ĐOÀN VĂN TRUNG

2121050739

**BÁO CÁO**

**THỰC TẬP DOANH NGHIỆP**

CHUYÊN NGÀNH KHOA HỌC MÁY TÍNH ỨNG DỤNG

**XÂY DỰNG BACKEND CHO HỆ THỐNG QUẢN LÝ CÔNG VIỆC**

**HÀ NỘI - 11/2025**

**BÁO CÁO**

**THỰC TẬP DOANH NGHIỆP**

CHUYÊN NGÀNH KHOA HỌC MÁY TÍNH ỨNG DỤNG

**XÂY DỰNG BACKEND CHO HỆ THỐNG QUẢN LÝ CÔNG VIỆC**

|  |  |  |
| --- | --- | --- |
| **Sinh viên thực tập** | **:** | **Đoàn Văn Trung** |
| **Mã sinh viên** | **:** | **2121050739** |
| **Đơn vị thực tập** | **:** | **Công Ty Cổ Phần BKPLUS SOFTWARE** |
| **Người phụ trách** | **:** | **Nguyễn Tùng Dương** |
| **Giảng viên hướng dẫn** | **:** | **GV.ThS Nguyễn Thị Phương Bắc** |

**HÀ NỘI - 11/2025**

# **MỤC LỤC**

[**MỤC LỤC** 1](#_Toc213552734)

[**DANH MỤC HÌNH ẢNH** 2](#_Toc213552735)

[**DANH MỤC BẢNG BIỂU** 4](#_Toc213552736)

[**DANH MỤC TỪ VIẾT TẮT** 5](#_Toc213552737)

[**LỜI CẢM ƠN** 6](#_Toc213552738)

[**MỞ ĐẦU** 7](#_Toc213552739)

[**1. Lý do chọn đề tài** 7](#_Toc213552740)

[**2. Mục tiêu đề tài** 7](#_Toc213552741)

[**3. Phạm vi và đối tượng nghiên cứu** 8](#_Toc213552742)

[**4. Ý nghĩa khoa học và thực tiễn của đề tài** 9](#_Toc213552743)

[**CHƯƠNG 1: GIỚI THIỆU CHUNG** 10](#_Toc213552744)

[**1.1.** **Giới thiệu đơn vị thực tập** 10](#_Toc213552745)

[**1.2. Vị trí thực tập** 11](#_Toc213552746)

[**1.3. Đề tài thực tập** 12](#_Toc213552747)

[**1.4. Lịch làm việc** 12](#_Toc213552748)

[**CHƯƠNG 2: NỘI DUNG THỰC TẬP** 17](#_Toc213552749)

[**2.1. Kiến thức được đào tạo** 17](#_Toc213552750)

[**2.2. Công cụ sử dụng** 33](#_Toc213552751)

[**2.3. Nội dung thực tập** 37](#_Toc213552752)

[**CHƯƠNG 3: ĐÁNH GIÁ KẾT QUẢ VÀ ĐỊNH HƯỚNG PHÁT TRIỂN** 54](#_Toc213552753)

[**3.1. Đánh giá kết quả thực tập** 54](#_Toc213552754)

[**3.2. Định hướng phát triển của đề tài** 56](#_Toc213552755)

[**KẾT LUẬN** 59](#_Toc213552756)

[**TÀI LIỆU THAM KHẢO** 60](#_Toc213552757)

# **DANH MỤC HÌNH ẢNH**

[Hình 1 Công Ty Cổ Phần BKplus Software 10](#_Toc213605822)

[Hình 2 Phát Triển Backend (Server-side Development) 17](#_Toc213605823)

[Hình 3 Python 20](#_Toc213605824)

[Hình 4 Django 22](#_Toc213605825)

[Hình 5 Cơ chế hoạt động của Django 25](#_Toc213605826)

[Hình 6 Django REST Framework (DRF) 27](#_Toc213605827)

[Hình 7 RESTful API 31](#_Toc213605828)

[Hình 8 Thành phần chính trong RESTful API 32](#_Toc213605829)

[Hình 9 Cơ chế hoạt động của RESTful API 32](#_Toc213605830)

[Hình 10 PyCharm 33](#_Toc213605831)

[Hình 11 Visual Studio Code 34](#_Toc213605832)

[Hình 12 GitHub 35](#_Toc213605833)

[Hình 13 postgreSQL 35](#_Toc213605834)

[Hình 14 PostMan 36](#_Toc213605835)

[Hình 15 Docker 37](#_Toc213605836)

[Hình 16 API đăng ký tài khoản người dùng 38](#_Toc213605837)

[Hình 17 API đăng nhập tài khoản 38](#_Toc213605838)

[Hình 18 API lấy tất cả người dùng 39](#_Toc213605839)

[Hình 19 API lấy chi tiết người dùng 39](#_Toc213605840)

[Hình 20 API tìm kiếm người dùng dựa vào username hoặc email 39](#_Toc213605841)

[Hình 21 API lấy tất cả dự án 40](#_Toc213605842)

[Hình 22 API tạo dự án mới 40](#_Toc213605843)

[Hình 23 API lấy chi tiết dự án 40](#_Toc213605844)

[Hình 24 API sửa thông tin dự án 41](#_Toc213605845)

[Hình 25 API thêm thành viên vào dự án 41](#_Toc213605846)

[Hình 26 API xóa thành viên khỏi dự án 41](#_Toc213605847)

[Hình 27 API xóa dự án 42](#_Toc213605848)

[Hình 28 API thêm công việc vào dự án 42](#_Toc213605849)

[Hình 29 API lấy tất cả các công việc 42](#_Toc213605850)

[Hình 30 API lấy chi tiết công việc 43](#_Toc213605851)

[Hình 31 API sửa thông tin công việc 43](#_Toc213605852)

[Hình 32 API chỉ định thành viên cho công việc 43](#_Toc213605853)

[Hình 33 API xóa công việc 44](#_Toc213605854)

[Hình 34 API thêm bình luận 44](#_Toc213605855)

[Hình 35 API lấy tất cả bình luận 44](#_Toc213605856)

[Hình 36 API lấy chi tiết bình luận 45](#_Toc213605857)

[Hình 37 API sửa bình luận 45](#_Toc213605858)

[Hình 38 API xóa bình luận 45](#_Toc213605859)

[Hình 39 API lấy nhật ký hoạt động của dự án 46](#_Toc213605860)

[Hình 40 API lấy nhật ký hoạt động của công việc 46](#_Toc213605861)

# **DANH MỤC BẢNG BIỂU**

[Bảng 1- 1: Lịch làm việc 11](#_Toc197871864)

[Bảng 2- 1: Mô tả công việc 46](#_Toc197871874)

[Bảng 2- 2: Các công cụ đã tìm hiểu 53](#_Toc197871875)

# **DANH MỤC TỪ VIẾT TẮT**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Từ viết tắt** | **Từ tiếng Anh** | **Nghĩa tiếng Việt** |
| 1 | API | Application Programming Interface | Giao diện lập trình ứng dụng |
| 2 | CRUD | Create, Read, Update, Delete | Tạo, Đọc, Cập nhật, Xóa |
| 3 | MVT | Model-View-Template | Mô hình Model-View-Template |
| 4 | REST | Representational State Transfer | Kiến trúc chuyển trạng thái có biểu diễn |
| 5 | JWT | JSON Web Token | Mã thông báo Web dạng JSON |
| 6 | CORS | Cross-Origin Resource Sharing | Chia sẻ tài nguyên giữa các nguồn gốc khác nhau |
| 8 | JSON | JavaScript Object Notation | Định dạng đối tượng JavaScript |
| 9 | SQL | Structured Query Language | Ngôn ngữ truy vấn có cấu trúc |
| 10 | RDBMS | Relational Database Management System | Hệ quản trị cơ sở dữ liệu quan hệ |
| 11 | IDE | Integrated Development Environment | Môi trường phát triển tích hợp |
| 14 | HTTPS | HTTP Secure | HTTP bảo mật |
| 17 | ORM | Object-Relational Mapping | Ánh xạ quan hệ |
| 18 | Framework | Framework | Khung công tác |
| 20 | Database | Database | Cơ sở dữ liệu |
| 21 | Server | Server | Máy chủ |
| 22 | Client | Client | Máy khách, bên sử dụng |
| 24 | DRF | Django REST Framework | Framework xây dựng API cho Django |
| 25 | Authentication | Authentication | Xác thực |
| 35 | PostgreSQL | PostgreSQL | Hệ quản trị cơ sở dữ liệu PostgreSQL |

# **LỜI CẢM ƠN**

Sau quá trình học tập, nghiên cứu và thực hiện đề tài này, em muốn bày tỏ lòng biết ơn chân thành đến tất cả những người đã giúp đỡ cũng như hỗ trợ em trong suốt quá trình này. Đầu tiên, em xin gửi lời cảm ơn tới khoa Công Nghệ Thông Tin và bộ môn Khoa học máy tính tại Trường Đại Học Mỏ - Địa Chất, những người đã chỉ dạy truyền đạt kiến thức và hướng dẫn em một cách tận tình trong suốt thời gian học tập tại đây.

Em muốn đặc biệt gửi lời cảm ơn đến Cô Nguyễn Thị Phương Bắc, người đã luôn hướng dẫn nhiệt tình em trong quá trình thực tập doanh nghiệp này.

Em cũng muốn cảm ơn đến Công Ty Cổ Phần BKPLUS SOFTWARE đã tạo điều kiện cho em được thực tập tại đây, nơi em đã được học tập, trải nghiệm cũng như áp dụng những kiến thức đã học vào trong thực tế. Em gửi lời cảm ơn đến anh Nguyễn Tùng Dương đã tận tình hướng dẫn, giúp đỡ cho em tận tình trong suốt thời gian thực tập.

Em nhận thức rằng kiến thức của em còn nhiều hạn chế và em chưa hoàn thiện trong việc áp dụng chúng vào thực tế. Vì vậy, em mong nhận được những lời đóng góp của các thầy cô để em có thể tiếp tục hoàn thiện và phát triển một cách tốt hơn nữa.

Em xin chân thành cảm ơn thầy cô!

Sinh viên thực hiện

Đoàn Văn Trung

# **MỞ ĐẦU**

## **1. Lý do chọn đề tài**

Trong môi trường làm việc hiện đại, việc quản lý dự án và theo dõi tiến độ công việc của các thành viên là yếu tố then chốt quyết định sự thành công. Tuy nhiên, nhiều đội nhóm vẫn đang sử dụng các phương pháp truyền thống như bảng tính (Excel, Google Sheets), email hoặc các ghi chú rời rạc. Các phương pháp này bộc lộ nhiều hạn chế: khó theo dõi tiến độ tổng thể, dễ xảy ra sai sót thông tin, thiếu tính cộng tác và không có khả năng tự động hóa báo cáo.

Bên cạnh đó, việc chuyển đổi số trong quản lý doanh nghiệp đang diễn ra mạnh mẽ. Các công cụ quản lý công việc chuyên nghiệp như Trello, Jira, Asana đã chứng minh hiệu quả vượt trội, nhưng chi phí bản quyền hoặc sự phức tạp trong triển khai có thể là rào cản đối với các đội nhóm nhỏ và vừa. Vì vậy, việc xây dựng một hệ thống backend vững chắc cho một ứng dụng quản lý công việc tùy chỉnh là một giải pháp thiết thực, giúp tự động hóa quy trình, tăng cường sự minh bạch và nâng cao hiệu suất làm việc.

Xuất phát từ nhu cầu thực tiễn trên và mong muốn vận dụng kiến thức về Python và Django vào một dự án có tính ứng dụng cao, em đã lựa chọn đề tài “**Xây dựng backend cho Hệ thống Quản lý Công việc**”. Đề tài này không chỉ giải quyết một bài toán thực tế mà còn là cơ hội để em đóng góp vào việc tạo ra các công cụ hỗ trợ làm việc hiệu quả hơn.

## **2. Mục tiêu đề tài**

* Mục tiêu chung:

Xây dựng và triển khai hoàn chỉnh phần backend cho hệ thống quản lý công việc, đảm bảo đáp ứng các yêu cầu về chức năng, hiệu năng, bảo mật và khả năng mở rộng, phản ánh đúng logic nghiệp vụ phức tạp của một ứng dụng thực tế.

