our server spec. Every app version is contained into a docker container, uploaded and released to AWS EC2.

7.2. Database, Libraries, Frameworks, and External Services

We chose to use PostgreSQL as our database. It is an object-relational database system(ORDBMS), and an open-source DBMS that provides many features that can only be seen in the next generation of DBMSs. There are many public reviews that PostgreSQL is the most adaptable open-source DBMS. These are the reasons why we chose to use PostgreSQL as our database.

For backend python dependencies of the project are managed with a traditional approach utilizing pip and maintaining requirements.txt. For frontend although npm is a widely adopted package manager for nodejs, it is well-known for slowness. So, we chose to use Yarn, which helped a lot when performance matters.

We brought most of our data for the ML feature from Aizu online judge(http://judge.uaizu.ac.jp/onlinejudge). Thankfully, they provided all of their data free of charge, when that is to be used as an educational objective. Of course, we used testing libraries, such as jest, enzyme, and python unit-test. There will be more specific explanations about them in the 5.3 Testing and CI.

Main libraries other than those libraries that are mentioned above are followings:

- uwsgi: application server

- scikit-learn: provides data preprocessing and SVM model for Machine Learning
- psycopg2: PostgreSQL connector in python
- material-ui : Material CSS framework for react

7.3. Testing and CI

Our code is divided into two parts as most software projects do: frontend, backend. For the front-end we used a well-known testing framework, jest & enzyme. We also used es-lint, which forces consistency of codes. Because of redux and other things that made our react containers deep, we had to use mock functions appropriately.

For Back-end, we used python unit-test as a testing framework for Django. And to make every code in our backend consistent, we also used py-lint. By using it, we could be able to force our code to be consistent in style. There were not many difficulties in testing the backend, compared to frontend testing, since Django views or models don't really have deep modeled structures.

It was not easy to ensure all frontend and backend unit tests over 80% every sprint. We employed Travis CI to enforce test coverage to be over 80%, and es-lint, py-lint. By employing it, every commit and pull request passes through CI so they are tested with several guidelines: es-lint, py-lint, front-end tests, backend-tests, and coverage report. The results of CI are reported onto both GitHub PR page and Slack chat room accordingly, once it is done on the travis CI server.

branch	Coveralls	Sonarcloud	Travis
master	coverage 94%	quality gate passed	build passing
dev	coverage 84%	quality gate passed	build passing

7.4. Deployment

We configured travis to access AWS and save built data into S3, then let the AWS codeDeploy service to launch our application in EC2.

(1) Before deployment, we build our application.

```
before_deploy:
  - |
    if [[ $TRAVIS_BRANCH != $TRAVIS_PULL_REQUEST_BRANCH && $TRAVIS_BRANCH = 'dev' ]]; then
      yarn build:dev
    else
      yarn build:prd
    fi
  - cp ../aws-codedeploy-scripts/frontend/* ./build
  - zip -r coding-mbti-webservice build/*
  - mkdir -p deploy
  - mv coding-mbti-webservice.zip deploy/coding-mbti-webservice.zip
```

(2) When the build process is done, we send our data to s3.

```
- provider: s3                          # frontend dev s3
  access_key_id: $AWS_ACCESS_KEY
  secret_access_key: $AWS_SECRET_KEY
  bucket: coding-mbti-1-deploy
  local_dir: deploy
  region: us-east-1
  skip_cleanup: true
  wait_until_deployed: true
  on:
    repo: swsnu/swpp2020-team16
    branch: dev
```

