Gatino - Innovation Gateway for Swinburne Vietnam

System Architecture Design and Research Report

GROUP 1

Name	Position	Email	Phone
Ta Quang Tung	Team Leader, Client Liaison	104222196@student.swin.edu.au	0921426803
Nguyen Quang Huy	Development Manager	104169507@student.swin.edu.au	0968751524
Tran Hoang Hai Anh	Support Manager	104177513@student.swin.edu.au	0866899518
Phan Sy Tuan	Quality Manager	104072029@student.swin.edu.au	0888052003
Duong Quang Thanh	Planning Manager	104167828@student.swin.edu.au	0976663084

COS40006, COMPUTING TECHNOLOGY PROJECT B, SPRING 2025

Document Change Control

Version	Date	Authors	Summary of Changes
1.0	01/11/2024	Ta Quang Tung, Nguyen Quang Huy, Tran Hoang Hai Anh, Phan Sy Tuan, Duong Quang Thanh	Initial version of the document.
2.0	6/4/2025	Ta Quang Tung, Duong Quang Thanh	Updated the document to better reflect the process of CTPB.

Document Sign Off

Name	Position	Signature	Date
Ta Quang Tung	Team Leader, Client Liaison	Quang Tung Ta	6/4/2025
Nguyen Quang Huy	Development Manager	Quang Huy Nguyen	6/4/2025
Tran Hoang Hai Anh	Support Manager	Hoang Hai Anh Tran	6/4/2025
Phan Sy Tuan	Quality Manager	Sy Tuan Phan	6/4/2025
Duong Quang Thanh	Planning Manager	Quang Thanh Duong	6/4/2025

Client Sign Off

Name	Position	Signature	Date
Dr. Le Minh Duc	Head of IT Department, Swinburne Vietnam Head of Innovation Lab, Swinburne Vietnam	Le, Minh Duc	6/4/2025
Organization: Innovation Lab, Department of Computer Science, Swinburne Vietnam			

Table of contents

1 - Introduction

1.1 - Overview

This document outlines the system architecture design and research conducted to support the development of *Gatino - Innovation Gateway for Swinburne Vietnam*. Gatino is a platform designed to manage, search, and demonstrate student projects across various domains, such as software development, Internet of Things (IoT). The platform aims to simplify the process of searching and showcasing these innovation projects, making them accessible to students, faculty, and potential industry partners.

The purpose of this document is to present the high-level architecture for Gatino, supported by relevant research on technologies and methodologies that align with project requirements and performance goals. This document is intended for the development team, system architects, and stakeholders within the Swinburne University Vietnam Software Lab to facilitate a shared understanding of the architectural choices and support informed decision-making as the system evolves.

1.2 - Definitions, Acronyms and Abbreviations

Acronym/Abbreviations	**Meaning**
IoT	Internet of Things
CNC	Component and Connector

2 - Problem Analysis

2.1 - System Goals and Objectives

Goal

Gatino's goal is to improve how students and instructors at Swinburne Vietnam organize and package innovative ideas for future reuse and exhibition. Gatino is now unable to perform this goal because it lacks project search capacity and has limited demonstration experience. The purpose of this initiative is to overcome these weaknesses and assist Gatino in achieving its objective.

Objectives

- Allow users to search for projects on the platform and apply particular search criteria for more granular results.
- Provide users with a seamless and robust project demonstration experience in which they can engage with and observe projects in real time.
- Allow users to use the search and demonstration tools from any location, using their browsers.
- Integrate the search and demonstration functions seamlessly with the rest of the Gatino system to make development and maintenance easier.

2.2 - Assumptions

- **Stable requirements:** We presume that the client's project requirements are stable and will not change significantly throughout the development process. Any significant changes to the requirements may affect the project timeframe.
- **Resource availability:** We expect that team members assigned to certain tasks will be available as scheduled. However, we recognize that unanticipated situations (such as illness or other crises) sometimes impact availability. To prevent this, we have assigned backup team members to important duties.
- **External dependencies:** Our project plan is based on the timely supply of external dependencies. Specifically, for client feedback, we rely on frequent feedback from clients to assess our work and make required changes. Delays in receiving client input may have an influence on succeeding phases. For the core Gatino team, the completion of the upload project functionality by the core Gatino team is critical to our integration phase. We presume that their development is consistent with our timeframe.
- **Infrastructure and environment:** We presume that the development, testing, and production environments will be configured as anticipated. Any delays in infrastructure provisioning might disrupt our timeline.

2.3 - Simplifications

Demo feature:

- As the types of projects that students develop can be very wide-ranging, supporting all of them in our system is not possible. Therefore, our system will limit the scope to the following demonstration types:
    - Web (full-stack) project demonstrations.
    - Physical device (Android) demonstrations for Android projects.
    - Data-streaming project demonstrations (for IoT projects, for example).
- We will try our best to support projects that can be adjusted to one of these types. Projects that are too different from these types will not be supported.

3 - High–Level System Architecture and Alternatives

3.1 - System Architecture

The system has been designed to satisfy the functional and non-functional requirements specified in the System Requirements Specifications. Refer to the Appendix to see the list of these requirements.

We have chosen the ***client-server architecture*** for this project because the client envisions the system to be a web application with a separate frontend and backend. Using this architecture makes the most sense because the frontend naturally maps to the client side and the backend maps to the server side. In addition, because the core Gatino team is also using this architecture, it satisfies the **Analyzability** requirement.

The server side makes use of a number of patterns and architectures, as shown by the CNC diagram. The server follows the ***service-oriented architecture***, breaking down the complex backend into a collection of services that work together to fulfill the required functions. These services are independently deployable and can be replicated for availability, enhancing **Replaceability** and **Fault Tolerance**, respectively. The ***API gateway pattern*** is used to connect the client to the different services on the backend. This effectively hides the backend services from the client while also allowing **Security** mechanisms such as rate-limiting and request filtering to be enforced. The decision to deploy many of the backend services on Docker, as shown in the Deployment diagram, allows the system to achieve the **Adaptability**, **Co-existence**, and **Replaceability** requirements.

The chosen architecture has been designed to adequately all the functional requirements. The following section describes how the included components fulfill each feature:

- **Search for products (Task 1.1):** The user enters the search query on **WebInterface**. **WebInterface** then sends the request to the **ReverseProxy**, which forwards it to the **ProductManager** and in turn to the **SearchService**. **SearchService** requests **SearchEngine** to return the matching products' metadata, then it returns this information to **ProductManager** to query the **MainDatabase**. **ProductManager** then returns this result to the user.
- **Add/update/delete product search information (Tasks 1.2 to 1.4):** The user makes a request to add/update/delete a product through the **WebInterface**. The **ReverseProxy** receives this request and forwards it to the **ProductManager**. The **ProductManager** updates this product in the **ProductDatabase** and invokes **SearchService** to reflect this new information in the **SeachDatabase**. In this process, **SearchService** will invoke **SearchDataExtractor** to extract the new search data.
- **Run a demonstration of a web product (Task 2.1):** The user accesses the URL to the web product through the **WebInterface**. The request goes through the **ReverseProxy**, then **ProductManager**, and to the **APIGateway**. **APIGateway** correctly routes the request to the corresponding **WebProductDemonstrator**. Each **WebProductDemonstrator** will be deployed as a Docker container as shown in the

Deployment diagram. This component will correspond to the student product and will handle the request based on how they programmed it.

- **Run a demonstration of a phyiscal Android product (Task 2.2):** The user accesses the URL to the Android product through the **WebInterface**. The request goes through the **ReverseProxy**, then **ProductManager**, and to the **APIGateway**. **APIGateway** correctly routes the request to the **PhysicalMobileViewer**. The viewer will request the **PhysicalMobileConnector** component to make the connection to the **PhysicalDemonstrator**, which is the Android device that runs the product.
- **Run a demonstration of an IoT product (Task 2.3):** These types of products are expected to produce data streams. These products typically have external sensors that operate outside of the Gatino system, but they should send the recorded data to the server to be demo-able. The data is sent directly to the **DataStreamManager**, which queries the **SecurityDatabase** to check if the product has permission to produce the stream. To run a demo of this product, the user accesses its URL through the **WebInterface**. The request goes through the **ReverseProxy**, then **ProductManager**, and to the **APIGateway**. **APIGateway** correctly routes the request to the corresponding **HeadlessDemonstrator**. This component is responsible for pulling data from the **DataStreamManager** and using it to create a visual (and possibly interactive) demonstration for the user. **HeadlessDemonstrators** will typically be developed by the owners of the IoT products.

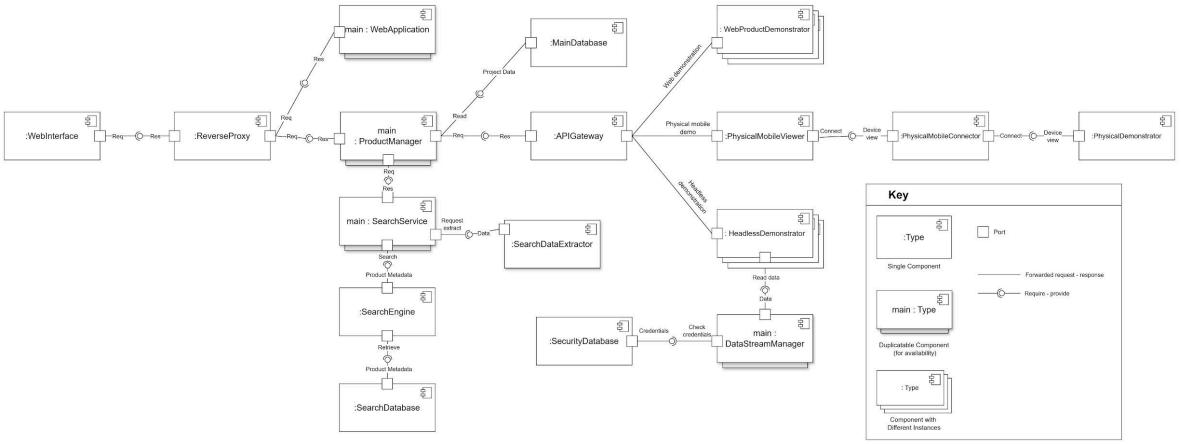


Fig. 1: The component-and-connector view of Gatino's architecture. It is loosely based on UML component diagram notation (Fakhroutdinov, n.d.). The full image can be seen here.

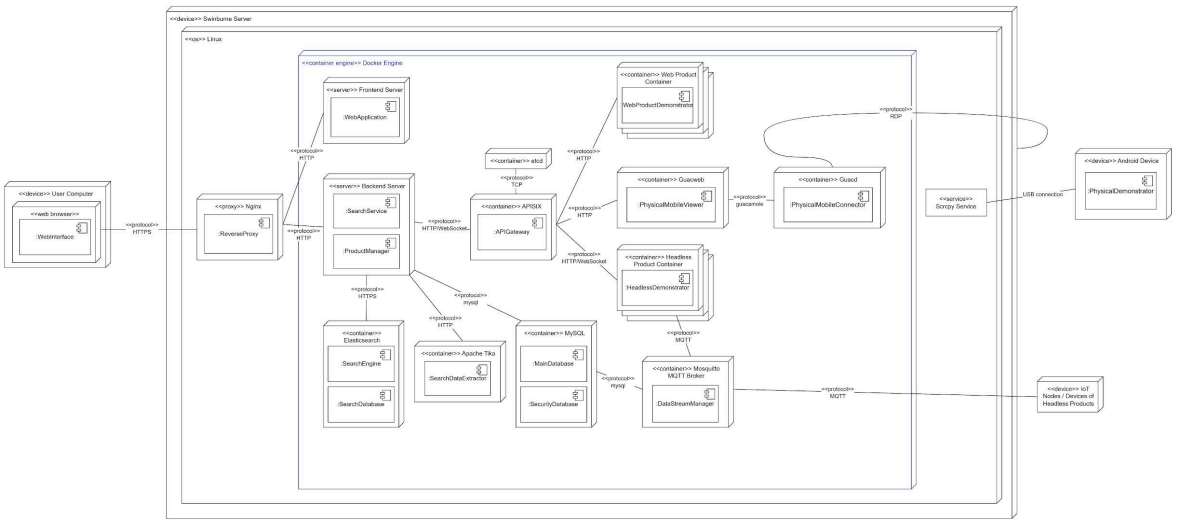


Fig. 2: The deployment allocation view of Gatino's architecture. It is loosely based on UML deployment diagram notation (Fakhroutdinov, 2016). The full image can be seen here.

