

---

# 빌드 및 배포

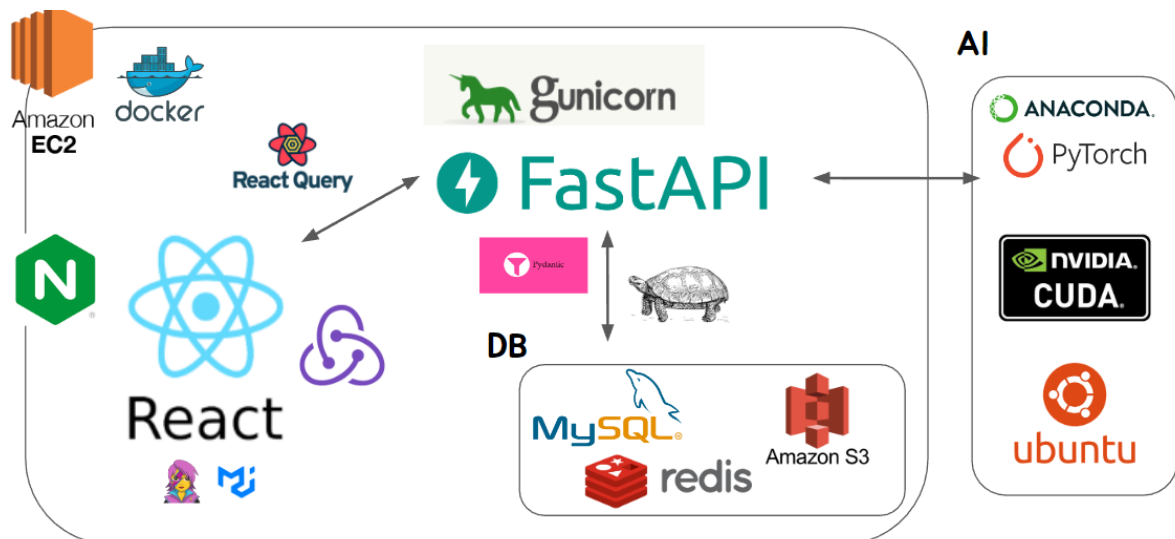
---



## 1. 기술 스택 및 버전

구분	사용 목적	사용 기술	기술 스택	버전
BackEnd	개발	Python		3.9
		Web framework	FastAPI	0.8
		DBMS	MySQL	8.0
			Redis	3.0
		ORM	Tortoise	0.19
		Cloud	S3	
FrontEnd	개발	Library	React	18.2
			React-query	3.39.2
			React-redux	8.0.2
			MUI	5.10
			emotion	11.10.4
AI	개발	Open source	Anaconda	3.0
		Library	Pytorch	1.12.1
			Scipy	1.9.1
			Speech_recognition	3.8.1
			Soundfile	0.10.3
			Numpy	1.23.3
			Librosa	0.9.2
Server	배포	Docker		3.3
		Web Server	Nginx	

## 2. 서비스 아키텍처



### 3. 빌드 및 배포 가이드

#### 1) Backend

##### (0) 디렉토리 구조

(/backend)

- /app
  - /enums
    - (not used)
  - /models : DB 모델 및 관련 메소드 정의
    - accounts.py
    - community.py
    - recipes.py
  - /routers : api 함수 정의
    - accounts.py
    - community.py
    - image.py
    - index.py
    - recipes.py
  - /schemas : pydantic schema 정의 (serializer)
    - accounts.py
    - common.py
    - community.py
    - recipes.py
  - /tests
    - (not used)
  - main.py
  - config.py
  - mail\_config.py
- /migrations
  - /b303
    - (.sql files)
- requirements.txt
- dockerfile

