Chương này trình bày các công nghệ và nền tảng được lựa chọn trong đồ án. Nội dung chương được tổ chức theo từng lớp của hệ thống. Với mỗi công nghệ, phần trình bày tập trung vào ba ý chính: công nghệ là gì và thường được dùng để giải quyết vấn đề nào; công nghệ đó đáp ứng các yêu cầu đã nêu ở Chương 2 ra sao; và trong số các hướng tiếp cận phổ biến, vì sao đồ án chọn phương án hiện tại.

0.1 Kiến trúc hệ thống

0.1.1 Monolith theo mô hình client–server

Trong phát triển hệ thống web, mô hình client–server là cách tổ chức quen thuộc: phía client chạy trên trình duyệt, chịu trách nhiệm hiển thị giao diện và tiếp nhận thao tác; phía server cung cấp API, xử lý nghiệp vụ và quản lý dữ liệu. Trên nền tảng đó, kiến trúc monolith được hiểu là backend được triển khai như một ứng dụng thống nhất (một codebase và một đơn vị triển khai), nhưng bên trong vẫn có thể tổ chức theo các module nghiệp vụ để tách bạch trách nhiệm. Ưu điểm lớn của monolith là tính đơn giản và trực tiếp: luồng xử lý ít tầng trung gian, dễ kiểm thử end-to-end, triển khai gọn, và thuận tiện khi cần đảm bảo tính nhất quán dữ liệu vì mọi thao tác nghiệp vụ tập trung trong một hệ thống.

Các yêu cầu ở Chương 2 cho thấy đồ án cần ưu tiên hoàn thiện các luồng nghiệp vụ chính và đảm bảo trải nghiệm người dùng. Hệ thống không chỉ có các thao tác thông thường như tạo và quản lý dữ liệu, mà còn phải tương tác với blockchain, cập nhật trạng thái giao dịch và phản hồi rõ ràng khi giao dịch thành công hoặc thất bại. Với phạm vi như vậy, monolith giúp giảm đáng kể chi phí thiết kế và vận hành so với mô hình phân tán: không phải giải quyết quá nhiều vấn đề giao tiếp giữa các dịch vụ, không phải xây dựng cơ chế đồng bộ dữ liệu phức tạp giữa nhiều hệ thống, và việc kiểm thử luồng từ giao diện đến cơ sở dữ liệu cũng trực tiếp hơn. Điều này đặc biệt phù hợp khi đồ án cần ổn định và hạn chế rủi ro trong tích hợp.

Microservices là hướng tiếp cận phổ biến khác, trong đó backend được tách thành nhiều dịch vụ nhỏ giao tiếp qua API hoặc hàng đợi tin nhắn. Cách làm này có lợi khi hệ thống ở quy mô lớn, nhiều nhóm phát triển song song, hoặc cần mở rộng độc lập theo từng chức năng. Tuy nhiên, đổi lại là độ phức tạp của một hệ phân tán: cấu hình, giám sát, truy vết luồng request, xử lý lỗi và đảm bảo nhất quán dữ liệu đều khó hơn và tốn công hơn. Trong bối cảnh đồ án, khi mục tiêu là xây dựng một marketplace chạy mạch lạc và tích hợp blockchain rõ ràng, lựa chọn monolith là hợp lý. Đồng thời, để tránh việc hệ thống trở nên cồng kềnh, backend vẫn được tổ chức theo các module nghiệp vụ, tạo nền tảng cho khả năng tách dần về sau nếu cần mở rộng.

0.1.2 Nguyên tắc phân tách on-chain và off-chain

NFT marketplace khác với các ứng dụng web thông thường ở chỗ tài sản và giao dịch được ghi nhận trên blockchain, nên quyền sở hữu và trạng thái mua bán cần có khả năng kiểm chứng công khai. Vì vậy, trong đồ án em lựa chọn cách tách rõ phần on-chain và off-chain: các thao tác làm thay đổi quyền sở hữu hoặc trạng thái giao dịch được thực hiện trong smart contract để đảm bảo tính tin cậy; còn các dữ liệu phục vụ hiển thị và có nhu cầu chỉnh sửa linh hoạt được lưu off-chain để thuận tiện thao tác và tránh phát sinh chi phí không cần thiết.