3.2 - Other Alternative Architectures Explored

In designing Gatino's architecture, two alternative approaches were considered:

1. **Monolithic Architecture**:

   This approach would encapsulate all system functionalities—search, product management, and demonstration—in a single, unified application. The monolithic design is straightforward and simplifies deployment and testing by consolidating all features within one codebase.

   Challenges: This structure is less flexible for updates, as any modifications would require the redeployment of the entire application. Moreover, scaling specific features, such as the search or demo functionalities, is challenging in a monolithic architecture due to interdependencies and limited modularity.

   Rationale for Rejection: Gatino's features have distinct resource requirements and user demand levels. The monolithic approach lacks the modularity needed for Gatino's diverse features, making it unsuitable for a dynamic platform that requires scalability, maintainability, and efficient resource allocation.

2. **Serverless Architecture with Managed Cloud Services**:

   This option would utilize a serverless architecture with managed cloud services, such as AWS Lambda or Google Cloud Functions, to handle each feature independently. Search, product demos, and user management could each run as individual functions, activated only when required, thereby minimizing idle resource costs.

   Challenges: This design offers scalability and efficient cost management but introduces latency in cold starts, particularly detrimental for real-time demo

functionalities. Additionally, a serverless approach introduces complexities in maintaining consistent state management across functions, making data retrieval slower and more complex.

Rationale for Rejection: Although serverless architecture reduces costs, the latency it introduces and challenges with state management make it less practical for Gatino, where quick response times and smooth user interactions are essential.

4 - Research and Investigations

4.1 - Research into Application Domain

Innovation gateways

Before developing Gatino, we explored several systems with similar functionalities, focusing on how they manage student innovation products. Among these, the Innovation Center by HUST stood out as closely aligned with our goals. This system showcases each product on a dedicated page, allowing users to interact and test the product instantly with a single button click. After careful inspection, we conclude that Gatino can adopt similar showcase features, including dedicated product pages with interactive elements, so users can easily engage with products in real-time. Below is an image of its product showcase page.

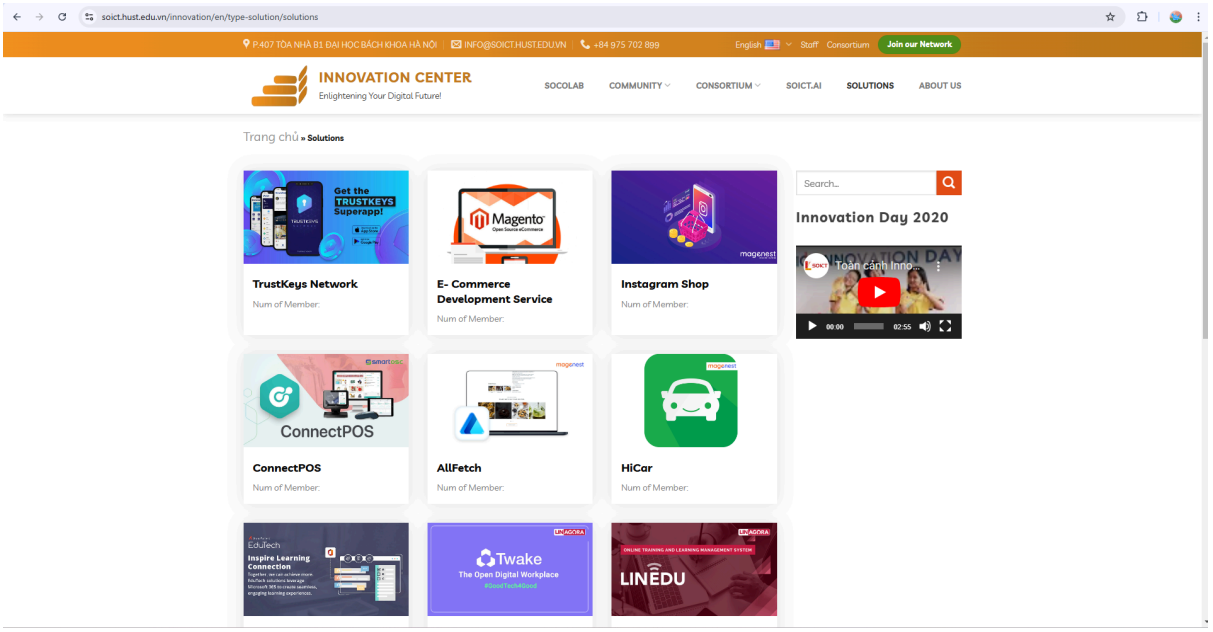


Fig 3: The innovation products on display with Innovation Center

Besides HUST Innovation Center, we have also conducted some further investigations into other similar innovation gateways locally and globally.

Globally, most innovation systems are found in universities. For example, the national innovation system for automation devices (*National Drone Innovation Gateway*, no date), co-managed by Cranfield University (UK), the Neuron Innovation Fund, and Ebeni, supports innovators and other stakeholders (including designers, manufacturers, operators, and integrators) in the ecosystem for automation and aerial mobile devices. Besides that, we also investigated the innovation systems at The Pennsylvania State University ('Innovation Gateway', no date) and the University of Delaware (*Innovation Gateway | University of Delaware*, no date), which serve as hubs that connect industry partners seeking expert knowledge to address business challenges with the universities' networks across various

research fields. Through those innovation gateways, universities enhance learning experiences and help students build strong problem-solving skills and networking.

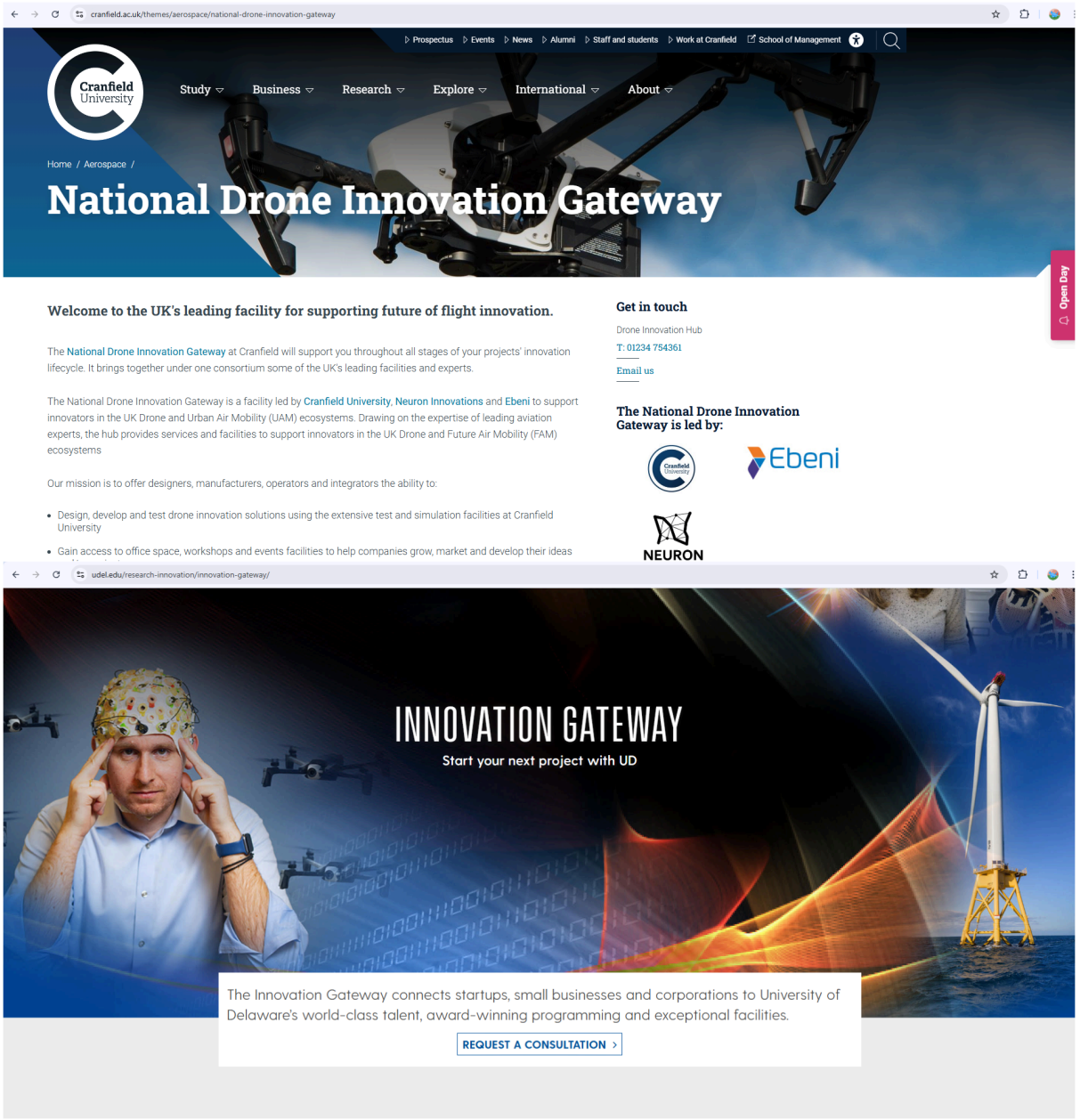


Fig 4: Existing innovation products by Cranfield and UD

In Vietnam, there have been existing systems organized by the government such as National Innovation Center (*Trung tâm Đổi mới sáng tạo Quốc gia | Vietnam National Innovation Center*, no date) and National Startup Support Center ('NSSC – Trung tâm hỗ trợ khởi nghiệp sáng tạo Quốc gia – Trung tâm hỗ trợ khởi nghiệp sáng tạo Quốc gia', no date) as well as innovation systems owned by corporation and university, namely Sunwah Innovation Center (*Sunwah Innovation - Office Space, Virtual Office and Workspace to Rent*, no date) and HueUni Innovation Gateway (*CEI-HUEUNI*, no date). However, according to the analysis of the study, innovation systems in Vietnam primarily function at the level of providing information about innovative products to stakeholders. Based on published

information, these systems haven't demonstrated the capability to manage the innovation process by itself.