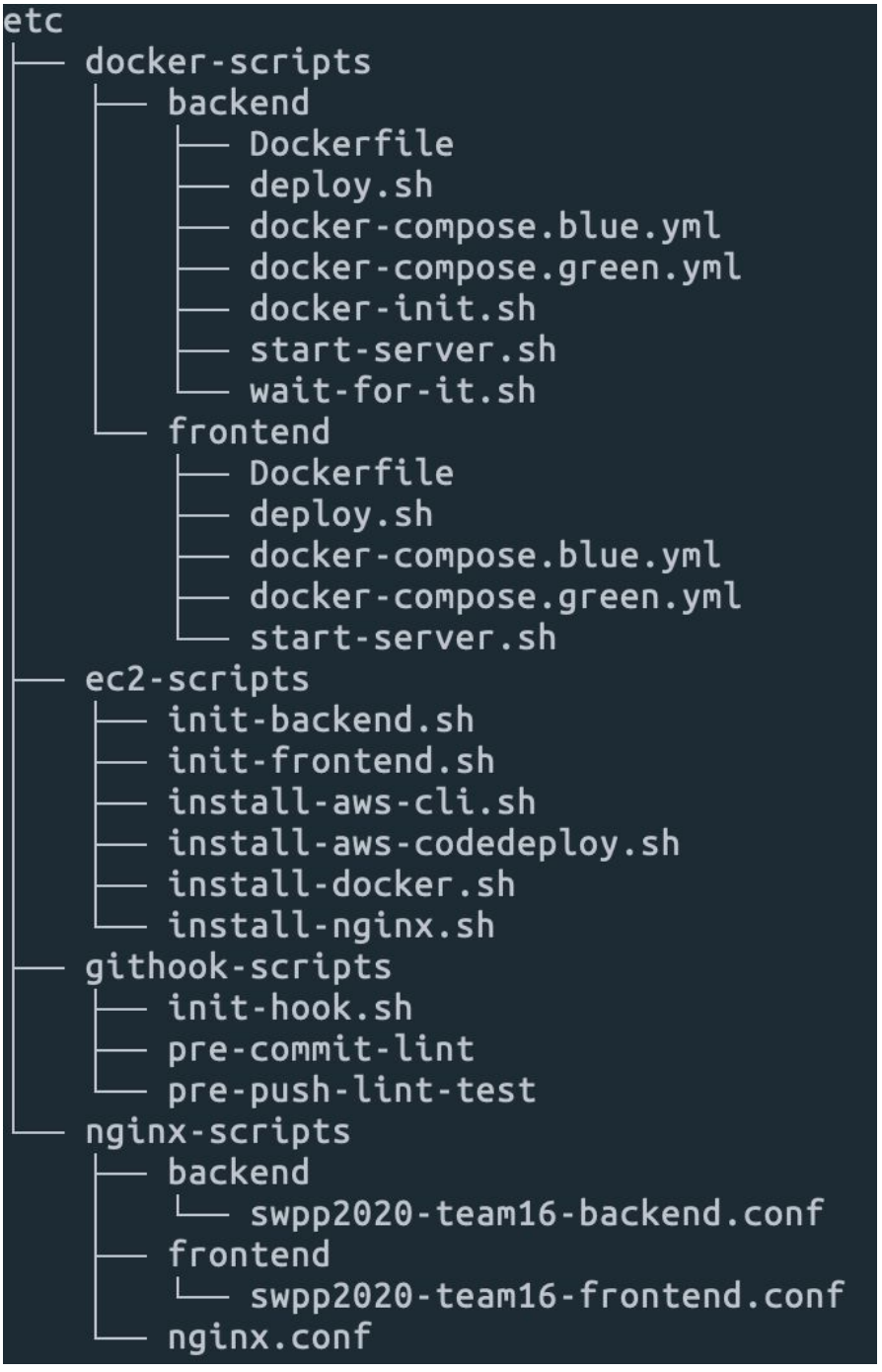
(3) After sending the built zip file to s3, travis notifies AWS codeDeploy service to catch the saved zip file in s3. codeDeploy service then fetch zipped built file and launches it to our ec2 server.

```
- provider: codedeploy                # frontend dev codedeploy
  access_key_id: $AWS_ACCESS_KEY
  secret_access_key: $AWS_SECRET_KEY
  bucket: coding-mbti-1-deploy
  key: coding-mbti-webservice.zip
  bundle_type: zip
  application: coding-mbti-1-deploy
  deployment_group: coding-mbti-1-deploy-group
  region: us-east-1
  wait_until_deployed: true
  on:
    repo: swsnu/swpp2020-team16
    branch: dev
```

As a result, developers in our team only had to care about the business code itself, but not about the deployment process.

Also, we managed all the ec2 server initialization scripts in shell script so as to save time to change or upgrade servers. In fact, on the last day of our project journey, we had to upgrade our server to handle concurrent requests. Though there was not enough time left, we could successfully change our server within 30 minutes. Compared with the time we spent on initial server settings, which took almost 3 days, saving all the scripts was effective.

We listed all the scripts related to ec2 as below. We only had to run ‘./ec2-scripts/init-backend.sh’, ‘./ec2-scripts/init-frontend.sh’ to configure a whole new server.

