ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO ĐỒ ÁN MÔN HỌC

MÔN HỌC: Đồ Án Tổng Hợp - Công nghệ Phần mềm

Đề tài: Grade Portal for students at HCMUT

HK241

Giảng viên hướng dẫn: Lê Đình Thuận
Lớp, nhóm hiện thực: L07 - Nhóm thầy Thuận
Sinh viên: Trần Đại Việt - 2213951
Phạm Văn Quốc Việt - 2213950
Nguyễn Nhật Khoa - 2211629
Phạm Việt Anh - 2210128
Nguyễn Gia Nguyên - 2212303
Lê Đăng Khoa - 2211599

HO CHI MINH CITY, SEPTEMBER 2024

Mục lục

1 Danh Sách Thành Viên Và Phân Công Nhiệm Vụ

MSSV	Họ và tên	Nhiệm vụ	Phần trăm công việc
2213951	Trần Đại Việt	PO	100%
2213950	Phạm Văn Quốc Việt	DEV	100%
2211629	Nguyễn Nhật Khoa	DEV	100%
2210128	Phạm Viết Anh	DEV	100%
2212303	Nguyễn Gia Nguyên	DEV	100%
2211599	Lê Đăng Khoa	DEV	100%

Bảng 1: *Danh sách sinh viên và nhiệm vụ*

2 Quản lý phiên bản (Version Control)

2.1 Lý Do Lựa Chọn GitHub

GitHub là một trong những nền tảng quản lý mã nguồn phổ biến nhất hiện nay, với nhiều tính năng mạnh mẽ, dễ sử dụng và tích hợp sẵn với các công cụ phát triển phần mềm. Các lý do chính khiến nhóm lựa chọn GitHub bao gồm:

- **Dễ dàng sử dụng**: GitHub cung cấp giao diện người dùng thân thiện, giúp cho việc quản lý và chia sẻ mã nguồn trở nên dễ dàng.
- **Quản lý phiên bản mạnh mẽ**: GitHub hỗ trợ Git, một công cụ quản lý phiên bản phân tán rất mạnh mẽ, giúp theo dõi và quản lý thay đổi mã nguồn theo thời gian.
- **Chia sẻ và cộng tác**: GitHub cho phép các thành viên trong nhóm cộng tác dễ dàng, với các tính năng như Pull Requests và Issues giúp theo dõi tiến độ và xử lý vấn đề.
- **Tính bảo mật và tính công khai**: GitHub cho phép tạo repository công khai hoặc riêng tư, đảm bảo tính bảo mật khi cần thiết và dễ dàng chia sẻ mã nguồn với cộng đồng khi sử dụng repository công khai.

Việc sử dụng GitHub giúp nhóm quản lý mã nguồn, tài liệu và các thay đổi một cách hiệu quả và dễ dàng. Các tính năng của GitHub không chỉ hỗ trợ trong việc cộng tác mà còn đảm bảo tính minh bạch và theo dõi tiến độ phát triển dự án một cách rõ ràng. Chúng em sẽ tiếp tục duy trì và cập nhật các repository này để đảm bảo dự án được phát triển liên tục và dễ dàng quản lý.

Dưới đây là đường dẫn đến repository của dự án:

- Repository dẫn tới Server của dự án (bao gồm mã nguồn và các tài liệu liên quan tới Backend dự án): `https://github.com/dath-241/grade-portal-be-java`

2.2 Vấn đề về việc mã hóa file `.env` khi đưa lên repo dự án

Trong các dự án phần mềm, file `.env` chứa các biến môi trường quan trọng như thông tin cấu hình hệ thống, khóa bảo mật, hoặc đường dẫn kết nối đến cơ sở dữ liệu. Việc để lộ file `.env` lên repository có thể dẫn đến rủi ro bảo mật nghiêm trọng. Để giải quyết vấn đề này, các công cụ mã hóa như **SOPS** kết hợp với **Age** thường được sử dụng để mã hóa các file chứa thông tin nhạy cảm.

Trường hợp bạn được cung cấp file khóa `key.txt`

Nếu bạn được cung cấp khóa Age trong file `key.txt`, cấu trúc file sẽ có dạng như sau:

```
# created: 2024-10-19T18:05:45+07:00
# public key: age1l4zc9ppdyvz6hvwj67uca0pfgyydm9efs7kgmmks3h9pz7xa9v3scrl3sn
AGE-SECRET-KEY-1T4Y4TH7S8GMAUW9J7SFMP6YXRG2FF0UFX08MSPWS540CA20NL40QVJ60VS
```

Thực hiện các bước sau để giải mã file chứa biến môi trường bị mã hóa (`dev.enc` hoặc `prod.enc`) và tạo file `.env`:

1. Đặt file `key.txt` vào thư mục `gradeportal`.