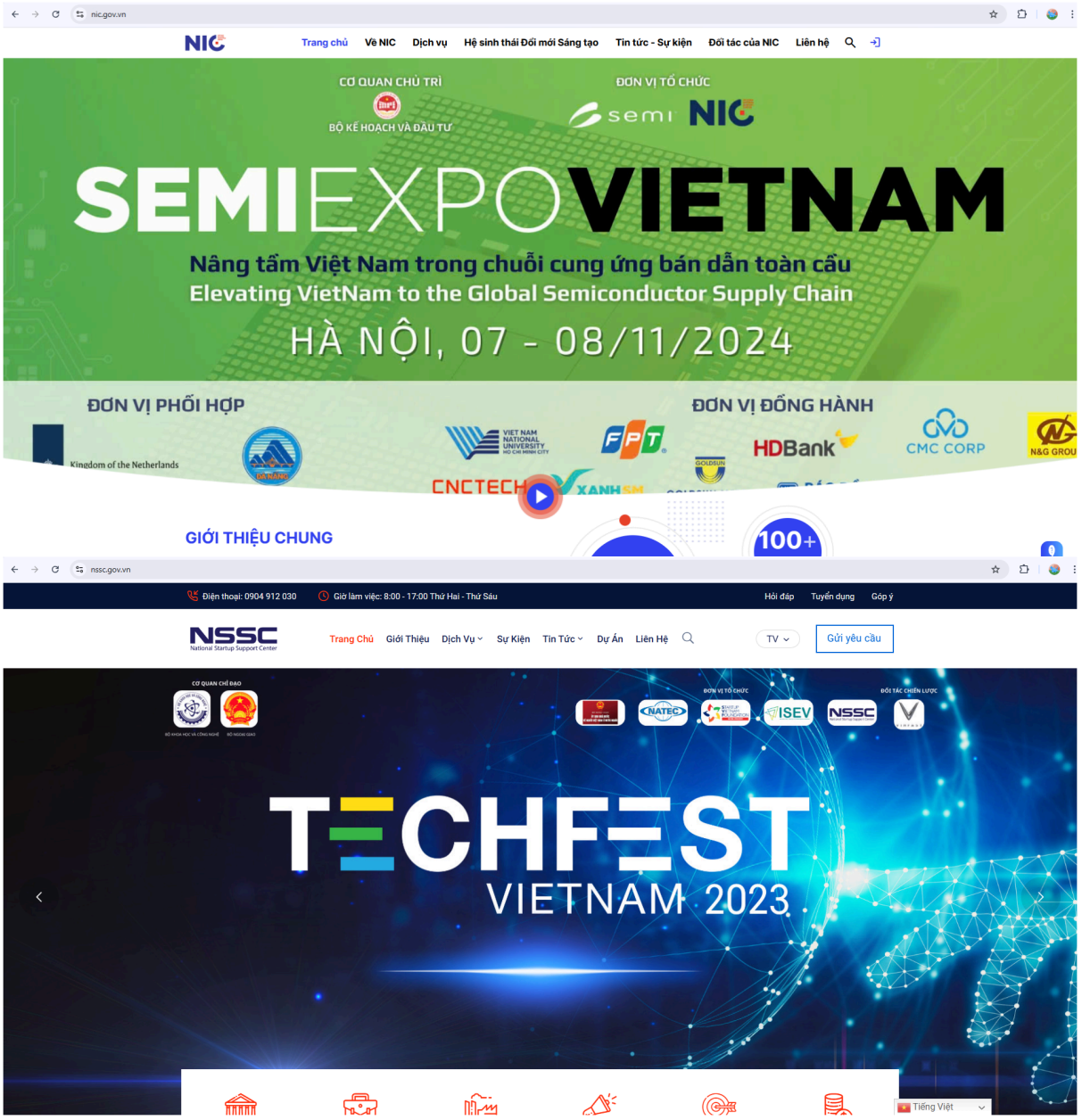


Fig 5: Existing national innovation products

Based on those existing systems, we concluded that Innovation management systems in Vietnam and around the world share some key features, including tasks like connecting stakeholders, training, consulting, research, and development, as well as advertising innovation products. However, these systems have certain limitations: they often lack functions for effectively showcasing products and do not support the integration of existing products into new ones, which limits reusability. Based on the strengths and limitations of current innovation management systems, Gatino can learn from existing ones by bridging the gap between students' innovation products with external business partners. Moreover, Gatino should include efficient web-based management of innovative products, allowing users to create, edit, delete, and search for products and support product demonstrations.

Search engines

To enrich the quality of our search module, we have conducted additional research on query-based searching by examining Google and Scopus's advanced searches. By analyzing how these search engines construct and process complex queries, we may delve into query parsing techniques, which can inform and potentially enhance our search mechanism.

Google

Google is a web search engine widely used for finding content across the internet. It supports both keyword-based searches and refined search options via filters for date, location, language, file type, and region, which can be accessed on the advanced search page. Google also allows Boolean operators like AND, OR, and NOT (or the minus sign - before a keyword) to refine searches further. Using an asterisk * enables a wildcard search for terms in between phrases. Wrapping keywords in quotation marks “ “ensures exact matches for phrases, enhancing precision. The image below shows the advanced search of Google in action and all of the options that we can modify to yield the most optimal results.

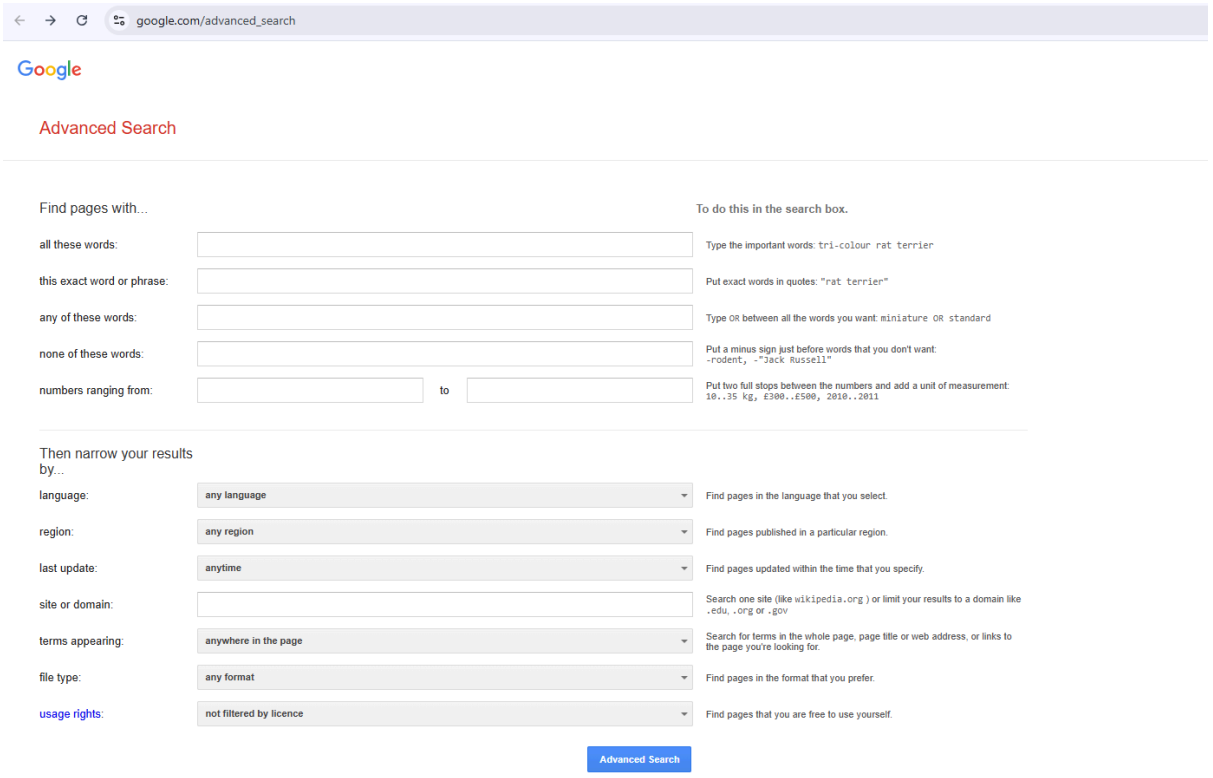


Fig 4: The advanced search of Google

Scopus

Scopus is a research-oriented search engine that indexes peer-reviewed articles, conference papers, and book chapters. Like Google, Scopus supports Boolean operators, but it also provides the added functionality of Field Codes, which are advanced search tags that allow users to filter specific attributes (e.g., author name, journal name, and publication year). The advanced search interface includes selectable fields, making it easy for users to construct queries. However, as operators are nested, search queries can become complex,

requiring careful syntax management. The image below shows the advanced search of Scopus in action and how complex the nested query can become.

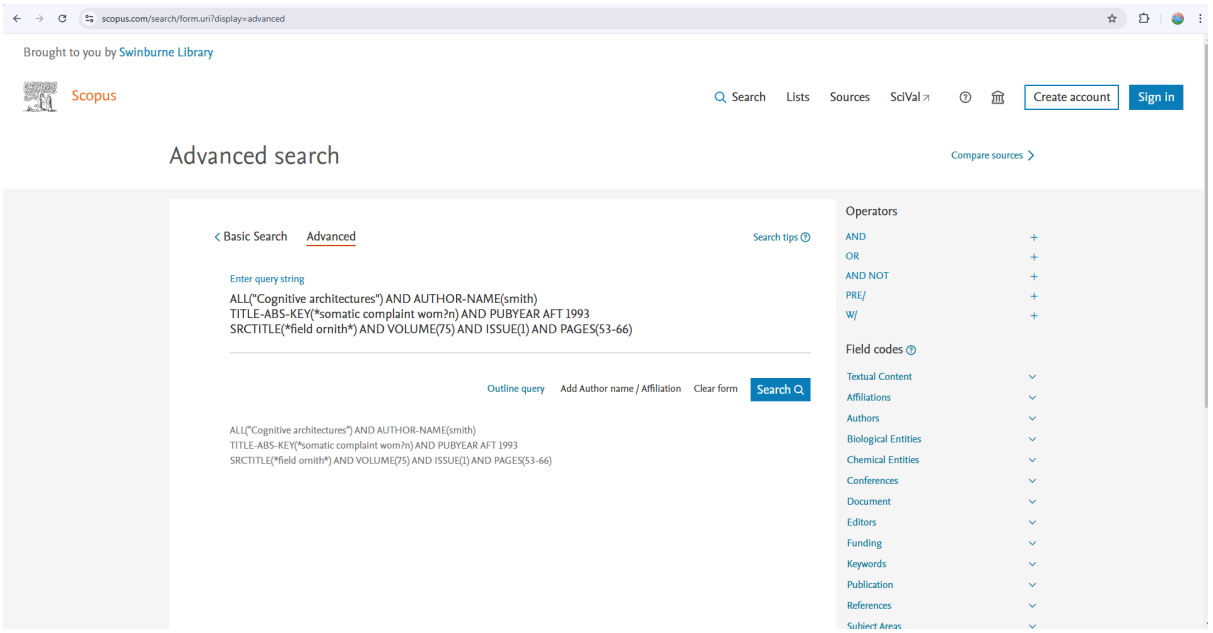


Fig 5: The advanced search of Scopus

Web product demonstrations

Our research into web product demonstrations has focused primarily on platforms like WordPress and Envato. These sites offer rich catalogs of themes and templates, each accompanied by live demos that showcase the design and functionality of the products. This interactive feature allows potential buyers to experience the look and feel of a template in real time, giving insight into how the design elements can be utilized for various types of products.

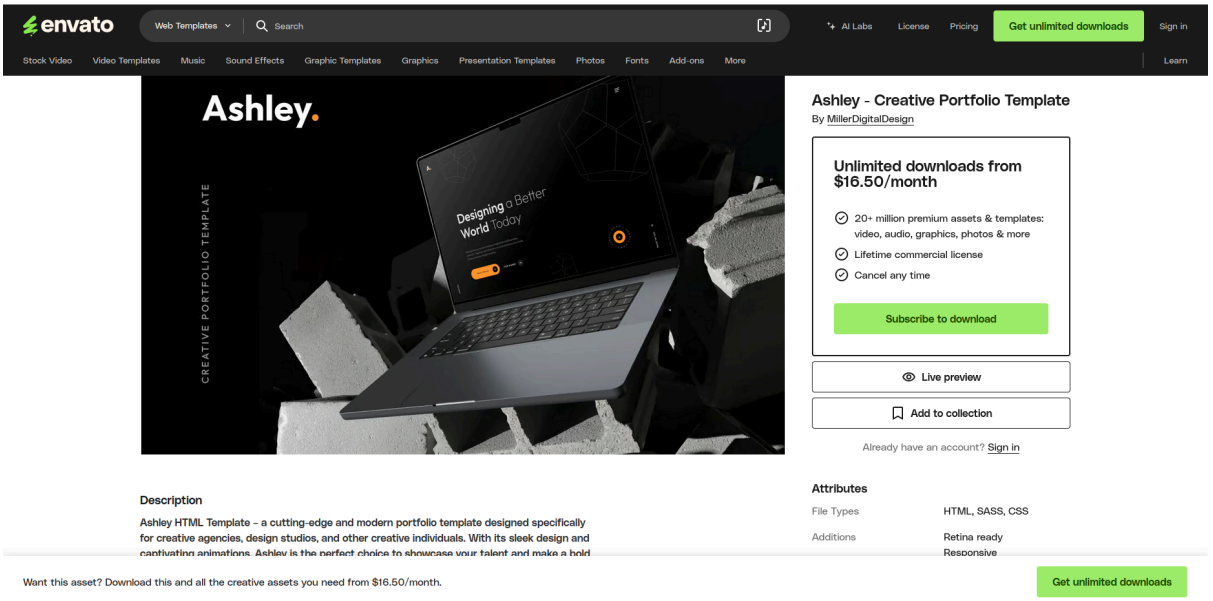


Fig. 6: A template description page on Envato. It provides detailed information about the template and offers the option to run a live preview.

