

VIETNAM NATIONAL UNIVERSITY, HANOI  
**INTERNATIONAL SCHOOL**

**GRADUATION PROJECT**



PROJECT NAME:

**BUILDING A WEBSITE FOR EPIDEMIOLOGY  
MANAGEMENT USING MONGODB DATABASE**

**Student's name**

Nguyễn Mạnh Thành

*Hanoi - Year 2024*

VIETNAM NATIONAL UNIVERSITY, HANOI  
**INTERNATIONAL SCHOOL**

**GRADUATION PROJECT**



PROJECT NAME:

**BUILDING A WEBSITE FOR EPIDEMIOLOGY  
MANAGEMENT USING MONGODB DATABASE**

SUPERVISOR: PhD. Nguyễn Đăng Khoa

STUDENT: Nguyễn Mạnh Thành

STUDENT ID: 19071627

COHORT: QHQ2019

SUBJECTCODE: INS401401

MAJOR: Informatics Computer and Engineering

*Hanoi - Year 2024*

## **ACKNOWLEDGEMENT**

I would like to express my deepest gratitude to Dr. Nguyen Dang Khoa for his enthusiastic guidance throughout the process of completing this design project. Your dedication and expertise have been instrumental in achieving the results we see today.

Your invaluable guidance and support throughout the project have been greatly appreciated. Your extensive knowledge, consistency, and dedication have helped me overcome challenges and develop a comprehensive skill set.

I also wish to acknowledge your invaluable assistance and advice in editing and refining the project. Your constructive feedback and detailed comments have deepened my understanding of the content and enhanced my self-evaluation skills.

Furthermore, I would like to extend my thanks to the entire school, my friends, family, and everyone who shared their knowledge, opinions, and encouragement during the project. The support from you and the community has been a significant source of motivation, contributing greatly to the successful completion of this project.

I am confident that the knowledge and skills acquired here will serve as a solid foundation for future development. Once again, I sincerely thank you and everyone who supported me on this journey.

Best regards,

## LETTER OF DECLARATION

I hereby affirm that this project “Building a website for epidemiology management using mongodb database” is my own research, conducted under the guidance of Dr. Nguyen Dang Khoa all results are honest and original, not copied from any other work.

All references are clearly sourced in the bibliography. I accept full responsibility for any violations of the school’s statutes. I have adhered to all regulations regarding research ethics and intellectual property rights. I understand that failure to comply can lead to serious consequences.

This project is a comprehensive study that involves critical thinking and innovative ideas. It contributes to the existing knowledge in my field. I am prepared to defend my work and learn from constructive criticism to improve my future research. I am committed to upholding the highest standards of academic integrity...

*Hanoi, June 2024*

Author

*Nguyen Manh Thanh*

## **ABSTRACT**

A comprehensive national epidemiology management website is essential for several reasons. Firstly, it centralizes the management of epidemic outbreaks across the country, providing a unified platform for health authorities, medical professionals, and the public. This system enables real-time data sharing and collaboration among different regions and agencies, ensuring that everyone has access to the latest information and resources. By integrating various data sources, such as hospitals, laboratories, and research institutions, the website can offer a holistic view of the epidemic landscape, facilitating more effective and coordinated responses to outbreaks. This nationwide approach helps in standardizing protocols and procedures, reducing discrepancies and improving the overall efficiency of epidemiology management efforts.

Secondly, an epidemiology management website enhances the ability to monitor, track, and predict epidemic trends. By aggregating data from various regions and health facilities, the system can provide accurate and timely statistics on the spread of epidemics. This data-driven approach enables health authorities to identify patterns and potential hotspots early, allowing for proactive measures to be taken before situations worsen. Predictive analytics can be employed to forecast the trajectory of outbreaks, informing policy decisions and resource allocation. This foresight is crucial in preparing for and mitigating the impact of future epidemics, ultimately saving lives and reducing the burden on healthcare systems.

Lastly, the website plays a vital role in minimizing the spread of epidemics and containing outbreaks. By providing daily updates on the status of various epidemics, it keeps the public informed and aware of current health risks. This transparency is critical in promoting adherence to public health guidelines and recommendations. Furthermore, the website can facilitate the dissemination of crucial information on preventive measures, treatment options, and vaccination campaigns, empowering individuals to take proactive steps in protecting their health. In times of crisis, timely and accurate information can make the difference between containment and escalation. The ability to

quickly update and disseminate information helps in mobilizing resources and personnel to areas most in need, effectively curbing the spread of epidemics and preventing them from becoming widespread epidemics.

## LIST OF ABBREVIATIONS

<b>Abbreviation</b>	<b>Definition</b>
ACID	Atomicity, Consistency, Isolation, Durability
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
GEOJSON	Geographic JavaScript Object Notation
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
I/O	Input/Output
JS	JavaScript
JWT	JSON Web Token
NoSQL	Not only SQL
SEO	Search Engine Optimization
SSO	Single Sign-On
RSA	Rivest, Shamir, and Adleman

## LIST OF FIGURES

Figure 2.1 Covid health risks [10] .....	15
Figure 2.2 Covid-19 pandemic [11] .....	16
Figure 2.3 Online epidemic management system [13] .....	17
Figure 2.4 Overview use case diagram .....	18
Figure 2.5 Decay usecase chart notification management .....	19
Figure 2.6 Decay usecase chart dynamic declaration .....	20
Figure 2.7 Decay usecase chart personal declaration .....	21
Figure 2.8 Decay usecase chart epidemic situation .....	22
Figure 3.1 Front-end technology [14] .....	26
Figure 3.2 How Express js works?.....	33
Figure 3.3 MongoDB database backup and restore process .....	37
Figure 3.4 Firebase [19] .....	40
Figure 3.5 Firebase storage [20].....	42
Figure 3.6 JWT bearer operating model [21].....	45
Figure 4.1 Client server.....	48
Figure 4.2 Design overview .....	49
Figure 4.3 Database design diagram .....	51
Figure 4.4 MongoDB deploy – deployment database.....	61
Figure 4.5 MongoDB deploy – create cluster .....	62
Figure 4.6 MongoDB deploy – connect from backend.....	62
Figure 4.7 MongoDB deploy successfully.....	63
Figure 4.8 Backend folder structure.....	64
Figure 4.9 Frontend folder structure .....	65
Figure 4.10 Deploy to Aptible with git push .....	66
Figure 4.11 Deploy to Aptible – add ssh key.....	67
Figure 4.12 Deploy to Aptible - environment.....	67
Figure 4.13 Deploy to Aptible – prepare the template.....	68
Figure 4.14 Deploy to Aptible - environment variables .....	68
Figure 4.15 Deploy to Aptible - view logs in real time .....	69
Figure 4.16 Deploy to Aptible - expose app .....	69



Figure 4.17 Deploy to Aptible successfully.....	70
Figure 4.18 Build in APIs (1).....	71
Figure 4.19 Build in APIs (2).....	72
Figure 4.20 Build in APIs (3).....	73
Figure 4.21 Build in APIs (4).....	74
Figure 4.22 Login interface.....	75
Figure 4.23 Register interface .....	76
Figure 4.24 Account approval interface.....	77
Figure 4.25 Personal information interface.....	78
Figure 4.26 Announcement/Post interface .....	79
Figure 4.27 Approve announcement/post interface .....	80
Figure 4.28 General declaration interface.....	81
Figure 4.29 General declaration list interface .....	81
Figure 4.30 Entry declaration interface.....	82
Figure 4.31 Entry declaration list interface.....	82
Figure 4.32 Move declaration interface .....	83
Figure 4.33 Move declaration list interface .....	83
Figure 4.34 Situation reports pandemic interface (1) .....	84
Figure 4.35 Situation reports pandemic interface (2) .....	84
Figure 4.36 Update situation pandemic interface .....	85
Figure 4.37 Pandemic map interface (1).....	86
Figure 4.38 Pandemic map interface (2).....	87
Figure 4.39 Pandemic chart interface (1).....	87
Figure 4.40 Pandemic chart interface (2).....	88
Figure 4.41 New pandemic interface .....	89
Figure 4.42 New pandemic list interface .....	89
Figure 4.43 Add new pandemic interface .....	90

## LIST OF TABLES

Table 4.1 User schema .....	52
Table 4.2 Admin_info schema .....	52
Table 4.3 Person_info schema .....	53
Table 4.4 Medical_info schema .....	54
Table 4.5 Domestic_guest schema .....	55
Table 4.6 Move_declaration schema .....	56
Table 4.7 Entry_declaration schema .....	58
Table 4.8 Unit Schema .....	60
Table 4.9 Notification schema .....	60

## TABLE OF CONTENTS

ACKNOWLEDGEMENT .....	1
LETTER OF DECLARATION .....	2
ABSTRACT .....	3
LIST OF ABBREVIATIONS .....	5
LIST OF FIGURES.....	6
LIST OF TABLES .....	8
TABLE OF CONTENTS .....	9
CHAPTER I: OVERVIEW OF THE TOPIC .....	12
1.1 Background of the topic .....	12
1.2 Objectives of the topic .....	12
1.3 Oriented.....	13
1.4 Conclusion.....	13
CHAPTER II: SURVEY AND REQUIREMENTS ANALYSIS .....	14
2.1 Current status survey.....	14
2.2 Function overview.....	18
2.2.1 Overview use case diagram.....	18
2.2.2 Decay use case diagram .....	19
2.3 Conclusion.....	23
CHAPTER III: OVERVIEW OF TECHNOLOGY.....	24
3.1 Frontend .....	24
3.1.1 jQuery.....	24
3.1.2 Leaflet technology.....	27
3.1.3 GeoJSON.....	29
3.1.4 CanvasJS .....	31
3.2 Backend.....	33
3.2.1 Node.js and express.js .....	33
3.2.2 MongoDB.....	36
3.2.3 Firebase .....	40
3.2.4 JWT bearer .....	44
3.3 Conclusion.....	47

CHAPTER IV: APPLICATION DEVELOPMENT AND DEPLOYMENT .....	48
4.1 Architectural design .....	48
4.1.1 Software architecture selection .....	48
4.1.2 Design overview.....	49
4.1.3 Database design.....	51
4.1.4 Description of the web development .....	61
4.1.5 Built-in APIs .....	71
4.2 System interface design .....	75
4.3 Conclusion.....	90
CHAPTER V: CONCLUSION.....	91
5.1 Results and evaluation.....	91
5.1.1 Results .....	91
5.1.2 Evaluation .....	92
5.2 Limitation .....	92
5.3 Future development.....	93
REFERENCES.....	94

## **CHAPTER I: OVERVIEW OF THE TOPIC**

This chapter introduces the topic, providing the background and rationale for choosing to develop epidemiology management website system. It also outlines the main objectives of the project.

### **1.1 Background of the topic**

In recent years, global warming has led to the ice phenomenon, accompanied by epidemics from ancient times preserved in blocks of ice. These epidemics have never appeared before, and there is no vaccine to prevent them or specific treatment methods. The consequences lead to rapid epidemic outbreaks.

Along with the development of technologies that humans have invented, we need to thoroughly apply those applications to the consequences that humans have indirectly created.

Hopefully “Building a website for epidemiology management using MongoDB database” can contribute to preventing unpredictable consequences that we can hardly imagine.

### **1.2 Objectives of the topic**

To build a system with such features, I chose to make a website system for three types of users:

- **Manager:** allows the management and approval of all accounts on the system, enables notifications to the entire system, updates the epidemic situation across units, and calculates the epidemic situation nationwide
- **Medical staff:** allows the creation of new dynamic forms to survey the epidemic situation from the public, collects and compiles the results of declarations from the public, views epidemic statistics and charts, and issues notifications such as vaccination information to all units and the entire country.

- Civilian: allows viewing the epidemic situation in the local area and nationwide, tracking the progression of the epidemic over time, viewing epidemic maps, filling out forms issued by healthcare staff, declaring entry if entering the country, declaring movement if traveling within the country, and posting questions or requests for assistance if needed.

### **1.3 Oriented**

Build a website system for the above three agents, accompanied by a server that provides APIs to be able to implement the functions and features.

The system will be divided into 2 main parts: Building frontend and backend.

- Frontend is to build a user interface specifically for user agents such as administrator, medical staff and civilian using HTML, CSS and JS technology.
- Backend will build a service that provides APIs specifically for businesses, using NodeJS technology and databases using MongoDB technology.

### **1.4 Conclusion**

The overview provided a clear understanding of the background and importance of epidemiology management system development. This information lays the foundation for approaching the next steps effectively and purposefully. Next, we will conduct a survey and requirements analysis, an important step to collect detailed information about specific needs and actual conditions, thereby determining the exact requirements for the system. This will ensure the system is designed and developed in accordance with actual needs.

## CHAPTER II: SURVEY AND REQUIREMENTS ANALYSIS

Following the overview, this chapter delves into the survey and requirements analysis. The goal is to gather necessary information and identify the specific requirements for the system.

### 2.1 Current status survey

The advent of the COVID-19 pandemic has underscored the vulnerability of global health systems and the importance of rapid, coordinated responses to health crises. This essay explores the various challenges posed by such pandemics and discusses potential solutions. This chapter provides the background and rationale for choosing to develop an epidemiology management website system. It also outlines the main objectives of the project.

#### **Health risks:**

The primary concern of any pandemic is the health risk it poses to the global population. COVID-19, for instance, has proven to be a highly infectious epidemic, causing severe illness and death in a significant number of cases. The virus primarily affects the respiratory system, but its impact extends to other organs, leading to a range of complications. The elderly and those with underlying health conditions are particularly at risk, further straining healthcare systems. Take a look at this insightful infographic which delves into the promising role of mesenchymal stem cells (MSCs) in COVID-19 treatment, showcasing the rapid global spread of the virus and the anticipated therapeutic approaches.