## (1) 주요 환경 변수

```
# Main DB(Mysql)
DB_URL={DB URL}
ROOT_PASSWORD={DB Root Password}
PYTHONUNBUFFERED=TRUE

# Sub DB(Redis)
REDIS_HOST={redis host url}
REDIS_PORT={redis port number}

# Mail config
MAIL_USERNAME={official mail name}
MAIL_PASSWORD={mail password}
MAIL_FROM={mail}
MAIL_PORT={mail port}
MAIL_SERVER={mail server, ex: naver.com}
MAIL_FROM_NAME={mail from name (project name)}

# S3
S3_BUCKET_NAME={S3 bucket name}
AWS_ACCESS_KEY_ID={aws access key id}
AWS_SECRET_ACCESS_KEY={aws secret access key}
REGION_NAME={region name}

# JWT
SECRET_KEY={jwt secret key}
ALGORITHM={encoding algorithm}
ACCESS_TOKEN_EXPIRE_MINUTES={expire minutes}
```

## (2) 도커라이징(빌드 및 배포 포함)

### ① 이미지 생성

- DB 이미지 생성(동일 경로에 .env 필요) 및 백그라운드 실행

```
docker-compose -f db-docker-compose.yaml --env-file ./env up -d
```

- Backend 이미지 생성(동일 경로에 Dockerfile 필요)
- 도커 허브를 이용할 경우, 이미지 이름을 {도커허브아이디}/{저장소이름}:{태그} 형식으로 맞춰줌

```
Docker build -t {image name} .
```

```
ex) docker build -t hamelin92/b303:ver .
```

## ② 도커 허브에 이미지 올리기

- 도커 로그인(터미널 또는 GUI에서)

```
Docker login
```

- 이미지 PUSH

```
docker push {도커허브아이디}/{저장소이름}:{태그}
```

```
ex) docker push hamelin92/b303:ver
```

## ③ 도커 허브에서 이미지 받아오기

- 이미지 PULL

```
docker pull {도커허브아이디}/{저장소이름}:{태그}
```

```
ex) docker pull hamelin92/b303:ver
```

## ④ 컨테이너 생성 및 실행

- FastAPI 이미지 기반 컨테이너 생성 및 실행

```
docker run --env-file ./env -d --name api -p 8000:8000 hamelin92/b303:ver
```

- -p: 포트 넘버 지정, --env-file: 환경변수 파일 지정, -d: 백그라운드 실행
- 위 명령어에서 api => 컨테이너 이름(임의로 지정)
- hamelin92/b303:ver => 이미지 이름

### (3) 도커 DB 마이그레이션

- tortoise\_orm을 데이터 베이스에 연결

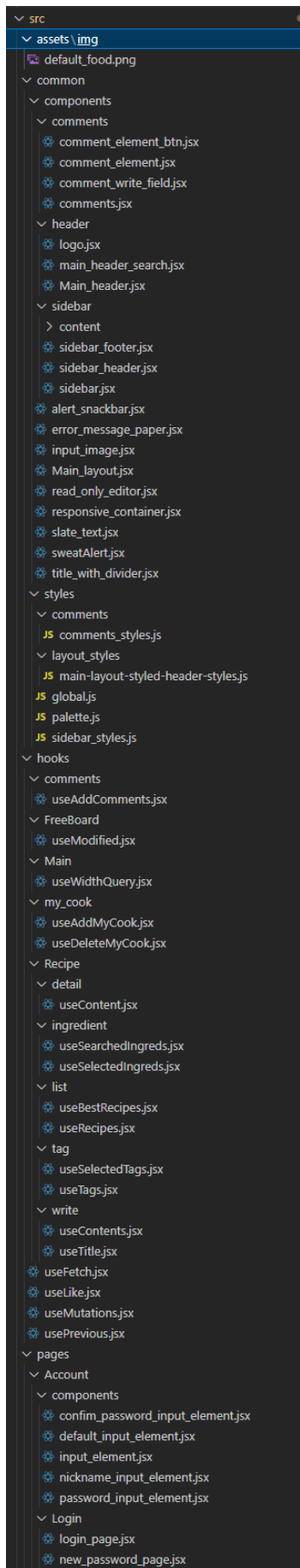
```
docker exec {백엔드컨테이너아이디} aerich init -t app.config.TORTOISE_ORM
```

