Setup dự án

Project gồm 3 thành phần: Front-end, Back-end và Database theo kiến trúc 3 lớp (three-tier)

Front-end

Cấu hình cơ bản

- React TypeScript với Vite
- TailwindCSS
- Formatter: Prettier. Linter: ESLint

Cấu trúc thư mục

Thư mục được thiết kế theo cấu trúc phổ biến của React:

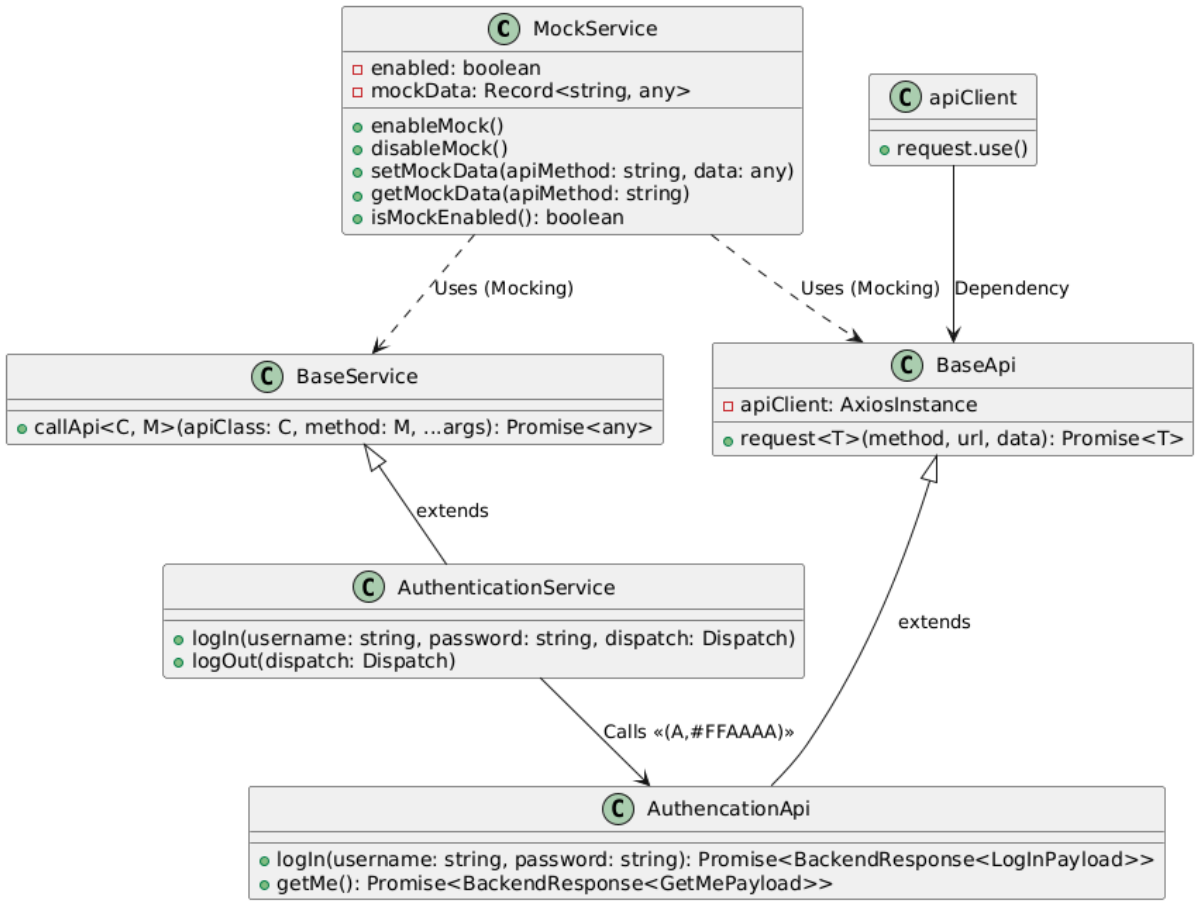
```
📂 public
    📄 (static assets: hình ảnh, favicon,...)
📂 src
 ├── 📂 @types
 │      📄 (type definitions cho API, models)
 │
 ├── 📂 assets
 │      📄 (ảnh, icon tĩnh)
 │
 ├── 📂 components (Các component được reuse cho nhiều trang)
 │
 ├── 📂 constants
 │      📄 (các hằng số: API URL, route name)
 │
 ├── 📂 lib
 │      📂 apis (gọi API, axios client)
 │      📂 redux (store, reducers)
 │      📂 services (xử lý logic, gọi API thông qua services)
 │      📂 utils (các hàm tiện ích dùng lại)
 │
 ├── 📂 pages (Thư mục chứa các trang - mỗi trang ứng với mỗi route)
 │
 ├── 📂 routes
 │      📄 (cấu hình React Router)
 │
 └── 📂 utils
 │      📄 (các tiện ích dùng toàn dự án)
 └── 📄 App.tsx (root component, setup routing, layout khung)
 └── 📄 Main.tsx (entry point, mount App vào HTML)
```