* Mục tiêu cụ thể:
* Thiết kế và triển khai cơ sở dữ liệu quan hệ với PostgreSQL, bao gồm các model: User, Project, Task, Comment, Attachment, và ActivityLog.
* Xây dựng hệ thống API theo chuẩn RESTful sử dụng Django REST Framework, bao gồm các endpoint cho tất cả các tài nguyên.
* Triển khai hệ thống xác thực an toàn sử dụng JSON Web Token (JWT) cho việc đăng ký, đăng nhập và làm mới token.
* Xây dựng hệ thống phân quyền (Permissions) chi tiết và phức tạp, quy định rõ các hành động người dùng (chủ dự án, thành viên, người được giao việc) được phép thực hiện trên từng tài nguyên.
* Tích hợp chức năng ghi nhật ký hoạt động (Activity Log) để theo dõi và lưu lại tất cả các thay đổi quan trọng trong hệ thống.
* Phát triển các tính năng phụ trợ như quản lý bình luận (Comment) và tệp đính kèm (Attachment) cho công việc.

## **3. Phạm vi và đối tượng nghiên cứu**

* Phạm vi nghiên cứu: Backend API của hệ thống quản lý công việc, tập trung vào các chức năng chính:
* Quản lý người dùng và xác thực JWT.
* CRUD cho Dự án (Project) và quản lý thành viên.
* CRUD cho Công việc (Task) với các thuộc tính trạng thái, độ ưu tiên, và người được giao.
* CRUD cho Bình luận (Comment) và Tệp đính kèm (Attachment).
* Hệ thống phân quyền truy cập chi tiết.
* Hệ thống ghi và truy xuất nhật ký hoạt động.
* Đối tượng nghiên cứu:
* Chủ dự án (Owner): Người có toàn quyền trên một dự án.
* Thành viên dự án (Member): Người tham gia vào dự án với các quyền hạn nhất định.
* Người được giao việc (Assignee): Người chịu trách nhiệm thực hiện một công việc cụ thể.
* Các kỹ sư, sinh viên quan tâm đến việc xây dựng backend API phức tạp với Django.

## **4. Ý nghĩa khoa học và thực tiễn của đề tài**

* Về ý nghĩa khoa học:
* Hệ thống là một ví dụ thực tiễn về việc áp dụng kiến trúc RESTful và các nguyên tắc thiết kế phần mềm vào việc xây dựng một ứng dụng có logic nghiệp vụ rõ ràng.
* Việc triển khai hệ thống phân quyền tùy chỉnh (custom permissions) và nhật ký hoạt động cung cấp một giải pháp tham khảo giá trị cho các bài toán tương tự.
* Về ý nghĩa thực tiễn:
* Cung cấp một bộ API backend vững chắc, sẵn sàng để tích hợp với bất kỳ giao diện frontend nào (web, mobile app).
* Hệ thống giúp tăng cường hiệu suất làm việc nhóm, minh bạch hóa quy trình và giảm thiểu sai sót trong quản lý.
* Là nền tảng công nghệ có thể dễ dàng mở rộng, giúp các đội nhóm, doanh nghiệp nhỏ nhanh chóng áp dụng số hóa vào quy trình quản lý.

# **CHƯƠNG 1: GIỚI THIỆU CHUNG**

## **Giới thiệu đơn vị thực tập**

### ***1.1.1. Giới thiệu chung về đơn vị***

Tên đầy đủ: Công Ty Cổ Phần BKplus Software

Website chính thức: [https://bkplussoft.com/](https://bkplussoft.com/?fbclid=IwZXh0bgNhZW0CMTAAYnJpZBExYWhtRGdkV2lCNkZlU0tDN3NydGMGYXBwX2lkEDIyMjAzOTE3ODgyMDA4OTIAAR7mfYg0ARgM67lthEhLT9zsTCdMId0wzZD4TaT1bqX4qMpb9mXO6i0HPmQAcw_aem_7fUo4Qi-d3jUv_Pz1lim0Q&brid=poSERw8SZs5dp8busJSptQ)

Email: contact@bkplussoft.com

Địa chỉ: Tầng 7, tòa nhà văn phòng số 104 Khuất Duy Tiến, Phường Nhân Chính, Quận Thanh Xuân, Thành phố Hà Nội, Việt Nam



Hình 1 Công Ty Cổ Phần BKplus Software

### ***1.1.2. Lịch sử phát triển***

Công Ty Cổ Phần BKPlus Software (BKPlus Software) được thành lập vào năm 2017 với định hướng trở thành đơn vị tư vấn và cung cấp giải pháp công nghệ sáng tạo cho doanh nghiệp trong đa dạng lĩnh vực.

Ngay từ những ngày đầu hoạt động, công ty đặc biệt chú trọng xây dựng đội ngũ kỹ sư công nghệ chất lượng cao, cùng với việc phát triển các dịch vụ tư vấn và thiết kế phần mềm theo yêu cầu, nhằm giúp doanh nghiệp tối ưu vận hành, nâng cao hiệu quả sản xuất và kinh doanh.

Với tầm nhìn hướng đến tương lai, BKPlus Software luôn duy trì chiến lược mở rộng hợp tác với các doanh nghiệp trong và ngoài nước, cập nhật xu hướng công nghệ mới nhất, tạo điều kiện nâng cao năng lực đội ngũ kỹ sư và cung cấp giải pháp công nghệ hiện đại cho khách hàng.

Trong quá trình phát triển, công ty đã triển khai nhiều dự án về chuyển đổi số, phát triển ứng dụng web & mobile cũng như các giải pháp phần mềm doanh nghiệp đa ngành. Nhờ đó, BKPlus Software từng bước khẳng định vị thế là một trong những đơn vị công nghệ trẻ, năng động và sáng tạo tại thị trường Việt Nam.

Định hướng trong giai đoạn tiếp theo, BKPlus Software đặt mục tiêu trở thành doanh nghiệp hàng đầu trong tư vấn và triển khai giải pháp công nghệ toàn diện, không ngừng đổi mới và sáng tạo để thích ứng với sự phát triển của nền công nghiệp số hiện nay.

### ***1.1.3. Quy mô hoạt động và lĩnh vực kinh doanh***

Trong quá trình hình thành và phát triển, BKPlus Software đã xây dựng một đội ngũ nhân lực có chuyên môn cao với quy mô khoảng 25–99 nhân sự, hoạt động chủ yếu tại văn phòng chính đặt tại: tầng 7, tòa nhà văn phòng số 104 Khuất Duy Tiến, Phường Nhân Chính, Quận Thanh Xuân, Thành phố Hà Nội, Việt Nam.

Hoạt động của công ty không chỉ tập trung vào phát triển phần mềm truyền thống mà còn hướng đến việc cung cấp các giải pháp công nghệ toàn diện cho doanh nghiệp trong và ngoài nước, bao gồm:

* Về nội bộ
* Tổ chức các nhóm phát triển phần mềm theo mô hình linh hoạt nhằm đảm bảo tiến độ và chất lượng sản phẩm.
* Ứng dụng quy trình quản lý dự án hiện đại (Agile/Scrum) trong phát triển sản phẩm và dịch vụ.
* Xây dựng môi trường làm việc sáng tạo, thúc đẩy năng lực nghiên cứu, sáng tạo và đổi mới công nghệ.
* Về cung cấp dịch vụ bên ngoài
* BKPlus Software tập trung cung cấp các sản phẩm – dịch vụ chủ lực:
* Tư vấn chuyển đổi số (Digital Transformation) cho doanh nghiệp và tổ chức.
* Phát triển phần mềm theo yêu cầu (custom software development).
* Gia công phần mềm (outsourcing) trong nước và quốc tế.
* Phát triển ứng dụng Web & Mobile theo yêu cầu.
* Cung cấp giải pháp phần mềm cho nhiều ngành nghề như:
* Y tế (Healthcare)
* Quản lý tài sản (Asset Management)
* Du lịch – khách sạn (Travel & Hospitality)
* Ẩm thực (F&B)
* Thương mại điện tử & bán lẻ (E-commerce & Retail)

Với đội ngũ kỹ sư giàu kinh nghiệm cùng định hướng dài hạn, BKPlus Software ngày càng khẳng định vai trò trong việc cung cấp giải pháp công nghệ, hỗ trợ doanh nghiệp thực hiện chuyển đổi số hiệu quả và bền vững.

## **1.2. Vị trí thực tập**

- Vị trí thực tập: Intern Backend

## **1.3. Đề tài thực tập**

Tên đề tài: Xây dựng Backend cho Hệ thống Quản lý Công việc

Mục tiêu của đề tài:

* **Về nghiên cứu và phân tích:** Tìm hiểu về các cấu trúc phụ trợ (cụ thể là MVT của Django), các framework (Django, Django REST Framework), ngôn ngữ (Python) cũng như hệ quản trị cơ sở dữ liệu (PostgreSQL) để phát triển hệ thống.
* **Về phát triển API**: Tìm hiểu và viết các API theo tiêu chuẩn RESTful, đảm bảo tất cả các chức năng cần thiết của công việc quản lý hệ thống đều được hỗ trợ đầy đủ.
* **Về phát triển chức năng:** Phát triển các nền tảng chức năng như quản lý người dùng (xác thực, phân quyền), quản lý dự án, quản lý công việc (tạo, phân công công việc, cập nhật trạng thái) và quản lý thành viên trong dự án.
* **Về kiểm tra và đánh giá**: Kiểm tra hiệu suất và kiểm tra tính bảo mật của hệ thống, thu thập phản hồi để cải thiện và hoàn thiện sản phẩm.

## **1.4. Lịch làm việc**

Công việc được giao theo tuần:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tuần** | **Công việc** | **Người hướng dẫn** | **Mức độ hoàn thành (SV tự đánh giá)** | **Ghi chú** |
| 1 | - Làm quen với đơn vị thực tập, tìm hiểu đơn vị thực tập.  - Cài đặt các công cụ cần thiết: Visual Studio Code, Git, Postman, PostgreSQL,…  - Cài đặt thư viện, xóa thư viện, chỉnh sửa thông tin project. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 2 | Làm quen Django  - Học: tạo project, tạo app.  + Hiểu flow: URL → View → Template.  + Cài admin, làm quen migration + SQLite.  + Thực hành: tạo blog app có model Post.  - Model & Template  + Học: ORM (CRUD), QuerySet cơ bản.  + Template: loop, if, render dữ liệu.  + Static files (CSS, JS), Media upload.  + Thực hành: mở rộng blog có Post + Comment.  🡺 Đầu ra: CRUD blog đầy đủ, chạy được local. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 3 | - Form & Authentication  + Học: Form (Django Form, ModelForm).  + Authentication: login, logout, register cơ bản.  + Middleware cơ bản.  + Thực hành: thêm tính năng login user để viết blog.  🡺 Đầu ra: web blog có user login, phân biệt user thường/admin. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 4 | - REST API Intro   * Cài DRF.   + Học: Serializer (ModelSerializer).  + ViewSet, Router.  + Thực hành: chuyển blog → thành API CRUD Post.  🡺 Đầu ra: test CRUD API bằng Postman: GET, POST, PUT, DELETE. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 5 | - CRUD API mở rộng  + Học: Generic APIView, mixins.  + Pagination, Filtering, Ordering.  + Thực hành: API blog có filter theo tác giả, phân trang.  🡺 Đầu ra: API chạy ổn, test bằng Postman ok. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 6 | - Auth trong DRF  + Học: Token Auth, JWT Auth dùng djangorestframework -simplejwt.  + Permission: custom, IsAdminUser/IsAuthenticated.  + Thực hành: API login JWT + phân quyền user thường admin.  🡺 Đầu ra: có login /logout API, token -based auth. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 7 | - Quan hệ model + Upload file  + Học: OneToMany, ManyToMany, nested serializer.  + API upload file (ảnh).  + Thực hành: mở rộng blog: Post có Category, user upload ảnh đại diện.  🡺 Đầu ra: API hỗ trợ quan hệ + upload file. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 8 | - Documentation  + Swagger/OpenAPI (drf-yasg hoặc drf-spectacular).  - Demo Project API hoàn chỉnh  + Chọn đề tài nhỏ (gợi ý: Task Manager API, E-commerce API mini, hoặc Car Listing API).  + Tính năng:  + User signup/login JWT.  + CRUD entity chính (Task/Product/Car).  + Có search, filter, pagination.  + Upload ảnh.  + Swagger doc.  🡺 Đầu ra: API có swagger doc, 1 project API chuẩn RESTful. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 9 | - Deploy & Best Practices  + .env config  + Debug vs Production  + PostgreSQL thay SQLite  + Deploy Render, Heroku, Railway  + Thực hành: Deploy demo project.  🡺 Đầu ra: link API online có thể share cho mọi người test. | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |
| 10 | - Làm báo cáo tổng kết các phần đã được học và thực hành | Nguyễn Tùng Dương | Hoàn thành nhiệm vụ |  |

Bảng 1- 1: Lịch làm việc

# **CHƯƠNG 2: NỘI DUNG THỰC TẬP**

## **2.1. Kiến thức được đào tạo**

### ***2.1.1. Phát triển Backend (Server-side Development)***

Phát triển Backend, hay còn gọi là Lập trình phía Máy chủ (Server-side Development), là nghệ thuật và khoa học xây dựng, duy trì phần cơ sở hạ tầng công nghệ mà người dùng không tương tác trực tiếp. Nếu ví một ứng dụng web như một nhà hàng, thì Frontend là khu vực ăn uống nơi thực khách ngồi, với bàn ghế, trang trí và thực đơn, trong khi Backend chính là toàn bộ nhà bếp, kho chứa, và đội ngũ đầu bếp. Đó là nơi nguyên liệu (dữ liệu) được lưu trữ, chế biến (xử lý logic), và nấu thành các món ăn (dữ liệu đã xử lý) để phục vụ cho thực khách (người dùng cuối) thông qua các "nhân viên phục vụ" (APIs).