2. Chạy lệnh để giải mã file `dev.enc`:

```
sops --age key.txt --decrypt dev.enc > .env
```

3. Trong trường hợp máy của bạn gặp lỗi khi sử dụng lệnh trên, bạn cần thiết lập biến môi trường Age Key trực tiếp như sau:

```
set SOPS_AGE_KEY=AGE-SECRET-KEY-1T4Y4TH7S8GMAUW9J7SFMP6YXRG2FF0UFX08MSPWS540CA20NL40QVJ60
```

4. Sau đó, giải mã file `dev.enc` hoặc `prod.enc` bằng lệnh:

```
sops --decrypt --input-type dotenv --output-type dotenv dev.enc > .env
```

Trường hợp bạn không được cung cấp file khóa `key.txt`

Nếu bạn không có khóa trong file `key.txt`, bạn có thể tự tạo file `.env` như sau:

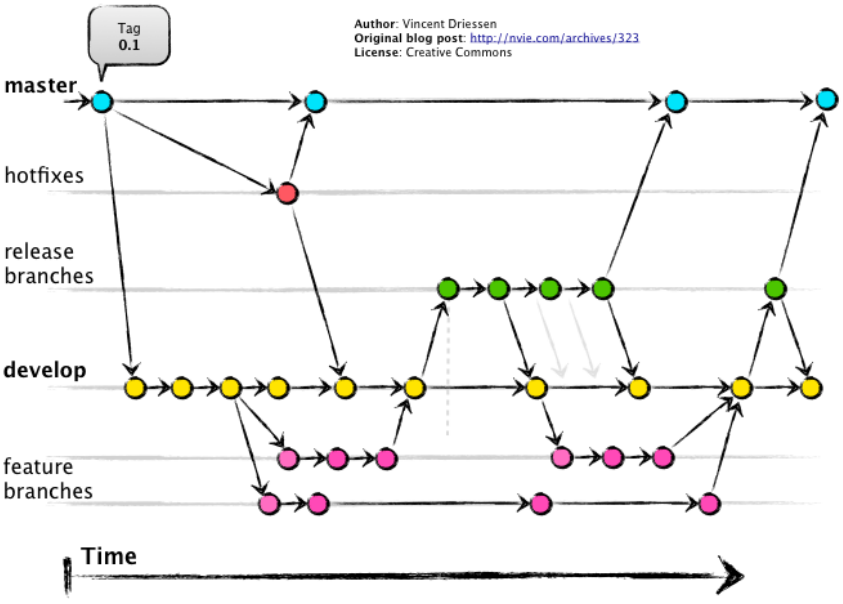
- Tạo file `.env` trong thư mục `gradeportal`.
- Điền các đường dẫn tài nguyên và thông tin cần thiết dựa trên file `example.env` được cung cấp sẵn.

Lợi ích của việc mã hóa file `.env`

- Bảo vệ thông tin nhạy cảm như khóa API, mật khẩu cơ sở dữ liệu, và các thông tin quan trọng khác.
- Giảm thiểu nguy cơ rò rỉ dữ liệu khi đẩy mã nguồn lên repository công khai hoặc trong quá trình làm việc nhóm.
- Cho phép người phát triển dễ dàng quản lý và chia sẻ biến môi trường một cách an toàn thông qua file mã hóa.

2.3 Gitflow

Trong dự án, **Gitflow** được sử dụng như một mô hình quản lý mã nguồn nhằm tránh xung đột giữa các thành viên trong nhóm và đảm bảo quy trình phát triển diễn ra một cách mạch lạc, rõ ràng.



Hình 1: *Gitflow hiện thực dự án*

1. Cấu trúc Branch trong dự án

- **Nhánh chính (Main Branches):**
  - **main**: Chứa mã nguồn ổn định đã được phát hành. Đây là nhánh chính phục vụ cho người dùng cuối.
  - **staging**: Chứa mã nguồn đang được kiểm thử trước khi phát hành. Các tính năng mới và hotfix sẽ được tích hợp vào nhánh này.
- **Nhánh phụ (Supporting Branches):**
  - **feature**: Được tạo từ nhánh `staging` để phát triển tính năng mới. Tên nhánh có dạng `feature/<tên-tính-năng>`.
  - **release**: Được tạo từ nhánh `staging` khi chuẩn bị phát hành phiên bản mới. Tên nhánh có dạng `release/<phiên-bản>`.
  - **hotfix**: Được tạo từ nhánh `main` để sửa lỗi khẩn cấp. Tên nhánh có dạng `hotfix/<tên-lỗi>`.