Cụ thể, hình ảnh NFT/collection được lưu trên AWS S3; các thông tin như tên và mô tả được lưu trong cơ sở dữ liệu để người dùng có thể cập nhật trực tiếp mà không phải ký giao dịch hay chờ xác nhận blockchain. Ngược lại, các thao tác list, unlist và transfer được xử lý on-chain nhằm đảm bảo trạng thái niêm yết và quyền sở hữu luôn minh bạch, tránh trường hợp dữ liệu off-chain không phản ánh đúng trạng thái thực tế trên blockchain. Cách tiếp cận này giúp tăng hiệu suất của hệ thống, đồng thời vẫn giữ đúng bản chất "giao dịch có ràng buộc" của một NFT marketplace.

0.2 Ngôn ngữ và nền tảng phát triển

0.2.1 TypeScript trong phát triển ứng dụng web

TypeScript là ngôn ngữ mở rộng từ JavaScript, bổ sung hệ kiểu tĩnh và các cơ chế hỗ trợ xây dựng phần mềm quy mô vừa và lớn như interface, generics, decorator và kiểm tra kiểu ở thời điểm biên dịch **typescript**. Trên thực tế, nhiều hệ thống web chọn TypeScript không phải vì hiệu năng cú pháp, mà vì cần kiểm soát rủi ro: dữ liệu qua API, dữ liệu UI state, và dữ liệu nghiệp vụ thường biến đổi liên tục trong quá trình phát triển; nếu chỉ dựa vào JavaScript thuần, lỗi kiểu dữ liệu thường xuất hiện và khó truy vết.

Trong đồ án, lượng mô hình dữ liệu không hề nhỏ. TypeScript giúp mô tả rõ cấu trúc request/response của API, giảm sai lệch giữa frontend và backend, đồng thời tăng độ an toàn khi xử lý các trường dễ nhầm như địa chỉ ví, chain id, giá, phí và trạng thái giao dịch. Đặc biệt với UI hiển thị lịch sử giá và thống kê, dữ liệu thường là mảng thời gian và phép tổng hợp; việc có kiểu dữ liệu rõ ràng giúp tránh lỗi không mong muốn và giúp quá trình phát triển thống nhất, đồng bộ hơn.

JavaScript thuần là một lựa chọn thay thế hợp lệ, nhất là khi muốn viết nhanh giai đoạn đầu. Tuy nhiên, trong bối cảnh dự án cần sự ổn định, TypeScript làm cho code dễ đọc hơn khi nhiều file cùng tham gia một luồng nghiệp vụ. Vì vậy, TypeScript được chọn làm ngôn ngữ thống nhất cho cả frontend và backend.

0.2.2 Node.js và mô hình I/O bất đồng bộ

Node.js là môi trường chạy JavaScript phía server dựa trên event loop, tối ưu cho các ứng dụng có nhiều tác vụ I/O như gọi cơ sở dữ liệu, gọi API, đọc/ghi file, hoặc giao tiếp mạng. Thế mạnh của Node.js không nằm ở xử lý tính toán nặng, mà ở khả năng xử lý đồng thời nhiều kết nối và nhiều request trong khi phần lớn thời gian là chờ I/O.

Đồ án có đặc điểm tương đối phù hợp với Node.js: số lượng API đọc nhiều (danh sách, chi tiết, search, lịch sử giao dịch, dashboard), còn các tác vụ ghi chủ yếu là cập nhật dữ liệu off-chain sau các hành động của người dùng. Ngoài ra, hệ sinh thái Web3 hỗ trợ Node.js rất tốt; việc tích hợp RPC, ký message, đọc event, và tương tác EVM trở nên thuận lợi nhờ các thư viện phổ biến (chẳng hạn ethers.js). Đây là lý do thực tiễn khiến Node.js phù hợp hơn trong đồ án so với việc dùng công nghệ khác.

Tất nhiên, backend có thể được viết bằng Java (Spring Boot) hoặc .NET; những nền tảng này rất mạnh trong doanh nghiệp và có hệ sinh thái lớn. Tuy nhiên, chọn chúng trong đồ án sẽ làm công nghệ sử dụng trở nên không đồng nhất, tăng thời gian tích hợp, trong khi mục tiêu là hoàn thiện một marketplace có luồng nghiệp vụ rõ ràng. Vì vậy, TypeScript/Node.js là lựa chọn hợp lý về cả kỹ thuật lẫn tiến độ.