Hình 2 Phát Triển Backend (Server-side Development)

Đây là một lĩnh vực phức tạp, bao gồm các trách nhiệm cốt lõi sau:

* Thiết kế và Quản lý Cơ sở dữ liệu: Đây là nền tảng của mọi ứng dụng. Công việc này không chỉ dừng lại ở việc tạo ra các bảng để lưu trữ dữ liệu, mà còn bao gồm việc thiết kế các mối quan hệ giữa chúng (ví dụ: một người dùng có thể có nhiều bài đăng), chuẩn hóa dữ liệu để tránh trùng lặp và đảm bảo tính nhất quán, cũng như viết các câu truy vấn hiệu quả để truy xuất và cập nhật dữ liệu một cách nhanh chóng.
* Xây dựng Logic Nghiệp vụ (Business Logic): Đây là "bộ não" của ứng dụng, nơi các quy tắc và quy trình cốt lõi được hiện thực hóa. Ví dụ, trong một trang thương mại điện tử, logic nghiệp vụ sẽ xử lý các việc như: kiểm tra số lượng hàng tồn kho khi người dùng đặt hàng, tính toán tổng giá trị đơn hàng bao gồm thuế và phí vận chuyển, xử lý thanh toán qua cổng thanh toán, và gửi email xác nhận cho khách hàng.
* Phát triển và Duy trì API (Application Programming Interface): APIs là tập hợp các quy tắc và giao thức cho phép các phần mềm khác nhau "nói chuyện" với nhau. Backend tạo ra các API để cung cấp một kênh giao tiếp chuẩn hóa cho phía Frontend (trang web, ứng dụng di động). Ví dụ, một API endpoint /api/products/123 sẽ trả về thông tin chi tiết của sản phẩm có ID là 123 dưới dạng dữ liệu có cấu trúc (thường là JSON).
* Xác thực và Phân quyền (Authentication & Authorization): Backend phải đảm bảo an ninh cho hệ thống. Xác thực là quá trình xác minh danh tính của người dùng (bạn là ai?), thường thông qua tên đăng nhập và mật khẩu hoặc các phương thức hiện đại như OAuth, JWT (JSON Web Tokens). Phân quyền là quá trình xác định những gì một người dùng đã được xác thực được phép làm (bạn được phép làm gì?). Ví dụ, một người dùng thông thường có thể xem sản phẩm, nhưng chỉ có quản trị viên mới có quyền thêm hoặc xóa sản phẩm.
* Tối ưu hóa Hiệu suất và Khả năng mở rộng (Performance & Scalability): Khi ứng dụng phát triển, Backend phải đảm bảo nó vẫn hoạt động nhanh và ổn định dưới tải trọng lớn. Điều này bao gồm việc tối ưu hóa các truy vấn cơ sở dữ liệu, sử dụng bộ nhớ đệm (caching) để lưu trữ các kết quả thường xuyên truy cập, và thiết kế hệ thống để có thể dễ dàng mở rộng bằng cách thêm nhiều máy chủ hơn (scaling).

Để xây dựng một hệ thống Backend hoàn chỉnh, các nhà phát triển dựa vào ba trụ cột công nghệ chính:

* Ngôn ngữ lập trình phía máy chủ: Là công cụ cơ bản để viết mã. Ngoài Python, các ngôn ngữ phổ biến khác trong thế giới backend bao gồm Java, C#, Node.js (JavaScript), Go, và PHP. Python được chọn cho dự án này vì sự đơn giản, hệ sinh thái mạnh mẽ và tốc độ phát triển nhanh.
* Framework hỗ trợ: Framework cung cấp một cấu trúc nền tảng và các công cụ được xây dựng sẵn, giúp chuẩn hóa quy trình phát triển và giải quyết các vấn đề phổ biến. Việc sử dụng framework như Django và Django REST Framework giúp lập trình viên tập trung vào logic nghiệp vụ thay vì phải xây dựng lại từ đầu các thành phần cơ bản như định tuyến URL, quản lý session, hay bảo mật.
* Công nghệ quản lý dữ liệu: Là nơi lưu trữ trái tim của ứng dụng - dữ liệu. Hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) như PostgreSQL được lựa chọn vì khả năng đảm bảo tính toàn vẹn dữ liệu mạnh mẽ, hỗ trợ các giao dịch phức tạp và tuân thủ tiêu chuẩn SQL, rất phù hợp cho các ứng dụng có dữ liệu cấu trúc rõ ràng.

### ***2.1.2. Python***

#### **2.1.2.1. Khái niệm về Python**

Python là một ngôn ngữ lập trình bậc cao, thông dịch, hướng đối tượng và có kiểu dữ liệu động. Ra mắt lần đầu vào năm 1991 bởi Guido van Rossum, Python được tạo ra với triết lý cốt lõi là nhấn mạnh vào khả năng đọc mã (code readability) và sự đơn giản. Triết lý này được thể hiện rõ trong "The Zen of Python" (Thiền của Python), với các nguyên tắc như "Đẹp đẽ tốt hơn xấu xí", "Rõ ràng tốt hơn tiềm ẩn", "Đơn giản tốt hơn phức tạp".



Hình 3 Python

Là một ngôn ngữ thông dịch, mã Python được thực thi từng dòng một bởi một chương trình thông dịch, khác với các ngôn ngữ biên dịch như C++ hay Java, nơi toàn bộ mã nguồn phải được chuyển đổi thành mã máy trước khi chạy. Điều này giúp chu trình phát triển (viết mã, chạy, gỡ lỗi) nhanh hơn đáng kể.

Tính đa năng của Python cho phép nó được ứng dụng trong một loạt các lĩnh vực:

* Phát triển Web: Với các framework mạnh mẽ như Django và Flask.
* Khoa học Dữ liệu và Phân tích: Với các thư viện như Pandas (xử lý dữ liệu), NumPy (tính toán số học), và Matplotlib (trực quan hóa).
* Trí tuệ Nhân tạo và Học máy: Là ngôn ngữ thống trị với các thư viện như Scikit-learn, TensorFlow, và PyTorch.
* Tự động hóa và Scripting: Dùng để tự động hóa các tác vụ lặp đi lặp lại của hệ thống.

#### **2.1.2.3. Ưu nhược điểm của Python**

Về ưu điểm:

* Dễ học và dễ đọc: Cú pháp của Python rất gần với ngôn ngữ tự nhiên, loại bỏ nhiều ký tự và cấu trúc phức tạp (như dấu ngoặc nhọn, dấu chấm phẩy) có trong các ngôn ngữ khác. Điều này không chỉ giúp người mới bắt đầu dễ tiếp cận mà còn giúp các lập trình viên dễ dàng bảo trì và làm việc nhóm trên cùng một cơ sở mã.
* Hệ sinh thái thư viện khổng lồ và trưởng thành: Python được hỗ trợ bởi PyPI (Python Package Index), một kho lưu trữ chứa hàng trăm nghìn gói phần mềm của bên thứ ba. Bạn muốn làm gì, gần như chắc chắn đã có người viết thư viện cho nó, từ việc xử lý file Excel, tạo file PDF, đến việc xây dựng các mô hình học máy phức tạp.
* Đa nền tảng (Cross-platform): Mã Python viết trên Windows có thể chạy trên macOS hoặc Linux mà không cần thay đổi (miễn là không sử dụng các thư viện đặc thù của hệ điều hành). Điều này giúp việc triển khai ứng dụng trở nên linh hoạt hơn.
* Cộng đồng lớn mạnh và hỗ trợ tuyệt vời: Python có một trong những cộng đồng lập trình viên lớn và tích cực nhất thế giới. Điều này có nghĩa là tài liệu hướng dẫn rất phong phú, và khi gặp vấn đề, bạn có thể dễ dàng tìm thấy câu trả lời trên các diễn đàn như Stack Overflow hoặc các nhóm cộng đồng.
* Tích hợp dễ dàng: Python có thể dễ dàng "kết dính" với các công nghệ khác. Nó có thể gọi các thư viện viết bằng C/C++ để tăng tốc độ xử lý, hoặc được nhúng vào các ứng dụng khác như một ngôn ngữ kịch bản.

Về nhược điểm:

* Tốc độ thực thi chậm hơn: Do bản chất là ngôn ngữ thông dịch và có kiểu dữ liệu động, Python thường chạy chậm hơn so với các ngôn ngữ biên dịch như C++, Java, hay Go. Trình thông dịch phải thực hiện nhiều công việc hơn tại thời điểm chạy. Tuy nhiên, đối với các ứng dụng web, thời gian xử lý thường bị chi phối bởi các yếu tố như mạng và truy vấn cơ sở dữ liệu, nên tốc độ của Python thường không phải là nút thắt cổ chai.
* Không phải lựa chọn hàng đầu cho phát triển di động: Python không phải là ngôn ngữ gốc cho iOS (Swift/Objective-C) hay Android (Kotlin/Java). Mặc dù có các framework như Kivy và BeeWare cho phép xây dựng ứng dụng di động bằng Python, chúng thường không đạt được hiệu năng và trải nghiệm người dùng mượt mà như các ứng dụng gốc.
* Tiêu thụ bộ nhớ cao: Các kiểu dữ liệu linh hoạt và việc quản lý bộ nhớ tự động của Python có thể dẫn đến việc sử dụng nhiều bộ nhớ hơn so với các ngôn ngữ quản lý bộ nhớ ở cấp thấp hơn.
* Global Interpreter Lock (GIL): GIL là một cơ chế trong trình thông dịch CPython (phiên bản Python phổ biến nhất) chỉ cho phép một luồng (thread) thực thi mã bytecode Python tại một thời điểm. Điều này làm cho việc tận dụng các bộ xử lý đa lõi cho các tác vụ tính toán song song (CPU-bound) trở nên khó khăn. Tuy nhiên, đối với các tác vụ liên quan đến I/O (như chờ phản hồi từ mạng hoặc cơ sở dữ liệu), GIL không phải là vấn đề lớn.

### ***2.1.3. Django***

#### **2.1.3.1. Khái niệm về Django**

Django là một web framework bậc cao theo triết lý "batteries-included" có nghĩa là nó cung cấp một bộ công cụ toàn diện và tích hợp sẵn để giải quyết hầu hết các vấn đề phổ biến trong phát triển web. Được phát triển trong một môi trường tòa soạn báo chí, Django được xây dựng để đáp ứng các yêu cầu về tốc độ phát triển nhanh và thiết kế sạch sẽ, thực dụng.