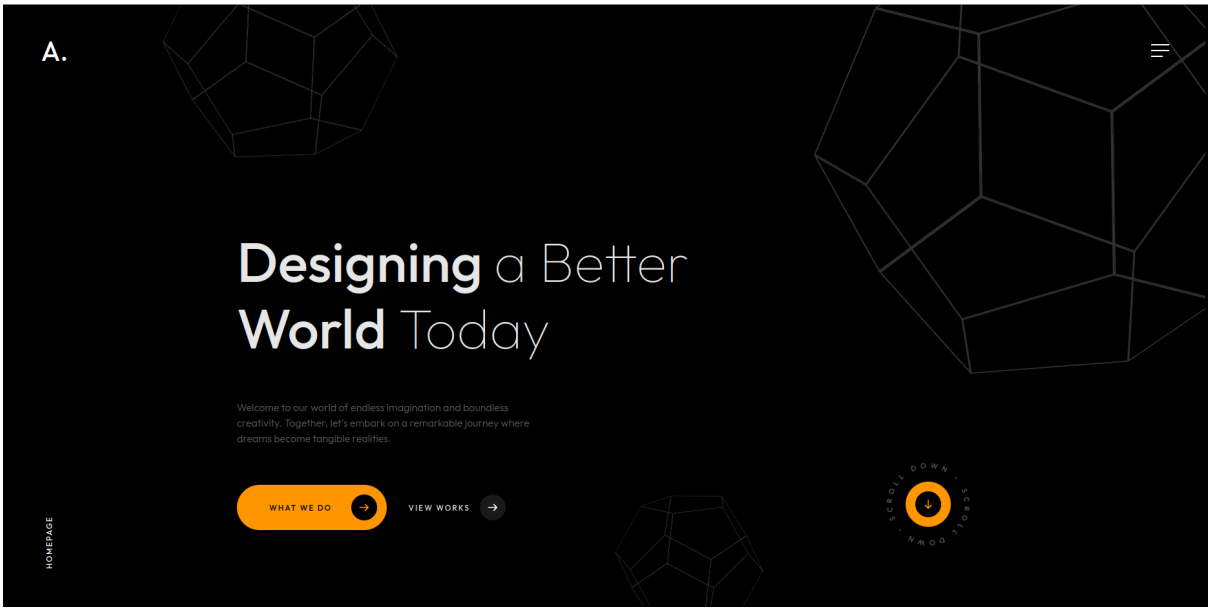


Fig. 7: The live demo of the above Envato template.

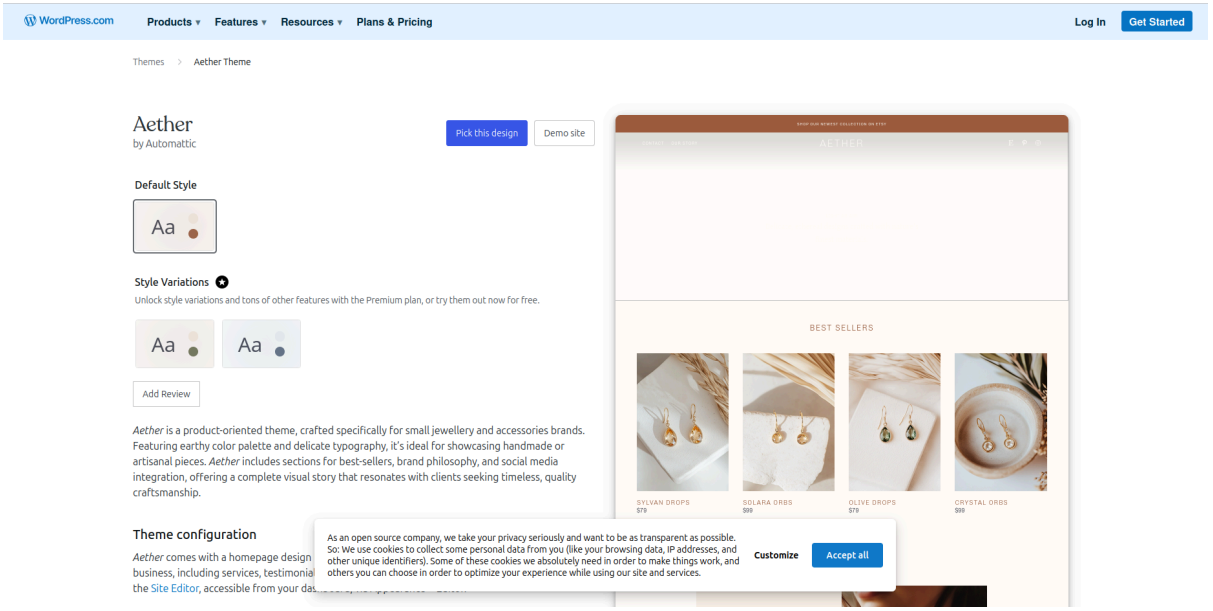


Fig. 8: A template description page on WordPress theme catalogue. It also provides detailed information about the template and offers the option to run a live preview.

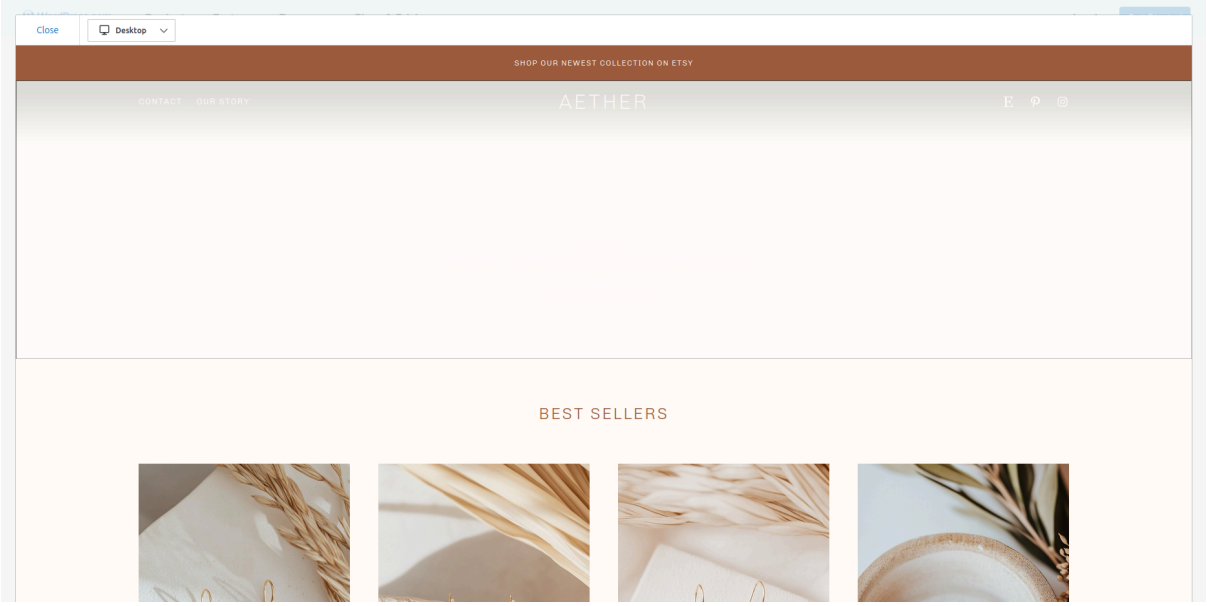


Fig. 9: The live demo of the above WordPress template.

We plan on developing a similar web demo mechanism for Gatino. That said, our product demonstrations will be more complex as many student web products will need to interact with the backend. A notable drawback of the live demos on Envato and WordPress is the absence of backend functionality. This limitation can lead to a disconnect between the demo experience and the reality of using the template in a live environment.

Mobile demonstrations

Our research on mobile (Android) demonstrations focused on applications that support browser-based Android demonstrations. One such example is Docker Android, which allows users to run an Android emulator within their web browser. This tool provides a straightforward way for developers to test Android apps without needing a local installation. However, a significant limitation of Docker Android is that it only supports one user connection at a time, meaning that multiple users cannot access the emulator simultaneously. Launching multiple emulators to support concurrent users is very costly as each emulator consumes a lot of resources.

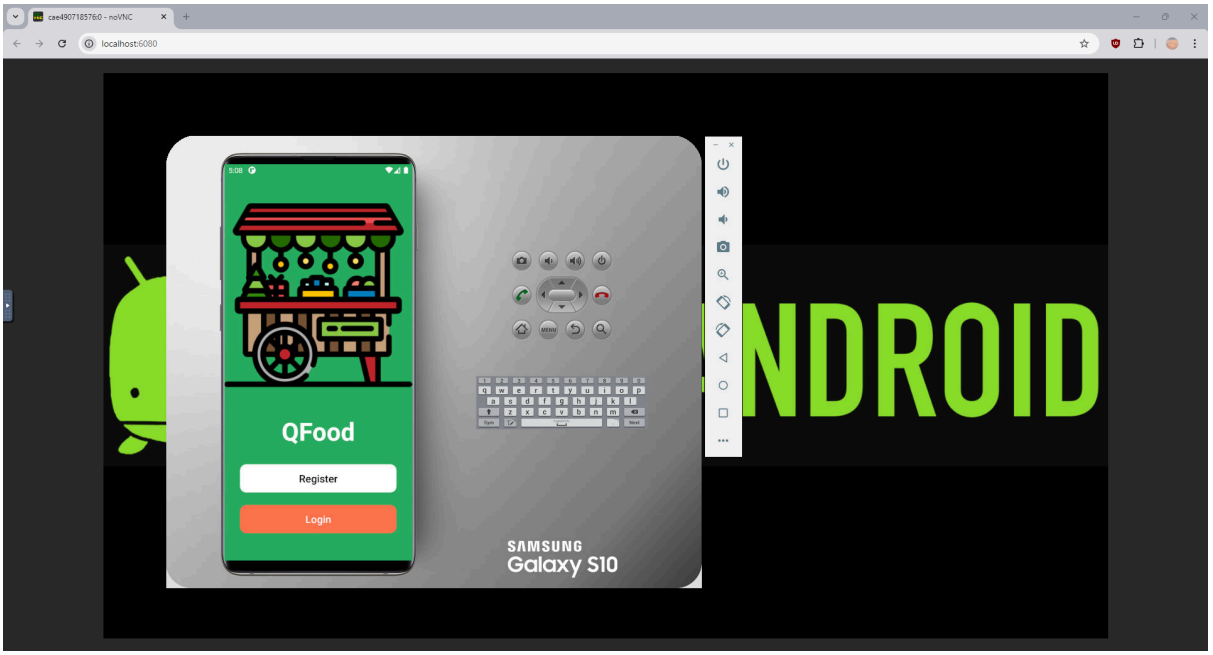


Fig. 10: A Docker Android emulator accessed from the browser.

Another service we looked into was Snack Player by Expo, which presents a more user-friendly solution for mobile app demonstrations. As a browser-based emulator specifically designed for React Native applications, Snack Player enables developers to test their apps in real-time, directly within their browsers. One of its standout features is the queue waiting system, which allows multiple users to access the emulator in an orderly fashion. This system alleviates the frustration of waiting for an available session and enhances collaborative workflows among developers and testers. Additionally, Snack Player includes an inactivity timeout feature, which automatically frees up resources if a user is not actively engaged. While we may not implement Snack Player directly in our products, its features provide valuable insights that could inform our approach to developing a mobile demonstration platform.