0.3 Công nghệ xây dựng giao diện người dùng

0.3.1 React và Next.js

React là thư viện xây dựng giao diện theo hướng thành phần (component), cho phép chia UI thành các khối độc lập, tái sử dụng và quản lý trạng thái theo cách có cấu trúc. Next.js là framework dựa trên React, cung cấp cơ chế routing theo file, tối ưu build, và hỗ trợ nhiều chế độ render (SSR/SSG/CSR) để cân bằng giữa tốc độ tải trang và tính tương tác. Trong nhiều dự án web hiện đại, Next.js được chọn vì nó cung cấp sẵn các tính năng sẵn có: cách tổ chức trang, cách build, cách tối ưu tài nguyên, và cách mở rộng.

Định hướng UI/UX ở Chương 2 yêu cầu giao diện đơn giản hơn các marketplace lớn. Với Next.js/React, giao diện dễ được tổ chức theo component. Việc tách UI theo component giúp duy trì sự nhất quán giữa các trang, ví dụ danh sách và chi tiết có cùng cách trình bày trạng thái của NFT, cùng kiểu hiển thị giá và phí, và cùng chuẩn thông báo lỗi.

Nếu dùng React SPA thuần (chẳng hạn Vite), dự án vẫn có thể hoàn thành, nhưng phải tự quyết nhiều thứ hơn về routing, tổ chức trang và tối ưu build. Angular là một lựa chọn khác, cung cấp framework đầy đủ với DI, routing, form; tuy nhiên

Angular thường nặng hơn và không phổ biến bằng React trong hệ sinh thái Web3. Vue/Nuxt cũng là phương án đáng cân nhắc, nhưng trong phạm vi đồ án, Next.js có lợi thế về mức độ phổ biến và tài liệu, giúp giảm rủi ro khi triển khai và khi viết báo cáo tham khảo.

0.3.2 Tương tác ví và trải nghiệm Web3 trên giao diện

Một trang web thông thường chỉ cần tương tác với trang web chính, còn Web3 thì khác: kết nối ví, ký message để đăng nhập, ký giao dịch để mint, ký giao dịch để list/unlist, rồi chờ mạng xác nhận. Nếu UI/UX không đươc thiết kế tốt, người dùng mới sẽ cảm thấy khó sử dụng và không hiểu luồng nó hoạt động như thế nào. Vì vậy, ngoài việc gọi RPC, giao diện phải thể hiện rõ trạng thái: đang yêu cầu ký, đang chờ xác nhận, đã thành công, hay thất bại và cần thử lại. Đây là phần trực tiếp đáp ứng yêu cầu trải nghiệm ở Chương 2: thao tác quan trọng phải có xác nhận rõ ràng và thông báo minh bạch.

Về mặt kỹ thuật, các thao tác này thường được thực hiện thông qua provider của ví và thư viện tương tác EVM (ví dụ ethers.js). Những thư viện này không tự làm UI; chúng chỉ giúp gọi giao dịch đúng cách. Vì vậy, lựa chọn Next.js/React không chỉ để phát triển giao diện, mà còn để tổ chức các trạng thái UI quanh những điểm cốt lõi để người dùng hiểu rõ hơn: phí gas, thời gian chờ, và các trường hợp người dùng từ chối ký.

0.4 Công nghệ xây dựng backend và API

0.4.1 NestJS và cách tổ chức backend theo module

NestJS là framework Node.js hướng kiến trúc, khuyến khích tổ chức dự án theo module và tách lớp rõ ràng giữa controller, service và data access, đồng thời cung cấp dependency injection để quản lý phụ thuộc. Trong thực tế, NestJS thường được chọn khi đội phát triển muốn tổ chức mã nguồn theo hướng nghiệp vụ dễ mở rộng: thay vì một tập hợp route handler rời rạc, hệ thống được tổ chức theo miền nghiệp vụ và có chuẩn chung cho validation, error handling, auth guard và logging.

Trong đồ án, backend đóng vai trò trung tâm để xử lý nghiệp vụ và điều phối dữ liệu giữa các thành phần của hệ thống. Cụ thể, backend quản lý dữ liệu off-chain trong PostgreSQL, sử dụng Redis để giảm tải cho các truy vấn đọc nhiều, làm việc với S3 để lưu trữ hình ảnh, đồng thời theo dõi và đồng bộ các thay đổi quan trọng từ smart contract lên cơ sở dữ liệu nhằm phục vụ hiển thị và thống kê.

