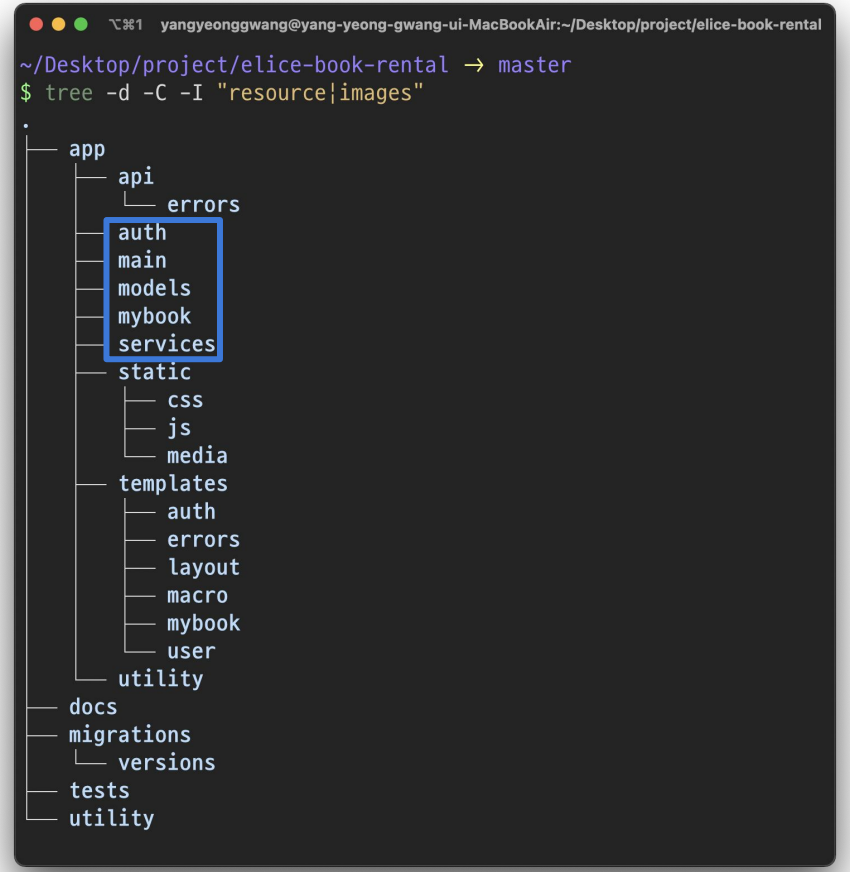
엘리스 개인 프로젝트

발표자 양영광

목차

1. 프로젝트 수행 과정

1. 프로젝트 설계
2. 개발 환경 설정
3. 코드 구현하기
4. 배포



설계

view —— route 를 처리

auth (로그인)
main (책)
mybook (내가 빌린 책)

services —— models를 이용해 DB 데이터를 CRUD 하는 로직

models —— 데이터베이스 모델

서비스 코드

```python
class BookService(object):
    @staticmethod
    def get_books(
        current_page: int, book_per_page: int, sort: str = BOOK_SORT_POPULARITY
    ) -> Pagination:

        query = BookService._book_sort(sort)

        pagination = query.paginate(current_page, book_per_page, error_out=False)
        return pagination

    @staticmethod
    def search_query(
        keyword: str,
        current_page: int,
        book_per_page: int,
        sort: str = BOOK_SORT_POPULARITY,
    ):
        query = BookService._book_sort(sort).filter(
            Book.book_name.like(f"%{keyword}%") | Book.description.like(f"%{keyword}%")
        )

        pagination = query.paginate(current_page, book_per_page, error_out=False)
        return pagination
    You, a day ago • refactor: 중복코드 메소드 추출, 검색기능 구현
    @staticmethod
    def _book_sort(sort):
        if sort not in sort_available:
            sort = DEFAULT_SORT

        if sort == BOOK_SORT_POPULARITY:
            query = (
                db.session.query(Book)
                .outerjoin(Review, Review.book_id == Book.id)
                .group_by(Book.book_name, Book.id)
                .order_by(desc(func.avg(Review.score)), desc(func.count(Book.review)))
            )
        elif sort == BOOK_SORT_REVIEW:
            query = (
```

view(route) 코드

```python
@main.route("/")
def index():
    query = request.args.get("q", default="", type=str)
    page = request.args.get("page", default=1, type=int)
    sort = request.args.get("sort", default=DEFAULT_SORT, type=str)
    style = request.args.get("style", default="list", type=str)

    book_per_page = current_app.config["BOOK_PER_PAGE"]

    if query:
        pagination = BookService.search_query(query, page, book_per_page, sort)
    else:
        pagination = BookService.get_books(page, book_per_page, sort)

    books = pagination.items

    return render_template(
        "book_list_line.html" if style == "list" else "book_list.html",
        query=query,
        style=style,
        sort=sort,
        book_list=books,
        pagination=pagination,
        enumerate=enumerate,
        get_score=BookService.get_score,
    )
```