2. Quy trình Gitflow

a. Làm việc với nhánh Feature:

1. Tạo nhánh `feature` từ nhánh `staging`:

```
git checkout -b feature/<tên-tính-năng> staging
```

2. Phát triển tính năng và commit các thay đổi:

```
git add .
git commit -m "Phát triển tính năng <tên-tính-năng>"
```

3. Push nhánh `feature` lên repository từ xa:

```
git push origin feature/<tên-tính-năng>
```

4. Tạo Pull Request để merge nhánh `feature` vào `staging`.
5. Sau khi merge, xóa nhánh `feature`:

```
git push origin --delete feature/<tên-tính-năng>
git branch -d feature/<tên-tính-năng>
```

b. Làm việc với nhánh Hotfix:

1. Tạo nhánh `hotfix` từ nhánh `main`:

```
git flow hotfix start <tên-lỗi>
```

2. Sửa lỗi, sau đó merge nhánh vào `main` và `staging`:

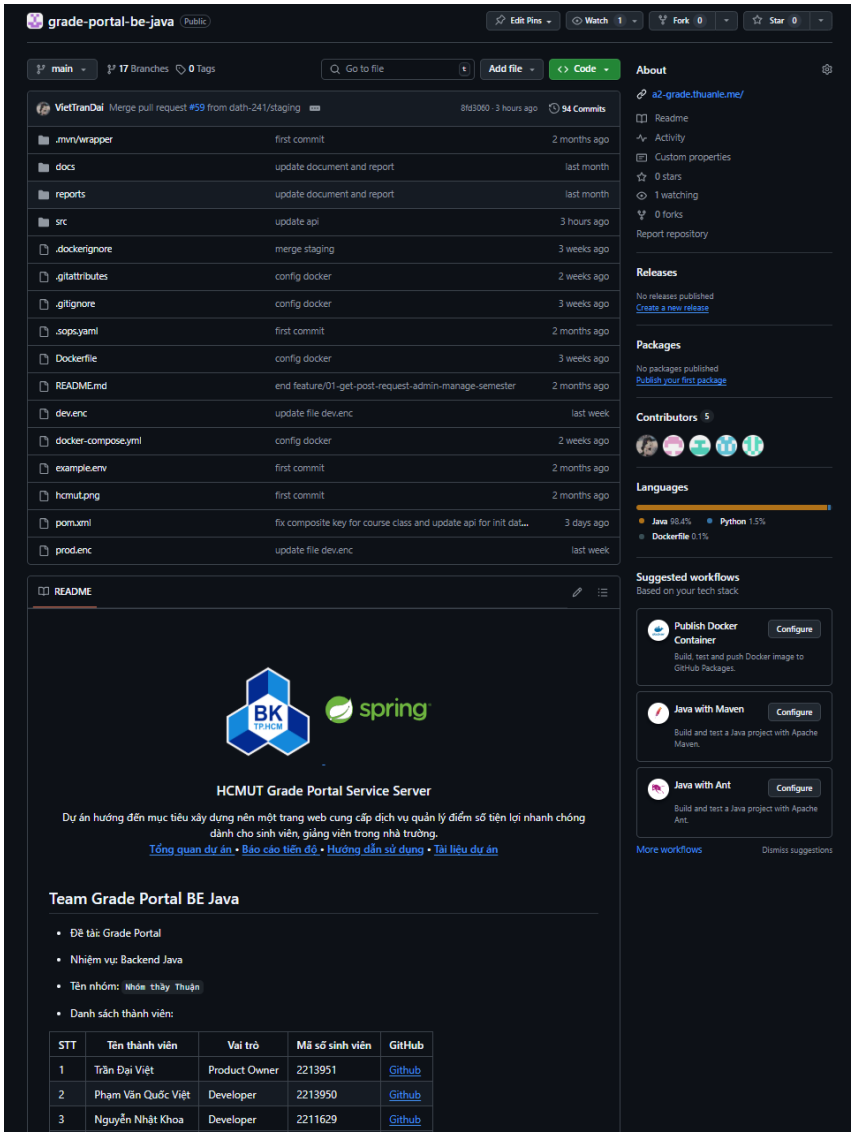
```
git flow hotfix finish <tên-lỗi>
```

3. Chú ý để tránh xung đột (Conflict) giữa các thành viên

- Luôn cập nhật nhánh `staging` hoặc `main` trước khi bắt đầu làm việc:

```
git checkout staging
git pull origin staging
```

- Sử dụng nhánh riêng biệt cho từng tính năng hoặc sửa lỗi.
- Đặt tên nhánh theo quy ước để dễ quản lý, ví dụ:
  - `feature/<tên-tính-năng>`
  - `hotfix/<tên-lỗi>`
  - `release/<phiên-bản>`
- Đẩy nhánh lên repository từ xa thường xuyên để tránh xung đột.
- Thực hiện `Code Review` trên các Pull Request để đảm bảo chất lượng mã nguồn và phát hiện xung đột sớm.
- Tránh commit trực tiếp vào các nhánh chính như `main` hoặc `staging`.



Hình 2: *Repo Github cho Backend của dự án*