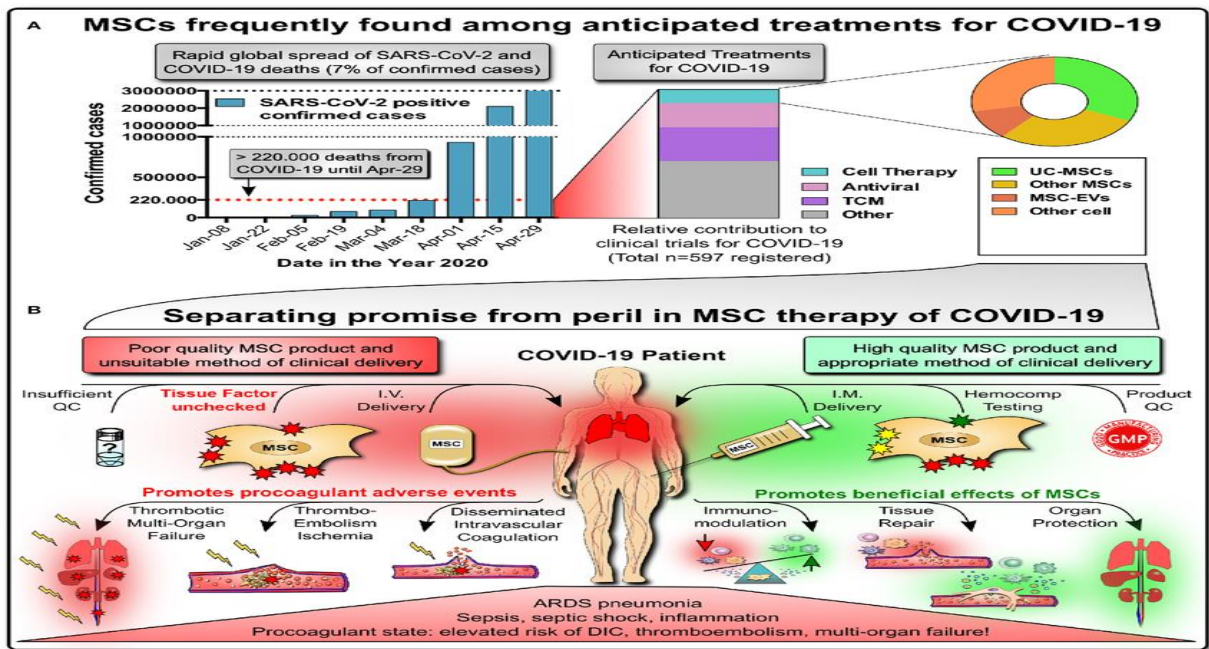


Figure 2.1 Covid health risks [10]

Figure 2.1 presents the rapid global spread of SARS-CoV-2 and associated deaths, anticipated treatments for COVID-19, and particularly Mesenchymal Stem Cells (MSCs) therapy. It shows how MSCs can be separated from periosteum tissue and their potential therapeutic effects against COVID-19. The infographic also discusses the risks associated with MSC therapy. It provides a comprehensive view of the role of MSC therapy in treating COVID-19.

### **Rapid spread:**

The rapid spread of COVID-19 has been facilitated by globalization and increased human mobility. The virus quickly traversed borders, infecting millions worldwide within a few months. This rapid spread has overwhelmed health systems, leading to shortages of medical supplies and healthcare workers, thereby exacerbating the crisis.

### **Epidemiology:**

Understanding the epidemiology of a pandemic is crucial for its management. Epidemiologists study the distribution and determinants of health-related states in specific populations and apply this study to control health problems. In the case of COVID-19, epidemiological studies have been instrumental in understanding the virus's



transmission dynamics, informing public health interventions. Look at this comprehensive visual representation which captures the early stages of the COVID-19 pandemic, detailing the initial case numbers, spread patterns, and key milestones in the outbreak's evolution.

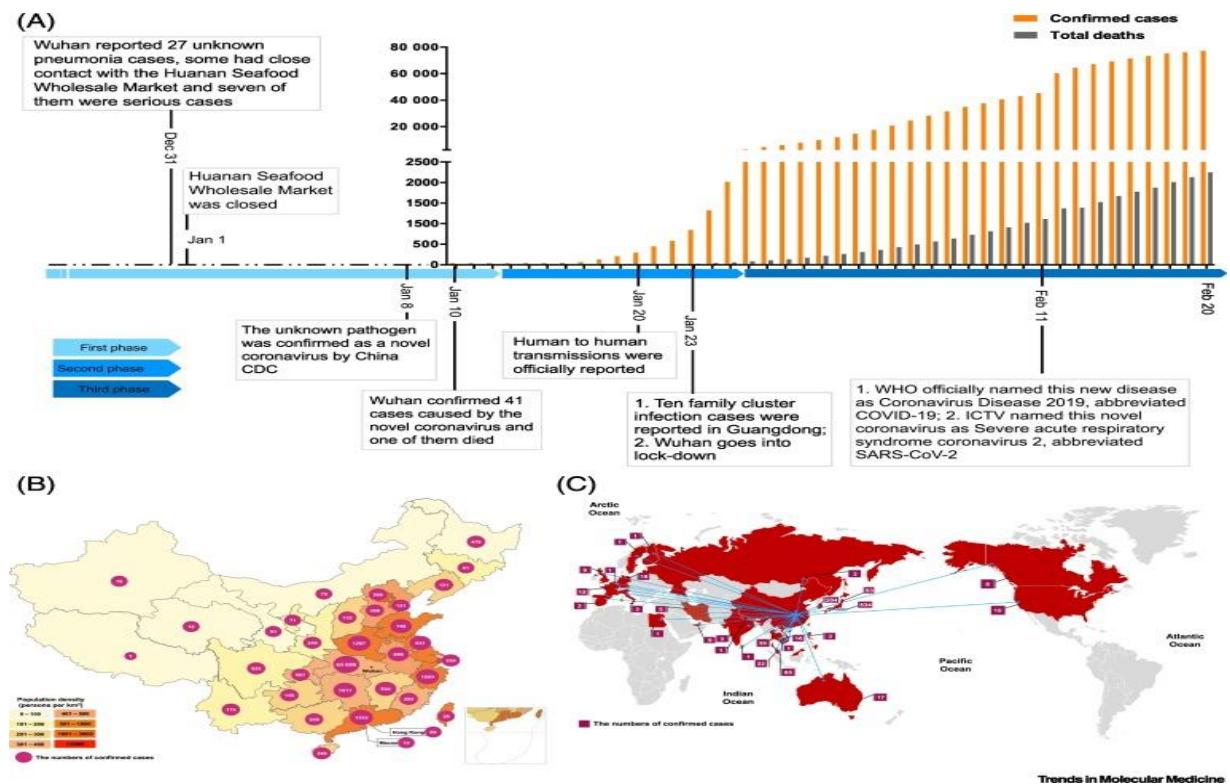


Figure 2.2 Covid-19 pandemic [11]

Figure 2.2 presents for three separate graphs related to the COVID-19 pandemic. Graph A shows a timeline from December 2019 to February 2020, indicating the number of confirmed cases and deaths, with key events such as the closure of the Wuhan seafood market and WHO announcements. Graph B is a map of China showing the distribution of confirmed cases across provinces. Graph C is a global map indicating the spread of COVID-19 to other countries. This image represents the early spread and impact of COVID-19 globally.

### **Isolation and treatment:**

Isolation and treatment are key strategies in managing a pandemic. Isolation helps to curb the spread of the virus, while treatment protocols ensure that those infected

receive appropriate care. However, these measures require substantial resources and pose significant logistical challenges. In the case of COVID-19, the sudden surge in cases led to a shortage of isolation facilities and essential medical supplies.

### **Public communication:**

Effective communication is vital during a pandemic. Authorities must keep the public informed about the situation, including the risks involved, preventive measures, and progress in combating the epidemic. During the COVID-19 pandemic, regular briefings and updates from health authorities have been crucial in managing public expectations and ensuring adherence to safety protocols.

### **Online epidemic management system:**

The COVID-19 pandemic has highlighted the importance of leveraging technology in epidemic management. An Online Epidemic Management System 4.0 can play a pivotal role in tracking the spread of the epidemic, managing resources, and facilitating communication between healthcare providers and the public. Such a system can provide real-time data, enabling swift responses to change in the pandemic’s trajectory. [9]

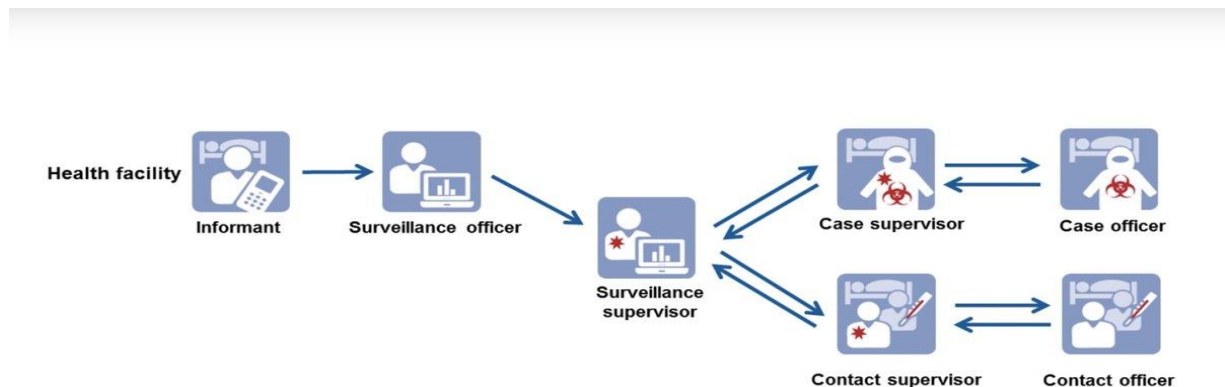


Figure 2.3 Online epidemic management system [13]

Figure 2.4 presents a health surveillance process. It starts with a “Health facility”, leading to an “Informant” and then to a “Surveillance officer.” From there, it splits into two paths: one leading to a “Case supervisor,” then to multiple “Case officers,” and another

path leading to a “Surveillance supervisor,” which further leads to a “Contact supervisor” and then multiple “Contact officers.” This flowchart outlines the hierarchy and workflow in health surveillance, crucial for understanding how health data is managed and how cases are followed up within an organization.

## 2.2 Function overview

### 2.2.1 Overview use case diagram

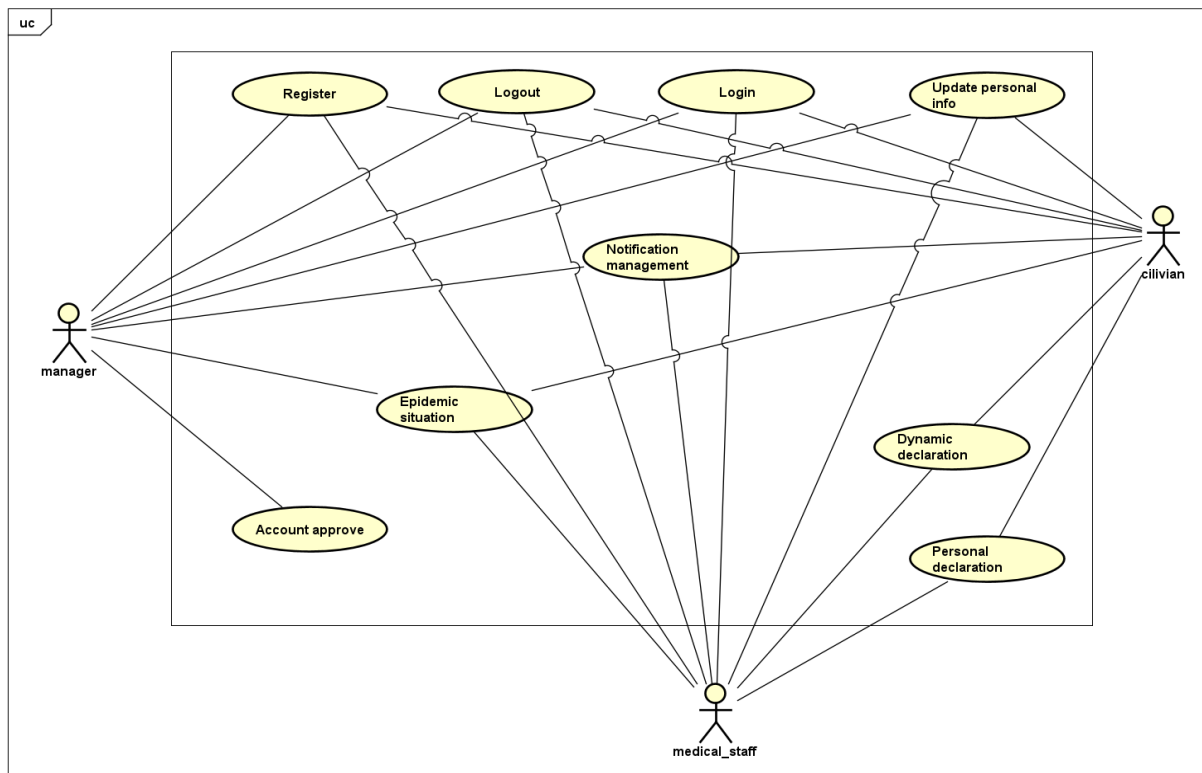


Figure 2.4 Overview use case diagram

Figure 2.5 presents for three main actors: manager, medical staff and civilian.[2]  
Description below:

- For manager, there are the following main use cases: register, login, logout, update personal info, notification management, approve account, update epidemic situation, etc.

- For medical staff, there are the following main use cases: register, login, logout, update personal info, post notification, epidemic situation, dynamic declaration, etc.
- For civilian, there are the following main use cases: register, login, logout, notification management, personal declaration, etc.