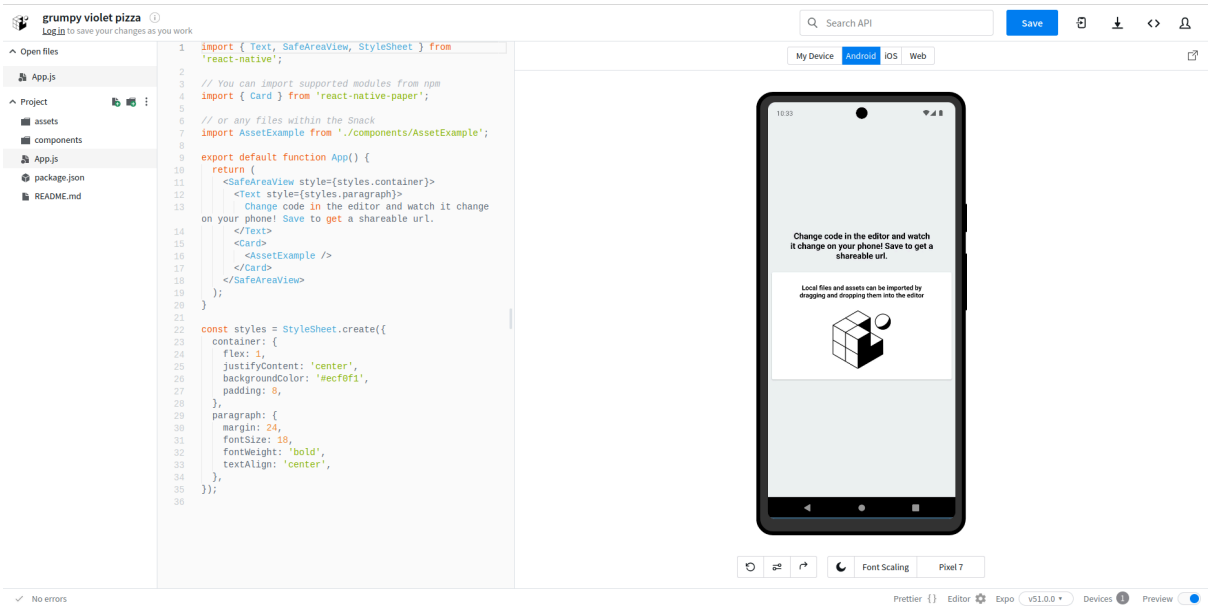


Fig. 11: A Snack Player Android emulator accessed from the browser. This emulator will time out after the user has not interacted with it for some period of time. It will also put the user in a queue and tell them to wait if other users are accessing the emulator.

IoT Headless Demo

Expanding into IoT demonstrations, we investigated MQTT-based solutions to simulate real-time device communication. Tools like Eclipse Mosquitto, Node-RED, and EMQX's sandbox environment enable virtual IoT interactions, though they often lack persistent state management. To address scalability and user experience challenges, we propose lightweight MQTT brokers with session isolation, state persistence via databases, and a drag-and-drop IoT playground for intuitive testing. By combining these approaches, Gatino's demo system will unify web, mobile, and IoT previews in a single dashboard, ensuring seamless, interactive product demonstrations while optimizing resource efficiency through automated cleanup mechanisms.

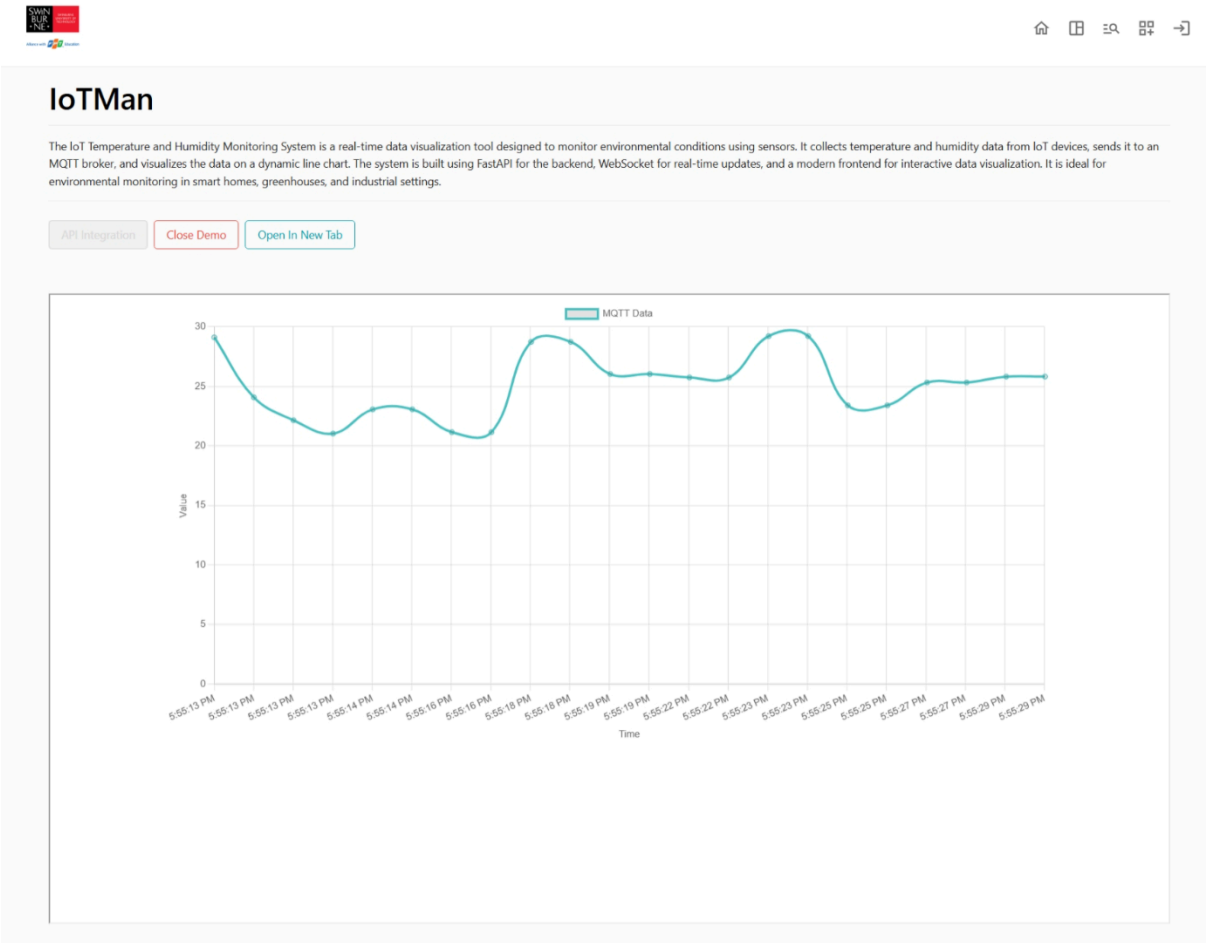


Fig. 12: Data streaming through MQTT Broker.

4.2 - Research into System Design

The system design of *Gatino - Innovation Gateway* focuses on building a scalable, modular, and user-friendly platform. Extensive research into system design approaches and

architectural patterns was conducted to ensure that Gatino can meet the demands of multiple user groups, support rapid deployment, and allow for efficient maintenance and scaling. The research focused on the following key areas:

1. Microservices Architecture

Overview: Microservices architecture was explored as a solution for developing Gatino's core functionalities—search, product demonstration, and user management—as distinct, independently deployable services. In this model, each core feature operates as an isolated service with dedicated resources, dependencies, and APIs.

Advantages: The microservices architecture enables Gatino to scale individual services based on demand, providing both flexibility and improved performance. By deploying each feature in a containerized environment, services can be independently updated or restarted without impacting the entire system.

Application for Gatino: Each of Gatino's primary functionalities (search, demo, and product management) will be encapsulated in its microservice, with an API gateway managing communications between them. This setup allows for seamless integration with other parts of the platform while ensuring that each module can be scaled or modified as needed.

2. API Gateway Pattern

Overview: An API gateway acts as an intermediary, routing requests from clients to the appropriate microservices. This pattern consolidates all communication, managing load balancing, security, and request handling.

Advantages: The API gateway offers several benefits for Gatino, including centralized management of user requests, streamlined access control, and reduced client-server interactions. It provides a unified interface for external communication while allowing the backend services to remain encapsulated.

Application for Gatino: The API gateway will serve as the central access point for all user requests, routing them to the relevant microservices. This allows for efficient handling of requests while abstracting the complexity of individual services from the end-user.

3. Containerization with Docker

Overview: Docker was chosen for containerizing Gatino's microservices, enabling consistent environments for each service and supporting efficient scaling and deployment. Each core functionality—such as the search engine (Elasticsearch) and document processing (Apache Tika)—will be containerized as an independent service.

Advantages: Docker containers provide isolation for each service, ensuring that each module operates with its specific dependencies without interference. This

reduces deployment issues and enhances resource efficiency by allowing services to run on a single host system or distributed infrastructure.

Application for Gatino: Docker's flexibility in deploying multiple containers aligns with Gatino's need for isolated services that can be updated independently. Additionally, Docker Compose will be used for orchestrating containers, simplifying configuration and deployment across environments.

4. Event-Driven Communication for Real-Time Updates

Overview: To provide real-time updates, particularly for Gatino's queue management in demonstrations, an event-driven communication model was explored. This design allows services to respond to real-time events, such as when a user requests or finishes a demo.

Advantages: Event-driven architectures are highly responsive and decoupled, enabling real-time updates and facilitating a queue system where multiple users may need to wait for product demonstrations.

Application for Gatino: An event-driven approach, potentially using WebSocket connections or an event streaming platform, will manage user queueing for the demonstration service. This setup ensures that users receive real-time feedback on queue positions and wait times, enhancing the demonstration experience.

5. Database Design and Caching

Overview: For efficient data retrieval, particularly in search functionality, research was conducted into optimal database structures and caching strategies. The primary database will store core product metadata, while Elasticsearch will handle search indexing and querying.

Advantages: A combined approach—storing structured product data in a relational database and searchable data in Elasticsearch—allows for fast, relevant search results. Additionally, implementing caching (e.g., using Redis) for frequently accessed data can reduce load times and improve response times for popular searches or demonstrations.

Application for Gatino: Product metadata will be stored in MySQL, while Elasticsearch will handle search queries. Caching mechanisms will be used where necessary to improve response times and ensure a smoother user experience during high-demand periods.

4.3 - Research into technical platforms, languages and tools

Elasticsearch

Elasticsearch is an open-source search and analytics engine, designed to handle large volumes of data in real-time. Built on top of Apache Lucene, Elasticsearch is known for its speed, scalability, and powerful full-text search capabilities, making it ideal for applications

requiring fast data retrieval and complex querying. In Gatino, it consists of large product's information data, when integrating Elasticsearch as a search engine, it could solve the search speed, most relevant, filter, which saves a lot of time compared to building from scratch. Moreover, Elasticsearch is widely used in applications such as website search, log and event data analytics, product catalogs, and other data-intensive scenarios where fast and scalable search solutions are essential. Compared to other search engines, ElasticSearch is known as the easiest to learn and most widely supported extensions. (Meher, Mishra and Sahoo, 2024). To implement ElasticSearch, users can easily download packet from the official website from a dedicated server, or nest Elasticsearch as a Docker image from configuration, many popular cloud services such as AWS, Azure,... already integrated Elasticsearch as a service, which let users implement it in their cloud easily.

Apache Tika

When a product is uploaded to Gatino, its search information must be added to the search database (this database is separate from the main MySQL database, which stores the primary product information). In addition to the information provided explicitly in the upload request's fields such as name and description, a lot of search data can be further extracted from the documents that are included in the request. To achieve this, we utilize Apache Tika deployed as a Docker container on port 9998, with communication over HTTP (Apache.org, 2024). Our data extraction module sends a PUT request to the Tika service for document processing. This request format includes the document in binary form, and Tika responds with extracted textual content, allowing us to parse data from various document formats (such as PDF, DOC, DOCX, and MD). After processing this content, We allow Gatino's search engine to use keywords extracted from the documents themselves, which improves search relevance.

GitHub API

The GitHub API provides powerful programmatic access to GitHub's resources, enabling developers to interact with repositories, issues, pull requests, users, and organizations through REST and GraphQL endpoints.

The GitHub API offers robust features for obtaining detailed repository insights, including contribution statistics and coding language proportions. Through the API, developers can observe **contributors** for any repository, allowing them to analyze user contributions such as the number of commits, additions, deletions, and active contributors over time. This information is accessible via endpoints that provide commit history, contributor stats, and individual contributions, which is valuable for understanding team involvement, code ownership, and activity levels across projects.

Additionally, the API enables retrieval of **coding language proportions**, offering a breakdown of the languages used in a repository and their respective sizes in bytes. This feature allows developers to visualize and assess the primary languages of a codebase, making it useful for projects needing language-specific optimization or categorization. With these features, the GitHub API helps automate project tracking, enhance insights into team contributions, and gain a deeper understanding of code composition by language distribution.