model 코드

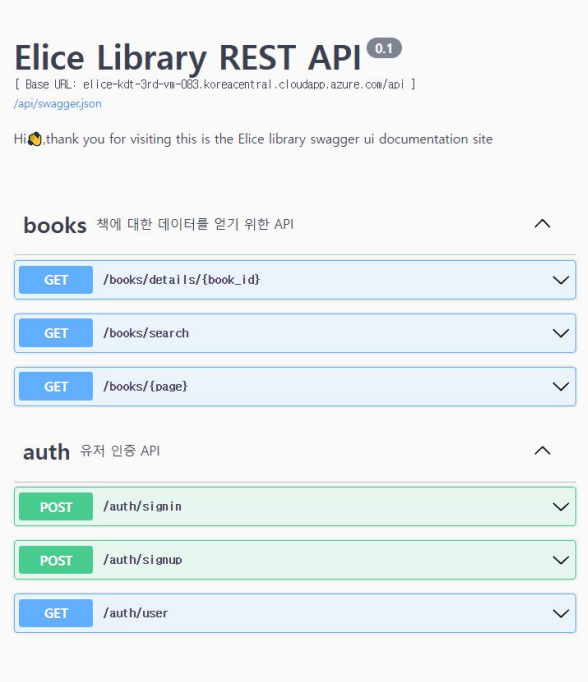
```python
class Book(db.Model):
    """책 Model"""

    __tablename__ = "books"

    id = db.Column(db.Integer, primary_key=True)
    book_name = db.Column(db.String(100), nullable=False)
    publisher = db.Column(db.String(64), nullable=False)
    author = db.Column(db.String(64), nullable=False)
    publication_date = db.Column(db.DateTime, nullable=False)
    pages = db.Column(db.Integer, nullable=False)
    isbn = db.Column(db.BigInteger, nullable=False)
    description = db.Column(db.Text, nullable=False)
    viewer = db.Column(db.Integer, default=0)  # 조회수
    link = db.Column(db.Text, nullable=False)
    image_url = db.Column(db.String(150), nullable=False)
    stock = db.Column(db.Integer, default=10)

    rental = db.relationship(Rental, backref="book")
    review = db.relationship(Review, backref="book")

    def to_dict(self):
```

REST API

REST API 코드

```python
from .response import Response

book_api = Namespace("books", description="책에 대한 데이터를 얻기 위한 API")

parser = book_api.parser()
parser.add_argument(
    "per_page",
    type=int,
    help="페이지당 반환할 책 데이터 갯수를 설정합니다.",
    location="args",
)
parser.add_argument(
    "sort",
    type=str,
    choices=["popularity", "review", "visit"],
    help="정렬 기준을 설정합니다. popularity(인기순) review(리뷰많은순) visit(많이찾은순)",
    location="args",
)


# book list
You, a day ago | 1 author (You)
@book_api.route("/<int:page>")
@book_api.doc(
    description="책 리스트를 반환합니다.",
    params={"page": "탐색할 페이지"},
    responses={200: "데이터 반환에 성공한 경우"},
)
class Books(Resource):
    @book_api.expect(parser)
    def get(self, page=1):
        # sort
        book_per_page = request.args.get(
            "per_page", default=current_app.config["BOOK_PER_PAGE"], type=int
        )
        sort = request.args.get("sort", default="popularity", type=str)

        pagination = BookService.get_books(page, book_per_page, sort)
        book_items = list(map(lambda x: x.to_dict(), pagination.items))
        return Response.make_response(
            {
                "count": len(book_items),
                "current_page": pagination.page,
                "last_page": pagination.pages,
                "has_next": pagination.has_next,
                "has_prev": pagination.has_prev,
                "books": book_items,
            }
        )
```

rest api swagger



라이브 데모

3. 배운 점

데이터베이스 테스트

설계 REST API

개발환경 배포환경

테스트

코드 각각의 부분(함수) 등

정확하게 동작하는지 확인한다.