### 2.2.2 Decay use case diagram

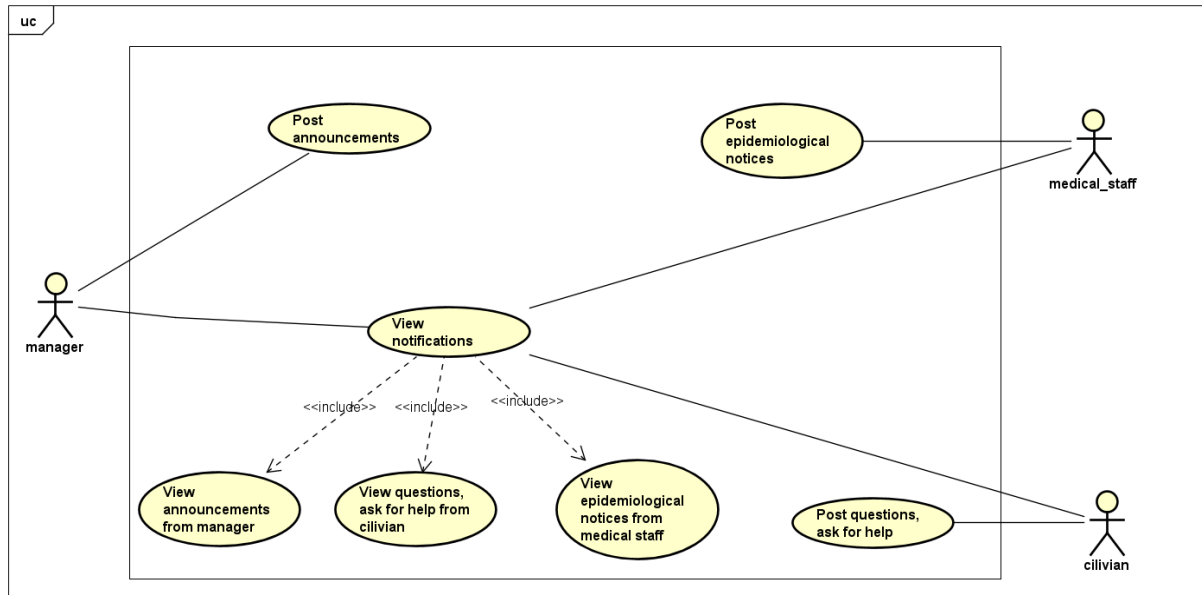


Figure 2.5 Decay usecase chart notification management

Figure 2.6 presents for actors: “manager,” “medical\_staff,” and “civilian”. Description below:

#### **Actor functions:**

- The “manager” actor has three use cases associated with them: “Post announcements,” “View notifications,” and “View announcements from medical staff.”
- The “medical\_staff” actor has four use cases: “Post epidemiological notices,” “View notifications,” “View epidemiological notes from medical staff,” and “Post questions, ask for help.”

- The “civilian” actor is connected to two use cases: “View notifications” and “View questions, ask for help from civilian.”

**Relationships between functions:**

- The “Post announcements” function of the “manager” actor seems to be included in the “View announcements from medical staff” function of the same factor, indicating that the manager can view the announcements they post.
- The “Post epidemiological notices” function of the “medical\_staff” factor is included in the “View epidemiological notes from medical staff” function, suggesting that medical staff can view the notices they post.
- The “View notifications” function is common to all factors, indicating that all factors can view notifications.
- The “Post questions, ask for help” function of the “medical\_staff” factor and the “View questions, ask for help from civilian” function of the “civilian” actor seem to be related, suggesting a platform where civilians can view and respond to questions posted by medical staff.

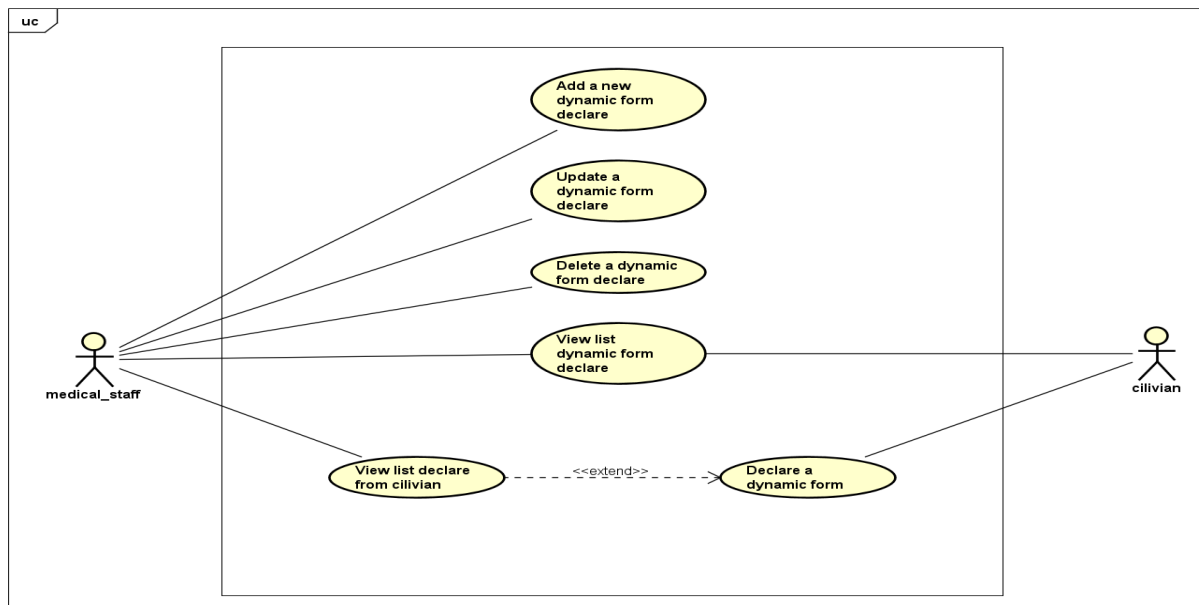


Figure 2.6 Decay usecase chart dynamic declaration

Figure 2.7 presents for actors: “medical\_staff” and “civilian”. Description below:

### Actor functions:

- The “medical\_staff” actor has three use cases associated with them: “View list declare from civilian,” and three other functions that are not fully visible due to the image cutoff.
- The “civilian” actor is connected to four use cases: “Add a new dynamic form declare,” “Update a dynamic form declare,” “Delete a dynamic form declare,” and “Declare a dynamic form.”

### Relationships between functions:

- The “Add a new dynamic form declare,” “Update a dynamic form declare,” and “Delete a dynamic form declare” functions of the “civilian” actor extend to the “View list declare from civilian” function of the “medical\_staff” actor, indicating that any additions, updates, or deletions made by civilians can be viewed by the medical staff.

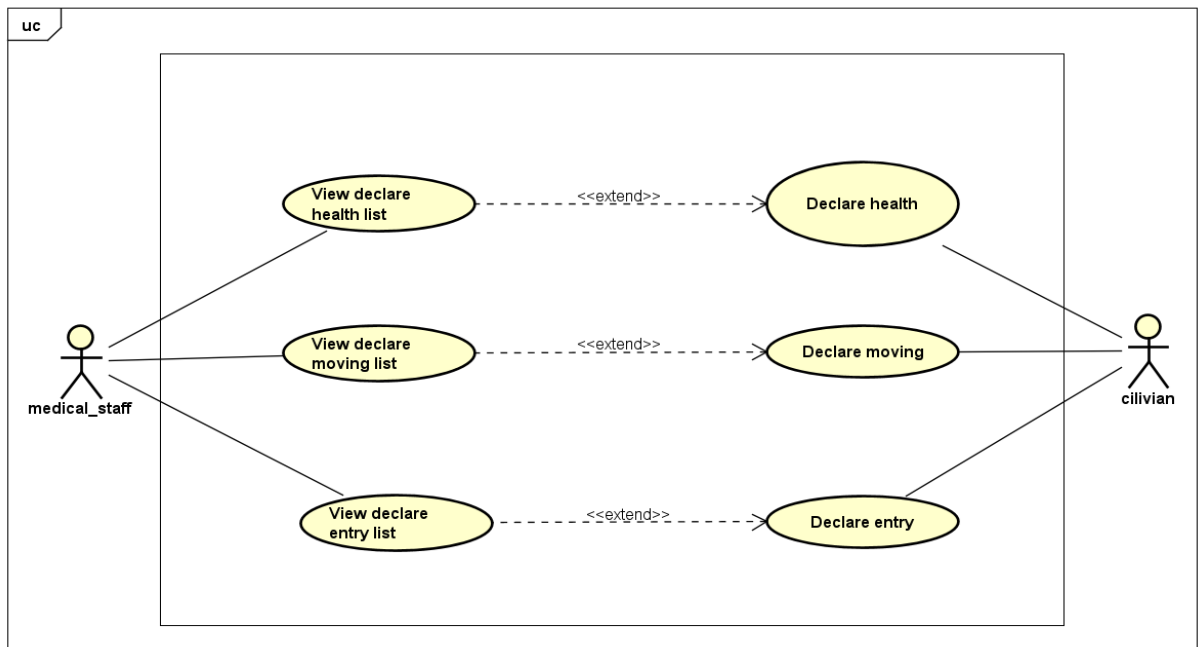


Figure 2.7 Decay usecase chart personal declaration

Figure 2.8 presents for actors: “medical\_staff” and “civilian”. Description below:

## Actor functions:

- The “medical\_staff” actor has three use cases associated with them: “View declare health list,” “View declare moving list,” and “View declare entry list.”
- The “civilian” actor is connected to three use cases: “Declare health,” “Declare moving,” and “Declare entry.”

## Relationships between functions:

- The “Declare health” function of the “civilian” actor extends to the “View declare health list” function of the “medical\_staff” actor, indicating that health declarations made by civilians can be viewed by the medical staff.
- Similarly, the “Declare moving” function of the “civilian” actor extends to the “View declare moving list” function of the “medical\_staff” actor, suggesting that moving declarations made by civilians can be viewed by the medical staff.
- The “Declare entry” function of the “civilian” actor extends to the “View declare entry list” function of the “medical\_staff” actor, indicating that entry declarations made by civilians can be viewed by the medical staff.

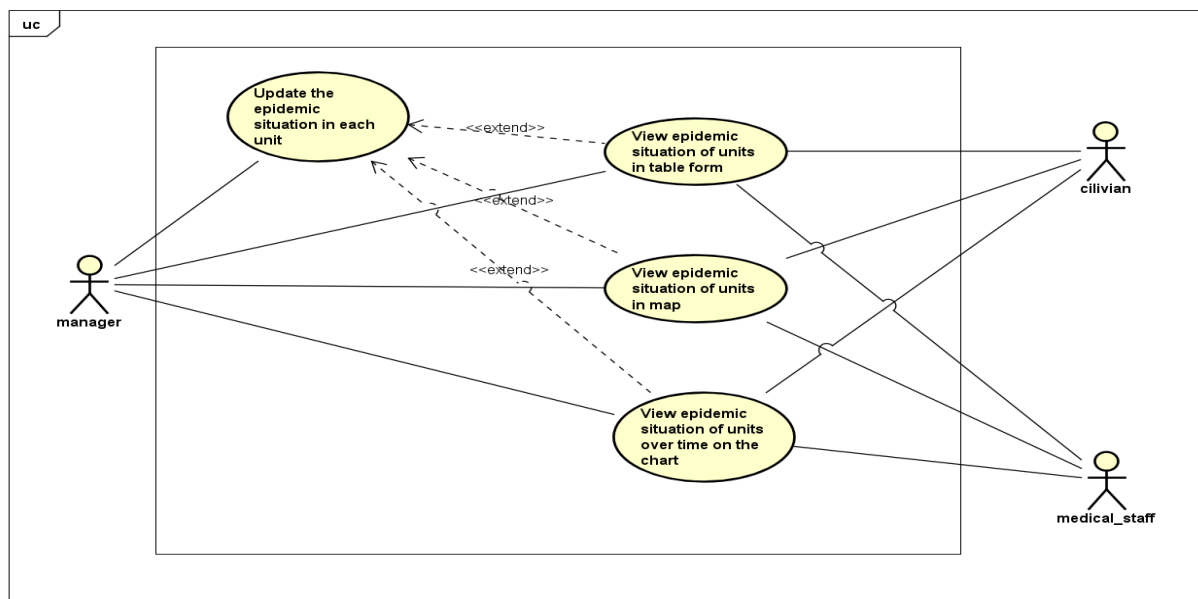


Figure 2.8 Decay usecase chart epidemic situation

Figure 2.9 presents for actors: “manager,” “medical\_staff,” and “civilian”. Description below:

**Actor functions:**

- The “manager” actor has one use case associated with them: “Update the epidemic situation in each unit.”
- The “medical\_staff” actor has two use cases: “View epidemic situation of units in table form” and “View epidemic situation over time on the chart.”
- The “civilian” actor is connected to two use cases: “View epidemic situation of units in table form” and “View epidemic situation of units in map.”

**Relationships between functions:**

- The “Update the epidemic situation in each unit” function of the “manager” actor seems to extend to the “View epidemic situation of units in table form” function of the “medical\_staff” actor, indicating that updates made by the manager can be viewed by the medical staff in a tabular format.
- Similarly, the “View epidemic situation of units in table form” function of the “medical\_staff” actor extends to the same function of the “civilian” actor, suggesting that civilians can also view the epidemic situation of units in a tabular format.
- The “View epidemic situation of units in map” function of the “civilian” actor seems to be an alternative way for civilians to view the epidemic situation.

**2.3 Conclusion**

The survey and requirements analysis have given us a detailed understanding of the essential components needed for the system. By gathering data from stakeholders and analyzing technical factors, we have identified specific requirements, guiding the system's development process. Next, we will explore the technology used for realizing the identified requirements.



## CHAPTER III: OVERVIEW OF TECHNOLOGY

With the requirements identified, this chapter presents the software and technology used. This is an important foundation for building the system.

### 3.1 Frontend

The front-end development of this project relies on various technologies and languages, including HTML, CSS, JavaScript, and jQuery. Despite not being the latest technologies, they effectively solve challenges related to User Interface (UI) and User Experience (UX).

**Initial Stage:** At the beginning, HTML, CSS, and JavaScript are used to create the basic structure of the website. HTML forms the backbone by defining the structure, CSS styles the user interface to make it visually appealing, and JavaScript adds dynamic interactions to make the site interactive.