Các luồng nghiệp vụ chính như tạo NFT, niêm yết hay mua bán đều cần backend phối hợp nhiều bước liên tiếp: vừa đảm bảo dữ liệu hiển thị đầy đủ, vừa đảm bảo trạng thái phản ánh đúng kết quả giao dịch on-chain. Vì vậy, nếu mã nguồn không

được tổ chức tốt, các phần xử lý dễ bị làm theo cách phức tạp và khó bảo trì khi mở rộng chức năng hoặc xử lý lỗi. NestJS được lựa chọn nhằm hỗ trợ cấu trúc dự án theo hướng module hoá, giúp tách bạch rõ các phần xử lý và phù hợp với yêu cầu bảo trì, mở rộng.

Express hoặc Fastify là lựa chọn thay thế phổ biến, nhẹ và linh hoạt. Tuy nhiên, dùng nền tảng thuần đồng nghĩa nhóm phải tự đặt chuẩn kiến trúc, tự tổ chức module, tự thống nhất cách trả lỗi và validation; gây ra tốn thời gian và không nhất quán khi dự án lớn dần. Spring Boot cũng là một đối thủ mạnh; nhưng với phạm vi của đồ án và nhu cầu tích hợp Web3 nhanh, NestJS giúp giảm thời gian phát triển và giữ công nghệ sử dụng được thống nhất.

0.4.2 RESTful API và tài liệu hoá API

REST là một phong cách thiết kế API theo hướng tài nguyên, trong đó mỗi loại dữ liệu chính của hệ thống được biểu diễn bằng một nhóm endpoint và được thao tác thông qua các phương thức HTTP quen thuộc; dữ liệu trao đổi thường ở dạng JSON. Cách tiếp cận này được sử dụng rộng rãi vì đơn giản, dễ kiểm thử và phù hợp với ứng dụng web. Trong đồ án, các nhóm chức năng được tổ chức thành các cụm API tương ứng với người dùng, NFT, collection, niêm yết và giao dịch. Phần lớn request từ phía giao diện là các truy vấn đọc để hiển thị danh sách, trang chi tiết, lịch sử giá, lịch sử giao dịch và các số liệu thống kê; bên cạnh đó là các API ghi phục vụ các thao tác off-chain như cập nhật hồ sơ, tạo và chỉnh sửa thông tin NFT/collection, cũng như các luồng đồng bộ dữ liệu sau khi trạng thái on-chain thay đổi.

Do hệ thống có nhiều màn hình và nhiều luồng nghiệp vụ, việc chuẩn hoá và tài liệu hoá API giúp giảm thời gian tích hợp giữa frontend và backend, đồng thời làm cho báo cáo có tính kiểm chứng cao hơn. OpenAPI/Swagger là chuẩn mô tả API phổ biến, cho phép sinh tài liệu và giao diện thử trực tiếp các endpoint. Trong phạm vi đồ án, phần tài liệu này giúp người đọc dễ đối chiếu mô tả chức năng của từng API có trong hệ thống.

Một hướng tiếp cận khác là GraphQL, thường phù hợp khi giao diện cần truy vấn dữ liệu linh hoạt và muốn tự chọn các trường dữ liệu cần lấy. Tuy nhiên, GraphQL đòi hỏi thiết kế schema và cơ chế kiểm soát truy vấn chặt chẽ để tránh ảnh hưởng hiệu năng, đặc biệt với các màn hình danh sách lớn có lọc, sắp xếp và phân trang. Với mục tiêu trình bày rõ ràng và triển khai trong đồ án, REST là lựa chọn phù hợp hơn.

0.5 Cơ sở dữ liệu và quản lý dữ liệu off-chain

0.5.1 PostgreSQL cho dữ liệu nghiệp vụ

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, được phát triển lâu đời và được sử dụng rộng rãi trong các hệ thống web hiện đại **postgresql**. Điểm mạnh của PostgreSQL nằm ở khả năng đảm bảo tính đúng đắn của dữ liệu thông qua giao dịch và các ràng buộc toàn vẹn, đồng thời hỗ trợ SQL tương đối đầy đủ và linh hoạt. Hệ thống cho phép mô hình hoá dữ liệu theo dạng bảng với khoá chính, khoá ngoại, ràng buộc duy nhất và các quy tắc kiểm tra, nhờ đó giảm đáng kể rủi ro sinh dữ liệu sai lệch khi nhiều chức năng cùng đọc/ghi. PostgreSQL cũng nổi bật ở khả năng xử lý đồng thời tốt nhờ cơ chế MVCC, nghĩa là việc đọc và ghi có thể diễn ra song song mà hạn chế tình trạng "chặn" nhau như các cơ chế khoá truyền thống, phù hợp với các ứng dụng có nhiều lượt truy cập.