Hình 4 Django

Kiến trúc Model-View-Template (MVT) của Django hoạt động như sau:

* Model: Là đại diện cho dữ liệu của bạn. Mỗi model trong Django tương ứng với một bảng trong cơ sở dữ liệu. Django sử dụng một hệ thống ORM (Object-Relational Mapping) mạnh mẽ, cho phép bạn định nghĩa cấu trúc dữ liệu bằng các lớp Python. Thay vì viết CREATE TABLE users (...), bạn chỉ cần định nghĩa một lớp User với các thuộc tính tương ứng. Django ORM sẽ tự động xử lý việc tạo bảng và chuyển đổi các thao tác trên đối tượng Python thành các câu lệnh SQL.
* View: Là nơi xử lý logic. Khi một người dùng truy cập một URL, Django sẽ tìm đến view tương ứng. View nhận yêu cầu (request), thực hiện các logic cần thiết (ví dụ: lấy dữ liệu từ Model, xử lý form), và sau đó trả về một phản hồi (response), thường là bằng cách render một template với dữ liệu đã được xử lý.
* Template: Là lớp trình bày, chịu trách nhiệm hiển thị dữ liệu cho người dùng. Django có một ngôn ngữ template riêng mạnh mẽ, cho phép nhúng các biến và logic đơn giản (như vòng lặp, điều kiện) vào trong file HTML. Nó cũng hỗ trợ các tính năng cao cấp như kế thừa template (template inheritance), giúp tái sử dụng các phần giao diện chung (như header, footer) một cách hiệu quả.

#### **2.1.3.2. Ưu nhược điểm của Django**

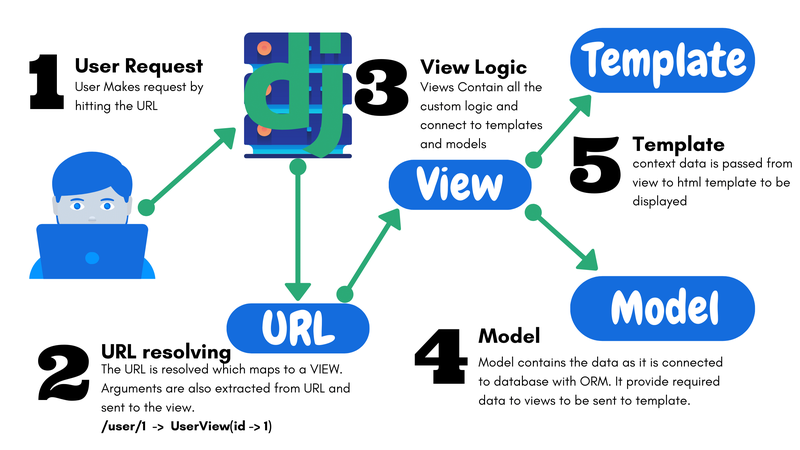
Về ưu điểm:

* Phát triển cực nhanh: Với triết lý "batteries-included", Django cung cấp sẵn các module cho hầu hết mọi thứ bạn cần: hệ thống xác thực người dùng, quản lý session, sitemap, RSS feeds, và đặc biệt là trang quản trị tự động. Điều này cho phép lập trình viên xây dựng các ứng dụng phức tạp trong thời gian ngắn.
* Bảo mật từ trong trứng nước: Django rất coi trọng vấn đề bảo mật và tích hợp sẵn các cơ chế bảo vệ chống lại các lỗ hổng phổ biến:
* SQL Injection: ORM của Django tự động "làm sạch" các tham số đầu vào, khiến việc tấn công qua SQL injection gần như là không thể nếu sử dụng đúng cách.
* Cross-Site Scripting (XSS): Hệ thống template của Django tự động thoát (escape) các ký tự HTML đặc biệt, ngăn chặn việc chèn mã độc vào trang web.
* Cross-Site Request Forgery (CSRF): Django cung cấp một middleware và template tag để bảo vệ chống lại các cuộc tấn công CSRF, đảm bảo rằng các yêu cầu thay đổi dữ liệu thực sự đến từ trang web của bạn.
* Khả năng mở rộng cao: Django được sử dụng bởi một số trang web có lưu lượng truy cập lớn nhất thế giới như Instagram và Pinterest. Cấu trúc module và khả năng tách biệt các thành phần giúp nó có thể mở rộng để đáp ứng nhu cầu tăng trưởng của ứng dụng.
* ORM mạnh mẽ và linh hoạt: ORM của Django là một trong những điểm mạnh nhất của nó. Nó không chỉ giúp trừu tượng hóa cơ sở dữ liệu (bạn có thể chuyển từ PostgreSQL sang MySQL chỉ bằng cách thay đổi một dòng trong file cấu hình) mà còn cung cấp một API truy vấn phong phú và trực quan.
* Trang Admin tích hợp sẵn: Đây là một trong những "tính năng sát thủ" của Django. Chỉ với vài dòng code để đăng ký model, Django sẽ tự động tạo ra một giao diện quản trị hoàn chỉnh, chuyên nghiệp và sẵn sàng cho sản xuất, cho phép những người không phải là lập trình viên có thể quản lý nội dung của trang web một cách dễ dàng.

Về nhược điểm:

* Nguyên khối (Monolithic) và Cồng kềnh: Vì cung cấp quá nhiều thứ, Django có thể trở nên quá mức cần thiết cho các dự án nhỏ hoặc các microservice đơn giản. Việc phải tuân theo cấu trúc dự án của Django có thể gây cảm giác nặng nề.
* Đường cong học tập dốc: Mặc dù Python dễ học, nhưng để thành thạo Django lại là một câu chuyện khác. Lập trình viên cần phải hiểu rõ về kiến trúc MVT, chu trình request-response, middleware, class-based views, và nhiều khái niệm khác.
* Ít linh hoạt hơn (Opinionated): Django có "cách làm của Django". Nó áp đặt nhiều quy ước và cấu trúc lên lập trình viên. Điều này giúp đảm bảo tính nhất quán nhưng cũng có thể hạn chế sự linh hoạt so với các micro-framework như Flask, nơi bạn có toàn quyền lựa chọn và kết hợp các thư viện.

#### **2.1.3.3. Cơ chế hoạt động của Django**



Hình 5 Cơ chế hoạt động của Django

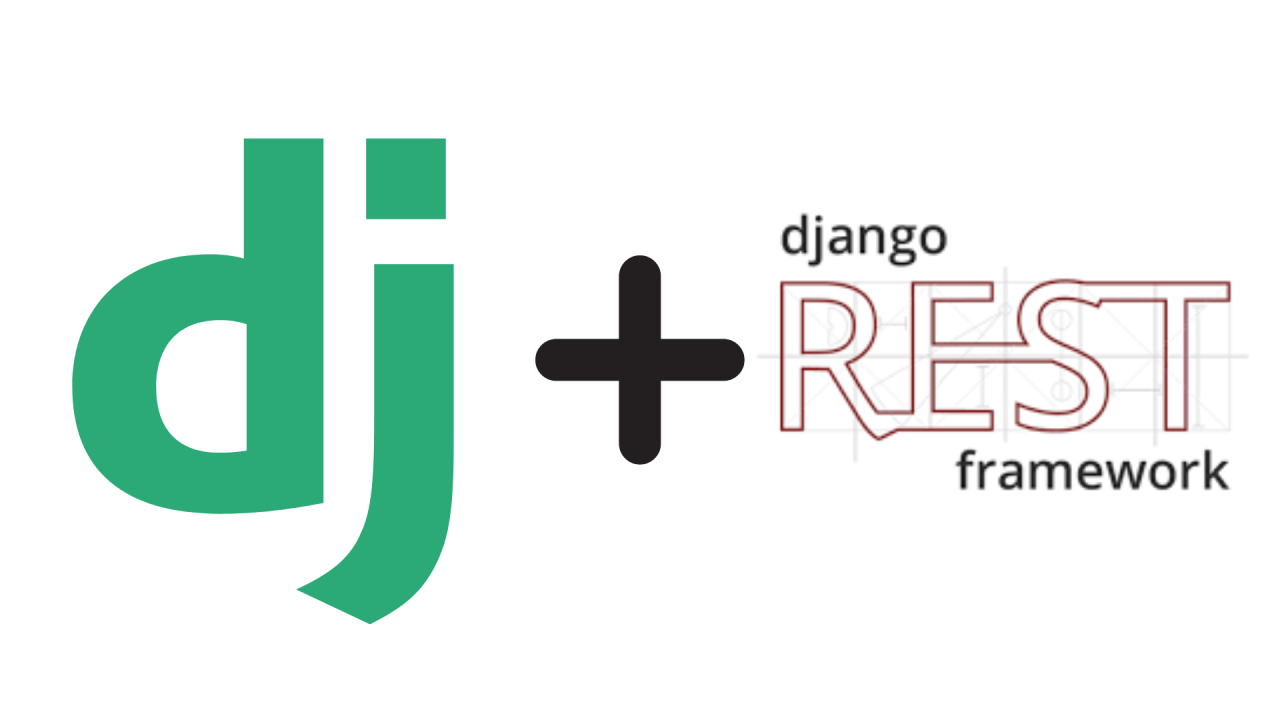
* Khi người dùng gửi một yêu cầu (HTTP Request) đến máy chủ, Django sẽ tiếp nhận và chuyển nó đến bộ điều phối URL (URL Dispatcher), được định nghĩa trong file urls.py.
* Bộ điều phối URL sẽ duyệt qua danh sách các mẫu URL đã đăng ký để tìm một mẫu khớp với đường dẫn của request.
* Khi tìm thấy một mẫu URL khớp, bộ điều phối sẽ gọi hàm hoặc lớp View tương ứng. View là nơi chứa logic nghiệp vụ chính của ứng dụng.
* Nếu cần truy xuất hoặc thay đổi dữ liệu, View sẽ tương tác với Model. Model là lớp đại diện cho cấu trúc dữ liệu và cung cấp một giao diện (thông qua Django ORM) để làm việc với cơ sở dữ liệu.
* Sau khi nhận được dữ liệu từ Model, View sẽ chuyển dữ liệu này cho một Template. Template chịu trách nhiệm định dạng và trình bày dữ liệu.
* Lưu ý quan trọng cho dự án API: Trong một dự án sử dụng Django REST Framework, vai trò của Template được thay thế bằng Serializer. Serializer sẽ chuyển đổi các đối tượng dữ liệu phức tạp (từ Model) thành định dạng JSON.
* Cuối cùng, View đóng gói kết quả (trang HTML đã được render từ Template hoặc dữ liệu JSON từ Serializer) vào một HTTP Response và gửi trả lại cho trình duyệt của người dùng.

Thông qua kiến trúc MVT, Django đạt được sự phân tách rõ ràng giữa logic xử lý dữ liệu (Model), logic nghiệp vụ (View), và logic trình bày (Template/Serializer). Cơ chế này giúp mã nguồn trở nên có tổ chức, dễ bảo trì, tăng khả năng tái sử dụng và cho phép các lập trình viên backend và frontend có thể làm việc song song một cách hiệu quả.

### ***2.1.4. Django REST Framework (DRF)***

#### **2.1.4.1. Khái niệm về Django REST Framework (DRF)**

Trong khi Django được thiết kế chủ yếu để xây dựng các trang web truyền thống (nơi server render HTML và gửi về cho trình duyệt), thế giới phát triển hiện đại ngày càng hướng tới việc tách biệt Backend và Frontend. Backend chỉ cung cấp dữ liệu thông qua API, còn Frontend (được xây dựng bằng React, Vue.js, Angular hoặc ứng dụng di động) sẽ tiêu thụ dữ liệu đó để hiển thị cho người dùng. Django REST Framework (DRF) ra đời để lấp đầy khoảng trống này. Nó là một thư viện mở rộng mạnh mẽ, giúp biến một dự án Django thành một máy chủ API RESTful chuyên nghiệp.