Apache Guacamole

In Gatino, Apache Guacamole is deployed using a custom Docker container to provide remote access to a virtual machine (VM) running on the server, allowing users to interact with and showcase mobile applications from their web browsers. The deployment is set up via a Dockerfile specifically configured for Gatino's environment, making Guacamole easy to manage, scalable, and consistent across instances (Apache.org, 2024). To access the VM, users log in through a pre-configured guest account, which we've set up to provide immediate access without additional permissions. Users connect directly through their browser via Guacamole, which establishes a secure RDP connection over port 3389 to the VM. This setup removes the need for complex client-side installations, simplifying access and allowing users to engage with better accessibility. Once connected, the VM runs scrcpy, a tool that mirrors and controls an Android device's screen, allowing users to view and interact with the mobile application directly. This service makes it easy for users to demonstrate mobile products in real time and enhances accessibility and usability in product showcases.

Scrcpy

Scrcpy (short for Screen Copy) is a desktop utility that enables the user to connect to and control real Android devices through the desktop interface. Testing with a USB connection, we have found that viewing and controlling the Android device through Scrcpy has almost the same response time as directly controlling it. Scrcpy's utility shines when it is used in combination with Apache Guacamole. This combination enables users to connect to a virtual desktop on the backend from their web browsers, after which they can view and control the physical Android device to run mobile demos.

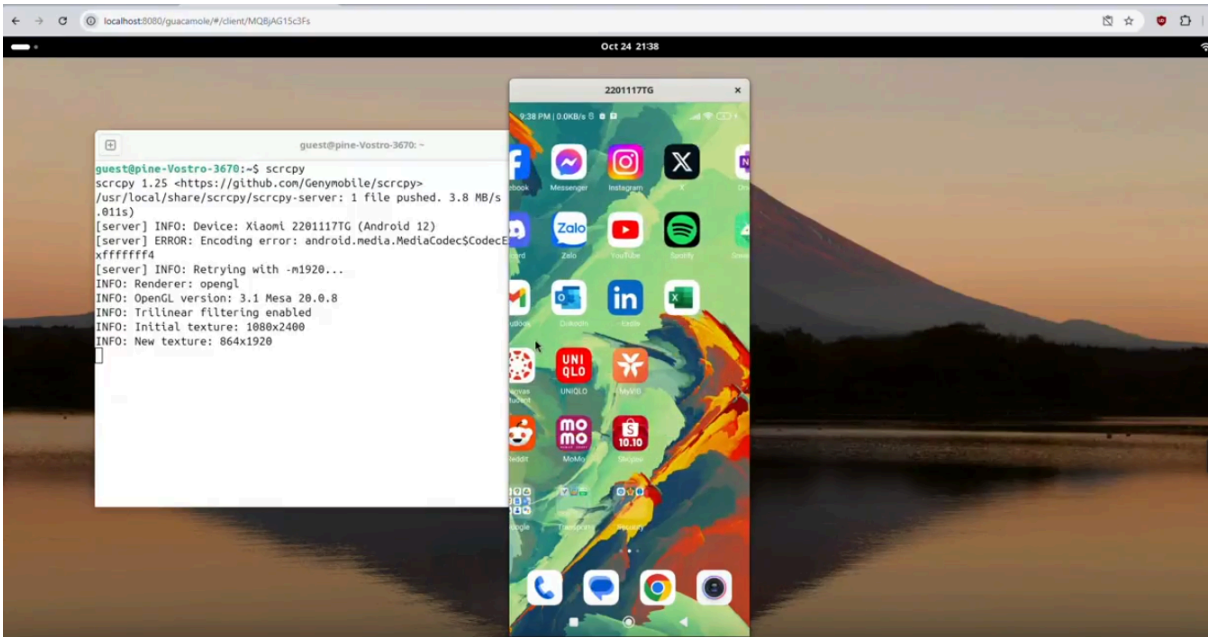


Fig. 13: Scrcpy allows the user to view and control a real Android device through a desktop interface. Also notice that this desktop interface is actually in the browser, made possible by Apache Guacamole.

Docker

Docker is a containerization platform that enables developers to package applications and their dependencies into isolated containers. By using Docker, Gatino's core features—search, product management, and demonstrations—can be separated into microservices that can run independently. Docker offers several advantages:

Environment Consistency: Docker ensures that Gatino's development, testing, and production environments are identical, which minimizes compatibility issues. By encapsulating each component with its specific dependencies, Docker promotes consistent behavior across various stages of development and deployment.

Modularity: Docker's containerization enables each functionality (such as the Elasticsearch search engine, Apache Tika for document processing, and Guacamole for remote access) to be encapsulated as an independent service. This modularity supports easier maintenance, as individual containers can be updated, restarted, or scaled without affecting the entire application.

Resource Efficiency: Docker's lightweight nature ensures that each service consumes only the necessary resources, which is essential for managing Gatino's anticipated high user concurrency. Containers can also be scaled up or down dynamically based on user demand, ensuring efficient resource utilization.

Scalability: With Docker, each core feature of Gatino can be deployed as a separate container, allowing the platform to respond to increased demand by scaling only the services that need more resources. For instance, the demo container can be scaled during events, while the search container can be independently scaled for regular use.

Apache APISIX

To enhance API management, security, and traffic control in Gatino, we integrate Apache APISIX as a high-performance API gateway. APISIX provides dynamic routing, load balancing, authentication (JWT, OAuth2), rate limiting, and observability (metrics, logging) for microservices architectures. Since Gatino relies on multiple services (Elasticsearch, Tika, MySQL), APISIX acts as a unified entry point, efficiently managing requests and improving scalability. APISIX is deployed as a reverse proxy, handling API requests before they reach backend services. It supports plugin-based extensions, allowing seamless integration with authentication providers, logging systems (e.g., Elasticsearch for logs), and serverless functions. By offloading cross-cutting concerns like SSL termination and request filtering to APISIX, Gatino's core services remain lightweight and focused on business logic.

MQTT Mosquitto

For real-time notifications and event-driven communication (e.g., product updates, user alerts), Gatino employs MQTT Mosquitto, a lightweight and scalable MQTT broker. Mosquitto facilitates publish-subscribe messaging, enabling efficient IoT-style communication between services with minimal overhead. In Gatino, Mosquitto is deployed

as a Docker container, allowing microservices to subscribe to topics (e.g., products/updates) and receive instant updates when new data is indexed in Elasticsearch or processed by Tika. This decoupled architecture ensures asynchronous, low-latency communication, improving system responsiveness. Additionally, Mosquitto's support for QoS levels and retained messages ensures reliable delivery, even in unstable network conditions.

5 - References

Apache.org. (2024). *Apache Tika – Getting Started with Apache Tika*. [online] Available at: https://tika.apache.org/2.6.0/gettingstarted.html [Accessed 1 Nov. 2024].

Apache.org. (2024). *Installing Guacamole with Docker — Apache Guacamole Manual v1.5.5*. [online] Available at: https://guacamole.apache.org/doc/gug/guacamole-docker.html [Accessed 1 Nov. 2024].

Kirill Fakhroutdinov (n.d.). *UML component diagram reference - components, provided and required interfaces, ports, relationships between them, etc.* [online] Available at: https://www.uml-diagrams.org/component-diagrams-reference.html.

Kirill Fakhroutdinov (2016). *UML Deployment Diagrams - Graphical Notation Reference*. [online] Uml-diagrams.org. Available at: https://www.uml-diagrams.org/deployment-diagrams-reference.html [Accessed 31 Oct. 2024].

Meher, A.K., Mishra, D. and Sahoo, A.K. (2024). Leveraging Towards Big Data Indexing for Industry Automation. 7, pp.1–6. doi:https://doi.org/10.1109/assic60049.2024.10507930.

CEI-HUEUNI (no date) *cei.hueuni.edu.vn*. Available at: https://cei.hueuni.edu.vn/ (Accessed: 2 November 2024).

'Innovation Gateway' (no date) *Invent Penn State*. Available at: https://invent.psu.edu/tools/innovation-gateway/ [Accessed: 2 November 2024].

Innovation Gateway | University of Delaware (no date). Available at: https://www.udel.edu/research-innovation/innovation-gateway/ (Accessed: 2 November 2024).

National Drone Innovation Gateway (no date). Available at: https://www.cranfield.ac.uk/themes/aerospace/national-drone-innovation-gateway (Accessed: 2 November 2024).

'NSSC – Trung tâm hỗ trợ khởi nghiệp sáng tạo Quốc gia – Trung tâm hỗ trợ khởi nghiệp

sáng tạo Quốc gia' (no date). Available at: https://nssc.gov.vn/ (Accessed: 2 November 2024).

Sunwah Innovation - Office Space, Virtual Office and Workspace to Rent (no date) *Sunwah Innovation*. Available at: https://swinno.com.vn/en/ (Accessed: 2 November 2024).

Trung tâm Đổi mới sáng tạo Quốc gia | Vietnam National Innovation Center (no date) *https://nic.gov.vn/*. Available at: https://nic.gov.vn/ (Accessed: 2 November 2024).

6 - Appendix

This appendix contains a list of functional and non-functional requirements specified in the SRS.

6.1 - Functional requirements

Work area 1 - Search for projects

General	Search is an important feature of the system as it allows users to discover projects on the platform, a precondition to several other features such as demonstration or integration. Search should allow complex filters to refine the results while also maintaining simplicity. Ideally, it should function similarly to popular search engines such as Google, allowing the user to enter an easy-to-write search query and specify additional search parameters. The search module must also add/update/delete a project's search information when it is added/updated/deleted. A project's search information is not only given explicitly by the student who uploads it but can also be extracted from various sources, such as any provided links and documents.
Actors involved	**1 - Swinburne students** Students upload/update their projects and expect their project information to be searchable as quickly as possible. In addition, students may also want to search for projects from other students, either for reference purposes or for integration into their own projects. This group typically have medium-to-high technical experience and they may want to search by feature, technology and language used, and API integration support. **2 - Swinburne lecturers/Heads of Department** Lecturers want to search for specific student projects to demonstrate in lectures. Typically, they will search by project name or project type. Heads of Department want to use the search feature to better understand the most common technology/language choices for a given type of project. They are also interested in seeing how many projects have been developed for each major. While this makes more sense as a separate visualization feature, the search feature can report some basic statistics for each search query to fulfil this need. This user group have high technical experience.

	3 - Businesses Businesses may partner with Swinburne Vietnam to purchase student's software for their companies. To do this, they need to use the search feature. This group has varying levels of technical expertise. Some business partners are not familiar with technical terms and will only search by project feature, often expressed as keywords. Others have more technical knowledge and will want to search by tech stack or programming language used.

Data model

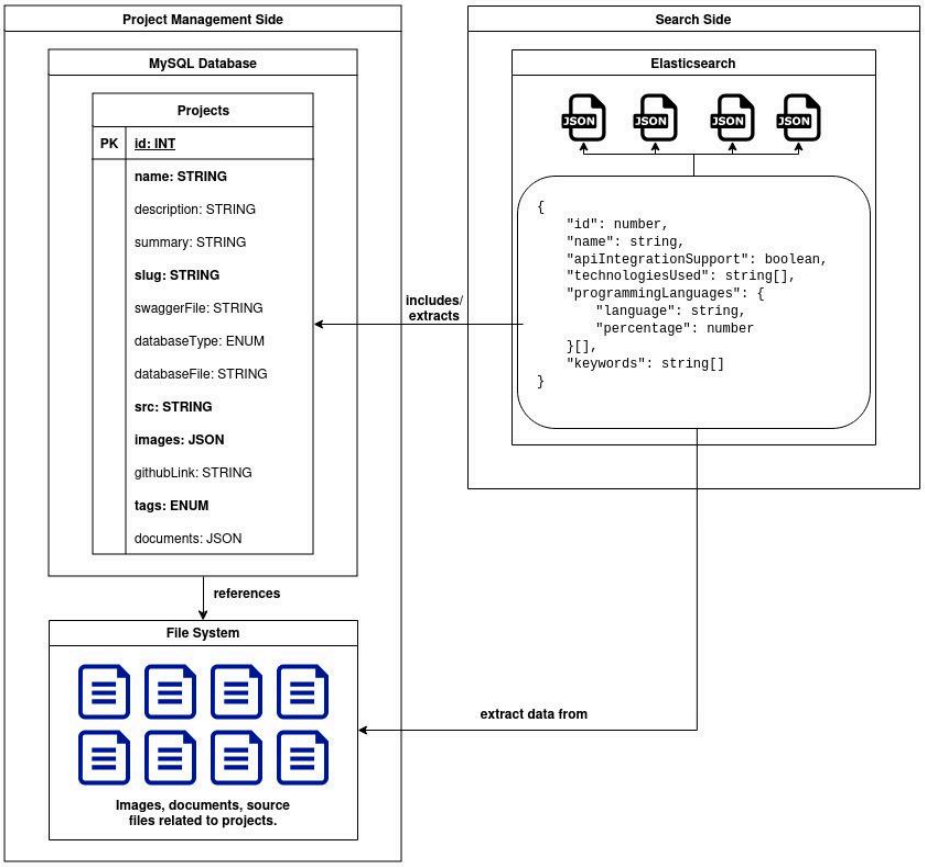


Fig. 1: The data model for the search feature. The full-resolution image can be seen here.