코드를 수정 할 일이 생긴다면

기존 코드와 동일하게

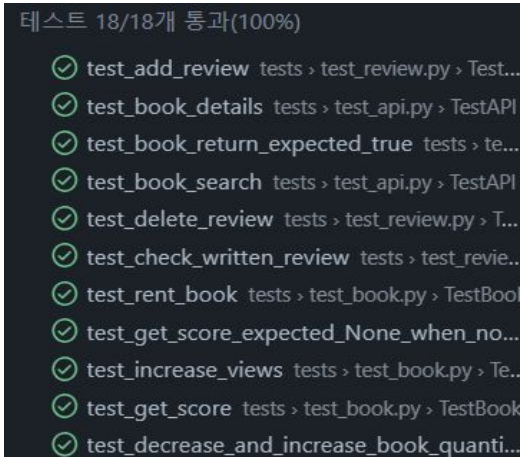
작동하는지 검증을 도와주므로

디버깅 시간을 단축할 수 있다

```python
def test_zz(self) -> None:
    EXPECTED = 3
    ANSWER = 4
    self.assertEqual(EXPECTED, ANSWER)
```

결과 3을 예상했지만
실제 결과는 4이므로 테스트는 실패

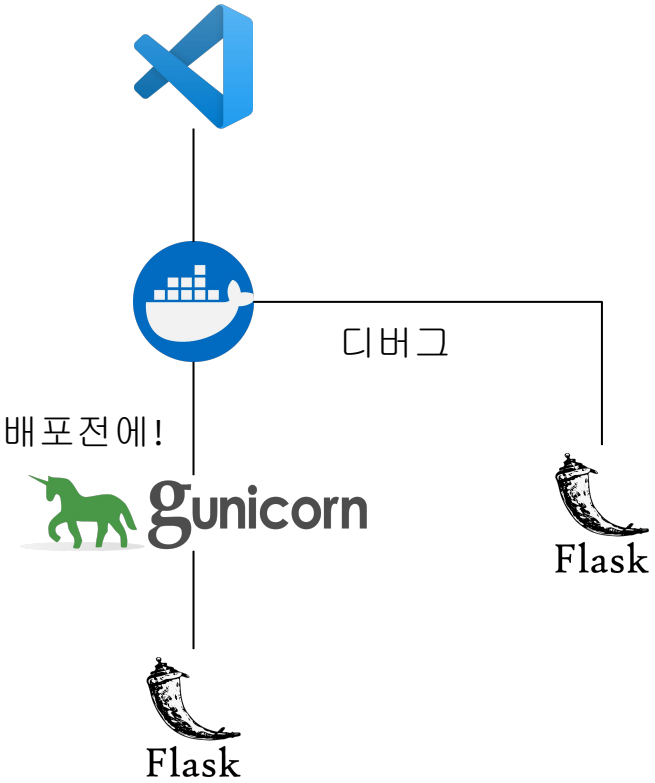
테스트 실패



테스트 성공

개발환경 배포환경

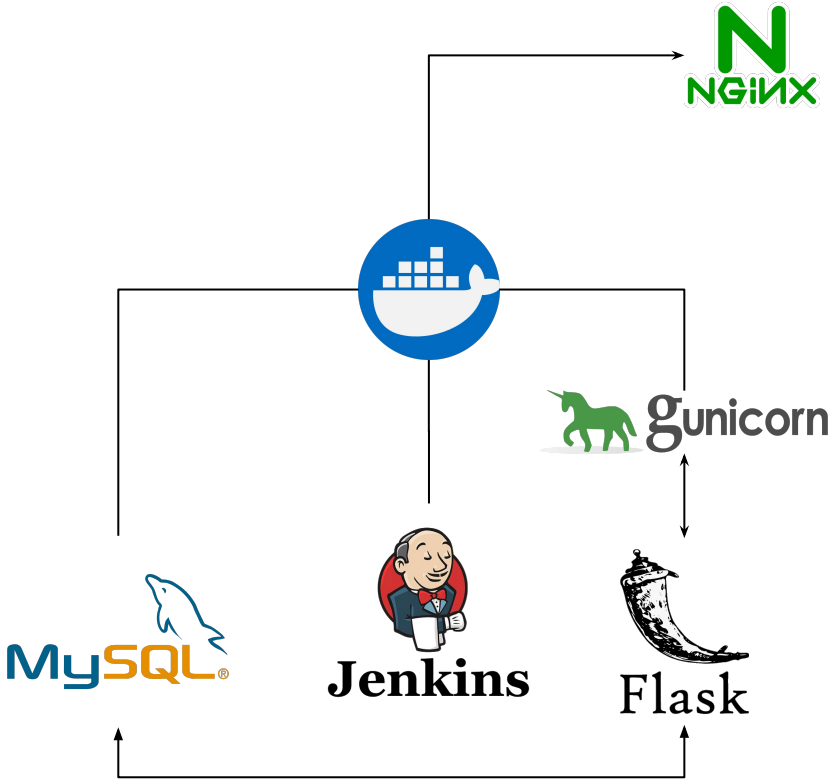
개발환경



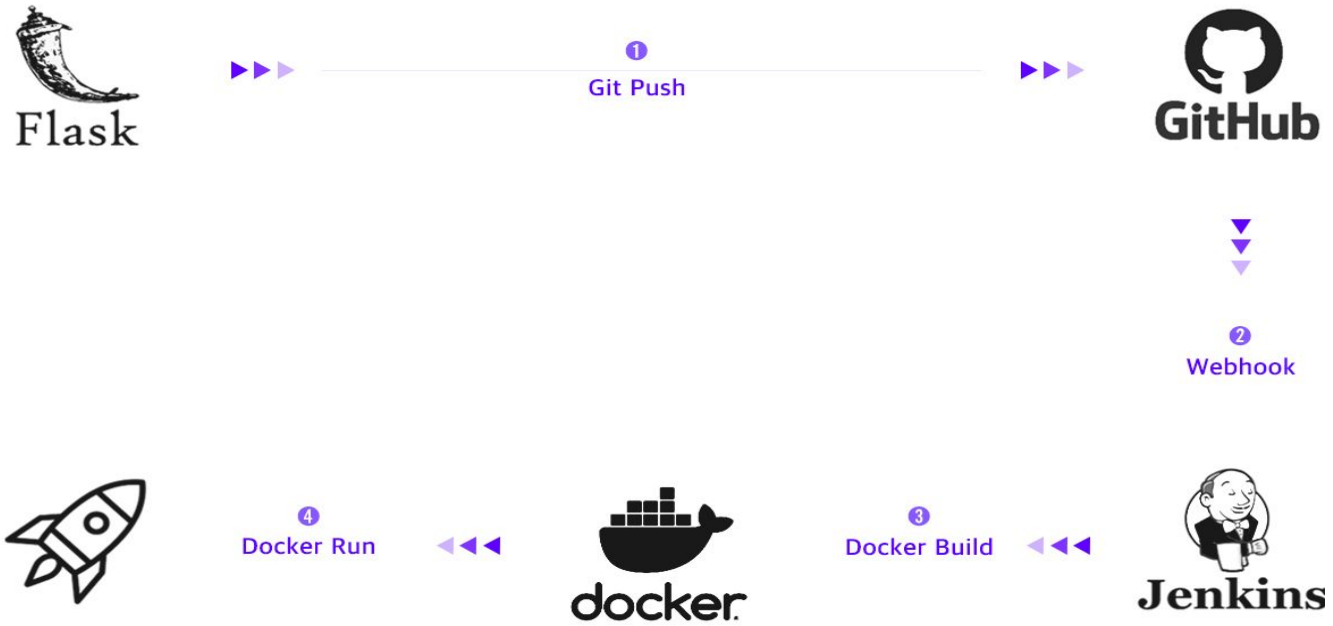
배포 환경

Docker 를 사용해서
개발 환경과 배포환경을
비슷하게 만들 수 있었다.

Jenkins 를 사용하여
git 저장소에 코드가 push 되면
Docker 빌드및 실행 자동화



배포 workflow


Flask
1
Git Push
GitHub
2
Webhook