#### 3.1.1 jQuery

**UI-UX Development Stage:** During the development phase focused on enhancing UI and UX, jQuery is used to add dynamic effects, animations, and interactions to the website without sacrificing performance. This lightweight library is easy to deploy, integrates seamlessly into projects, and aids in SEO optimization.

#### **Advantages of jQuery:**

jQuery, a popular JavaScript library, simplifies complex tasks with its concise syntax, reducing the amount of code required for common operations. It addresses cross-browser compatibility issues, ensuring consistent behavior across different web browsers and saving developers from handling these inconsistencies. jQuery's intuitive API makes DOM manipulation, event handling, and animation straightforward for both beginners and experienced developers. Additionally, jQuery enhances AJAX request efficiency, simplifying asynchronous communication. Its extensive plugin library allows developers to quickly add pre-built functions to their projects, saving development time and effort.

Overall, jQuery facilitates more efficient and maintainable code, making it a valuable tool for creating dynamic and interactive web applications.

### **Further benefits of jQuery:**

- Lightweight, improving page load speeds.
- Easy to deploy and integrate.
- Enhances SEO for better search engine performance.

### **Disadvantages of jQuery:**

While jQuery has been beneficial in simplifying web development, it also has its downsides. One notable drawback is its potential impact on page load times, as it adds to the overall size of the JavaScript files that need to be downloaded and executed. This can be particularly problematic on mobile devices with slower network connections. In modern web development, where performance optimization is critical, the additional weight of jQuery can be a disadvantage. Moreover, as browsers have advanced, native JavaScript has become more powerful, reducing the necessity for jQuery in some scenarios. Developers skilled in modern JavaScript frameworks might find jQuery adds unnecessary complexity and overhead to their projects. Additionally, jQuery may not perform optimally with complex web applications that require high flexibility. While jQuery remains useful in certain situations, developers should weigh its benefits against these potential drawbacks and consider alternatives when appropriate.

### **Responsive Design:**

Cross-Platform Deployment Stage: Responsive design ensures that the website is adaptable and accessible across various devices, from mobile phones and tablets to desktops. This approach guarantees a consistent and user-friendly experience regardless of the device being used.

## Future plan:

Using ReactJS: The long-term strategy involves transitioning to ReactJS for the front-end development. ReactJS was chosen for its high performance and code reusability, which will enhance the user experience and reduce application complexity. This move aims to improve scalability and streamline code management, aligning with modern development standards.



Figure 3.1 Front-end technology [14]

Figure 3.1 presents eight different web development technologies or concepts. These include “HTML 5 Semantic”, “CSS 3 More Action”, “Bootstrap Flexibility”, “Javascript Event Based”, “JQuery CrossBrowsing”, and “AJAX Asynchronous”. Each icon symbolizes a key element of modern web design and programming, highlighting the unique features of each technology. This image is relevant for those interested in web development as it visually summarizes important technologies used to create dynamic and responsive websites.

### 3.1.2 Leaflet technology

Leaflet is a widely used open-source JavaScript library for interactive maps, offering a simple and lightweight solution for embedding maps on web pages. It supports various map layers, including tiled maps, vector layers, and markers, enabling developers to create rich, interactive map experiences. Leaflet's design emphasizes performance, usability, and ease of use, making it ideal for both novice and experienced developers. [5]

One of the standout features of Leaflet is its responsiveness and compatibility with mobile devices. It provides smooth animations and seamless interaction on both desktop and mobile platforms, ensuring a consistent user experience across different devices. The library also supports a wide range of map providers, such as OpenStreetMap, Mapbox, and Google Maps, giving developers flexibility in choosing the best map service for their application.

Leaflet's extensibility is another key advantage. It offers numerous plugins that extend its core functionality, allowing for features like heatmaps, clustering, and geocoding. This extensibility makes it easy to customize and enhance the map according to specific project requirements. Additionally, Leaflet's straightforward API and extensive documentation facilitate quick learning and implementation, reducing the development time.

In summary, Leaflet's lightweight nature, cross-platform compatibility, and extensive plugin ecosystem make it a powerful tool for integrating interactive maps into web applications. Its user-friendly approach and robust performance capabilities ensure that developers can deliver high-quality, responsive maps with minimal effort.

#### **Advantages:**

- **Lightweight and Efficient:** Leaflet is known for being lightweight, which means it has a smaller file size compared to other mapping libraries like Google Maps

API or OpenLayers. This efficiency leads to faster loading times and improved performance, particularly beneficial for web applications with limited resources.

- **Easy to Use:** Leaflet offers a simple and intuitive API, making it easy for developers to integrate and use. Its straightforward syntax and well-documented examples allow even those with minimal experience in mapping technologies to get started quickly.
- **Customizable:** Leaflet is highly customizable. Developers can easily add custom map layers, markers, popups, and other features. The extensive plugin ecosystem allows for adding functionalities such as heatmaps, routing, and drawing tools without much effort.
- **Open Source:** Being open source, Leaflet is free to use and modify. This fosters a collaborative community where developers can contribute to the library, report issues, and share plugins and extensions, enhancing the overall functionality and reliability of the tool.
- **Responsive Design:** Leaflet supports responsive design out of the box, ensuring that maps look good and are usable on a variety of devices, from desktops to mobile phones. This adaptability is crucial in modern web development.

### **Disadvantages:**

- **Limited Advanced Features:** While Leaflet excels in simplicity and ease of use, it lacks some of the advanced features found in more robust mapping solutions like Google Maps. For example, it does not natively support features like indoor mapping, street view, or detailed 3D visualization.
- **Dependency on Third-Party Services:** Leaflet often relies on third-party tile providers (e.g., OpenStreetMap, Mapbox) for map tiles. This dependency can introduce limitations or costs, especially if the external service has usage restrictions or requires a subscription for higher levels of access.
- **Performance with Large Datasets:** When dealing with very large datasets or high-frequency real-time data, Leaflet's performance can degrade. Unlike some

commercial alternatives optimized for handling extensive data sets efficiently, Leaflet may require additional optimization techniques or plugins to maintain performance.

- **Less Support and Documentation Compared to Major Alternatives:** While Leaflet has good documentation and community support, it is not as extensive as that of major commercial mapping solutions like Google Maps API. This can sometimes make it harder to find solutions to complex problems or get support for less common use cases.

### 3.1.3 GeoJSON

GeoJSON is a widely used format for encoding a variety of geographic data structures using JavaScript Object Notation (JSON). It is a powerful tool for representing geographical features, their attributes, and even their spatial extents. GeoJSON supports various geometry types, such as points, lines, polygons, multi-points, multi-lines, and multi-polygons, enabling it to describe complex spatial relationships. This format is lightweight and human-readable, making it an excellent choice for web-based mapping applications. [8] Below is an example of GeoJSON to Map.

#### **Advantages of GeoJSON in building web maps:**

- **Simplicity and Readability:** GeoJSON's use of JSON makes it easy to read and understand for both humans and machines. This simplicity facilitates quick debugging and seamless data exchange between the server and the client.
- **Compatibility:** GeoJSON is natively supported by many web mapping libraries, including Leaflet, Mapbox, and OpenLayers. This broad compatibility ensures that developers can integrate spatial data into web maps with minimal effort.
- **Interactivity:** The JSON format allows for the inclusion of rich metadata alongside geographic coordinates. This feature enhances the interactivity of web maps, enabling features such as pop-ups, tooltips, and custom styling based on attribute data.

- Performance: GeoJSON is lightweight, which can lead to faster load times and better performance, especially for applications that require quick rendering of spatial data in the browser.
- Versatility: GeoJSON can represent both simple and complex geometries, making it suitable for a wide range of mapping applications, from simple point markers to detailed multi-polygon regions.

### **Disadvantages of GeoJSON in building web maps:**

- Scalability Issues: While GeoJSON is suitable for small to medium-sized datasets, its performance can degrade with very large datasets. The verbosity of JSON can lead to large file sizes, which may impact load times and browser performance.
- Limited Precision: GeoJSON stores coordinate as floating-point numbers, which can result in precision loss for highly detailed spatial data. This limitation might be significant for applications requiring high-precision geographic information.
- Lack of Advanced Features: GeoJSON does not natively support more advanced geographic features like projections, topology, or spatial indexes. These limitations can necessitate additional processing or the use of complementary formats and tools for advanced spatial analysis.
- Security Concerns: As with any data format used in web applications, there are security considerations. GeoJSON, being a text-based format, can be susceptible to injection attacks if not properly sanitized and validated.
- Browser Dependency: The performance and capabilities of GeoJSON rendering can vary depending on the browser and device. While modern browsers handle GeoJSON well, older or less powerful devices may struggle with complex or large datasets.

### 3.1.4 CanvasJS

CanvasJS is a versatile and powerful JavaScript library designed to create interactive and dynamic charts and graphs. It leverages the HTML5 Canvas element to render high-performance charts, making it an excellent choice for visualizing data in web applications. CanvasJS supports a wide variety of chart types, including line charts, bar charts, pie charts, and more, and it is known for its ease of use, flexibility, and smooth animations. This makes CanvasJS a popular tool for developers looking to embed real-time data visualization into their websites and applications. [6]

#### **Advantages of using CanvasJS for drawing charts to represent epidemic data:**

- **Interactivity:** CanvasJS charts are highly interactive, allowing users to hover, click, and zoom into specific data points. This interactivity can be particularly useful in epidemic data visualization, where users may need to explore detailed trends and patterns.
- **Performance:** CanvasJS is optimized for performance, providing smooth rendering even with large datasets. This is crucial when dealing with extensive epidemic data that can include numerous data points and complex time-series visualizations.
- **Ease of Use:** The library is user-friendly, with a straightforward API that simplifies the process of creating and customizing charts. Developers can quickly set up and integrate charts into their web pages, reducing development time and effort.
- **Customization:** CanvasJS offers extensive customization options, allowing developers to tailor the look and feel of the charts to match the specific requirements of epidemic data visualization. This includes custom colors, labels, tooltips, and more.
- **Cross-Browser Compatibility:** CanvasJS ensures that charts look consistent and perform well across different web browsers, which is essential for reaching a



broad audience. This reliability makes it a robust choice for public-facing epidemic data dashboards.

### **Disadvantages of using CanvasJS for drawing charts to represent epidemic data:**

- **Learning Curve:** While CanvasJS is user-friendly, there is still a learning curve associated with mastering its API and features. New users may require some time to become proficient in using the library effectively.
- **Limited Free Features:** CanvasJS offers a limited free version, with some advanced features and functionalities restricted to the paid version. This can be a limitation for developers or organizations with budget constraints.
- **Performance with Extremely Large Datasets:** Although optimized for performance, rendering extremely large datasets can still pose challenges. In such cases, additional techniques like data aggregation or server-side processing might be necessary to maintain performance.
- **Dependency on JavaScript:** CanvasJS relies heavily on JavaScript, which means that users with JavaScript disabled in their browsers will not be able to view the charts. Additionally, developers need to ensure that their JavaScript code is secure and optimized.
- **Complex Customizations:** For very specific or complex customizations, developers might find CanvasJS to be somewhat restrictive compared to more comprehensive data visualization libraries. Advanced customizations may require additional coding and workaround solutions.

## 3.2 Backend

### 3.2.1 Node.js and express.js

Node.js is a powerful open-source server environment that runs on various platforms, including Windows, Linux, and Mac OS. Built on Chrome's V8 JavaScript engine, Node.js uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient. This model is particularly well-suited for real-time applications, such as chat applications and live data feeds, where maintaining numerous simultaneous connections is essential. [4] Below is an illustration of how the express.js application framework for Node.js.

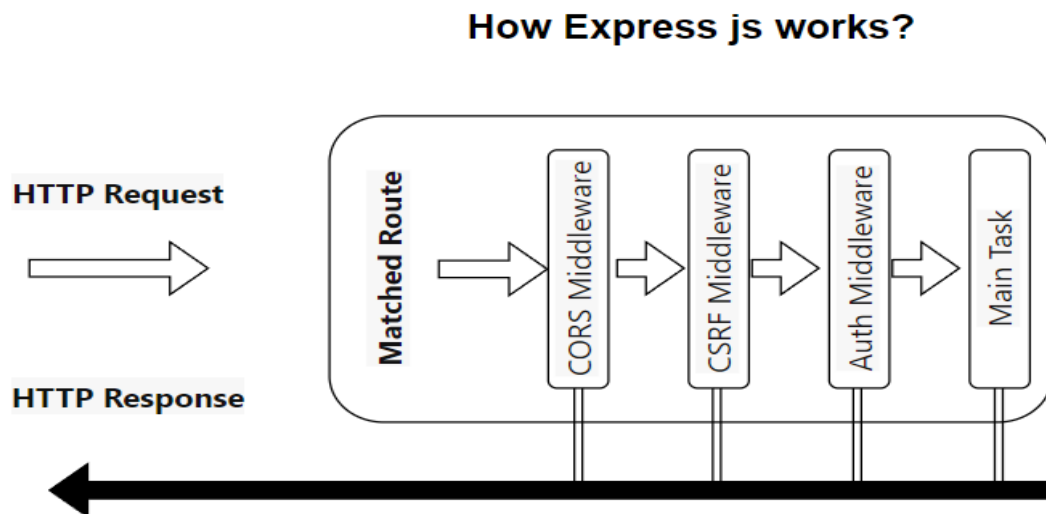


Figure 3.2 How Express js works?

Figure 3.5 presents how Express.js, a web application framework for Node.js, works. It shows the flow of an HTTP request through various middleware functions before it reaches the main task and then returns an HTTP response. The process begins with an HTTP Request being sent to a Matched Route, then sequentially passing through CORS Middleware, CSRF Middleware, Auth Middleware, and finally reaching the Main Task. After processing in the Main Task, an HTTP Response is generated and sent back. This diagram is significant as it visually represents the middleware pattern used in Express.js, which is crucial for developers to understand for building efficient web applications.