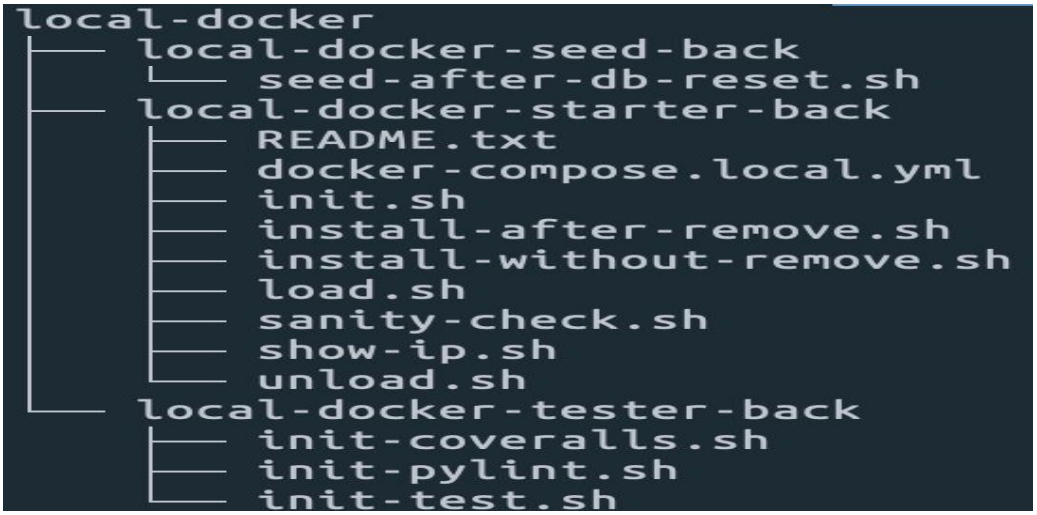


7.5. Dockerized application

We used docker service to resolve environment inconsistencies. As a result, the environment in each developer's local machine for running our application had become consistent. Also, we did not have to care about ec2 server's environment inconsistencies, because that would also be the same. We managed all the necessary commands in a shell script.

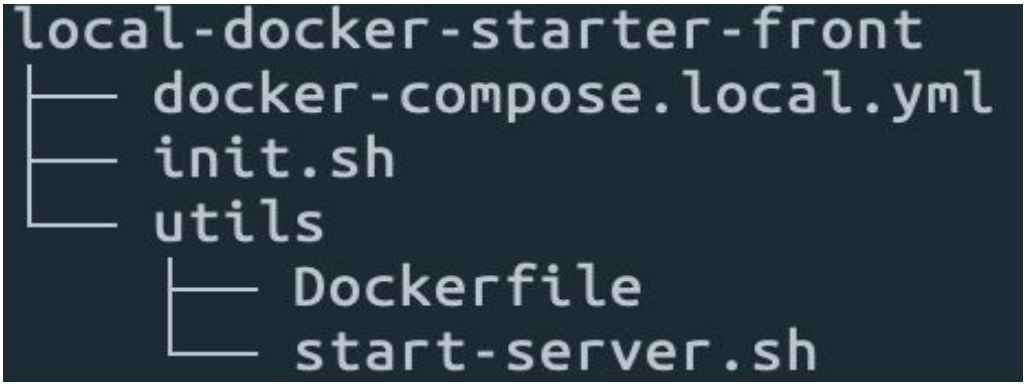
(1) dockerized backend

Initializing backend django application, testing backend django application, seeding backend postgresql were all managed in shell scripts. Any developers in our team could use this script so there was not much of a learning curve in utilizing docker service.



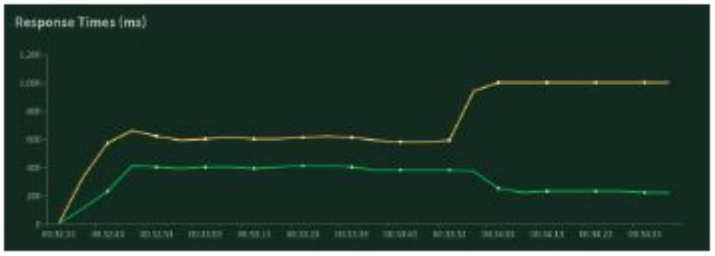
(2) dockerized frontend

For frontend, the same rule applies. We managed front app initialization with shell scripts.