Hình 6 Django REST Framework (DRF)

Các thành phần cốt lõi của DRF bao gồm:

* Serializers: Đây là trái tim của DRF. Chúng thực hiện hai nhiệm vụ chính:
* Serialization (Tuần tự hóa): Chuyển đổi các kiểu dữ liệu phức tạp của Python, như các đối tượng model của Django, thành các định dạng dữ liệu gốc có thể dễ dàng truyền qua mạng, phổ biến nhất là JSON.
* Deserialization (Giải tuần tự hóa) và Validation (Xác thực): Chuyển đổi dữ liệu nhận được (ví dụ: JSON từ một yêu cầu POST) trở lại thành các đối tượng Python. Trong quá trình này, serializer cũng thực hiện việc xác thực dữ liệu, đảm bảo rằng dữ liệu đầu vào tuân thủ các quy tắc đã định nghĩa (ví dụ: email phải có định dạng hợp lệ, mật khẩu phải đủ dài).
* Views (APIView, ViewSets): DRF cung cấp các lớp View được xây dựng sẵn để xử lý logic cho các API endpoint. APIView là lớp cơ bản, trong khi ViewSet cung cấp một mức độ trừu tượng cao hơn, tự động xử lý các hành động CRUD (Create, Read, Update, Delete) tiêu chuẩn, giúp giảm thiểu đáng kể lượng code cần viết.
* Authentication & Permissions: DRF cung cấp một hệ thống xác thực và phân quyền linh hoạt, có thể cắm và chạy. Bạn có thể dễ dàng cấu hình API của mình để sử dụng các phương thức xác thực khác nhau như Session, Token, hoặc JWT. Hệ thống quyền hạn cho phép bạn kiểm soát ai có thể truy cập vào endpoint nào và thực hiện hành động gì.
* Routers: Router tự động tạo ra các mẫu URL cho ViewSets của bạn. Thay vì phải định nghĩa thủ công từng URL cho việc lấy danh sách, lấy chi tiết, tạo mới, cập nhật, và xóa một tài nguyên, bạn chỉ cần đăng ký ViewSet với một router, và nó sẽ làm tất cả công việc đó cho bạn.
* Browsable API: Một tính năng cực kỳ hữu ích cho việc phát triển. DRF tự động tạo ra một giao diện web thân thiện cho API của bạn. Bạn có thể truy cập các endpoint bằng trình duyệt, xem kết quả dưới dạng JSON được định dạng đẹp mắt, và thậm chí gửi các yêu cầu POST, PUT, DELETE thông qua các form HTML, giúp việc kiểm thử và gỡ lỗi trở nên vô cùng tiện lợi.

#### **2.1.4.2. Ưu nhược điểm của Django REST Framework**

Ưu điểm:

* Tích hợp chặt chẽ với Django: DRF được xây dựng dựa trên nền tảng Django, do đó nó tận dụng và mở rộng các thành phần cốt lõi như ORM, hệ thống xác thực, và phân quyền một cách liền mạch. Điều này giúp giảm đáng kể thời gian phát triển.
* Tài liệu hóa API tự động: DRF có khả năng tự động tạo ra các schema API (theo chuẩn OpenAPI/Swagger). Các thư viện như drf-spectacular có thể dựa vào đó để sinh ra các trang tài liệu API tương tác, giúp việc kiểm thử và tích hợp với frontend trở nên dễ dàng hơn rất nhiều.
* Serializers mạnh mẽ: Hệ thống Serializer của DRF không chỉ giúp chuyển đổi dữ liệu giữa các định dạng một cách dễ dàng mà còn tích hợp sẵn một cơ chế xác thực (validation) mạnh mẽ, đảm bảo tính toàn vẹn của dữ liệu đầu vào.
* Hệ thống xác thực và phân quyền linh hoạt: DRF cung cấp sẵn nhiều cơ chế xác thực (Session, Token, JWT) và các lớp phân quyền có thể tùy chỉnh. Điều này cho phép xây dựng các hệ thống bảo mật phức tạp, đáp ứng các yêu cầu nghiệp vụ đa dạng.
* Giao diện API có thể duyệt (Browsable API): Một trong những tính năng độc đáo và hữu ích nhất của DRF là tự động tạo ra một giao diện web cho API. Giao diện này cho phép các lập trình viên có thể tương tác, kiểm thử các endpoint trực tiếp trên trình duyệt mà không cần đến các công cụ bên ngoài như Postman.
* Cộng đồng lớn và tài liệu tốt: DRF có một cộng đồng người dùng đông đảo và được hỗ trợ tích cực. Tài liệu chính thức của nó rất chi tiết, rõ ràng và đầy đủ ví dụ, giúp người mới bắt đầu dễ dàng tiếp cận.

Nhược điểm:

* Đường cong học tập (Learning Curve): Mặc dù tài liệu tốt, DRF có rất nhiều khái niệm và thành phần (Serializers, ViewSets, Routers, Permissions,...). Đối với người mới bắt đầu, việc hiểu và sử dụng thành thạo tất cả các thành phần này có thể mất thời gian.
* "Ma thuật" ẩn sau các lớp trừu tượng: Các lớp cấp cao như Routers và  ModelViewSet giúp tạo ra các endpoint CRUD một cách nhanh chóng, nhưng chúng cũng che giấu đi nhiều logic bên trong. Điều này có thể gây khó khăn khi cần tùy chỉnh sâu hoặc gỡ lỗi các hành vi không mong muốn.
* Hiệu suất với các tác vụ phức tạp: Mặc dù DRF rất hiệu quả cho hầu hết các trường hợp, việc sử dụng Serializer cho các cấu trúc dữ liệu lồng nhau (nested) quá sâu hoặc các truy vấn cơ sở dữ liệu phức tạp có thể dẫn đến vấn đề "N+1 query", làm giảm hiệu suất. Lập trình viên cần có kiến thức về tối ưu hóa truy vấn Django (ví dụ: sử dụng prefetch\_related và select\_related) để khắc phục.
* Phụ thuộc hoàn toàn vào Django: DRF không phải là một thư viện độc lập. Nó được thiết kế để hoạt động bên trong một dự án Django, do đó không thể sử dụng nó với các web framework Python khác.

#### **2.1.4.3. Ứng dụng của Django REST Framework**

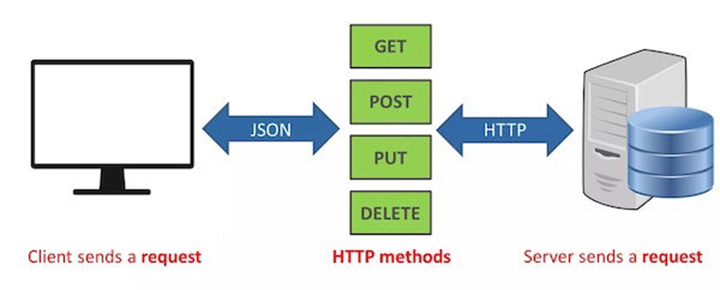
Nhờ vào sự mạnh mẽ và linh hoạt, Django REST Framework được sử dụng rộng rãi để xây dựng phần backend cho nhiều loại hình ứng dụng khác nhau:

* Backend cho các ứng dụng Single Page Application (SPA): Đây là ứng dụng phổ biến nhất. DRF cung cấp một API RESTful để các framework frontend hiện đại như React, Angular, hoặc Vue.js có thể giao tiếp, lấy dữ liệu và thực hiện các hành động mà không cần tải lại trang.
* Backend cho ứng dụng di động (Mobile Apps): Các ứng dụng trên iOS và Android cần một nơi để lưu trữ và đồng bộ hóa dữ liệu. DRF cung cấp các endpoint an toàn và hiệu quả để các ứng dụng di động có thể gửi và nhận dữ liệu từ máy chủ.
* Xây dựng các Microservices: Trong kiến trúc microservices, các dịch vụ nhỏ và độc lập giao tiếp với nhau thông qua API. DRF là một lựa chọn tuyệt vời để xây dựng các dịch vụ này một cách nhanh chóng, đặc biệt là các dịch vụ yêu cầu tương tác với cơ sở dữ liệu.
* Cung cấp API cho các dịch vụ bên thứ ba: Các doanh nghiệp thường xây dựng API để cho phép các đối tác hoặc khách hàng tích hợp với hệ thống của họ. DRF giúp tạo ra các API công khai (Public API) được tài liệu hóa tốt và bảo mật.
* Ứng dụng Internet of Things (IoT): Các thiết bị IoT (cảm biến, thiết bị thông minh) cần gửi dữ liệu thu thập được về một máy chủ trung tâm để xử lý và lưu trữ. DRF có thể được sử dụng để xây dựng các endpoint nhận dữ liệu từ hàng ngàn thiết bị này một cách hiệu quả.
* Nền tảng cho các hệ thống nội bộ và công cụ quản trị: DRF được sử dụng để xây dựng backend cho các hệ thống quản lý nội bộ (ví dụ: CRM, ERP, hệ thống quản lý kho) nơi dữ liệu cần được truy cập và thao tác thông qua một giao diện web tùy chỉnh.

### ***2.1.5. RESTful API***

#### **2.1.5.1. Khái niệm RESTful API**

RESTful API là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động…), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP [5].



Hình 7 RESTful API

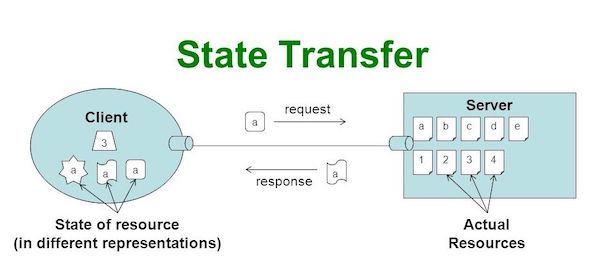
REST API thường vẫn sử dụng giao thức HTTP/1 kèm theo các định nghĩa trước đó mà cả client và server cần tuân thủ. Hiện tại, các REST API dùng JSON rất phổ biến. Một ít có thể vẫn còn dùng XML.

#### **2.1.5.2. Các thành phần chính trong RESTful API**

REST (Representational State Transfer) là một dạng chuyển đổi cấu trúc dữ liệu giữa client và server dựa trên giao thức HTTP. Mỗi tài nguyên được xác định duy nhất bởi một URL,và mọi tương tác như tạo mới, đọc, cập nhật, xóa bỏ đều thông qua các phương thức HTTP như GET, POST, PUT/PATCH, DELETE để xử lý dữ liệu.

API (Application Programming Interface) là một tập hợp các quy tắc với cơ chế là một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.

Qua đó, RESTful API là một tiêu chuẩn dùng để thiết kế các API cho các ứng dụng web để quản lý các resource. RESTful API là kiểu thiết kế API được sử dụng khá phổ biến và rộng rãi hiện nay để cho các ứng dụng như website, mobile… giao tiếp với nhau.



Hình 8 Thành phần chính trong RESTful API

#### **2.1.5.3. Cách thức hoạt động của RESTful API**

A diagram of a api

AI-generated content may be incorrect.

Hình 9 Cơ chế hoạt động của RESTful API

REST hoạt động chủ yếu dựa vào giao thức HTTP tận dụng toàn bộ các đặc tính có sẵn để trao đổi cũng như quản lý tài nguyên giữa client và server. Mỗi thao tác cơ bản trong mô hình CRUD đều được phản hồi trực tiếp lên các phương thức HTTP như POST, GET, PUT/PATCH, DELETE giúp làm rõ ý nghĩa của từng yêu cầu khi thiết kế các API.

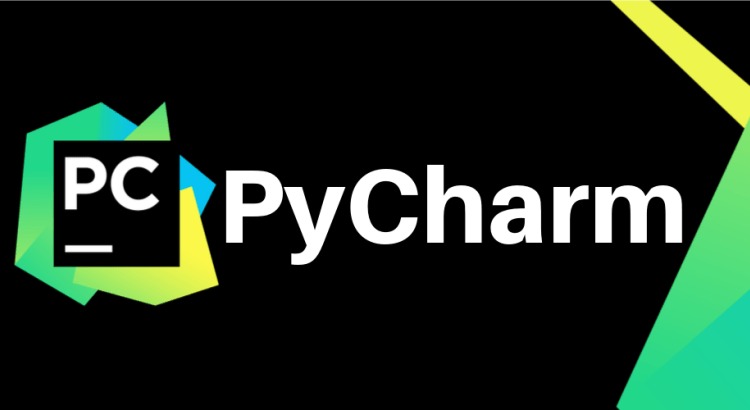
* **GET (Read/SELECT):** Truy vấn và trả về một resource cụ thể hoặc danh sách resource.
* **POST (CREATE):** Tạo mới một resource. Body của request chứa representation của resource cần tạo.
* **PUT (Update/Replace):** Cập nhật toàn bộ resource. Nếu resource chưa tồn tại, có thể dùng để tạo mới.
* **PATCH (Partial Update):** Cập nhật một phần resource, chỉ gửi những trường cần thay đổi.
* **DELETE (DELETE):** Xoá một Resource.

Những phương thức hay hoạt động này thường được gọi là CRUD tương ứng với các chức năng chính là Create (Tạo), Read (Đọc), Update (Cập nhật, Sửa), Delete (Xóa). Những phương thức này không chỉ làm cho API trở nên dễ hiểu hơn mà còn đảm bảo tính nhất quán, khả năng mở rộng và dễ bảo trì.