- 마이그레이션의 업데이트 내용을 데이터베이스에 적용

```
docker exec {백엔드컨테이너아이디} aerich upgrade
```

## 2) Frontend

### (0) 디렉토리 구조

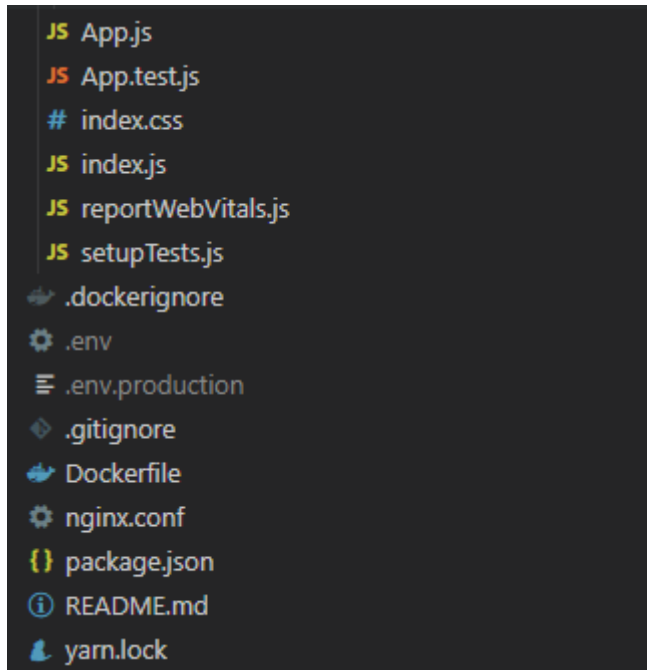


- ▼ Mypage
  - ▼ follow
    - 🔗 profile\_follow\_item.jsx
    - 🔗 profile\_follow.jsx
    - 🔗 profile\_follower\_list.jsx
    - 🔗 profile\_following\_list.jsx
  - ▼ gallery
    - 🔗 profile\_gallery\_mycook\_list.jsx
    - 🔗 profile\_gallery\_recipe\_list.jsx
    - 🔗 profile\_gallery.jsx
  - ▼ introduce
    - 🔗 profile\_introduce.jsx
  - ▼ modify
    - 🔗 profile\_modify\_introduce.jsx
    - 🔗 profile\_modify\_page.jsx
    - 🔗 profile\_page.jsx
  - ▼ Signup
    - 🔗 signup\_page.jsx
- ▼ community
  - ▼ FreeBoard
    - ▼ Detail
      - components
        - 🔗 free\_board\_detail\_page.jsx
    - ▼ List
      - ▼ components
        - ▼ item
          - 🔗 free\_board\_list\_item\_bottom\_info.jsx
          - 🔗 free\_board\_list\_item\_comment\_chip.jsx
          - 🔗 free\_board\_list\_item\_title.jsx
          - 🔗 free\_board\_list\_item.jsx
        - ▼ notice
          - 🔗 free\_board\_list\_notice\_chip.jsx
          - 🔗 free\_board\_list\_notices\_items.jsx
          - 🔗 free\_board\_list\_items\_container.jsx
          - 🔗 free\_board\_list\_pagination.jsx
      - 🔗 free\_board\_list\_page.jsx
    - ▼ Write
      - ▼ components
        - ▼ editor
          - 🔗 free\_board\_write\_editor\_container.jsx
          - 🔗 free\_board\_write\_editor\_toolbar.jsx
          - 🔗 free\_board\_write\_editor\_with\_image.jsx
          - 🔗 free\_board\_write\_admin\_chkbox.jsx
        - ▼ styles
          - # free\_board\_write\_editor.css
          - 🔗 free\_board\_write\_page.jsx
  - ▼ my\_cook
    - ▼ components
      - ▼ detail
        - 🔗 my\_cook\_detail\_page.jsx
      - ▼ list
        - 🔗 list.jsx
      - ▼ write
        - 🔗 recipe\_modal.jsx
        - 🔗 UploadImageArea.jsx
        - 🔗 write\_bottom\_bar.jsx
        - 🔗 write\_text.jsx
        - 🔗 write.jsx
      - ▼ styles
        - ▼ list
          - JS list\_style.js
        - ▼ write
          - JS write\_page\_styles.js
          - JS write\_styles.js
      - JS my\_cook\_container.js
  - ▼ Main
    - ▼ components
      - 🔗 search\_bar.jsx
      - 🔗 tag\_list.jsx
    - ▼ styles
      - JS search\_bar\_styles.js
      - JS tag\_list\_styles.js
      - JS main\_page\_styles.js
      - 🔗 main\_page.jsx