Về hiệu năng, PostgreSQL cung cấp hệ thống chỉ mục phong phú và nhiều kỹ thuật tối ưu truy vấn. Ngoài chỉ mục B-tree phổ biến cho các truy vấn so sánh và sắp xếp, PostgreSQL còn hỗ trợ các loại chỉ mục khác như GIN/GiST, hữu ích trong những trường hợp tìm kiếm theo tập hợp, dữ liệu bán cấu trúc hoặc truy vấn theo điều kiện phức tạp. Bên cạnh mô hình quan hệ, PostgreSQL cũng hỗ trợ kiểu dữ liệu JSON/JSONB, giúp lưu trữ một số trường dữ liệu linh hoạt khi cần mà vẫn có thể truy vấn và lập chỉ mục ở mức nhất định. Đây là lý do PostgreSQL thường được lựa chọn trong các dự án cần vừa chặt chẽ về quan hệ dữ liệu, vừa có một phần dữ liệu có thể thay đổi theo thời gian.

Trong đồ án, PostgreSQL được sử dụng làm kho dữ liệu chính cho phần off-chain, phục vụ lưu trữ thông tin nghiệp vụ và cung cấp dữ liệu cho các màn hình hiển thị. Nhờ cơ chế truy vấn và tổng hợp tốt, hệ thống có thể trích xuất lịch sử giao dịch theo thời gian để xây dựng lịch sử giá và các thống kê dashboard. Ngoài ra, việc lưu metadata như tên và mô tả ở cơ sở dữ liệu giúp thao tác cập nhật nội dung diễn ra thuận tiện hơn, trong khi các phần ràng buộc sở hữu và trạng thái giao dịch vẫn được đảm bảo bởi smart contract.

MySQL là phương án thay thế trực tiếp và cũng rất phổ biến trong các ứng dụng web. Với các thao tác CRUD cơ bản, cả MySQL và PostgreSQL đều đáp ứng tốt. Tuy nhiên, trong phạm vi đồ án, PostgreSQL được ưu tiên vì thuận lợi hơn khi xây dựng các truy vấn tổng hợp và thống kê, đặc biệt ở các chức năng như dashboard và lịch sử giá theo thời gian.

Một lựa chọn khác là MongoDB, phù hợp khi dữ liệu cần lưu dưới dạng document và cấu trúc thay đổi linh hoạt. Dù vậy, bài toán marketplace thường có nhiều quan hệ rõ ràng giữa các thực thể và nhu cầu tổng hợp dữ liệu theo nhiều chiều xuất hiện

thường xuyên, nên mô hình quan hệ giúp quản lý ràng buộc chặt chẽ hơn và việc truy vấn theo nghiệp vụ cũng dễ kiểm soát hơn.

0.6 Cache và tối ưu hiệu năng

0.6.1 Redis

Redis là một hệ thống lưu trữ dữ liệu dạng key–value hoạt động chủ yếu trên bộ nhớ, vì vậy tốc độ đọc/ghi thường rất nhanh **redis**. Redis được dùng phổ biến trong vai trò cache cho ứng dụng web nhờ cơ chế lưu trữ đơn giản, hỗ trợ đặt thời gian hết hạn cho dữ liệu và có thể giảm đáng kể số lần truy vấn về cơ sở dữ liệu chính. Ngoài cache, Redis cũng hay được tận dụng cho các dữ liệu "ngắn hạn" như nonce phục vụ xác thực, giới hạn tần suất truy cập hoặc các trạng thái tạm trong một khoảng thời gian nhất định, tuỳ theo cách thiết kế hệ thống.

Trong đồ án, Redis được sử dụng nhằm hỗ trợ yêu cầu hiệu năng đã nêu ở Chương 2, đặc biệt với các luồng đọc như danh sách NFT/collection, trang chi tiết và các số liệu thống kê. Đặc trưng của marketplace là số lượt đọc thường lớn hơn nhiều so với số lượt ghi; nếu mỗi lần tải trang đều phải truy vấn và tổng hợp lại toàn bộ từ PostgreSQL thì độ trễ sẽ tăng và hệ thống dễ bị quá tải khi lượng truy cập lớn. Khi có Redis, một số dữ liệu được truy cập lặp lại có thể được lưu tạm trong cache và phục vụ nhanh hơn; dữ liệu sẽ được làm mới theo thời gian hết hạn hoặc theo cơ chế xoá cache khi có cập nhật. Nhờ đó, thời gian phản hồi ổn định hơn và trải nghiệm người dùng cũng mượt hơn.