## **2.2. Công cụ sử dụng**

### ***2.2.1. PyCharm***

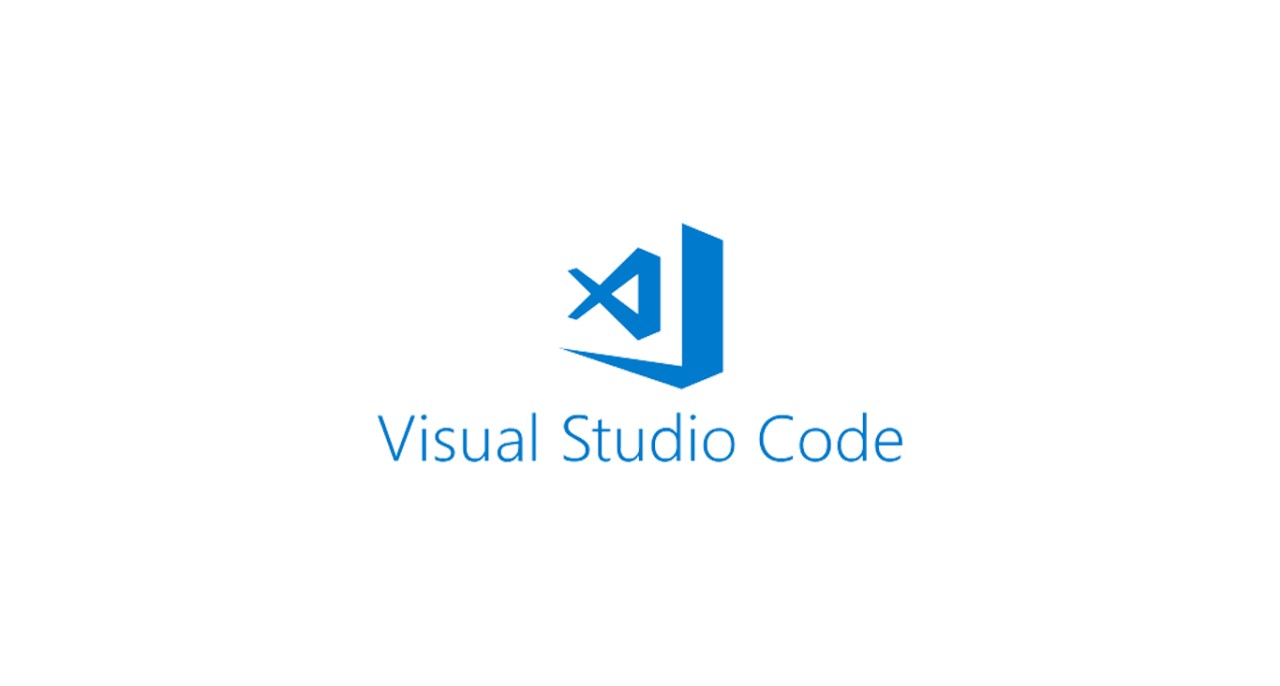
PyCharm là một Môi trường phát triển tích hợp (IDE) mạnh mẽ và chuyên dụng cho ngôn ngữ lập trình Python, được phát triển bởi công ty JetBrains. IDE này cung cấp một bộ công cụ toàn diện cho các lập trình viên chuyên nghiệp, bao gồm trình gỡ lỗi đồ họa, trình chạy kiểm thử, tích hợp sâu với các hệ thống quản lý phiên bản và hỗ trợ mạnh mẽ cho các framework phát triển web lớn như Django. Với khả năng phân tích mã nguồn thông minh và gợi ý cú pháp hiệu quả, PyCharm giúp tăng tốc độ phát triển và giảm thiểu lỗi trong quá trình xây dựng các ứng dụng phức tạp.



Hình 10 PyCharm

### ***2.2.2. Visual Studio Code***

Visual Studio Code (VS Code) là một trình soạn thảo mã nguồn miễn phí, nhẹ và mạnh mẽ được phát triển bởi Microsoft, hoạt động trên đa nền tảng bao gồm Windows, macOS và Linux. Với giao diện thân thiện và khả năng tùy biến cao thông qua một hệ sinh thái tiện ích mở rộng (extensions) khổng lồ, VS Code trở thành một công cụ cực kỳ linh hoạt. Chương trình tích hợp sẵn các tính năng thiết yếu như terminal, quản lý Git, và trình gỡ lỗi trực quan. Đối với dự án này, các tiện ích mở rộng cho Python và Django đã cung cấp khả năng gợi ý mã thông minh (IntelliSense), giúp việc phát triển web trở nên hiệu quả và nhanh chóng.



Hình 11 Visual Studio Code

### ***2.2.3. GitHub***

GitHub là một nền tảng chuyên lưu trữ cũng như quản lý mã nguồn trực tuyến dựa trên Git, được phát hành bởi Microsoft vào năm 2008. Người dùng có thể theo dõi lịch sử thay đổi, làm việc nhóm đồng thời quản lý lỗi qua issue tracker và dễ dàng tự động hóa kiểm thử, build và deploy. Nó cung cấp cả phiên bản trả tiền lẫn miễn phí cho các tài khoản. Ngoài ra nó còn là nơi lưu trữ hàng triệu dự án mã nguồn mở giúp cho các developer học hỏi cũng như là tương tác với nhau.

A logo of a cat

AI-generated content may be incorrect.

Hình 12 GitHub

### ***2.2.4. PostgreSQL***

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ đối tượng (ORDBMS) mã nguồn mở, nổi tiếng về sự ổn định, mạnh mẽ và tuân thủ nghiêm ngặt các tiêu chuẩn SQL. Do khả năng xử lý các truy vấn phức tạp, độ tin cậy cao và khả năng mở rộng tốt, PostgreSQL thường được lựa chọn cho các ứng dụng lớn và các môi trường sản phẩm (production). Trong dự án này, PostgreSQL đóng vai trò là nơi lưu trữ toàn bộ dữ liệu của hệ thống, và được tương tác một cách an toàn và hiệu quả thông qua lớp ORM của Django.



Hình 13 postgreSQL

### ***2.2.5. PostMan***

Postman là công cụ phổ biến nhất được sử dụng trong kiểm thử API được phát hành vào năm 2012 bởi Abhinav Asthana. Postman đơn giản hóa từng bước trong vòng đời API và làm hiệu quả, dễ dàng hơn cũng như là cung cấp môi trường cho người dùng thực hiện các hoạt động như tạo, chia sẻ, kiểm thử và quản lý các API.

A logo of a person in a suit

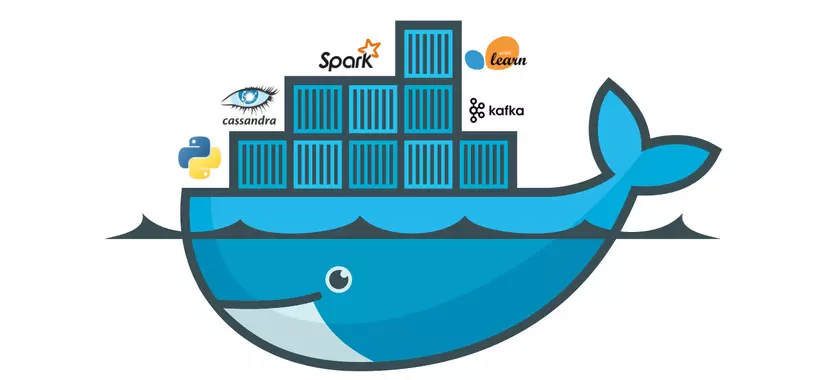
AI-generated content may be incorrect.

Hình 14 PostMan

Postman có những tính năng như tạo yêu cầu HTTP (GET, POST, PUT, DELETE, …), kiểm thử tự động, quản lý biến môi trường và xem phản hồi ngay lập tức. Giúp người dùng tiết kiệm được thời gian trong việc phát triển và kiểm thử API. Postman còn có cơ chế tương tác thông minh với API, giúp người dùng hiểu rõ về cách thức hoạt động và phản hồi của API.

### ***2.2.6. Docker***

Docker là một nền tảng cho lập trình viên để develop, deploy và run application với container. Docker cho phép tạo các môi trường độc lập và tách biệt để khởi chạy và phát triển ứng dụng và môi trường. Nó cũng hỗ trợ việc xây dựng, quản lý giúp đơn giản hóa quá trình phát triển, thử nghiệm và triển khai ứng dụng theo mô hình microservices.



Hình 15 Docker

## **2.3. Nội dung thực tập**

### ***2.3.1. Mô tả công việc***

|  |  |
| --- | --- |
| **Tuần** | **Mô tả** |
| **1** | * Tìm hiểu và làm quen với đơn vị thực tập: Lịch sử, cơ cấu tổ chức, lĩnh vực hoạt động, các dự án, đối tác,… * Cài đặt các công cụ cần thiết: PyCharm/VS Code, PostgreSQL, Postman, Git, Docker. |
| **2 - 5** | * Tìm hiểu về ngôn ngữ lập trình Python từ cơ bản đến nâng cao * Tìm hiểu về kiến trúc web và giao thức HTTP. * Tìm hiểu về SQL và cách thiết kế cơ sở dữ liệu quan hệ. * Training: * Nghiên cứu về kiến trúc MVT của Django. * Nghiên cứu cách viết Models, Views, và Serializers trong Django REST Framework. * Nghiên cứu viết API CRUD, phân quyền (Permissions), và xác thực (Authentication) với JWT * Nghiên cứu xây dựng RESTful API. |
| **6 - 9** | * Tiến hành thực hiện dự án: * Dự án backend cho Hệ thống Quản lý Công việc: * Nghiên cứu yêu cầu về các chức năng. * Lập kế hoạch phát triển các module. * Bắt đầu thực hiện code các chức năng chính. * Hoàn thiện và kiểm tra bằng Postman * Minh họa các chức năng đã xây dựng (thông qua Postman)  1. API quản lý người dùng :     Hình 16 API đăng ký tài khoản người dùng    Hình 17 API đăng nhập tài khoản    Hình 18 API lấy tất cả người dùng    Hình 19 API lấy chi tiết người dùng    Hình 20 API tìm kiếm người dùng dựa vào username hoặc email  2. API quản lý dự án :    Hình 21 API lấy tất cả dự án    Hình 22 API tạo dự án mới    Hình 23 API lấy chi tiết dự án    Hình 24 API sửa thông tin dự án    Hình 25 API thêm thành viên vào dự án    Hình 26 API xóa thành viên khỏi dự án    Hình 27 API xóa dự án  3. API quản lý công việc:    Hình 28 API thêm công việc vào dự án    Hình 29 API lấy tất cả các công việc    Hình 30 API lấy chi tiết công việc    Hình 31 API sửa thông tin công việc    Hình 32 API chỉ định thành viên cho công việc    Hình 33 API xóa công việc  4. API Quản lý bình luận và nhật ký hoạt động:   * **Bình Luận**     Hình 34 API thêm bình luận    Hình 35 API lấy tất cả bình luận    Hình 36 API lấy chi tiết bình luận    Hình 37 API sửa bình luận    Hình 38 API xóa bình luận   * **Nhật Ký hoạt động**     Hình 39 API lấy nhật ký hoạt động của dự án    Hình 40 API lấy nhật ký hoạt động của công việc |
| **10** | - Báo cáo tiến độ dự án  - Chuẩn bị và làm bài báo cáo thực tập doanh nghiệp và xin đánh giá từ phía người hướng dẫn tại đơn vị. |