  - ▼ NotFound
    - 🔗 not\_found\_page.jsx



```

  Recipe
  List
  components
  bests
  recipe_list_bests_card.jsx
  recipe_list_bests.jsx
  filter
  ingredient
  tag
  recipe_list_filter_btn.jsx
  recipe_list_filter_container.jsx
  recipe_list_fab.jsx
  recipe_list_item_like_btn.jsx
  recipe_list_item.jsx
  recipe_list_loading_spinner.jsx
  recipe_list.jsx
  recipe_list_page.jsx
  recipe_detail
  components
  recipe_detail_content.jsx
  recipe_detail_help.jsx
  recipe_detail_ingredient_item.jsx
  recipe_detail_ingredient.jsx
  recipe_detail_like_btn.jsx
  recipe_detail_popover.jsx
  recipe_detail_title.jsx
  ai_area.jsx
  ai_content_wrapper.jsx
  ai_content.jsx
  ai_controller_area.jsx
  ai_listen_area.jsx
  ai_listen_recorder.jsx
  ai_listen.jsx
  ai_voice_request.jsx
  ai_voice_result.jsx
  ai_voice_timer.jsx
  recipe_detail_page.jsx
  styles
  JS recipe_ai_styles.js
  JS recipe_detail_styles.js
  Write
  components
  bottombar
  recipe_write_bottombar_btn.jsx
  recipe_write_bottombar.jsx
  content
  recipe_write_content_block.jsx
  recipe_write_content_img.jsx
  recipe_write_content_text.jsx
  ingredient
  recipe_write_ingredient_input.jsx
  recipe_write_ingredient_item.jsx
  recipe_write_ingredient_list.jsx
  title
  recipe_write_title_input.jsx
  recipe_write_add_content_bar.jsx
  recipe_write_box.jsx
  recipe_write_page.jsx
  store
  module
  JS accountReducer.js
  JS AiReducer.js
  JS recipeReducer.js
  JS index.js
  utils
  JS audio_listen_backup.js
  JS CustomConst.js
  JS http-commons.js
  JS http-multipart.js
  JS JWT-token.js
  JS regex.js
  JS voice-recognition.js
  M
```



## (1) 주요 환경변수 설정

```
// 프론트엔드 서버  
  
REACT_APP_URL=http://localhost:3000  
  
// 백엔드 서버  
  
REACT_APP_BACKEND_URL=https://j7b303.p.ssafy.io/api
```

## (2) 도커라이징(빌드 및 배포 포함)

### ① 도커 이미지 빌드 준비

#### ● 도커파일 생성

```
# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.  
  
FROM nginx  
  
# root 에 app 폴더를 생성  
  
RUN mkdir /app  
  
# work dir 고정  
  
WORKDIR /app
```

```
# work dir 에 build 폴더 생성 /app/build

RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사

ADD ./build ./build

# nginx 의 default.conf 를 삭제

RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사

COPY ./nginx.conf /etc/nginx/conf.d

# 도커에서 80 포트 오픈

EXPOSE 80

# container 실행 시 자동으로 실행할 command. nginx 시작함

CMD ["nginx", "-g", "daemon off;"]
```