Về hướng tiếp cận thay thế, hệ thống có thể cache trực tiếp trong bộ nhớ của ứng dụng. Cách này đơn giản nếu chỉ chạy một instance, nhưng khi cần mở rộng nhiều instance thì cache bị phân tán và dễ không nhất quán. Memcached cũng là một lựa chọn phổ biến cho cache thuần, tuy nhiên Redis thường linh hoạt hơn và được hỗ trợ tốt trong hệ sinh thái Node.js, đồng thời tài liệu và cộng đồng cũng đông đảo giúp cho phần sửa lỗi trong quá trình phát triển dễ dàng hơn.

0.7 Lưu trữ nội dung số

0.7.1 Amazon S3 và cơ chế presigned URL

Amazon S3 là dịch vụ lưu trữ đối tượng của AWS, được thiết kế để lưu trữ dữ liệu dạng file với khả năng mở rộng lớn và độ bền cao **s3**. S3 phù hợp với các nội dung tĩnh như hình ảnh vì cho phép tách phần lưu file ra khỏi tầng dữ liệu nghiệp vụ. Nếu lưu ảnh trực tiếp trong cơ sở dữ liệu, dung lượng và chi phí vận hành sẽ tăng, việc sao lưu hoặc khôi phục cũng nặng hơn, trong khi dữ liệu ảnh vốn không cần được xử lý như dữ liệu quan hệ. Với một marketplace, hình ảnh là yếu tố hiển thị quan trọng, nhưng về bản chất nên được đặt ở một lớp lưu trữ chuyên biệt để hệ thống gọn và dễ vận hành hơn.

Trong đồ án, S3 được dùng để lưu ảnh của NFT và collection. Quy trình upload được triển khai theo hướng phổ biến là sử dụng presigned URL: backend tạo một đường dẫn tải lên có thời hạn, frontend dùng đường dẫn này để upload trực tiếp lên S3, sau đó hệ thống lưu lại object key hoặc đường dẫn ảnh trong cơ sở dữ liệu. Cách làm này giúp backend tránh phải xử lý file dung lượng lớn, đồng thời hạn chế rủi ro bảo mật vì phía client không cần nắm giữ thông tin truy cập AWS. Nhìn tổng thể, đây là lựa chọn phù hợp với yêu cầu về hiệu năng và bảo mật đã đặt ra trong Chương 2.

Về phương án thay thế, IPFS thường được nhắc đến trong các hệ thống Web3 vì lưu trữ theo hướng phi tập trung và tham chiếu nội dung bằng mã băm. Tuy nhiên, để đảm bảo dữ liệu luôn sẵn sàng, IPFS thường cần thêm cơ chế pinning và việc vận hành có thể phức tạp hơn trong phạm vi đồ án. Một số dịch vụ như Cloudinary hỗ trợ xử lý ảnh rất tiện, nhưng sẽ làm hệ thống phụ thuộc vào nhà cung cấp bên ngoài. Với định hướng triển khai đơn giản trên AWS và mục tiêu tập trung vào nghiệp vụ marketplace, S3 là phương án cân bằng và dễ triển khai.

0.8 Blockchain và Smart Contract

0.8.1 Solidity, EVM và chuẩn NFT

Solidity là ngôn ngữ được dùng phổ biến để phát triển smart contract trên các blockchain tương thích EVM. EVM có thể hiểu như một môi trường chạy chương trình được chuẩn hoá: cùng một đoạn mã hợp đồng sẽ được thực thi theo cùng một quy tắc trên mọi nút mạng, và kết quả được toàn mạng xác nhận. Nhờ đó, các quy tắc về quyền sở hữu và điều kiện giao dịch không còn phụ thuộc vào một máy chủ trung tâm, mà được bảo đảm bởi cơ chế đồng thuận của blockchain. Đây cũng là lý do các ứng dụng gắn với tài sản số thường đặt phần logic cốt lõi lên smart contract, vì mọi thao tác đều để lại dấu vết công khai và có thể kiểm chứng.

Trong bài toán NFT, các chuẩn token như ERC-721 đóng vai trò như một bộ quy ước chung để mô tả cách mint, transfer và truy vấn chủ sở hữu của từng token. Khi tuân theo chuẩn, hợp đồng sẽ tương thích tốt hơn với hệ sinh thái xung quanh như ví, trình duyệt blockchain, cũng như các thư viện kết nối từ ứng dụng web. Việc chuẩn hoá này giúp giảm đáng kể chi phí tích hợp, vì frontend và backend có thể dựa vào các hàm và sự kiện đã được quy ước sẵn và có thể áp dụng cho nhiều dự án khác nhau.