Jenkins
3
Docker Build
docker
4
Docker Run

4. 어려웠던 점

개발환경과 배포 환경을 동일하게 맞추는게 힘들었다.
-> 로컬에서 작동하는 코드가 원격에서 잘 작동된다는 보장이 없다.

SQLAlchemy ORM이 만능인줄 알았다.
-> ORM을 사용해도 각 DBMS (MySQL, SQLite) 의 특성을 잘 파악해야할 거같다..
같은 ORM 코드라도 SQL구문은 DB 마다 해석이 다르므로
ORM이 항상 동일한 작동을 보장하지 않는 거 같다.

템플릿 코드 작성이 어려웠다..
-> 백엔드 개발자던 프론트엔드 개발자던 HTML CSS JS 는 기본이 될거같아요

새로운 기술,라이브러리를 사용하면서 이게 맞나?? 싶은 부분이 많았다.

5. 나만의 팁

도커를 사용해보자!

개발 환경과 배포환경의 차이를 줄여준다

배포 단계에서 흔히 겪는 **로컬 개발환경**과의

차이에서 생긴 문제를 줄여준다.

라이브러리를 잘 쓰자..!

직접 구현하는것도 좋지만 내가 생각한 로직은

이미 라이브러리로 구현되어 있었다! 가져다 쓰면되고

개발 시간을 그 만큼 줄여준다.

Q & A