САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная Работа №3

Выполнил:

Кобелев Л.К.

K33401

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Необходимо реализовать отдельный микросервис, выполняющий какую-либо содержательную функцию из всего арсенала функций вашего приложения.

Ход работы

Предположим, что мы делаем некий сервис, в котором есть основной бэк-сервис и `ML`-сервис. Примером может выступить сервис для рекомендаций какого-то контента. В таком случае основной бэк отвечает за выдачу различного контента / фильтрацию / поиск и так далее, а ещё один за обращение к `ML` модели. Очевидно, что `ML` модели сейчас чаще всего пишутся на `python`, а бэк к ним на `FastAPI`.

Базовый `API` модели:

```python
@app.post("/recommendations")
async def get_recommendations(
    request: Request,
    game_id: int = 0,
    liked: int = 1,
    top: int = 10,
    offset: int = 0,
):
    """Get new recommendations based on a current game"""
    request = await request.json()
    predicted_vector = request["vector"]
    content = model.predict(predicted_vector, game_id, liked, top, offset)
    return JSONResponse(content=jsonable_encoder(content))


@app.get("/recommendations")
async def set_selected_games(request: Request, games_ids: List[int] = Query(None)):
    """Set selected games"""
    vector = model.set_initial_vector(games_ids)
    content = {"vector": vector}
    return JSONResponse(content=jsonable_encoder(content))
```

На основном бэке заведем специальные роуты и контроллер.

`RecommendationRoutes`:

```
import express from 'express'
import RecommendationController from '../controllers/RecommendationController'

const router: express.Router = express.Router()

const recommendationsController = new RecommendationController()

router.route( prefix: '/').get(recommendationsController.getRecommendations)

router.route( prefix: '/').post(recommendationsController.setRecommendations)

export default router
```

`RecommendationController`:

```
class RecommendationController {
    mlUrl: string = String(process.env.ML_HOST)

    setRecommendations = async (request: Request, response: Response) => {
        const mobile = request.query.mobile || false

        const params = new URLSearchParams()
        request.body.games_ids.forEach(function (game_id) {
            params.append('games_ids', game_id)
        })

        const recommendations = await fetch(
            url: this.mlUrl + 'recommendations?' + params,
            init: {
                method: 'GET',
                headers: {
                    Accept: 'application/json',
                    'Content-Type': 'application/json',
                },
            }
        )

        const recommendationsData = await recommendations.json()

        if (mobile == false) {
            const vector = recommendationsData['vector']
            response.cookie( name: 'vector', vector)
            response.cookie( name: 'recommendedGames', request.body['games_ids'])
        }
        return response.send(recommendationsData)
    }
```

Таким образом, запрос к модели идет через открытый API (считаем, что API модели скрыт через nginx и обратиться к нему можно только на локальном уровне, например, в рамках компоуза), который в свою очередь может как-то дополнительно обработать входные / выходные данные. В примере выше, например, работаем с `cookies`. `ML Host`, конечно же, удобнее всего обозначить в `.env`.

Вывод

Изучена базовая архитектура микросервисных приложений.