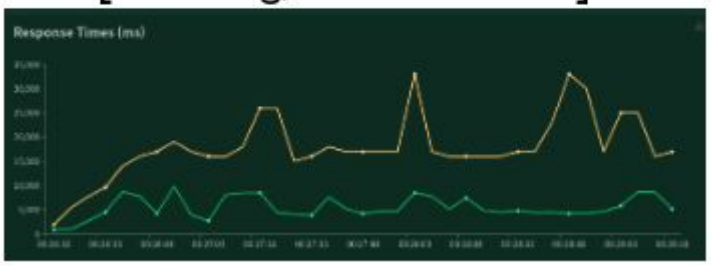


8. Performance

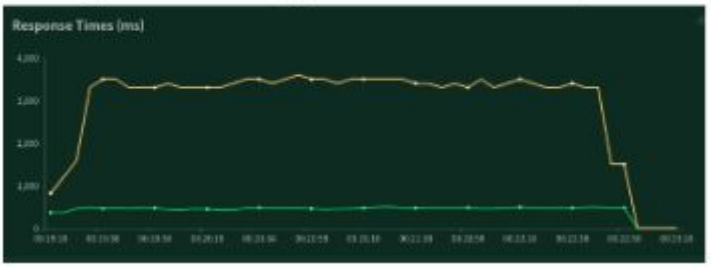
localhost [0.4 sec avg, 1.0 sec worst]



AWS before optimization
[6 sec avg, 35.0 sec worst]



AWS after optimization
[0.6 sec avg, 3.5 sec worst]



We apply several server backend optimization techniques. With locust, we set 100 users repeatedly sent requests for a global research report, coder's problem and solution requests in 25000 times. In local development environments, rapid response time is ensured but with aws, response time is 15 times worse in 6 seconds to 35 seconds in total. We make it change db connection code to directly apply SQL code and some other optimization and upgrade ec2 instances. So finally we ensure for the same test, 10 times faster responses.

9. What we've learned

Throughout this software development project, we learned and practiced both technical and conceptual parts of the software development process.

- Technical part of software development process

We learned tools that have been used a lot in the real world software development field, helping develop software. Such as React, Redux, Django, CSS, HTML, Docker, AWS and so on.

To be more specific :

We've learned how to use frameworks such as React, Redux ,Django, HTML and CSS basics. Based on the frameworks mentioned right before, we've also

learned how to make software look beautiful using MUI that someone had already made for that purpose. We also learned how to well integrate a Database(Especially PostgreSQL).

We've also learned what a test is and how to do a unit test, functional test. While doing a test, we were able to know what is mocking, why we need it and how to do it. During the entire development procedure, our development environment was constructed on the docker container, we could be more friendly with the docker environment and actually felt why it has been used by so many people.

We've learned what CI is, and how we put it into our software development project. By using es-lint, and py-lint we were able to understand why forcing consistent code writing style in the entire team is needed and it is indeed helpful.

And for deployment, we learned how to deploy software on AWS EC2 server. After deployment, we had to check our server's scalability and robustness, in that procedure we were able to learn what is load test and how to optimize our server and make it robust and scalable.

Not only that we also learned how to use some services that are provided by AWS such as SageMaker, and S3.

- Conceptual part of software development process

We actually practiced the main concepts of software development that we've learned during the class.

To be more specific :

We've learned how to make a decent team documents before actually starting a project. We also actually experienced how an Iterative process of software development(AGILE) works and experienced how it is like working as a team. We were also able to broaden our view about software, Bird's-eye view on service architecture. We could see how MVC(Model, View, and Controller) structure actually works. We experience what is SAAS(Software as a service) and how we achieve it successfully. We also saw how to integrate design patterns in the actual development, and how it made impact for efficiency.

10. Conclusion

Our project, Coding MBTI, is a web service which provides chances for coders to inspect their code, get insight on their coding habit and level compared to others and construct their personalized strategy to empower coding skills.

Our frontend design consists of component and service; each component provides a view of and control over a page, and services provide functionalities that multiple components share. Similarly, the backend design consists of model, service, and view; services provide functionalities for views. For implementation, we used frameworks like React, Redux and Django, jest and pytest for testing, aws ec2 and docker for deployment, and other tools to keep our code clean and consistent.

For future work, we are considering following things:

1) Add more code problems to provide more accurate and advanced analysis
2) Introduce more complex Algorithm & ML models using deep learning and NLP
3) Provide an API service instead of Web IDE for coders who is an expert
4) Add more various and useful functions for Managers and Researchers

Along the course and project, we could obtain techniques and concepts for software development. We could also have a small experiment on the impact of databases, achieving considerable improvement upgrading servers. Compared to the current problem-solving website, Coding MBTI is mostly differentiated in that it provides a more personalized report, and people connecting system.