Node.js's package ecosystem, npm (Node Package Manager), is one of the largest and most vibrant in the world. It offers a vast range of libraries and tools that streamline development tasks, allowing developers to focus more on coding and less on configuration and setup. This ecosystem's richness contributes significantly to Node.js's popularity among developers. [4]

Express.js, a minimalist web framework for Node.js, provides a robust set of features for building web and mobile applications. It simplifies the development of server-side applications by offering a thin layer of fundamental web application features, without obscuring Node.js's core functionalities. Express.js is known for its simplicity and flexibility, allowing developers to organize their application in a way that suits their needs.

Express.js handles a variety of HTTP requests and supports robust routing. It allows developers to define routes for different HTTP methods and URLs, making it easier to manage different parts of a web application. Middleware functions in Express.js can process requests before reaching the final route handler, which is useful for tasks like authentication, logging, and data validation.

The combination of Node.js and Express.js provides an efficient environment for developing scalable web applications. Node.js's asynchronous nature ensures that applications can handle a high number of simultaneous connections with minimal overhead. Express.js adds structure and organization to Node.js applications, enabling developers to create robust, maintainable code. Together, they are a powerful duo for modern web development, particularly in building RESTful APIs and full-stack applications.

### **Advantages:**

- **Fast and Efficient:** Node.js uses the V8 JavaScript engine from Google, which compiles JavaScript directly to machine code, making it extremely fast. Its non-blocking, event-driven architecture allows it to handle multiple requests

simultaneously, which is highly efficient for I/O-heavy operations like API requests or accessing databases.

- **Single Programming Language:** With Node.js, developers can use JavaScript for both client-side and server-side development. This unification simplifies the development process, allows for code reuse, and makes it easier for front-end developers to transition to back-end development.
- **Extensive Ecosystem:** Node.js has a rich ecosystem with npm (Node Package Manager), which provides access to a vast repository of open-source libraries and modules. This allows developers to quickly add functionality to their applications without reinventing the wheel.
- **Scalability:** Node.js is designed for building scalable network applications. It can handle a large number of simultaneous connections with high throughput, making it suitable for real-time applications like chat apps and live streaming services.
- **Active Community:** Node.js has a large and active community that contributes to its ongoing development and support. This means frequent updates, a plethora of tutorials, and a wealth of shared knowledge that developers can tap into.
- **Express.js Framework:** Express.js, built on top of Node.js, is a minimalist web framework that simplifies the development of web applications and APIs. It provides a robust set of features for building single-page, multi-page, and hybrid web applications.

### **Disadvantages:**

- **Callback Hell:** Node.js heavily relies on callbacks to handle asynchronous operations, which can lead to deeply nested callback structures known as "callback hell". This can make the code difficult to read and maintain. However, this issue can be mitigated by using Promises or async/await syntax.
- **Single-threaded Limitations:** While Node.js's single-threaded nature is efficient for I/O-bound operations, it can be a limitation for CPU-bound tasks. Heavy computation tasks can block the event loop, leading to performance bottlenecks. Solutions like worker threads or offloading tasks to separate processes can help, but they add complexity.
- **Immaturity of Tooling:** Compared to more mature ecosystems like Java or .NET, some developers may find Node.js's tooling and libraries to be less polished or

mature. While this is improving rapidly, it can still be a concern for larger, enterprise-level applications.

- **Lack of Strong Typing:** Node.js and Express.js are primarily JavaScript-based, which is a dynamically typed language. This can lead to issues with type safety and can make large codebases harder to manage. However, using TypeScript with Node.js can mitigate this issue by adding static type checking.
- **Security Concerns:** As with any web technology, security can be a concern. Node.js applications, especially when using numerous third-party packages, can be vulnerable to security threats if not properly managed. Regular updates and security audits are essential to mitigate these risks.
- **Concurrency Model:** Node.js's concurrency model, which uses the event loop, is not suited for applications requiring heavy multi-threading. While suitable for I/O-bound tasks, it may not perform as well as multi-threaded environments for CPU-intensive applications.

### **3.2.2 MongoDB**

MongoDB is a popular open-source NoSQL database known for its high performance, scalability, and flexibility. Unlike traditional relational databases, MongoDB stores data in flexible, JSON-like documents, which allows for a more dynamic schema. This flexibility makes MongoDB an excellent choice for applications that require quick iterations and frequent changes to data structures. [3]

One of the key features of MongoDB is its ability to handle large volumes of data and high-throughput operations. Its distributed architecture supports horizontal scaling, meaning that as your data grows, you can distribute it across multiple servers or clusters. This scalability is crucial for applications that anticipate significant growth and need to maintain performance under heavy loads.

MongoDB also excels in handling complex data types and hierarchical data models. Its document-oriented storage model allows embedding of documents within documents, making it easier to represent complex relationships in a natural and intuitive

way. This feature reduces the need for expensive joins and can lead to more efficient query performance.

The query language in MongoDB is another strength, offering a rich set of operators and expressions for filtering and transforming data. MongoDB's aggregation framework supports operations such as filtering, grouping, sorting, and reshaping data, enabling sophisticated data analysis directly within the database. Additionally, MongoDB provides powerful indexing capabilities, including geospatial and text search indexes, which enhance query performance and support diverse application needs.

For developers, MongoDB offers a range of tools and integrations. The MongoDB Atlas cloud service provides automated deployment, scaling, and backups, reducing the operational overhead. The MongoDB Stitch backend-as-a-service platform allows developers to connect to various services and create serverless applications. Moreover, the community around MongoDB is vibrant, with extensive documentation, tutorials, and a wealth of third-party libraries and tools available. Below is an image depicting the MongoDB database backup and restore process.

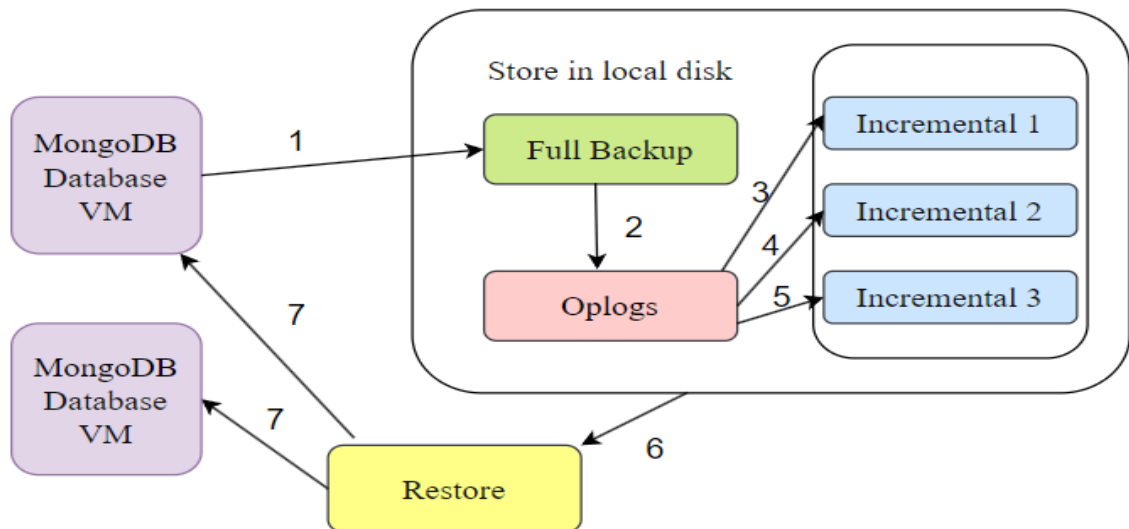


Figure 3.3 MongoDB database backup and restore process

Figure 3.8 presents for depicts a MongoDB database backup and restore process. It involves a full backup stored on a local disk, and three incremental backups. The database and 'Database VM' are central components. The 'Oplogs' box represents operation logs. The process also includes a restore operation from the local disk to the Database VM.

### **Advantages:**

- **Flexible Schema Design:** MongoDB uses a document-oriented data model, which allows for flexible schema design. Unlike traditional relational databases, MongoDB doesn't require a predefined schema, enabling developers to store and manage data in a JSON-like format (BSON). This flexibility is ideal for applications where data structures can evolve over time.
- **Scalability:** MongoDB is designed with horizontal scalability in mind. It supports sharding, which allows data to be distributed across multiple servers or clusters. This makes it easy to scale out as the data volume grows, providing high availability and redundancy.
- **High Performance:** MongoDB's architecture is optimized for read and write performance. Its ability to handle large volumes of unstructured data and perform operations quickly makes it suitable for high-traffic applications. Indexing, replication, and the in-memory storage engine further enhance performance.
- **Rich Query Language:** MongoDB offers a powerful and flexible query language that supports a wide range of operations, including filtering, sorting, and aggregations. This makes it easy to perform complex queries and retrieve the necessary data efficiently.
- **Ease of Use:** MongoDB's intuitive document model aligns closely with how developers work with data in modern applications. The syntax is simple and familiar to those who have experience with JavaScript, making it easier to learn and implement.
- **Strong Community and Ecosystem:** MongoDB has a large and active community, along with extensive documentation and a wealth of third-party tools and libraries. This vibrant ecosystem provides ample resources for learning and troubleshooting, as well as numerous integrations with other technologies.

## **Disadvantages:**

- **Memory Usage:** MongoDB can be memory-intensive, especially when handling large volumes of data. It often requires more RAM to store indexes and frequently accessed data in memory for faster access. This can lead to higher operational costs for larger deployments.
- **Data Redundancy:** Due to its flexible schema, MongoDB often leads to data redundancy. This denormalization can result in larger database sizes and potential inconsistencies. Unlike relational databases, where normalization minimizes redundancy, MongoDB requires careful schema design to manage this issue.
- **Lack of ACID Transactions:** While MongoDB has made strides in supporting multi-document ACID (Atomicity, Consistency, Isolation, Durability) transactions in recent versions, it historically lacked full ACID compliance. This can be a drawback for applications requiring complex transactions across multiple documents or collections.
- **Complexity in Handling Relationships:** MongoDB is less suited for applications with complex relationships and joints between data entities. While it supports embedded documents and references, handling relational data can be more complex and less efficient compared to traditional relational databases.
- **Limited Reporting and Analysis Tools:** While MongoDB provides powerful querying capabilities, it lacks the built-in reporting and analytical tools found in some relational databases. This can necessitate additional tools or integrations for comprehensive data analysis and reporting.
- **Consistency Concerns:** MongoDB's default configuration favors eventual consistency over immediate consistency. While this improves performance and scalability, it can lead to scenarios where read operations may not immediately reflect the most recent write operations. This behavior may not be suitable for applications requiring strict data consistency.

In summary, MongoDB's flexibility, scalability, and rich feature set make it a compelling choice for modern application development. Its ability to handle diverse data types, combined with powerful query and aggregation capabilities, supports the creation of sophisticated, high-performance applications. Whether used as the primary database



or as part of a broader data strategy, MongoDB provides the tools and flexibility needed to manage and analyze large-scale data efficiently.

### 3.2.3 Firebase

Firebase is a comprehensive app development platform backed by Google, designed to help developers build high-quality apps quickly and efficiently. It provides a suite of cloud-based tools and services, enabling developers to focus more on creating compelling user experiences and less on managing infrastructure. Firebase’s capabilities span across various essential functions, including real-time databases, authentication, cloud storage, and analytics. [1] Below is an illustration presenting the contrasting traditional and Firebase application architectures.

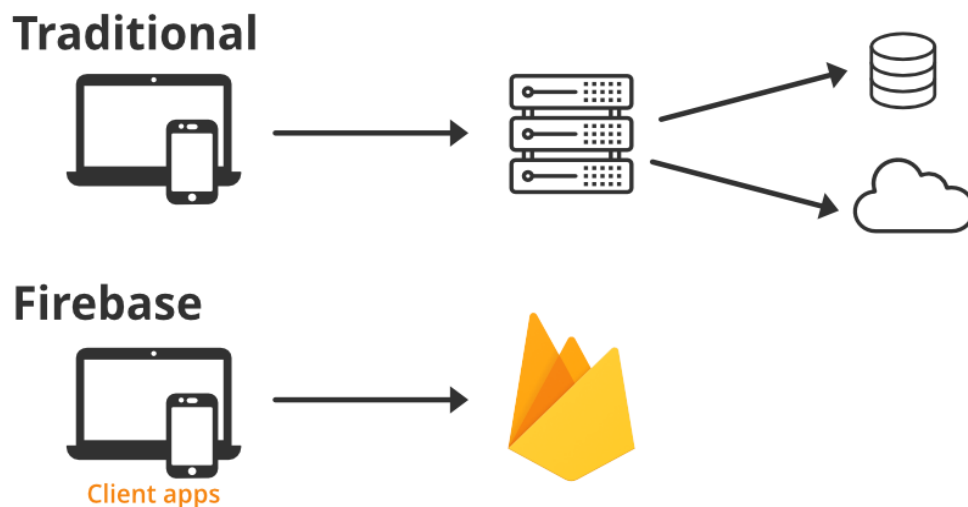


Figure 3.4 Firebase [19]

Figure 3.9 presents contrast traditional and Firebase application architectures. In the Traditional model, data flows from a mobile device to server racks, then to a cloud service, and finally to a database. The Firebase model simplifies this process by directly connecting client apps to Firebase, eliminating intermediate steps. This highlights Firebase’s efficiency and simplicity for app development.

One of Firebase's core features is the Realtime Database, a NoSQL cloud database that stores data in JSON format and synchronizes it in real-time with all connected clients. This is particularly useful for applications that require instant updates, such as chat applications, collaborative tools, or live data feeds. Firebase also offers Firestore, a more flexible and scalable database, which supports complex querying and hierarchical data structures.

Authentication is another crucial service provided by Firebase. It simplifies the process of managing user authentication and authorization. With Firebase Authentication, developers can easily integrate various sign-in methods, including email and password, phone authentication, and social logins such as Google, Facebook, and Twitter. This service helps to enhance security while providing a smooth user experience.

Firebase Cloud Storage enables developers to store and serve user-generated content such as photos, videos, and other media files. It is built for robust, secure file handling, and scales automatically as the app grows. Firebase also includes Cloud Functions, which allows developers to run backend code in response to events triggered by Firebase features or HTTPS requests.