- **nginx 설정: nginx.conf 파일 생성**

```
server {

    listen 80;

    location / {

        root    /app/build;

        index   index.html;

        try_files $uri $uri/ /index.html;

    }

}
```

- **프론트엔드 모듈 파일설치**

```
- yarn install
```

- **빌드 파일 생성**

```
- yarn build
```

## ② 도커 이미지 빌드

- **docker 이미지 생성**

```
- docker build -t [도커허브아이디]/[도커저장소이름]:[이미지 태그명] .
```

- **docker 로그인**

```
- docker login
```

- **docker hub에 생성한 이미지 push**

```
-docker push [도커허브아이디]/[도커저장소이름]:[이미지 태그명]
```

## ③ 도커 허브에서 이미지 받아오기

- **docker 이미지 pull**

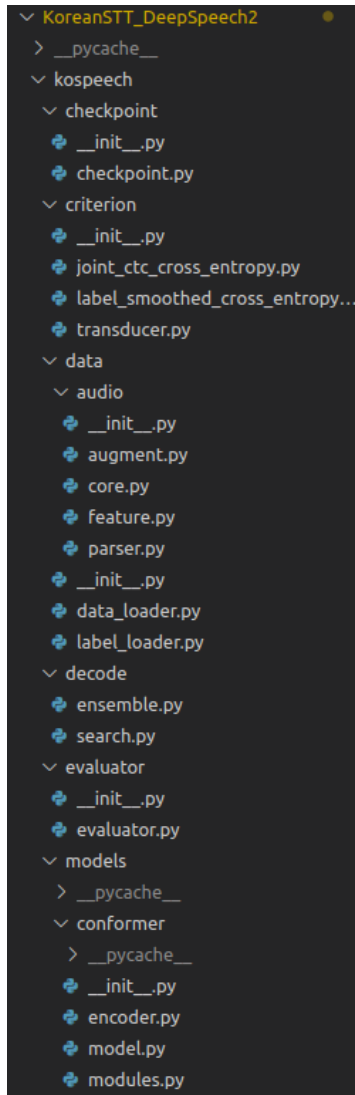
```
- docker pull [도커허브아이디]/[도커저장소이름]:[이미지 태그명]
```

## ④ 도커 이미지 컨테이너 실행

```
- docker run --name atti-front -d -p 3000:80 [도커허브이아이디]/[도커저장소이름]: [이미지 태그명]
```

### 3) AI

#### (0) 디렉토리 구조

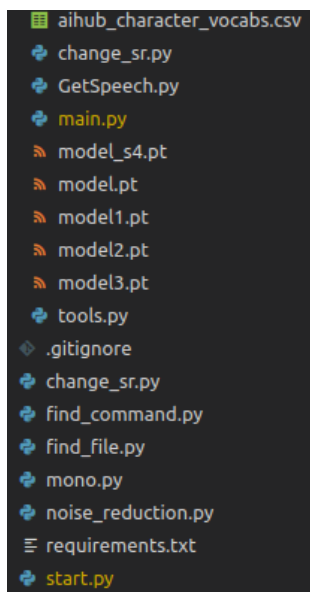
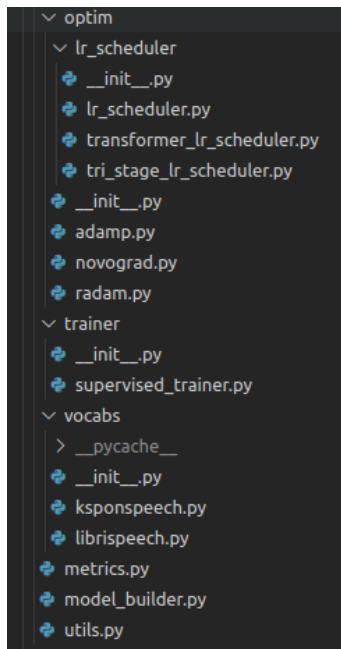


```

KoreanSTT_DeepSpeech2
├── __pycache__
├── kospeech
│   ├── checkpoint
│   │   ├── __init__.py
│   │   └── checkpoint.py
│   ├── criterion
│   │   ├── __init__.py
│   │   ├── joint_ctc_cross_entropy.py
│   │   ├── label_smoothed_cross_entropy.py
│   │   └── transducer.py
│   ├── data
│   │   ├── audio
│   │   │   ├── __init__.py
│   │   │   ├── augment.py
│   │   │   ├── core.py
│   │   │   ├── feature.py
│   │   │   ├── parser.py
│   │   │   ├── __init__.py
│   │   │   ├── data_loader.py
│   │   │   └── label_loader.py
│   │   ├── decode
│   │   │   ├── ensemble.py
│   │   │   └── search.py
│   │   ├── evaluator
│   │   │   ├── __init__.py
│   │   │   └── evaluator.py
│   │   └── models
│   │       ├── __pycache__
│   │       └── conformer
│   │           ├── __pycache__
│   │           ├── __init__.py
│   │           ├── encoder.py
│   │           ├── model.py
│   │           └── modules.py