Bảng 2- 1: Mô tả công việc

### ***2.3.2. Các công cụ đã tìm hiểu***

|  |  |
| --- | --- |
| **Công cụ đã tìm hiểu** | **Mô tả** |
| **GitHub** | * **Khái niệm:** * GitHub là nền tảng lưu trữ và quản lý mã nguồn dựa trên việc sử dụng hệ thống kiểm soát từ Git. GitHub cho phép lập trình viên và nhóm phát triển, quản lý mã nguồn, theo dõi thay đổi và quản lý các phiên bản của dự án. * **Hoạt động:** * **Repository:** Người dùng có thể tạo ra một repository mới để chứa toàn bộ mã nguồn hoặc sao chép một repository có sẵn để làm việc. * **Commit và Push:** Thực hiện các thay đổi trong mã nguồn, tạo commit để lưu lại các thay đổi, và đẩy (Push) các commit này lên repository trên GitHub. * **Branching và Merging:** Tạo nhánh mới để phát triển các tính năng hoặc sửa lỗi riêng biệt. Tạo pull request để hợp nhất (Merge) các thay đổi từ nhánh đó vào nhánh chính. * **Collaborative Development:** Sử dụng các công cụ thảo luận, xem xét mã (Code review), và quản lý vấn đề (Issue tracking) để làm việc nhóm hiệu quả. * **Ưu điểm:** * **Về quản lý mã nguồn:** Cung cấp công cụ mạnh mẽ để lưu trữ và quản lý các thay đổi trong mã nguồn, dễ dàng quay lại bất kỳ trạng thái nào. * **Về hợp tác:** Pull request giúp các thành viên làm việc trong nhóm hiệu quả hơn khi thông qua việc thảo luận trước khi hợp nhất các thay đổi. * **Về lưu trữ và chia sẻ mã nguồn:** Cung cấp nền tảng để lưu trữ cũng như chia sẻ mã nguồn. * **Về tích hợp công cụ:** Tích hợp tốt với nhiều công cụ phát triển phần mềm, kiểm thử và triển khai khác nhau, giúp cho việc phát triển phần mềm trở nên dễ dàng hơn. * **Về mở rộng và bảo mật:** Hỗ trợ các tính năng bảo mật như kiểm soát việc truy cập, tìm lỗ hổng bảo mật,… giúp bảo vệ dự án khỏi các vấn đề nguy hiểm. * **Về cộng đồng:** Đây là nơi tập trung của cộng đồng developer rất lớn lên tới hàng triệu người dùng. Là nơi tìm kiếm và học hỏi từ các dự án trở nên dễ dàng hơn. * **Nhược điểm:** * **Về tính năng nâng cao:** Các tính năng nâng cao như repository riêng tư không giới hạn, tích hợp CI/CD nâng cao,… sẽ phải trả phí bắt buộc. * **Về sự phụ thuộc vào nền tảng:** Phải phụ thuộc vào một hoặc nhiều dịch vụ bên ngoài. Nếu gặp vấn đề sự cố thì sẽ có thể ảnh hưởng đến quá trình làm của người dùng cũng như của đội nhóm. * **Về khả năng học hỏi:** Đối với những người mới, việc làm quen với GitHub có thể sẽ gặp nhiều khá khó khăn vì phải học và hiểu được những lệnh cơ bản của nó. * **Về bảo mật dữ liệu:** Việc lưu trữ mã nguồn trên một nền tảng công cộng có thể gây ra nhiều lo ngại về vấn đề bảo mật đối với các dự án quan trọng. * **Về khả năng mở rộng:** Đối với những dự án lớn, việc quản lý các repository lớn với nhiều nhánh và commit sẽ trở nên phức tạp khó khăn hơn. |
| **PyCharm/VS Code** | * **Khái niệm :**   + Đây là các Môi trường Phát triển Tích hợp (IDE) và trình soạn thảo mã nguồn hàng đầu, được sử dụng làm không gian làm việc chính để viết mã, gỡ lỗi và quản lý dự án. Cả hai đều có thể được tùy biến mạnh mẽ thông qua các plugin và tiện ích mở rộng để hỗ trợ chuyên sâu cho Python và Django. * **Vai trò và Ứng dụng trong Dự án :**   + **Viết mã và Gợi ý thông minh (IntelliSense):** Đây là chức năng quan trọng nhất. IDE/Editor hiểu rõ cấu trúc của Django và DRF, giúp tự động hoàn thành mã khi viết các truy vấn ORM, định nghĩa các trường trong serializers.py, hay gọi các thuộc tính của đối tượng request trong views.py. Điều này giúp giảm thiểu lỗi cú pháp và tăng tốc độ code.   + **Gỡ lỗi (Debugging):** Sử dụng trình gỡ lỗi đồ họa để đặt các điểm dừng (breakpoints) trong các file quan trọng như views.py hoặc permissions.py. Khi một API được gọi, chương trình sẽ dừng lại tại breakpoint, cho phép kiểm tra giá trị của các biến tại thời điểm đó (ví dụ: request.user, queryset, kwargs), giúp tìm và sửa lỗi logic một cách trực quan và hiệu quả.   + **Tích hợp Terminal:** Sử dụng terminal tích hợp sẵn để chạy các lệnh quản lý của Django (python manage.py runserver, makemigrations, migrate) và các lệnh Git mà không cần phải chuyển đổi cửa sổ.   + **Quản lý Môi trường và Phụ thuộc:** Hỗ trợ quản lý môi trường ảo (venv) và nhận diện các thư viện đã được cài đặt trong requirements.txt, đảm bảo môi trường phát triển luôn nhất quán. * **Ưu điểm:**   + **Tăng năng suất:** Các tính năng gợi ý mã và gỡ lỗi giúp tiết kiệm rất nhiều thời gian so với việc viết mã trên các trình soạn thảo đơn giản.   + **Quản lý dự án tập trung:** Mọi thứ từ viết mã, chạy lệnh, gỡ lỗi, đến quản lý phiên bản đều có thể thực hiện trong một cửa sổ duy nhất.   + **Nhược điểm:**   + **Tài nguyên:** Các IDE đầy đủ tính năng như PyCharm có thể tiêu tốn nhiều tài nguyên hệ thống (RAM, CPU).   + **Cấu hình:** VS Code đòi hỏi người dùng phải tự cài đặt và cấu hình các tiện ích mở rộng cần thiết để có được trải nghiệm tốt nhất. |
| **PostgreSQL** | * **Khái niệm:** * PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ - đối tượng (ORDBMS) mã nguồn mở, nổi tiếng về sự ổn định, mạnh mẽ và tuân thủ nghiêm ngặt các tiêu chuẩn SQL. * **Hoạt động trong dự án:** * **Lưu trữ dữ liệu:** Đóng vai trò là nơi lưu trữ toàn bộ dữ liệu của hệ thống (users, projects, tasks,...) một cách có cấu trúc và bền vững. * **Tương tác qua ORM:** Ứng dụng Django không tương tác trực tiếp với PostgreSQL bằng SQL thuần, mà thông qua lớp ORM. Django ORM sẽ dịch các thao tác trên đối tượng Python (ví dụ: Project.objects.filter(...)) thành các câu lệnh SQL tương ứng để thực thi trên PostgreSQL. * **Quản lý quan hệ**: Xử lý hiệu quả các mối quan hệ phức tạp như ForeignKey (1-n) và ManyToManyField (n-n) giữa các bảng. * **Ưu điểm:** * **Về tính toàn vẹn dữ liệu:** Rất mạnh mẽ trong việc đảm bảo tính nhất quán và toàn vẹn của dữ liệu thông qua các ràng buộc (constraints) và transactions. * **Về khả năng mở rộng:** Hỗ trợ tốt cho các ứng dụng lớn với lượng dữ liệu phức tạp, sẵn sàng cho việc phát triển trong tương lai. * **Nhược điểm:** * **Về cấu hình:** Có thể phức tạp hơn trong việc cài đặt và cấu hình ban đầu so với các CSDL nhẹ hơn như SQLite. |
| **PostMan** | * **Khái niệm:** * Postman là một trong những công cụ phổ biến trong việc kiểm thử API, giúp các nhà phát triển có thể thiết kế, thử nghiệm, tài liệu hóa và theo dõi các API. Postman có rất nhiều tính năng đa dạng và phong phú hỗ trợ trong việc quản lý cũng như phát triển vòng đời API. * **Hoạt động:** * **Cấu hình:** Người dùng đăng nhập hoặc tạo tài khoản postman để có thể lưu trữ cũng như đồng bộ hóa dữ liệu trên các thiết bị của người dùng. * **Quản lý request:** Postman cho phép người dùng sử dụng các lệnh phương thức HTTP như GET, POST, PUT, DELETE, PATCH, v.v.. Người dùng phải nhập URL thêm các tiêu đề và thân của trước gửi yêu cầu về máy chủ. * **Collections:** Postman cho phép người dùng tổ chức các API vào collections để có thể dễ dàng quản lý cũng như sửa đổi. * **Mock Servers và Monitors:** Postman cho phép người dùng tạo các máy chủ giả lập để thử nghiệm API cũng như là thiết lập các giám sát định kỳ để kiểm tra hiệu suất của API. * **Ưu điểm:** * **Về giao diện:** Postman cung cấp người dùng một giao diện trực quan và dễ sử dụng, giúp người dùng có thể làm quen và bắt đầu làm việc nhanh chóng với API. * **Về tính năng:** Postman cung cấp cho người dùng nhiều tính năng mạnh mẽ, vượt trội như tạo và quản lý các collection, test scripts, mock servers, monitors,.... * **Về khả năng hỗ trợ:** Postman hỗ trợ cho người dùng nhiều phương thức HTTP như GET, POST, PUT, DELETE, PATCH,… và các định dạng dữ liệu như JSON, XML, HTML,… * **Về khả năng chia sẻ:** Postman cung cấp cho người dùng tính năng chia sẻ cho phép nhiều người dùng chia sẻ collection, môi trường và tài liệu API trong thời gian thực thông qua Workspace. * **Về khả năng tích hợp:** Postman có thể tích hợp với các công cụ khác nhau như GitHub, Jenkins, Travis CI,… giúp tự động hóa và quản lý vòng đời API. * **Nhược điểm:** * **Về chi phí**: Postman tuy có nhiều tính năng hữu ích cho người dùng, nhưng các tính năng cao cấp như phân tích nâng cao, giám sát không giới hạn,…. yêu cầu trả một mức phí nhất định để có thể sử dụng các tính năng cao cấp đó. * **Về độ phức tạp**: Postman yêu cầu người dùng phải biết những kiến thức nâng cao hơn khi sử dụng các tính năng nâng cao vào việc cấu hình và quản lý thành phần trong đó. * **Về tài nguyên**: Postman sẽ cần tiêu thụ một lượng lớn tài nguyên hệ thống khi phải làm việc với các bộ sưu tập API lớn hoặc dùng nhiều monitors và test API cùng lúc. * **Về sự phụ thuộc khác**: Postman có một vài chức năng chỉ có ở máy khách và không thể dùng trực tiếp nên sẽ gây ra một số bất tiện cho người dùng. |

Bảng 2- 2: Các công cụ đã tìm hiểu

# **CHƯƠNG 3: ĐÁNH GIÁ KẾT QUẢ VÀ ĐỊNH HƯỚNG PHÁT TRIỂN**

## **3.1. Đánh giá kết quả thực tập**

Kết quả của quá trình thực tập

* Về kiến thức:
* **Kiến thức nền tảng:**Trong quá trình thực tập với dự án Python & Django, em đã củng cố và nâng cao kiến thức về lập trình hướng đối tượng, các nguyên tắc thiết kế phần mềm như SOLID, và kiến trúc REST.
* **Kiến thức chuyên sâu:** Em đã học hỏi và hiểu rõ về Django REST Framework, bao gồm cách xây dựng các APIView, Serializers, và đặc biệt là các khái niệm nâng cao như:
  + Xây dựng hệ thống phân quyền tùy chỉnh (custom permissions) phức tạp
  + Xây dựng hệ thống lọc và tìm kiếm động sử dụng thư viện django-filter với các logic tùy chỉnh.
  + Thiết kế và triển khai một hệ thống ghi log (auditing) để theo dõi hoạt động người dùng.
  + Xử lý xác thực và bảo mật API qua JWT (JSON Web Token).
* Về kỹ năng (bao gồm kỹ năng mềm):
* **Kỹ năng lập trình**: Kỹ năng viết mã Python sạch, có cấu trúc và tuân thủ các chuẩn đã được cải thiện đáng kể.
* **Kỹ năng giải quyết vấn đề**: Em đã phát triển kỹ năng phân tích yêu cầu, chia nhỏ vấn đề và tìm kiếm giải pháp hiệu quả trong quá trình xây dựng các tính năng phức tạp như quản lý thành viên hay phân quyền.
* **Kỹ năng quản lý thời gian**: Em đã biết phân chia độ ưu tiên công việc, lên các kế hoạch hợp lý và chi tiết, và đã biết sử dụng các công cụ nhắc nhở để đảm bảo hoàn thành đúng hạn.
* **Kỹ năng lập kế hoạch:** Em đã học cách xác định mục tiêu, phân chia nhiệm vụ nhỏ hơn và điều chỉnh phương án kịp thời khi quá trình làm việc có sự cố hoặc cần thay đổi.