Project information will be stored in two places: the Project Management Side and the Search Side. The Project Management Side stores the project data that can be seen and directly modified by the user. Here, the data includes the project's metadata in a MySQL table and any files associated with the project in the server's file system.

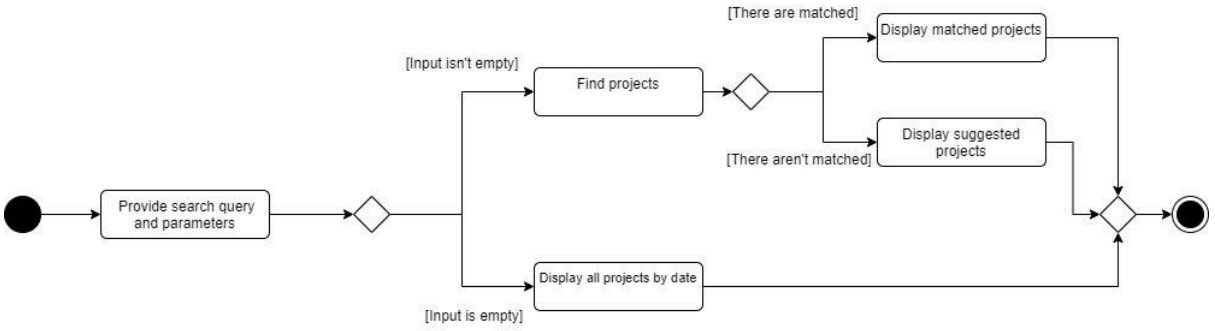
The Search Side stores the project search information directly inside Elasticsearch as JSON documents. Every project has one associated document in Elasticsearch. This search data is built on the metadata stored in MySQL and the metadata extracted from project files and links. The system only searches on the Elasticsearch data, but it may ask the Management Module for additional project data when it returns the result to the user.

Task 1.1. - Search for projects

Purpose	Allow the user to search for student projects that have been uploaded to Gatino.
Trigger/precondition	A user opens the Projects page and initiates a search.
Frequency	10 searches/minute.
Critical	During a demonstration event where the audience members may want to search for a project simultaneously.

Sub-tasks	**Example solution**
1 - Provide search query and parameters. **Problems:** ● Users are not aware of the search parameters. ● The search query can represent one or more fields. ● The user may enter keywords that are synonymous to the ones in the database.	The system displays a search bar along with search parameters in the form of dropdowns checkboxes.
2 - Find projects that match the query and parameters. **Problem:** ● Some projects may not have been indexed and thus are unsearchable.	The system scans the database for projects that match the search query and returns the results in pages.
Variants	**Example solution**
1a - The user provides no search query or parameters.	Returns a list of all projects, sorted by date uploaded.
2a - No projects match the search query.	Informs the user that no projects match their search query, or shows them a list of random projects.

Workflow:


[There are matched]
Display matched projects
[Input isn't empty]
Find projects
Display suggested projects
[There aren't matched]
Provide search query and parameters
Display all projects by date
[Input is empty]

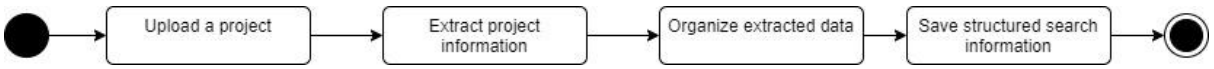
Task 1.2. - Add project search information

Purpose	Make a project searchable by adding its search information to the database.
Trigger/precondition	A student uploads a project to the platform. The upload includes basic details about the project (such as name and description) as well as files and links through which more search data can be extracted.
Frequency	10 projects/month.
Critical	● When Gatino is first launched, a large number of projects will be uploaded as students and lecturers migrate from existing platforms like Canvas. ● The search module may not be informed of the project upload due to network errors, etc.

Sub-tasks	**Example solution**
1 - Extract search information from the project. **Problems:** ● Search information can come from a variety of sources, such as the project's GitHub repository or uploaded documents. ● The uploaded documents may contain too much text.	The system extracts more information from the provided links and files using a variety of tools/services such as the GitHub API or Apache Tika. The system may also filter the data or transform it to reduce the amount of storage needed.
2 - Organize the extracted data into a common structure. **Problem:** The structure of the information may change in the future.	The system defines a common structure for the search data (JSON or YAML, etc.), including all the fields that will be included, then puts the obtained data into that structure.
3 - Save the structured search information. **Problem:** The search information may have to be saved separately from the main database, and the two databases have to be synchronized.	The system adds the organized project data into the search database.

Variants	Example solution
None.	

Workflow:



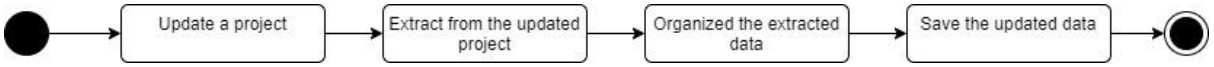
Task 1.3. - Update project search information

Purpose	Reflect the latest information about the project after it has been changed.
Trigger/precondition	A student updates a project that they have uploaded. The update may change the basic details about the project (such as name and description) as well as files and links through which more search data can be extracted.
Frequency	5 updates/month.
Critical	The search module may not be informed of the project update due to network errors, etc.

Sub-tasks	**Example solution**
1 - Extract search information from the updated project. **Problems:** ● Search information can come from a variety of sources, such as the project's GitHub repository or uploaded documents. ● The uploaded documents may contain too much text.	The system extracts updated information from the provided links and files using a variety of tools/services such as the GitHub API or Apache Tika. The system may also filter the data or transform it to reduce the amount of storage needed.
2 - Organize the extracted data into a common structure. **Problem:** The structure of the information may change in the future.	The system defines a common structure for the search data (JSON or YAML, etc.), including all the fields that will be included, then puts the obtained data into that structure.
3 - Save the updated search information. **Problems:** ● Some of the old search information may have to be deleted. ● The search information may have to be saved separately from the main	The system updates the search information in the database, removing any old data as necessary.

database, and the two databases have to be synchronized.	
Variants	**Example solution**
None.	

Workflow:

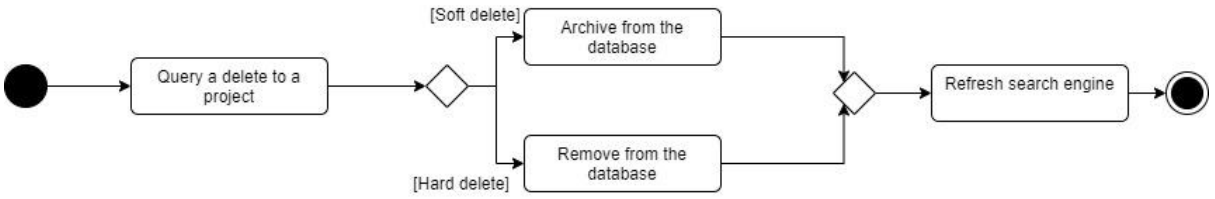


Task 1.4. - Delete project search information

Purpose	Stop a deleted project from being searchable.
Trigger/precondition	A student or admin archives or deletes a project.
Frequency	Very rarely, 5 projects/year.
Critical	The search module may not be informed of the project delete due to network errors, etc.

Sub-tasks	**Example solution**
1 - Erase the project's search information.	The system sends a query to delete the project information from the search database.
2 - Refresh the search engine to reflect the changes.	The system sends a query to refresh the search engine if necessary, or the engine refreshes itself.
Variants	**Example solution**
1a - The project has only been archived (soft-deleted) but not permanently deleted.	The system sets a boolean flag in the project's search information to make the engine ignore it.

Workflow:



Work area 2 - Demonstrate projects

General	Demonstration is a key feature of the platform as it enables users to run and interact with a project instead of just viewing its static information, helping them better understand its features and capabilities. Depending on the nature of the project, it may be partially or fully demonstrable. In addition, due to the lack of hardware/software resources, some projects can only be run by one user at a time. Projects can fall into many categories, although this project covers the following: ● **Web**: Web projects are typically fully viewable and interactive. Multiple users can access a web project at the same time with minimal interference. ● **Android (mobile):** Android projects are run on either an Android emulator or a physical device, the former being easier and more convenient to implement. Only one person can control the same emulator or physical device at a given time. These projects are fully viewable and interactive. ● **IoT/AI**: These projects may or may not run on Gatino's server and often make heavy use of external APIs. In addition, some projects may involve physical sensors that are not controlled by Gatino. It is quite rare for these projects to have a user interface, unless they are part of a bigger web or Android project (i.e. a smart home app). Typically, these projects expose APIs that produce data (either as a continuous stream or on demand) and can be visualized.
Actors involved	All user groups, from students to lecturers and businesses, will want to access this feature. Their technical expertise ranges from very low to very high. They all expect to be able to view and interact with the demonstration as quickly as possible, and some will not tolerate any waiting time.

Data model

With regards to the Demo feature, the system does not store much information related to the projects. In addition, different project types will have the system concerned with different types of information. More specifically:

- For **web** projects, the system stores their source code and deploys them as Docker containers that can be turned on or off.
- For **Android** projects, the system uploads their APK files to a Docker Android emulator container that can be turned on or off.
- For **IoT** projects, the system stores their source code and, depending on the configuration of the uploader, deploys them as Docker containers that can be turned

on or off. Some projects may run outside of the Gatino platform. These projects may expose data-producing APIs that Gatino can use to build visualizations for the user interface.

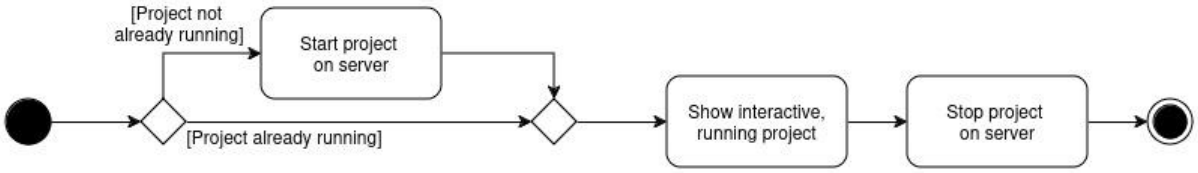
Task 2.1. - Run a demonstration of a web project

Purpose	Enables a user to view and interact with a web project in operation.
Trigger/precondition	A user clicks on the “Run demo” button in the web project’s details page.
Frequency	20 concurrent demos/hour.
Critical	During an important showcase event (such as a lecture or Swinburne Experience Day), a large audience (assumed to be about 50-75) may want to run demos simultaneously.

Sub-tasks	**Example solution**
1 - Start the project on the server (optional). **Problem:** A project may take a few minutes to start.	The server issues a command to switch on the project’s Docker container when a demo is requested. Ahead of an important event, the system can allow the system admin to switch the project on in advance, saving time.
2 - Show the interactive, running project.	The server presents a web-based user interface through which the user can view the demonstration in action. The user can interact with the web interface to interact with the project itself.
3 - Stop the project on the server (optional). **Problem:** Other users may want to run the project demonstration after the first one.	The server issues a command to switch off the project’s Docker container. To be available for near future users, the container can stay on for a given period of time before being automatically shut off to save resources.