```

- ▼ deepspeech2
  - > \_\_pycache\_\_
  - 🔗 \_\_init\_\_.py
  - 🔗 model.py
- ▼ jasper
  - > \_\_pycache\_\_
  - 🔗 \_\_init\_\_.py
  - 🔗 configs.py
  - 🔗 model.py
  - 🔗 sublayers.py
- ▼ las
  - > \_\_pycache\_\_
  - 🔗 \_\_init\_\_.py
  - 🔗 decoder.py
  - 🔗 encoder.py
  - 🔗 model.py
- ▼ rnnt
  - > \_\_pycache\_\_
  - 🔗 \_\_init\_\_.py
  - 🔗 decoder.py
  - 🔗 encoder.py
  - 🔗 model.py
- ▼ transformer
  - > \_\_pycache\_\_
  - 🔗 \_\_init\_\_.py
  - 🔗 decoder.py
  - 🔗 embeddings.py
  - 🔗 encoder.py
  - 🔗 mask.py
  - 🔗 model.py
  - 🔗 sublayers.py
  - 🔗 \_\_init\_\_.py
  - 🔗 activation.py
  - 🔗 attention.py
  - 🔗 beam\_search.py
  - 🔗 convolution.py
  - 🔗 decoder.py
  - 🔗 encoder.py
  - 🔗 model.py
  - 🔗 modules.py