Additionally, Firebase Analytics provides free, unlimited reporting on up to 500 distinct events. This helps developers understand user behavior, measure the impact of changes, and make informed decisions based on real-time data. Coupled with Firebase's Crashlytics, developers can monitor and fix stability issues to improve app performance and user satisfaction. Below is an illustration of backing up a project's database and media files.

# Firestore + Storage bucket back up

( Short guide on how to back up your project database + media files)



Figure 3.5 Firebase storage [20]

Figure 3.10 presents for backing up a project's database and media files using Firebase's Firestore and Storage bucket. It shows the process of transferring files to a secure location, emphasizing the importance of data integrity and recovery. This is crucial for preserving data in case of loss or damage. The image visually summarizes the process of securing data in cloud services. It features the Firebase logo, Firestore and Storage icons, and an illustration of file transfer.

## **Advantages:**

- **Real-time Database:** Firebase's real-time database allows for instantaneous data synchronization across all clients. This is particularly advantageous for applications requiring real-time updates, such as chat applications, live streaming services, and collaborative tools. Changes made in the database are immediately reflected on all connected clients.
- **Backend-as-a-Service (BaaS):** Firebase provides a comprehensive suite of backend services, including authentication, cloud storage, hosting, and cloud functions. This all-in-one solution eliminates the need for managing backend infrastructure, allowing developers to focus more on the front-end and application logic.
- **Scalability:** Firebase is built on Google's infrastructure, which means it can scale seamlessly to handle large amounts of data and high traffic loads. This makes it suitable for both small startups and large-scale enterprise applications.

- **Cross-platform Support:** Firebase offers extensive support for both web and mobile platforms, including iOS, Android, and the web. This cross-platform capability allows developers to use a single backend service to support multiple client platforms, streamlining development and maintenance.
- **Easy Integration and Use:** Firebase's SDKs are designed to be easy to integrate into applications. Its user-friendly console and detailed documentation further simplify the development process. Features like Firebase Authentication provide pre-built UI components that can be easily customized.
- **Security:** Firebase provides robust security features, including Firebase Authentication for user identity verification and Firebase Security Rules for controlling data access. These tools help ensure that only authorized users can access and manipulate data, enhancing the overall security of the application.

### **Disadvantages:**

- **Vendor Lock-in:** One of the main drawbacks of using Firebase is the potential for vendor lock-in. Since Firebase is a proprietary platform, migrating to another service can be challenging and time-consuming. This dependency can be a significant concern for businesses that require flexibility in their technology stack.
- **Limited Querying Capabilities:** While Firebase's real-time database is powerful, it has limited querying capabilities compared to traditional SQL databases. Complex queries involving multiple conditions or large datasets can be cumbersome and inefficient. Developers may need to implement workarounds, which can complicate the codebase.
- **Pricing:** Firebase offers a free tier with limited features, but costs can quickly escalate as the application's user base and data storage needs grow. Services like Firebase Realtime Database and Firebase Cloud Storage charge based on usage, which can become expensive for high-traffic applications.
- **Data Structure Constraints:** Firebase's NoSQL data structure can be less intuitive for developers accustomed to relational databases. Organizing data in a

hierarchical structure and managing complex relationships between data entities can be challenging and may require a different approach to data modeling.

- **Limited Server-side Logic:** While Firebase provides cloud functions to run server-side logic, there are limitations in terms of execution time, memory usage, and available runtime environments. This can be restrictive for applications requiring extensive backend processing or custom server-side operations.
- **Offline Capabilities:** Although Firebase supports offline data synchronization, handling offline scenarios can be complex, especially for applications with intricate data relationships and conflict resolution needs. Ensuring data consistency and managing conflicts when the connection is restored can require additional development effort.

In summary, Firebase offers a rich ecosystem of tools and services that cover all aspects of app development, from backend infrastructure and user authentication to real-time data synchronization and analytics. Its integration with other Google services and extensive documentation makes it a powerful choice for both small and large-scale applications.

### **3.2.4 JWT bearer**

JWT (JSON Web Token) Bearer is a widely used standard for securing APIs and transmitting information between parties as a JSON object. It is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

One of the main advantages of JWT is its simplicity and compactness. A JWT typically consists of three parts: a header, a payload, and a signature. The header contains metadata about the token type and the signing algorithm. The payload contains the

claims, which are statements about an entity (typically, the user) and additional data. The signature is used to verify the token's integrity and authenticity.

JWT Bearer tokens are commonly used for authorization. When a user logs in, a server generates a JWT and sends it to the client. The client then includes this token in the Authorization header of subsequent requests to access protected resources. The server verifies the token's signature and claims to ensure the request is authenticated.

In essence, JWT Bearer tokens offer a secure and efficient way to handle user authentication and authorization in modern web applications. Their stateless nature eliminates the need for server-side session storage, reducing overhead and enhancing scalability. Below is an illustration of the JWT Bearer Operating Model.

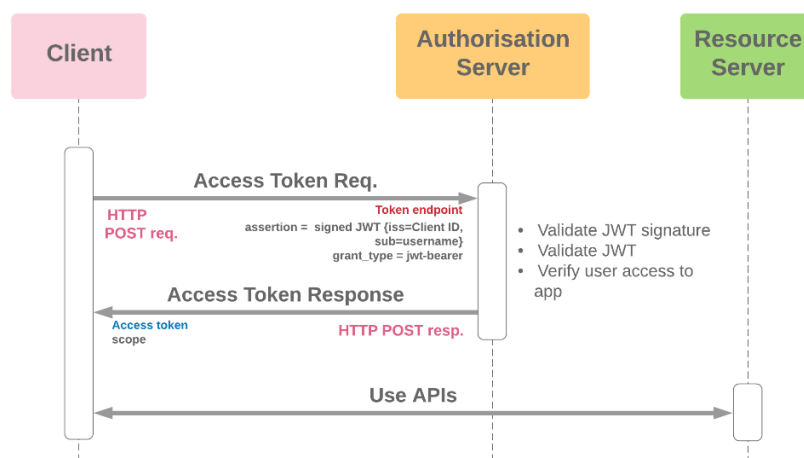


Figure 3.6 JWT bearer operating model [21]

Figure 3.11 presents the process of obtaining and using an access token in a system with separate client, authorization server, and resource server entities. JWT Bearer operating model.

### **Advantages:**

- **Statelessness:** JSON Web Tokens (JWT) enable stateless authentication, meaning the server does not need to store session information. This reduces server load and simplifies the scaling process. Each token contains all the necessary information, so the server can verify the token without maintaining session data.

- **Compact and Portable:** JWTs are compact and can be easily transmitted via URLs, POST parameters, or HTTP headers. Their JSON format makes them easy to read and generate, ensuring seamless integration with web and mobile applications.
- **Security:** JWTs can be signed using a secret (HMAC) or a public/private key pair (RSA or ECDSA). This ensures the token's integrity and authenticity, preventing tampering. When properly implemented, JWTs provide a secure mechanism for transmitting information between parties.
- **Versatile and Flexible:** JWTs are versatile and can store various types of claims, such as user roles and permissions. This makes them suitable for different authentication scenarios, including user authentication, API authentication, and single sign-on (SSO).
- **Cross-domain Authentication:** JWTs are particularly useful for cross-domain authentication. Since they do not require cookies, they can be easily used across different domains, making them ideal for microservices architectures and applications with multiple subdomains.
- **Reduced Server Load:** Since JWTs are self-contained, the server only needs to verify the token's signature to authenticate the user. This reduces the need for frequent database queries and can significantly improve the performance of the authentication system.

### **Disadvantages:**

- **Security Risks:** If not implemented correctly, JWTs can introduce security vulnerabilities. For example, storing sensitive information in the payload without proper encryption can expose it to potential attackers. Additionally, weak secret keys or improper management of key rotation can compromise the security of the tokens.
- **Token Size:** While JWTs are compact, they can still become relatively large if too much information is stored in the payload. This can impact performance, especially when used in HTTP headers or URL parameters, potentially leading to longer load times and increased bandwidth usage.
- **No Automatic Revocation:** One of the significant drawbacks of JWTs is the lack of built-in token revocation mechanisms. Once a token is issued, it remains valid

until it expires. If a token is compromised, there is no straightforward way to invalidate it immediately. This requires additional mechanisms, such as maintaining a token blacklist, to manage revocation.

- **Complexity in Managing Expiry:** Managing token expiry and refresh mechanisms can be complex. Short-lived tokens improve security but require implementing a seamless refresh process to ensure a smooth user experience. Long-lived tokens reduce the need for frequent refreshments but increase the risk if a token is compromised.
- **Overhead in Token Parsing:** Every request containing a JWT requires the server to parse and verify the token, which adds some computational overhead. While this is generally minimal, it can become significant in high-traffic applications where every request must be authenticated.
- **Not Suitable for Large Payloads:** JWTs are not ideal for storing large amounts of data. The more information included in the payload, the larger the token becomes, which can negatively affect performance and efficiency.

### **3.3 Conclusion**

This chapter has provided a comprehensive view of the technologies, from selecting appropriate technologies to designing system and planning development. These decisions are critical for the project's success, ensuring the system will operate efficiently and meet the defined requirements. Next, we will move on to the actual development and deployment of the application, a crucial step to turn plans and designs into a complete product.



## CHAPTER IV: APPLICATION DEVELOPMENT AND DEPLOYMENT

After outlining the architecture and technology, this chapter focuses on application development and deployment. This is an important step in incorporating specific requirements into the system.

### 4.1 Architectural design

#### 4.1.1 Software architecture selection

The system consists of three main components:

- Client: executes on the user's web browser, where the interface is displayed for user interaction with the system.
- Server: where requests from the Client are processed.
- Database: where data is stored.

For the basic functions of the system, the data flow will be as follows: the user interacts with the system through the device screen, then the Client sends HTTP Requests to the server. The server receives the request and processes it (may retrieve data from the Database if needed), then returns the information to the Client in the form of HTTP response. The Client receives the Response and extracts relevant information to generate HTML/CSS pages to display to the user. Below is an illustration of the client-server model in web communications.

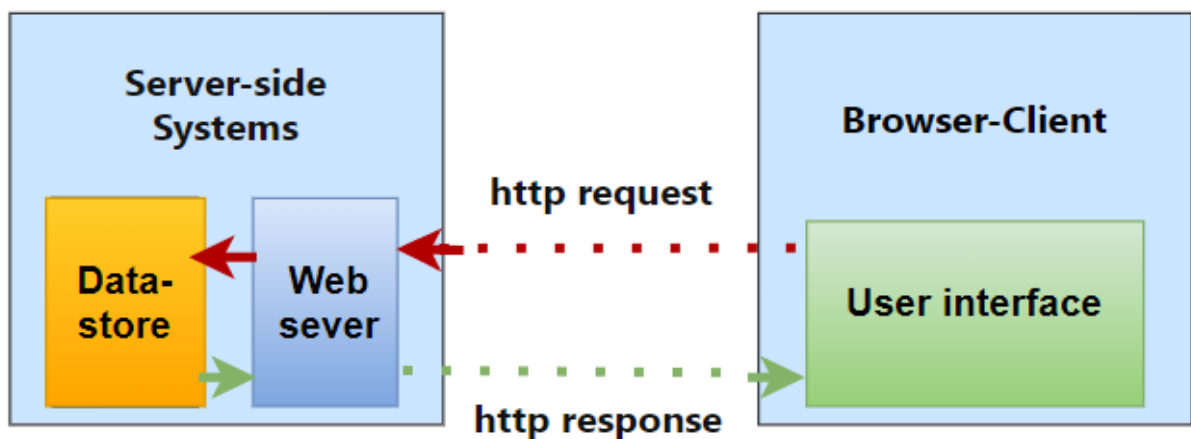


Figure 4.1 Client server

Figure 4.1 presents the client-server model in web communications. It shows server-side systems, including a data-store and a web server. The browser-client sends an HTTP request to the server, which responds with data. This data is then displayed on the user interface. This model is fundamental to understanding web technology.

#### 4.1.2 Design overview

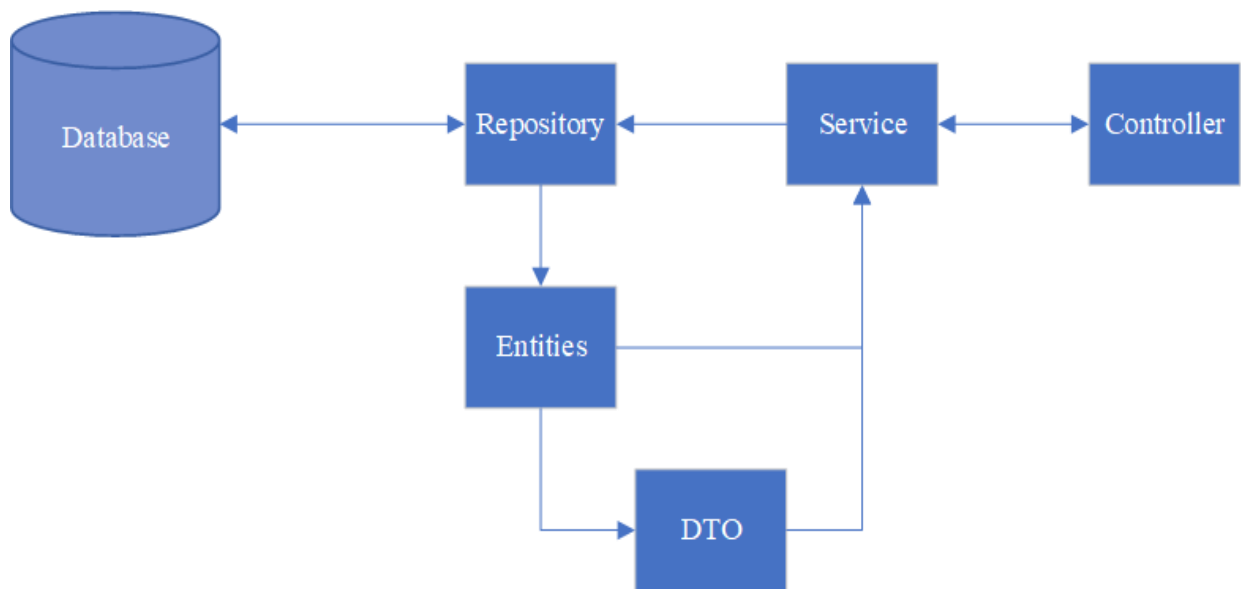


Figure 4.2 Design overview

Figure 4.2 presents a flowchart of a software application's data management system. It includes five components: Database, Repository, Service, Controller, and Entities and Data Transfer Object (DTO). The arrows show the data flow. The Database connects to the Repository, which links to the Service and Entities. The Service connects to the Controller and Entities. Entities link to DTO. This represents a layered architecture in software design. Draw by draw.io

Express.js, a minimalist web application framework for Node.js, paired with MongoDB, a NoSQL database, forms a powerful combination for building dynamic and scalable web applications. Understanding their operational model sheds light on how they work seamlessly together to handle requests, manage data, and deliver content efficiently.