Khả năng mở rộng của code:

Khi cần thêm một trang vào frontend:

- Tạo một route mới trong index.tsx của thư mục /routes
- Tạo một file chứa UI của trang trong thư mục /pages
    - Ưu tiên sử dụng lại các component đã được thiết kế với mục đích tái sử dụng trong thư mục /components
    - Gọi các hàm hỗ trợ trong utils
    - Lấy dữ liệu từ backend thông qua class Service
        - Service cần gọi API và format/tính toán lại để ra được dữ liệu cần có

Đối với API và Services:



Workflow:
Page / Component gọi phương thức của một service:

1. Check xem MockService có được bật?
2. Check xem dữ liệu của API đã được mock
3. Trả về dữ liệu mock
4. Trả về kết quả của lớp API khi không có mock

Thiết kế trên nhằm giúp cho việc mock dữ liệu được dễ dàng hơn, có kèm theo việc kiểm tra phương thức đã mock có tồn tại (check API có phải thừa kế BaseAPI, check phương thức có thuộc API)

Backend

Cấu trúc thư mục

```
📂 (root)
├── 📂 app
│   ├── 📂 config          # Load config, env
│   ├── 📂 controller      # Xử lý request từ client
│   ├── 📂 middleware      # CORS, logger, recover
│   ├── 📂 model           # Định nghĩa struct dữ liệu
│   ├── 📂 repository      # Giao tiếp DB (MongoDB, JSON)
│   │   ├── 📂 student
│   │   └── 📂 user
│   ├── 📂 routes          # Khai báo và quản lý route
│   ├── 📂 service         # Xử lý business logic
│   └── 📂 utils           # Tiện ích chung
│
├── 📂 data                # Chứa file dữ liệu JSON mock
│
├── 📂 docs                # Tài liệu Swagger API
│
└── 📂 pkg
    └── 📂 response        # Định nghĩa phản hồi API
```

Backend được thiết kế theo mô hình MVC, kiến trúc lớp:
Request -> Middleware -> Controller -> Services -> Repository -> Database

Với services và repository, các lớp được thiết kế theo hướng dependency injection:

```
type IStudentRepository interface {
    GetStudents() ([]model.Student, error)
    AddStudent(req model.CreateStudentRequest) (model.Student, error)
    UpdateStudent(student model.Student) (model.Student, error)
    DeleteStudent(id int) error
    GetStudentByID(id int) (model.Student, error)
}
```

Repository sẽ được định nghĩa các lớp dưới dạng interface. Repository cụ thể sẽ được hiện thực thông qua các struct:

```
type MongoDBStudentRepository struct {
    collection *mongo.Collection
}
```

```
func NewMongoDBStudentRepository() IStudentRepository {
    return &MongoDBStudentRepository{
        collection: utils.StudentCollection,
    }
}
```

Với thiết kế dưới dạng dependency injection, backend có thể dễ dàng thay đổi database hơn (ví dụ chuyển từ MongoDB ở development sang DynamoDB khi chuyển sang production mode) mà không cần phải thay đổi cấu trúc code bên trong Controller / Service.