## (1) 서버 세팅

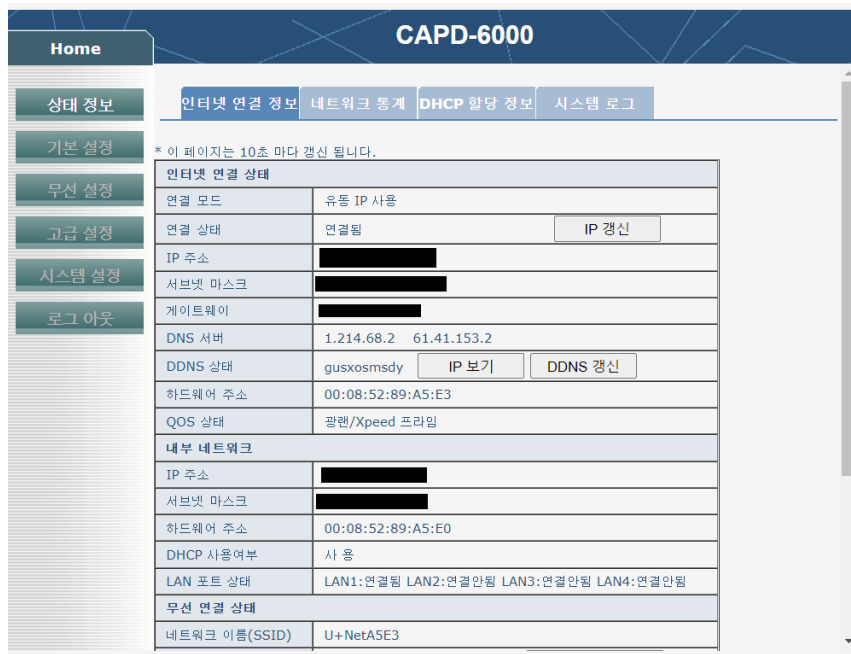
### ① Ubuntu(Linux) 설치

- USB에 Ubuntu 설치 파일 받아 설치 준비
- 기존 Windows를 삭제하면서 위에서 만든 USB를 이용하여 Ubuntu 설치

## ② 포트포워딩

공유기를 사용하고 있을 경우 필요한 작업

- 현재 자신이 사용하고 있는 공유기 확인
- 공유기 별 관리자 페이지 접속 후 포트포워딩 설정 접근  
(현 AI 서버의 경우 LG 공유기 사용)



The screenshot shows the 'CAPD-6000' web interface, specifically the '인터넷 연결 정보' (Internet Connection Information) tab. The left sidebar contains navigation links: Home, 상태 정보 (Status Info), 기본 설정 (Basic Settings), 무선 설정 (Wireless Settings), 고급 설정 (Advanced Settings), 시스템 설정 (System Settings), and 로그 아웃 (Logout). The main content area displays various network settings. A note at the top states: '\* 이 페이지는 10초 마다 갱신됩니다.' (This page is updated every 10 seconds).

인터넷 연결 상태	
연결 모드	유동 IP 사용
연결 상태	연결됨 <span>IP 갱신</span>
IP 주소	[Redacted]
서브넷 마스크	[Redacted]
게이트웨이	[Redacted]
DNS 서버	1.214.68.2 61.41.153.2
DDNS 상태	gusxosmsdy <span>IP 보기</span> <span>DDNS 갱신</span>
하드웨어 주소	00:08:52:89:A5:E3
QOS 상태	광랜/Xspeed 프라임
내부 네트워크	
IP 주소	[Redacted]
서브넷 마스크	[Redacted]
하드웨어 주소	00:08:52:89:A5:E0
DHCP 사용여부	사용
LAN 포트 상태	LAN1:연결됨 LAN2:연결안됨 LAN3:연결안됨 LAN4:연결안됨
무선 연결 상태	
네트워크 이름(SSID)	U+NetA5E3

- 서버에서 사용하고자 하는 포트 번호를 포트포워딩



The screenshot shows the 'CAPD-6000' web interface, specifically the '포트 포워딩' (Port Forwarding) tab under the 'NAT 설정' (NAT Settings) section. The left sidebar is the same as the previous screenshot. The main content area has sub-tabs: 포트 포워딩, DMZ 서버, 비정규 FTP, and NAT-T. The '포트 포워딩' sub-tab is active, showing a form to add a new port forwarding rule. Below the form is a table of existing rules.

\*최대 입력값은 32개 입니다.

No.	ON/OFF	프로토콜	시작 포트	마지막 포트	내부 IP 주소	내부 포트
1	On	TCP	20022	20022	[Redacted]	22
2	On	TCP	38000	38000	[Redacted]	8000

Buttons at the bottom: 적용 (Apply), ON/OFF, 삭제 (Delete), 수정 (Edit).



### ③ FastAPI를 이용한 코드 작성

```
from fastapi import FastAPI, Request, UploadFile, File
from fastapi.middleware.cors import CORSMiddleware
```

```
app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"]
)

@app.get('/')
async def index():
    return {"message": "Hello Test World"}

@app.post('/stt')
def result(request: Request, file: UploadFile = File(...)):
    start = time.time()
    path = "."
    new_file = open(f'{path}/{file.filename}', 'wb+')
    new_file.write(file.file.read())
    new_file.close()
    ...
    ...
```

### ④ Uvicorn을 이용한 서버 개설

- 서버 실행 명령어 입력

```
uvicorn start:app --host [ip주소] --port [허용할 포트번호]
```