Trong đồ án, smart contract được triển khai trên Binance Smart Chain Testnet, phù hợp cho mục đích thử nghiệm và trình diễn do chi phí thấp và dễ triển khai. Từ góc nhìn thiết kế, điểm quan trọng là những hành động mang tính ràng buộc tài sản cần có khả năng kiểm chứng công khai. Vì vậy, các thao tác như list, unlist và

transfer được đặt trên smart contract để trạng thái niêm yết và quyền sở hữu luôn bám theo logic on-chain, giảm phụ thuộc vào việc server tự gán trạng thái. Cách làm này giúp mô hình giao dịch rõ ràng hơn: khi người dùng nhìn thấy một NFT đang được niêm yết, trạng thái đó phản ánh đúng điều kiện trong hợp đồng; khi giao dịch mua bán hoàn tất, quyền sở hữu được cập nhật trực tiếp trên blockchain và có thể đối chiếu lại bằng các công cụ của mạng.

Trong thực tế, nhiều marketplace còn áp dụng cơ chế tối ưu hơn như off-chain order hoặc lazy listing, tức là người bán chỉ ký một lệnh bán và chỉ tạo giao dịch on-chain khi có người mua khớp lệnh. Cách tiếp cận này giảm số lần phải trả phí cho list hoặc unlist và hỗ trợ giao dịch nhanh, nhưng đi kèm độ phức tạp cao hơn: cần thiết kế chữ ký, chống replay, quản lý danh sách lệnh và xử lý các tình huống cạnh tranh khi nhiều người mua cùng nhắm vào một NFT. Với phạm vi đồ án hướng đến luồng nghiệp vụ mạch lạc và dễ trình bày, lựa chọn đưa list và unlist lên on-chain giúp hệ thống đơn giản hơn, đồng thời vẫn giữ đúng tính chất giao dịch có ràng buộc của marketplace.

0.9 Xác thực và bảo mật

0.9.1 SIWE (EIP-4361) cho đăng nhập bằng ví

SIWE (Sign-In With Ethereum) là một chuẩn đăng nhập bằng ví dựa trên cơ chế ký số, được đặc tả trong EIP-4361. Thay vì tạo tài khoản bằng mật khẩu, người dùng chứng minh mình thực sự sở hữu một địa chỉ ví bằng cách ký một thông điệp đăng nhập. Thông điệp này thường chứa các thông tin như tên miền của ứng dụng, địa chỉ ví, một giá trị nonce và một số trường mô tả phiên làm việc. Sau khi nhận chữ ký, phía server kiểm tra tính hợp lệ của chữ ký để xác nhận người đang đăng nhập chính là chủ sở hữu của địa chỉ ví đó. Nonce đóng vai trò quan trọng để tránh việc chữ ký cũ bị kẻ khác lấy lại và dùng lặp lại cho một lần đăng nhập khác.

Trong đồ án NFT marketplace, đăng nhập bằng ví là nền tảng vì hầu hết chức năng gắn với danh tính theo địa chỉ ví, chẳng hạn tạo NFT/collection, quản lý tài sản và xem dashboard cá nhân. SIWE được lựa chọn vì nó đưa ra một khuôn dạng thông điệp thống nhất, giúp quá trình xác thực rõ ràng và giảm rủi ro sai sót so với việc tự đặt ra format ký tuỳ ý. Ở mức triển khai, nonce có thể được lưu tạm để đảm bảo mỗi nonce chỉ sử dụng một lần, từ đó đáp ứng yêu cầu bảo mật đã nêu ở Chương 2. Đồng thời, cách đăng nhập này cũng phù hợp với trải nghiệm Web3: người dùng chỉ cần kết nối và ký xác nhận bằng ví, không phải tạo thêm mật khẩu hay ghi nhớ thông tin đăng nhập riêng.

Nếu dùng OAuth2 như đăng nhập bằng Google hoặc Facebook, người dùng có thể thấy quen thuộc hơn, nhưng cơ chế này không chứng minh được quyền sở hữu

địa chỉ ví, trong khi đây lại là yếu tố cốt lõi của marketplace. Một hướng khác là tự thiết kế cơ chế ký message, tuy nhiên dễ phát sinh sai sót ở những chi tiết quan trọng như nonce, ràng buộc theo domain hoặc cách chuẩn hoá nội dung thông điệp. Vì vậy, SIWE là lựa chọn phù hợp để vừa đúng bản chất xác thực bằng ví, vừa dễ trình bày và kiểm chứng.