Variants	**Example solution**
None	

Workflow:


[Project not already running]
Start project on server
[Project already running]
Show interactive, running project
Stop project on server

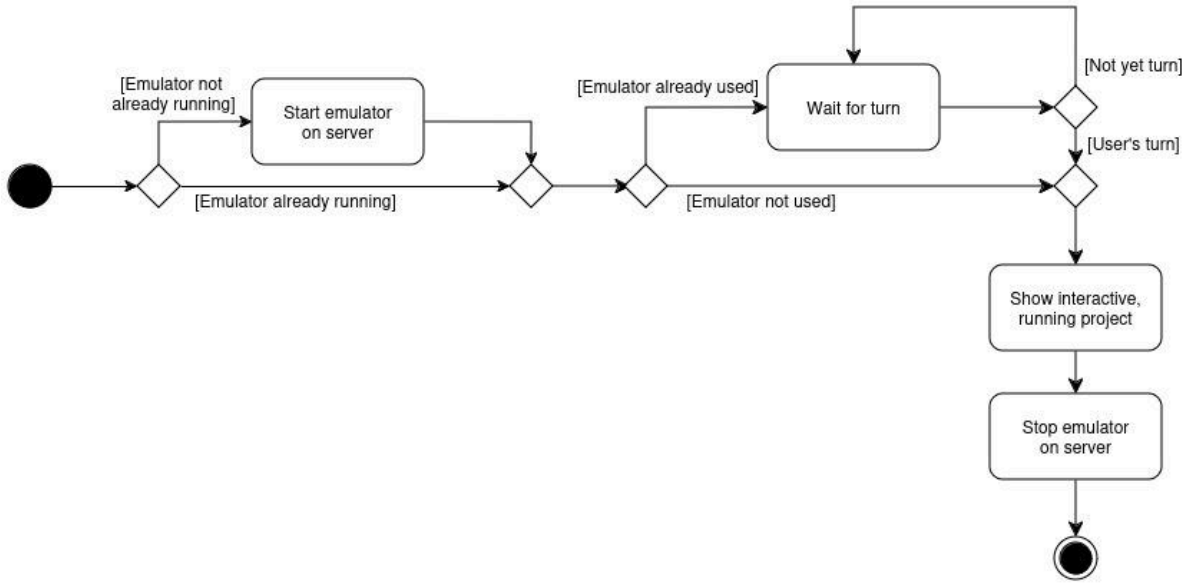
Task 2.2. - Run a demonstration of an Android project

Purpose	Enables a user to view and interact with an Android project in operation.
Trigger/precondition	A user clicks on the “Run demo” button in the Android project’s details page.
Frequency	20 concurrent demos/hour.
Critical	During an important showcase event (such as a lecture or Swinburne Experience Day), a large audience (assumed to be about 50-75) may want to run demos simultaneously.

Sub-tasks	**Example solution**
1 - Start the Android emulator on the server (optional). **Problem:** The emulator may take a few minutes to start. One emulator might not be enough if there are too many demo requests.	The server issues a command to switch on the Docker container running the Android emulator when a demo is requested. Ahead of an important event, the system can allow the system admin to switch the emulator on in advance, saving time.
2 - Show the interactive, running project.	The server presents a web-based user interface through which the user can view the demonstration in action. The user can interact with the web interface to interact with the project itself.
3 - Stop the Android emulator on the server (optional). **Problem:** Other users may want to run the project demonstration after the first one.	The server issues a command to switch off the emulators’ Docker container. To be available for near future users, the container can stay on for a given period of time before being automatically shut off to save resources.
Variants	**Example solution**
2a - Another user is already accessing the emulator.	The system places incoming users into a queue so that they can wait for their turn. The system will inform the user of their position in their queue and the estimated

	waiting time. To ensure fairness, allow only a few minutes for each user.

Workflow:



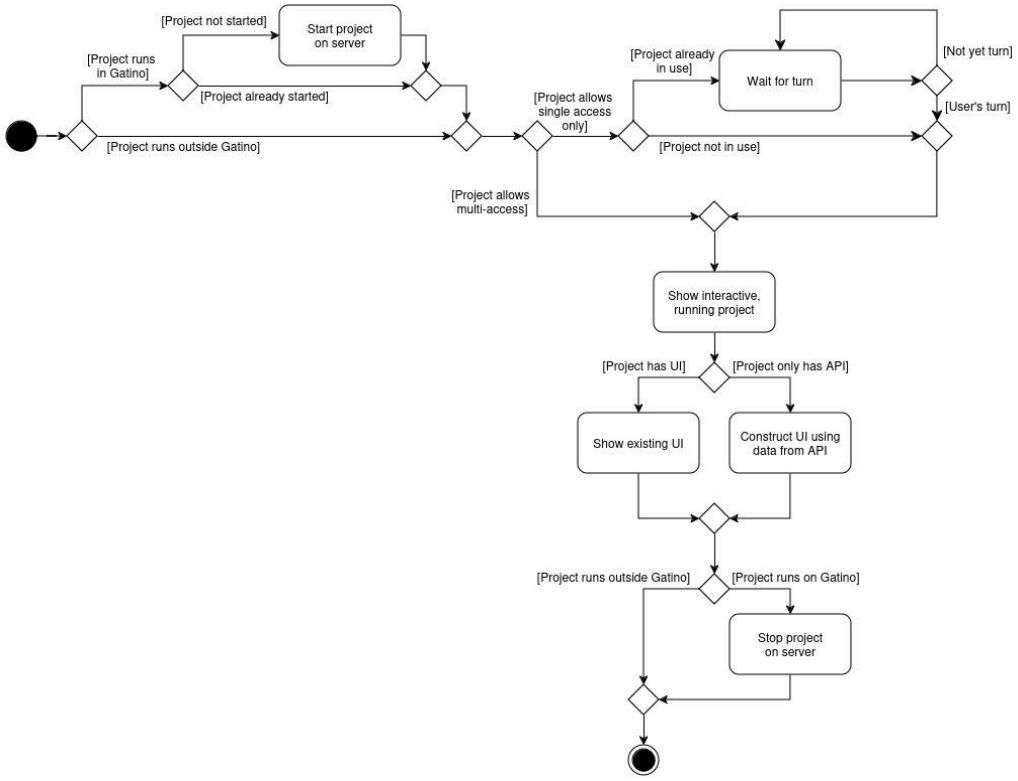
Task 2.3. - Run a demonstration of an IoT project

Purpose	Enables a user to view and possibly interact with an IoT project in operation.
Trigger/precondition	A user clicks on the “Run demo” button in the IoT project’s details page.
Frequency	20 concurrent demos/hour.
Critical	During an important showcase event (such as a lecture or Swinburne Experience Day), a large audience (assumed to be about 50-75) may want to run demos simultaneously.

Sub-tasks	**Example solution**
1 - Start the project on the server (optional). **Problems:** A project may take a few minutes to start. Some projects may not run on the server and thus cannot be directly controlled.	The server issues a command to switch on the project’s Docker container when a demo is requested. Ahead of an important event, the system can allow the system admin to switch the project on in advance, saving time.
2 - Show the running (and possibly interactive) project. **Problems:** Some projects can only be controlled by one person at a time or may not be controllable at all. Some projects do not come with a user interface.	The server presents a web-based user interface through which the user can view the demonstration in action. The user may interact with the web interface to interact with the project itself.
3 - Stop the project on the server (optional). **Problem:** Other users may want to run the project demonstration after the first one.	The server issues a command to switch off the project’s Docker container. To be available for near future users, the container can stay on for a given period of time before being automatically shut off to save resources.
Variants	**Example solution**
2a - The demonstration can only be controlled by one user and cannot be	The system places incoming users into a queue so that they can wait for their turn. The system will inform the user of their

replicated.	position in their queue and the estimated waiting time. To ensure fairness, allow only a few minutes for each user.
2b - Some projects cannot be interacted with completely or at all.	The system presents as much information as it can show and allows as much interaction as it can support. In cases where a project cannot be interacted with at all, the system can present the user with a fallback video showing the developers demonstrating the project.
2c - Some projects do not have a built-in user interface.	If the project exposes, or links to, a data-producing API, the system can query the API for data to construct a visualization for the user. This visualization can be data charts or live video/image feeds, depending on the type of the project. In cases where a project cannot be viewed at all, the system can present the user with a fallback video showing the developers demonstrating the project.

Workflow:



6.2 - Non-Functional (Quality) Requirements

Based on the ISO/IEC 9126 quality model (*ISO/IEC 9126-1*, n.d.), we present a structured presentation of the non-functional (quality) requirements, covering all of the defined characteristics and sub-characteristics:

Functionality

- Requirement**:** The system must provide complete functionality as specified in the requirements document.
    - **Suitability:** The search and demo features must be operational by the end of the development phase.
    - **Accuracy:** Search results must achieve at least 95% relevance as measured by user testing.
    - **Interoperability:** The system must successfully integrate with at least two external tools (e.g., authentication systems) by deployment.
    - **Security:** Implement role-based access control to ensure only authorized users can access sensitive features, validated through a security audit before launch.
    - **Functionality Compliance:** All functions must comply with specifications, verified by the Innovation Lab and product owner Dr Le Minh Duc.
- Verification: Conduct functional testing and gather feedback from stakeholders during the development process with a target completion date of two weeks before the final deployment.

Reliability

- Requirement: The system must maintain an uptime of 99.5% over any given month, with recovery from failures in under 10 minutes.
    - **Maturity:** All components must pass stability testing with no critical failures during the testing period.
    - **Fault Tolerance:** The system should handle up to 3 simultaneous failures without service interruption.
    - **Recoverability:** The system must restore operations within 10 minutes after any failure, validated through recovery drills conducted quarterly.
    - **Reliability Compliance:** Achieve at least 95% compliance with reliability metrics during testing.
- Verification: Monitor system uptime and conduct reliability testing frequently during the testing phase.

Usability

- Requirement: The user interface must achieve a usability score of at least 80% based on System Usability Scale (SUS) assessments from at least 30 users.
    - **Understandability:** At least 90% of users should complete key tasks without assistance during usability testing.
    - **Learnability:** New users should complete uploading their projects with an average time of less than 10 minutes and searching for less than 2 minutes.
    - **Operability:** Users must complete critical functions within 2 clicks on average.
    - **Attractiveness:** The interface must receive a satisfaction rating of at least 80% for visual appeal in user surveys.
- Verification**:** Conduct usability testing sessions within three weeks of the initial development phase, and collect client and user feedback.

Efficiency

- Requirement: The system must support 100 concurrent users, with response times of less than 5 seconds for search queries and demo requests.
    - **Time Behaviour:** At least 95% of interactions must respond within the specified time frame under load.
    - **Resource Utilization:** CPU and memory usage must not exceed 75% under peak load conditions.
    - **Efficiency Compliance:** Achieve compliance with efficiency metrics during load testing.
- Verification: Use load testing tools to simulate user interactions and gather performance data two weeks before deployment.

Maintainability

- Requirement: The system must allow for critical updates to be implemented within 1 hour and feature updates within 24 hours.
    - **Analyzability:** All of the code must follow the conventional coding standard and Gatino coding standard, facilitating easy understanding.
    - **Changeability:** Changes to features should require no more than 5 hours of developer time on average.
    - **Stability:** No more than 2 critical issues should arise from modifications in 30 days following an update.
    - **Testability:** At least 80% of the code must have automated tests in place to ensure quick validation of changes.

- Verification: Review the coding rules and conduct maintenance tests one month after deployment.

Portability

- Requirement: The system must be deployable across at least two different operating systems (e.g., Linux and Windows) with minimal configuration changes.
    - **Adaptability:** The system must function correctly in each environment with less than 5 configuration changes required.
    - **Installability:** The installation process should not exceed 30 minutes for both operating systems.
    - **Co-existence:** The system should operate without conflicts alongside existing applications in the target environments.
    - **Replaceability:** Components should be replaceable without requiring more than 1 hour of downtime.
- Verification: Conduct compatibility testing on both operating systems, documenting configuration requirements and installation times within three weeks of deployment.