Đánh giá mức độ áp dụng những kiến thức đã được học tại trường với thực tế công việc trong quá trình thực tập

* Những kiến thức (học phần) đã học ở trường được dùng trong công việc thực tập:
  + **Cơ sở dữ liệu:** Qua môn cơ sở dữ liệu, em đã biết cách thiết kế mô hình quan hệ, viết các câu lệnh SQL, giúp em dễ dàng hơn trong việc thiết kế Django Models và hiểu cách ORM hoạt động.
  + **Phát triển ứng dụng Web:** Qua môn này, em đã có kiến thức cơ bản về cách thiết kế và triển khai các API cho các chức năng CRUD, phân quyền.
  + **Mạng máy tính:** Môn học này giúp em hiểu được các khái niệm cơ bản về giao thức HTTP, client-server và cách cấu hình HTTPS, hỗ trợ cho việc triển khai ứng dụng web.
  + **Phân tích và thiết kế hệ thống:** Qua môn học này, em đã có kiến thức về việc phân tích yêu cầu và thiết kế hệ thống, giúp em hiểu được cấu trúc và luồng nghiệp vụ của ứng dụng.
* Những kiến thức mới đã được tiếp cận, tìm hiểu trong quá trình thực tập:
* **Django, Django REST Framework:** Em đã học hỏi và tiếp cận sâu với các framework chuyên dụng cho phát triển backend, đây là những kiến thức thực chiến quan trọng.
* **Bảo mật web:** Em đã học hỏi và tiếp cận với middleware như CORS và xác thực bằng JSON Web Token (JWT).
* **Containerization với Docker:** Em đã được học một công nghệ rất hiện đại là Docker, giúp đóng gói và triển khai ứng dụng một cách nhất quán.

Đánh giá khả năng đáp ứng công việc của bản thân về kiến thức và kỹ năng

* **Về kiến thức:** Em tự tin rằng kiến thức đã được học tại trường đã giúp em có một nền tảng vững chắc để tiếp thu cũng như áp dụng các kiến thức mới trong thực tế. Những kiến thức mới về Python, Django, và các kỹ thuật API nâng cao đã giúp em hoàn thành tốt công việc.
* **Về kỹ năng:** Các kỹ năng lập trình, làm việc nhóm, giải quyết vấn đề và thuyết trình của em đã được cải thiện rõ rệt hơn. Em có thể làm việc hiệu quả, giao tiếp và phối hợp tốt với mọi người đồng thời có thể tự giải quyết các vấn đề phức tạp khác.

## **3.2. Định hướng phát triển của đề tài**

### ***3.2.1. Hướng làm đồ án tốt nghiệp (ĐATN)***

Về hướng làm đồ án tốt nghiệp, em sẽ phát triển đề tài thực tập này thành một hệ thống ứng dụng web hoàn chỉnh (Full-stack Application), không chỉ có backend mà còn tích hợp cả một giao diện người dùng (Frontend) hiện đại và đầy đủ chức năng. Hệ thống sẽ được mở rộng về phạm vi và logic nghiệp vụ để trở thành một sản phẩm hoàn thiện, sẵn sàng cho việc triển khai thực tế.

Mục tiêu của đồ án là hoàn thiện các chức năng quản lý, bổ sung các tính năng nâng cao như báo cáo, thống kê, đồng thời củng cố và hoàn thiện tài liệu hóa quy trình phát triển. Các mục tiêu cụ thể bao gồm:

* **Phân tích và Thiết kế Hệ thống:**
* Xác định chi tiết các chức năng mới và bổ sung các luồng nghiệp vụ phức tạp hơn (ví dụ: quy trình phê duyệt công việc, quản lý vai trò người dùng).
* Thiết kế kiến trúc hệ thống hoàn chỉnh, bao gồm cả kiến trúc frontend và cách nó tương tác với backend API.
* Xây dựng các module quản lý người dùng, phân quyền truy cập chi tiết, chức năng báo cáo thống kê, và các chức năng khác liên quan đến dự án và công việc.
* **Triển khai Giao diện (Frontend):**
* Xây dựng một Single Page Application (SPA) hoàn chỉnh sử dụng một framework JavaScript hiện đại như React hoặc Vue.js.
* Thiết kế giao diện người dùng (UI/UX) thân thiện, trực quan và responsive, đảm bảo hoạt động tốt trên cả máy tính và các thiết bị di động.
* Tích hợp các thư viện quản lý trạng thái (state management) như Redux hoặc Vuex để quản lý dữ liệu phía client một cách hiệu quả.
* **Phát triển Xử lý ở Server (Backend):**
* Triển khai thêm các API endpoint mới để hỗ trợ các tính năng của frontend (ví dụ: API cho báo cáo, API cho thông báo).
* Tối ưu hóa hiệu suất của các API hiện có, đặc biệt là các truy vấn cơ sở dữ liệu phức tạp, bằng cách sử dụng các kỹ thuật như select\_related và prefetch\_related của Django.
* Nâng cấp hệ thống xác thực để hỗ trợ các luồng phức tạp hơn như "Quên mật khẩu" hoặc đăng nhập bằng tài khoản mạng xã hội (OAuth2).
* **Kiểm thử và Tài liệu:**
* Thực hiện kiểm thử toàn diện cho cả backend (Unit Test, Integration Test) và frontend (End-to-End Test).
* Hoàn thiện tài liệu API (sử dụng Swagger/OpenAPI), tài liệu thiết kế kiến trúc, và tài liệu hướng dẫn sử dụng để đảm bảo chất lượng và khả năng bảo trì của đồ án.

### ***3.2.2. Hướng phát triển đề tài***

Về mặt hướng phát triển dài hạn, để đề tài thực tập “Xây dựng backend cho Hệ thống Quản lý Công việc” trở thành một sản phẩm cạnh tranh và hoàn thiện, em sẽ mở rộng nó thành một nền tảng quản lý công việc toàn diện. Nền tảng sẽ được tối ưu về khả năng quản trị, bảo mật, hiệu năng và khả năng tích hợp. Sau đây là một số định hướng cụ thể để phát triển đề tài:

* **Tích hợp Hệ thống Thông báo Real-time và Email:**
* Sử dụng Django Channels để tích hợp công nghệ WebSocket, cho phép gửi thông báo tức thì (real-time notifications) đến người dùng khi có các sự kiện quan trọng (ví dụ: được gán việc mới, có bình luận mới, công việc sắp hết hạn).
* Xây dựng một hệ thống gửi email tự động (sử dụng Celery và Redis) để gửi các thông báo, báo cáo định kỳ hoặc các lời nhắc quan trọng.
* **Xây dựng Module Báo cáo và Thống kê (Dashboard):**
* Phát triển các API chuyên dụng để tổng hợp và phân tích dữ liệu, cho phép tạo ra các báo cáo và biểu đồ trực quan (ví dụ: biểu đồ Burndown chart, biểu đồ phân bổ công việc) giúp người quản lý có cái nhìn tổng quan về tiến độ dự án và hiệu suất làm việc của đội nhóm.
* **Nâng cao Bảo mật:**
* Tích hợp các cơ chế bảo mật nâng cao như Xác thực hai yếu tố (2FA).
* Mã hóa end-to-end các dữ liệu nhạy cảm, đặc biệt là các tệp đính kèm.
* Áp dụng các biện pháp phòng chống tấn công DDoS và các kỹ thuật rate-limiting nâng cao.
* **Mở rộng Khả năng Tích hợp (Integrations):**
* Xây dựng các API cho phép tích hợp với các công cụ làm việc phổ biến khác như Slack (để gửi thông báo), Google Calendar (để đồng bộ hóa deadline), hoặc GitHub/GitLab (để liên kết công việc với các commit và pull request).
* **Tối ưu hóa Hiệu năng và Khả năng Mở rộng:**
* Tích hợp hệ thống caching (sử dụng Redis) để lưu trữ các kết quả truy vấn thường xuyên, giảm tải cho cơ sở dữ liệu và tăng tốc độ phản hồi của API.
* Thiết kế kiến trúc để sẵn sàng cho việc mở rộng theo chiều ngang (horizontal scaling), cho phép hệ thống có thể xử lý hàng triệu người dùng đồng thời.

Với những định hướng phát triển chi tiết và có tầm nhìn như trên, em tin rằng sản phẩm từ đề tài này sẽ trở nên đầy đủ, mạnh mẽ, an toàn và có khả năng đáp ứng được các yêu cầu khắt khe của môi trường doanh nghiệp.

# **KẾT LUẬN**

Trong khoảng thời gian thực hiện quá trình thực tập tại Công ty Cổ phần BKPLUS SOFTWARE, em đã được tìm hiểu, học hỏi và phát triển nhiều kiến thức chuyên môn cũng như các kỹ năng cần thiết. Từ đó, em đã rút ra cho bản thân những bài học sâu sắc:

* Về kinh nghiệm lập trình: Em đã có cơ hội thực hành các kỹ năng lập trình từ cơ bản đến nâng cao và áp dụng vào việc tạo ra một sản phẩm backend thực tế, có logic phức tạp.
* Về kỹ năng mềm: Quá trình thực tập đã giúp em rèn luyện các kỹ năng mềm quan trọng như giao tiếp, xử lý vấn đề và làm việc nhóm, giúp em tự tin hơn trong môi trường công việc chuyên nghiệp.
* Hiểu rõ hơn về quy trình phát triển phần mềm: Em đã được trải nghiệm các giai đoạn quan trọng trong quy trình phát triển phần mềm, bao gồm thiết kế, phát triển, kiểm thử và triển khai.

Trong thời gian thực tập đã giúp em tìm hiểu và học hỏi được nhiều kiến thức bổ ích, đồng thời rút ra nhiều kinh nghiệm thực tế. Do vấn đề về thời gian cũng như kinh nghiệm còn hạn chế, sản phẩm không tránh khỏi những thiếu sót. Em xin chân thành cảm ơn sự hướng dẫn tận tình của các thầy cô và anh chị tại đơn vị thực tập, và rất mong nhận được những ý kiến đóng góp để hoàn thiện hơn nữa.

Cùng với những kiến thức đã học tại trường, em đã hoàn thành tốt đợt thực tập. Qua đó, em đã học được thêm nhiều kiến thức và kinh nghiệm bổ ích để sau này khi ra trường có thể vận dụng vào thực tế. Em xin cảm ơn sự hướng dẫn của các thầy cô trên lớp cũng như ở đơn vị đã giúp em hoàn thành tốt đợt thực tập này.

# **TÀI LIỆU THAM KHẢO**

1. MDN Web Docs. Server-side website programming overview. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side
2. FreeCodeCamp. Backend Development Explained: Server, Database, APIs. Available: https://www.freecodecamp.org/news/backend-development/
3. Python Software Foundation. Python 3.12 Documentation. Available: https://docs.python.org/3/
4. W3Schools. Python Programming Basics. Available: https://www.w3schools.com/python/
5. TutorialsPoint. Python Programming Language Tutorial. Available: https://www.tutorialspoint.com/python/
6. Django Software Foundation. Django Documentation (v5.0). Available: https://docs.djangoproject.com/
7. TutorialsPoint. Python and Django Full Stack Development Guide. Available: https://www.tutorialspoint.com/django/
8. Django REST Framework. Official REST API Framework for Django. Available: https://www.django-rest-framework.org/
9. GeeksforGeeks. Introduction to Django REST Framework. Available: https://www.geeksforgeeks.org/django-rest-framework-tutorial/
10. Real Python. Building RESTful APIs with Django REST Framework. Available: https://realpython.com/django-rest-framework-quick-start/
11. MDN Web Docs. HTTP Overview and RESTful API Design. Available: https://developer.mozilla.org/
12. FreeCodeCamp. How to Build and Deploy a Django REST API. Available: https://www.freecodecamp.org/
13. JetBrains. PyCharm IDE Documentation. Available: https://www.jetbrains.com/pycharm/documentation/
14. Microsoft. Visual Studio Code Documentation. Available: https://code.visualstudio.com/docs
15. GitHub Docs. Collaborating with GitHub Repositories. Available: https://docs.github.com/
16. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. Available: https://www.postgresql.org/docs/
17. Postman. API Development and Testing Tool Documentation. Available: https://learning.postman.com/
18. Docker Inc. Docker Overview and User Guide. Available: https://docs.docker.com/