At the core of Express.js lies its middleware system, which intercepts and processes HTTP requests. Middleware functions can perform tasks such as parsing request bodies, authenticating users, and logging requests. This modular architecture allows developers to structure their applications in a clear and organized manner, chaining middleware functions to handle requests sequentially.

When a client sends a request to an Express.js server, the request is routed to the appropriate route handler based on the URL and HTTP method. Route handlers are functions that execute specific logic and generate a response to the client. Express.js provides a simple and intuitive syntax for defining routes and handling various types of requests, making it easy to develop RESTful APIs and web applications.

MongoDB serves as the data storage layer for Express.js applications, offering a flexible and scalable solution for managing structured and unstructured data. MongoDB stores data in JSON-like documents, making it well-suited for handling complex data structures and nested relationships. Express.js applications can interact with MongoDB using the official MongoDB Node.js driver or higher-level libraries like Mongoose, which provide a schema-based approach for defining data models and performing CRUD operations.

The operational model of Express.js - MongoDB revolves around asynchronous programming and non-blocking I/O, leveraging the event-driven architecture of Node.js to handle concurrent requests efficiently. Express.js applications can handle a large number of concurrent connections without blocking the event loop, ensuring optimal performance and responsiveness.

Integration between Express.js and MongoDB is seamless, thanks to the asynchronous nature of both technologies. Express.js route handlers can interact with MongoDB databases asynchronously, performing database operations such as querying, inserting, updating, and deleting data without blocking the execution of other requests.

This asynchronous model allows Express.js applications to handle complex data operations without sacrificing performance or responsiveness.

### 4.1.3 Database design

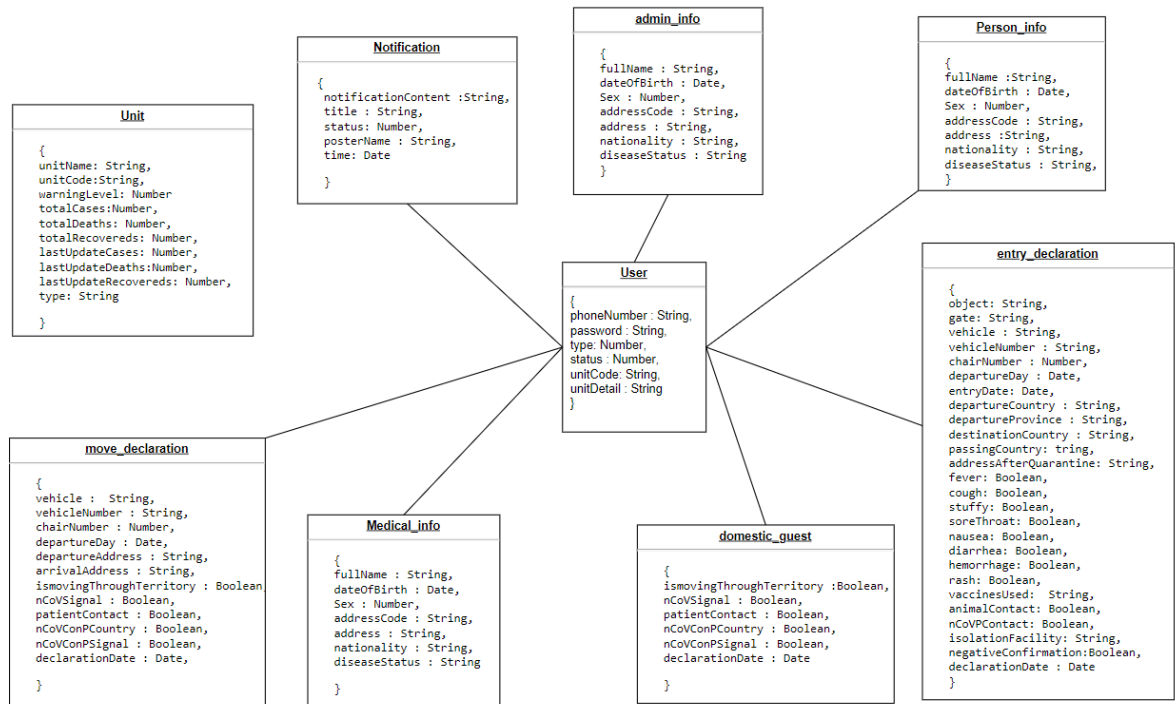


Figure 4.3 Database design diagram

Figure 4.3 presents for the database includes tables of users, notification, unit, person\_info, medical\_info, admin\_info, entry\_declaration, move\_declaration, domestic\_guest.

#### User schema

Field	Type	Null
phoneNumber	string	not null
password	string	not null
type	number	not null

status	number	not null
unitCode	string	not null
unitDetail	string	not null

Table 4.1 User schema

User schema: To store user account information.

- phoneNumber: stores user phone number information
- password: stores user password hash code information
- type: saves the user type. The 3 categories corresponding to values 0,1,2 are residents, managers and medical staff
- status: saves the status of the account, 0 is active and 1 otherwise
- unitCode: stores the user's unit code
- unitDetail: stores the name of the user unit

### **Admin info schema**

Field	Type	Null
fullName	string	not null
dateofBirth	date	not null
Sex	bool	not null
addressCode	string	not null
address	string	
nationality	string	not null
epidemicStatus	number	

Table 4.2 Admin\_info schema

Admin\_info schema: To store administrator information.

- fullName: To store the manager's name
- dateOfBirth: To store the manager's date of birth

- Sex: To Store Gender Manager
- addressCode: To store the manager address code
- address: To store the manager's text address
- nationality: To store the country name of the manager
- epidemicStatus: To store the manager's status. 0 is normal and 1 is infected

**Person info schema**

Field	Type	Null
fullName	string	not null
dateOfBirth	date	not null
Sex	bool	not null
addressCode	string	not null
address	string	
nationality	string	not null
epidemicStatus	number	

Table 4.3 Person\_info schema

Person\_info schema: To store civilian information.

- fullName: To store people's names
- dateOfBirth: To store people's birthdays
- Sex: To store the sex of the people
- addressCode: To store people's address codes
- address: To store people's text addresses
- nationality: People's Country

- epidemicStatus: People's status, 0 is normal and 1 is infected

**Medical info schema**

Field	Type	Null
fullName	string	Not null
dateOfBirth	date	Not null
Sex	bool	Not null
addressCode	String	Not null
address	String	
nationality	String	Not null
epidemicStatus	number	

Table 4.4 Medical\_info schema

Medical\_info schema: To store medical staff information.

- fullName: Save the name of the medical staff
- dateOfBirth: Date of birth of medical staff
- Sex: Gender of Medical Staff
- addressCode: Medical staff address code
- address: Text address of medical staff
- nationality: Nationality of Medical Staff
- epidemicStatus: medical staff's medical condition, 0 is normal and 1 is infected

### **Domestic\_guest schema**

Field	Type	Null
ismovingThroughTerritory	bool	not null
nCoVSignal	bool	not null
patientContact	bool	not null
nCoVConPCountry	bool	not null
nCoVConPSignal	bool	not null
declarationDate	date	not null

Table 4.5 Domestic\_guest schema

Domestic\_guest: To store general declaration information.

- ismovingThroughTerritory: Do you pass through the epidemic area
- nCoVSignal: whether there are signs of the epidemic
- patientContact: Have you been in contact with a sick person or are suspicious
- nCoVConPCountry: Have you been in contact with people from epidemic countries
- nCoVConPSignal: having contact with people who show signs of epidemic
- declarationDate: declaration date

### **Move declaration schema**

Field	Type	Null
vehicle	string	not null
vehicleNumber	string	not null



chairNumber	number	not null
departureDay	date	not null
departureAddress	string	not null
arrivalAddress	string	not null
ismovingThroughTerritory	bool	not null
nCoVSignal	bool	not null
patientContact	bool	not null
nCoVConPCountry	bool	not null
nCoVConPSignal	bool	not null
declarationDate	date	not null

Table 4.6 Move\_declaration schema

Move\_declaration schema: To store move declaration information.

- vehicle: Transportation
- vehicleNumber: Vehicle number
- chairNumber: Number of seats
- departureDay: Departure date
- departureAddress: Departure Address
- arrivalAddress: Destination Address
- ismovingThroughTerritory: Do you pass through a country or territory
- nCoVSignal: whether there are signs of the epidemic
- patientContact: having contact with a sick person or suspecting
- nCoVConPCountry: having contact with people from epidemic countries
- nCoVConPSignal: having contact with people who show signs of epidemic

- declarationDate: declaration date

**Entry declaration schema**

Field	Type	Null
object	string	not null
gate	string	not null
vehicle	string	not null
vehicleNumber	string	not null
chairNumber	number	not null
departureDay	date	not null
entryDate	date	not null
departureCountry	string	not null
departureProvince	string	not null
destinationCountry	string	not null
passingCountry	string	not null
addressafterQuarantine	string	not null
fever	bool	not null
cough	bool	not null
stuffy	bool	not null
soreThroat	bool	not null
nausea	bool	not null

diarrhea	bool	not null
hemorrhage	bool	not null
rash	bool	not null
vaccineused	string	not null
animalContact	bool	not null
nCoVPCocontact	bool	not null
isolationfacility	string	not null
negativeconfirmation	bool	not null
declarationDate	date	not null

Table 4.7 Entry\_declaration schema

Entry\_declaration schema: To store immigration declaration information.

- object: Object
- gate: Border Gate
- vehicle: vehicle entry
- vehicleNumber: Vehicle number
- chairNumber: Number of seats
- departureDay: Departure date
- entryDate: Date of Entry
- departureCountry: Country of departure
- departureProvince: Departure City
- destinationCountry: Country of Arrival
- passingCountry: What country has it traveled through
- addressafterQuarantine: Accommodation address after concentrated quarantine

- fever: fever or not
- cough: cough or not
- stuffy: stuffy or not
- soreThroat: sore throat or not
- nausea: nausea or not
- diarrhea: diarrhea or not
- hemorrhage: hemorrhage or not
- rash: rash or not
- vaccineused: type of vaccine used
- animalContact: Have contact with wildlife
- nCoVPContact: Have you been in contact with people with symptoms of the epidemic
- isolationfacility: isolation facility
- negativeconfirmation: negative confirmation
- declarationDate: declaration date

**Unit schema**

Field	Type	Null
unitName	string	not null
unitCode	string	not null
waringLevel	number	not null
totalCases	number	
totalDeaths	number	
totalRecoveredds	number	
lastUpdateCases	number	

lastUpdateDeaths	number	
lastUpdateRecoveredds	number	
type	string	not null

Table 4.8 Unit Schema

Unit schema: To store local unit information. (From small to large units: ward - district - city)

- unitName: Unit Name
- unitCode: Unit Code
- waringLevel: epidemic level, there are levels 0,1,2,3
- totalCases: total number of cases
- totalDeaths: Total Deaths
- totalRecoveredds: total number of recoveries
- lastUpdateCases: number of new cases
- lastUpdateDeaths: new deaths
- lastUpdateRecoveredds: number of new recoveries
- type: unit type, 3 values are w,d,p corresponding to ward, district, province

**Notification schema**

Field	Type	Null
notificationContent	string	not null
title	string	not null
status	number	not null
posterName	string	not null
time	date	not null

Table 4.9 Notification schema

Notification schema: To store announcement/post information

- notificationContent: The content of the announcement
- title: Announcement Title
- status: Post Status
- posterName: Name of the person who posted the announcement
- time: Announcement posting time

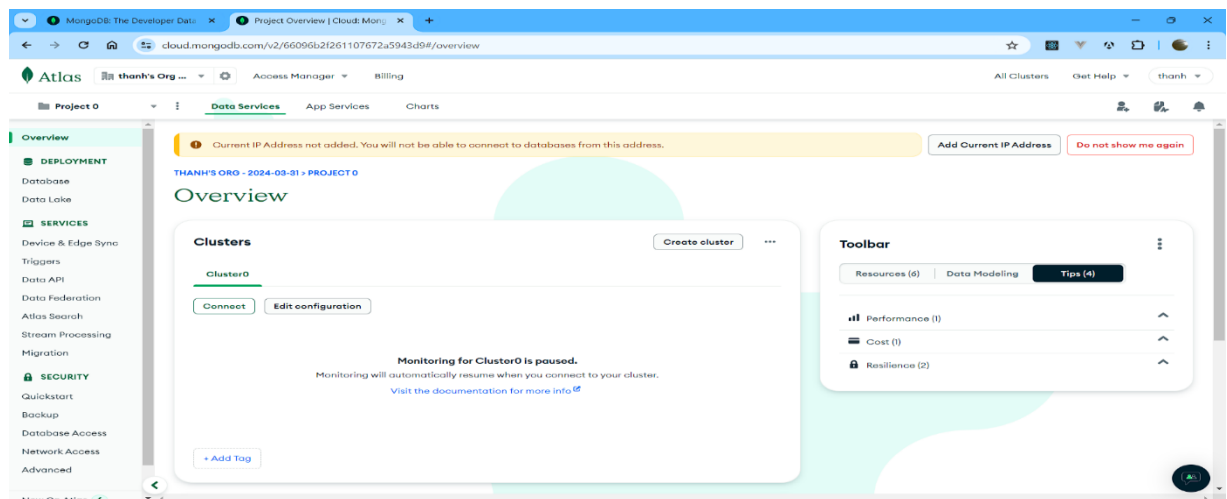
#### 4.1.4 Description of the web development

##### Database: MongoDB

About MongoDB: MongoDB is a non-relational, open-source, document-based database system designed to be flexible and easily scalable. It uses a JSON document structure to store data, making application development flexible and fast. MongoDB Cloud is a cloud service platform provided by MongoDB, providing easy and flexible MongoDB database management solutions in the cloud environment, helping to reduce the burden of system administration and increase availability and flexibility for applications.