0.9.2 JWT (RFC 7519) để bảo vệ API

JWT (JSON Web Token) là một chuẩn token được mô tả trong RFC 7519. Token có cấu trúc gọn, mang theo một số thông tin định danh dưới dạng JSON và đi kèm chữ ký số để phía server có thể kiểm tra xem dữ liệu có bị sửa trong quá trình truyền hay không. Một ưu điểm thường được nhắc đến của JWT là tính stateless: thay vì phải lưu thông tin phiên cho từng người dùng trên server, hệ thống có thể dựa vào token mà client gửi kèm để xác thực và phân quyền cho mỗi request. Cách làm này phù hợp với mô hình API hiện đại vì đơn giản hoá tầng backend và dễ mở rộng khi lượng truy cập tăng.

Trong đồ án, SIWE được dùng như bước xác thực ban đầu dựa trên chữ ký ví. Khi chữ ký hợp lệ, backend phát hành JWT để frontend sử dụng trong các request tiếp theo tới những API cần quyền truy cập. Luồng kết hợp này khá tự nhiên: người dùng chỉ cần ký một lần để đăng nhập, sau đó có thể thực hiện các thao tác off-chain như cập nhật hồ sơ, chỉnh sửa metadata, xem dashboard mà không phải ký lại cho từng thao tác nhỏ. Ngược lại, các hành động tạo giao dịch thật trên blockchain như mint, list hoặc buy vẫn yêu cầu người dùng xác nhận trong ví. Nhờ vậy, hệ thống vừa giữ đúng nguyên tắc an toàn cho các thao tác ràng buộc tài sản, vừa tránh làm trải nghiệm bị nặng nề vì phải ký quá nhiều lần.

Một phương án khác trong các hệ thống web truyền thống là xác thực theo session: server lưu session và client giữ cookie. Session có lợi thế ở chỗ dễ thu hồi và quản lý tập trung, nhưng sẽ phát sinh thêm phần quản lý trạng thái phía server, đặc biệt khi triển khai nhiều instance hoặc cần cơ chế chia sẻ session. Với định hướng triển khai gọn và mô hình API của đồ án, JWT là lựa chọn phù hợp vì nhẹ, phổ biến và dễ tích hợp với NestJS.

0.10 Triển khai và vận hành(cái này cần update lại, hiện tại đang chưa làm)

0.10.1 Docker và định hướng triển khai trên AWS

Docker là nền tảng container hoá, cho phép đóng gói ứng dụng và phụ thuộc thành image để chạy nhất quán ở nhiều môi trường. Trong thực tế phát triển, Docker giúp giảm các lỗi do lệch phiên bản runtime, thư viện và cấu hình. Với đồ án, việc tái lập môi trường nhất quán rất quan trọng: triển khai demo cần ổn định và ít phụ thuộc "máy cài thế nào".

Trong định hướng triển khai, hệ thống có thể đặt frontend và backend lên một compute service (EC2 hoặc dịch vụ container), cơ sở dữ liệu PostgreSQL có thể dùng dịch vụ quản lý như RDS để giảm công vận hành, Redis dùng ElastiCache để ổn định và dễ cấu hình, còn ảnh được lưu trên S3. Đây là mô hình triển khai khá điển hình và dễ giải thích, phù hợp yêu cầu "cấu hình có thể thay đổi mà không cần sửa mã nguồn" ở Chương 2 thông qua biến môi trường và cấu hình triển khai. Kubernetes là một lựa chọn mạnh khi cần điều phối cụm container và tự động mở rộng, nhưng vượt nhu cầu của đồ án và làm tăng độ phức tạp. Vì vậy, Docker được xem như công cụ đóng gói và tái lập môi trường, còn triển khai AWS theo hướng tối giản để đảm bảo hệ thống chạy ổn định.

0.11 Kết luận chương

Chương này đã làm rõ hướng tiếp cận công nghệ của đồ án và cách các lựa chọn kỹ thuật bám sát các yêu cầu đã nêu ở Chương 2. Trọng tâm của hệ thống là cân bằng giữa tính tin cậy của các thao tác gắn với blockchain và tính linh hoạt, hiệu năng của phần dữ liệu và trải nghiệm trên web. Từ nền tảng đó, chương tiếp theo sẽ trình bày về kết quả thực nghiệm của hệ thống.