Installation Steps: Use mongodb cloud to be able to install a mongodb instance through the steps:

- Create an account and log in at: [mongodb.com](https://mongodb.com)



- Go to the mongodb cloud console and select Deployment > Database

Figure 4.4 MongoDB deploy – deployment database

- Create a new cluster database instance

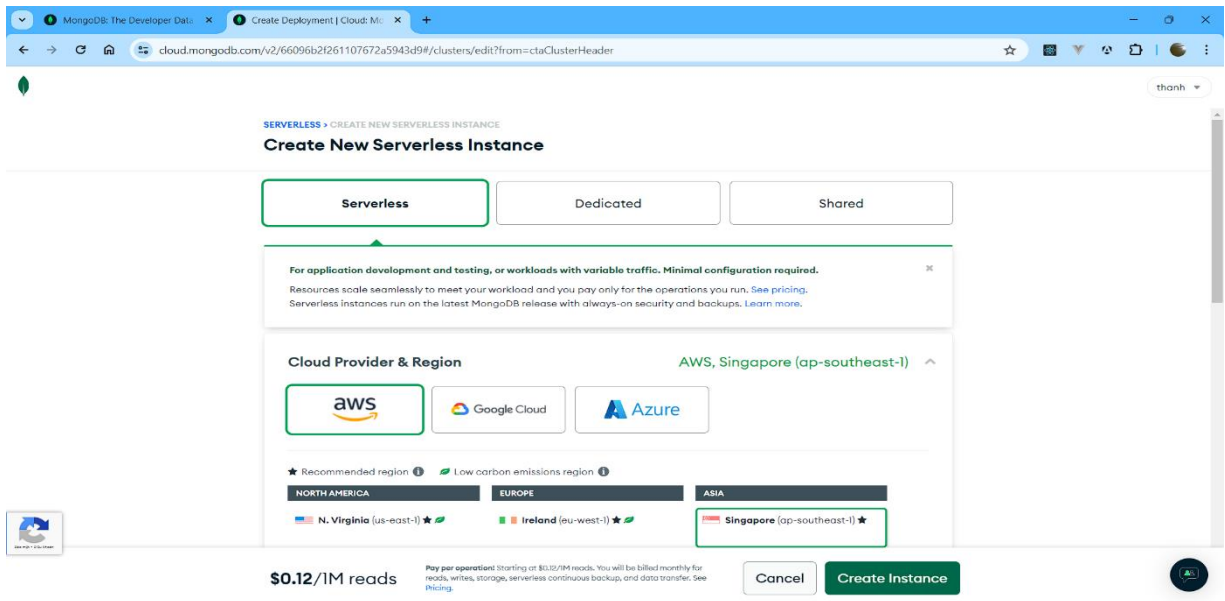


Figure 4.5 MongoDB deploy – create cluster

- Copy connection string to connect from backend

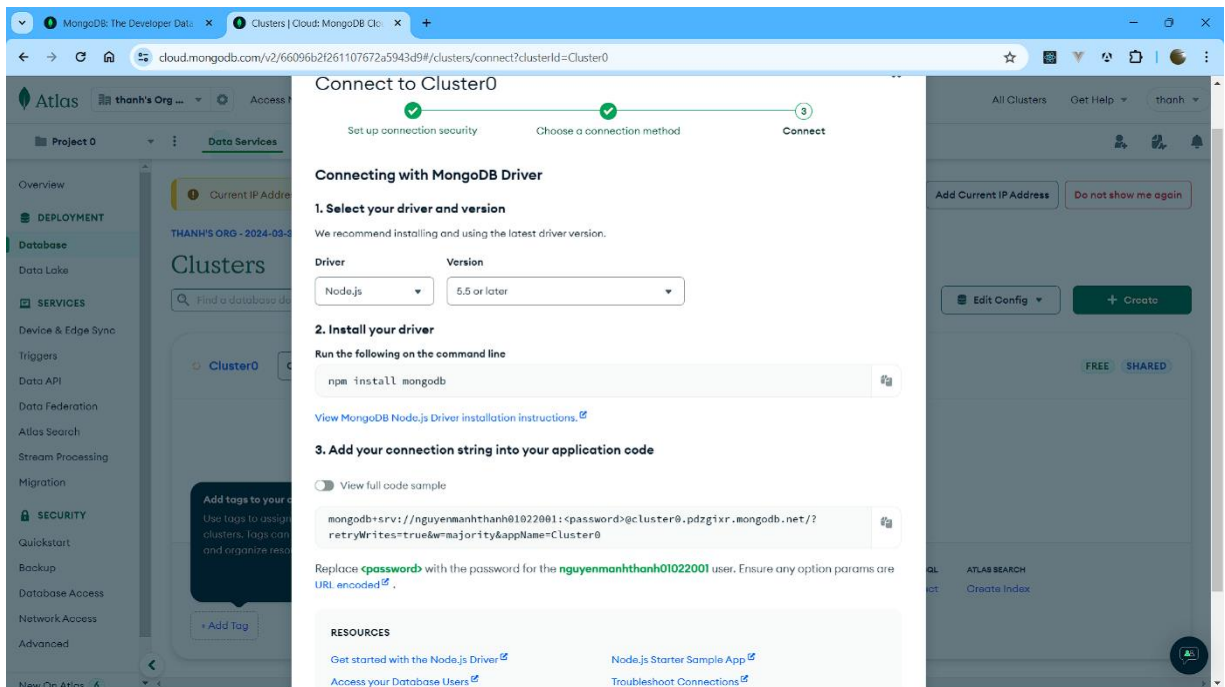


Figure 4.6 MongoDB deploy – connect from backend

Browse collections - similar to tables in sql. Each collection corresponds to 1 model in the nodejs backend.

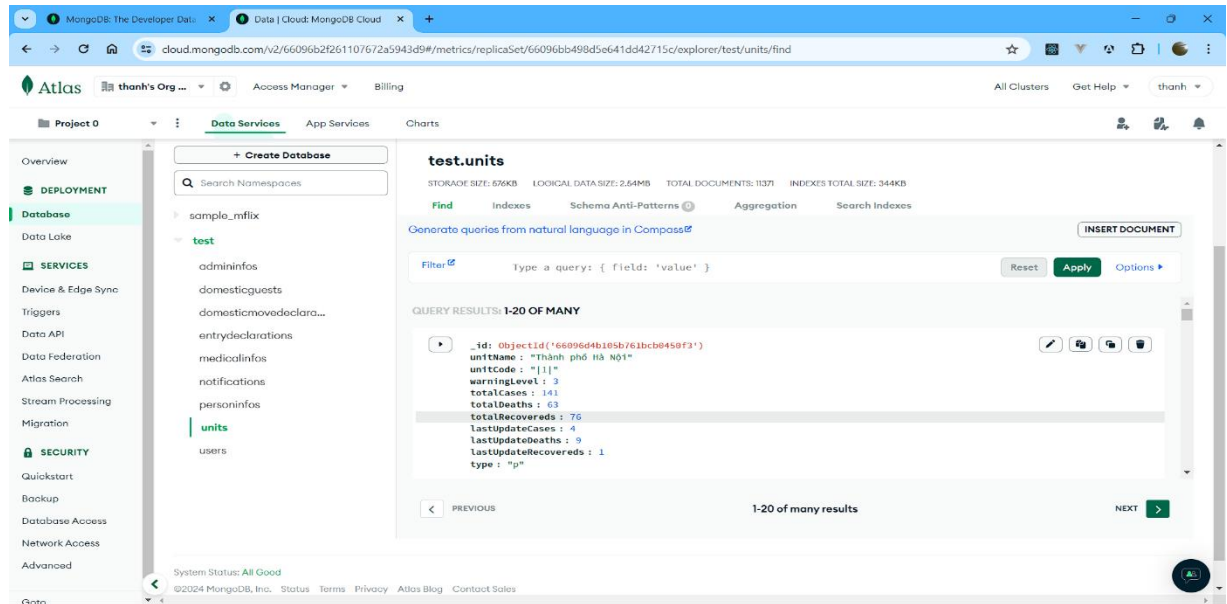


Figure 4.7 MongoDB deploy successfully

## **Backend: NodeJS x ExpressJS**

Node.js is an open-source execution environment built on a JavaScript platform for developing easily scalable network applications. With Node.js, developers can use JavaScript both server-side and client-side, creating a uniform development environment for web applications.

Express.js is an extremely flexible and powerful Node.js web application framework that is commonly used to build web applications and APIs. With Express.js, building web applications becomes simpler by providing basic features such as routing, middleware, and request and response management.

Node.js and Express.js are often used together to create efficient web applications with extensive, modular, and easily scalable source code. With a large community and



diverse support, Node.js and Express.js have become a popular pair of technologies in contemporary web application development.

Install the environment and backend code according to the steps:

- Install node.js from [nodejs.org](https://nodejs.org)
- Install express.js: open a terminal or command prompt and enter the following command  
`npm install express--save`
- Create a new expressjs app for the backend use the express command to generate a new project skeleton, or you can manually create the necessary files  
`express pandemic-management-be`
- The structure of the node.js backend project will follow the following main structure:

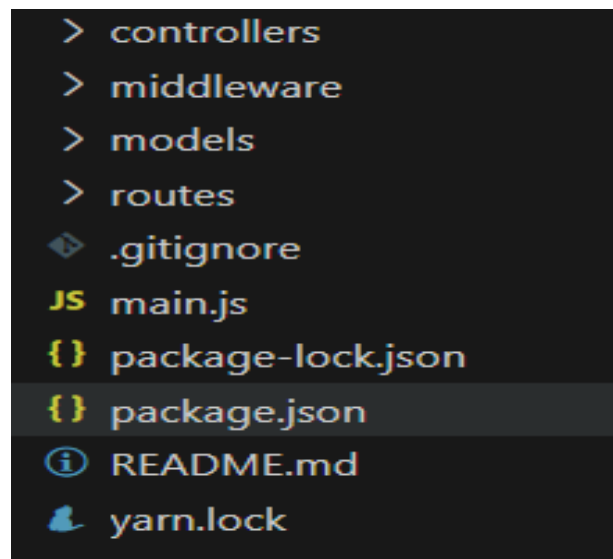


Figure 4.8 Backend folder structure

In which, the main file of the expressjs app is main.js

- Configure MongoDB: Install MongoDB on your machine from the [official MongoDB website](https://www.mongodb.com) and configure the connection in your project.
- Create new Mongoose model schemas: Use Mongoose to create schemas for your data models: `npm install mongoose --save`

- Create controllers: Define the processing logic for each route in controller files.
- Create routes and configure APIs for Express: Define API endpoints in route files and link them with corresponding controllers.
- Create middleware for authentication: Use packages like passport or jsonwebtoken to build user authentication middleware.
- Run the project: Use the node or nodemon command to start the server:
  - node app.js or nodemon app.js

**Conclusion:**

Using mongodb and nodejs, expressjs to build server APIs is simple structure and write APIs quickly through the following steps: create a data model, create a controller to process the APIs, create a route to route the APIs to the controllers and configure them in the main.js

**Frontend: HTML, CSS, JS:**

Frontend uses basic technologies such as: html, css, and js to build user interfaces and interactions vs. backend APIs to complete functions. Steps to build a frontend project:

- Frontend project folder structure:

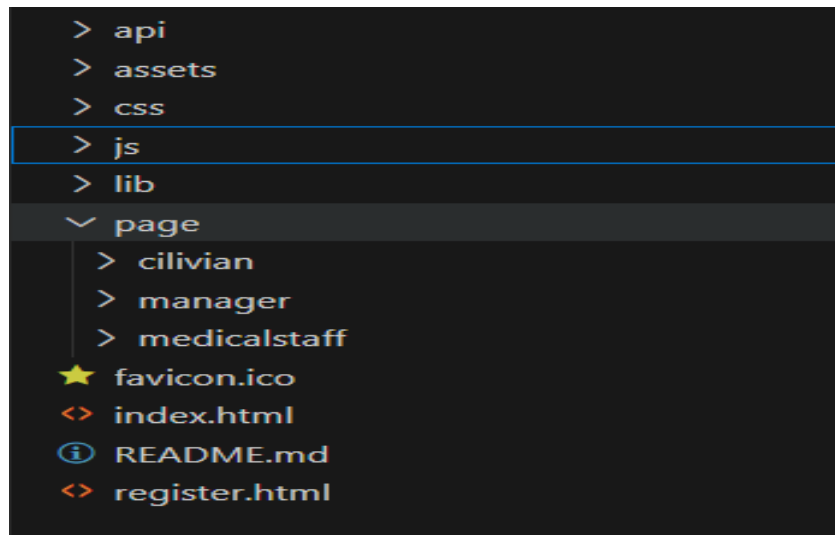


Figure 4.9 Frontend folder structure

In which, the page folder contains the html files of the pages, the js folder contains javascript files that handle interactions and operations, the css folder contains the css files to create styles for the website, the api folder contains the js files that use fetch to call the APIs from the backend, Finally, the asset folder contains the fonts and images needed for the interface.

## **Flows deploy backend to aptible cloud**

Using the Deploy Code tool in the Aptible Dashboard, you can deploy the Express Template. The tool will guide you through the following: [7]

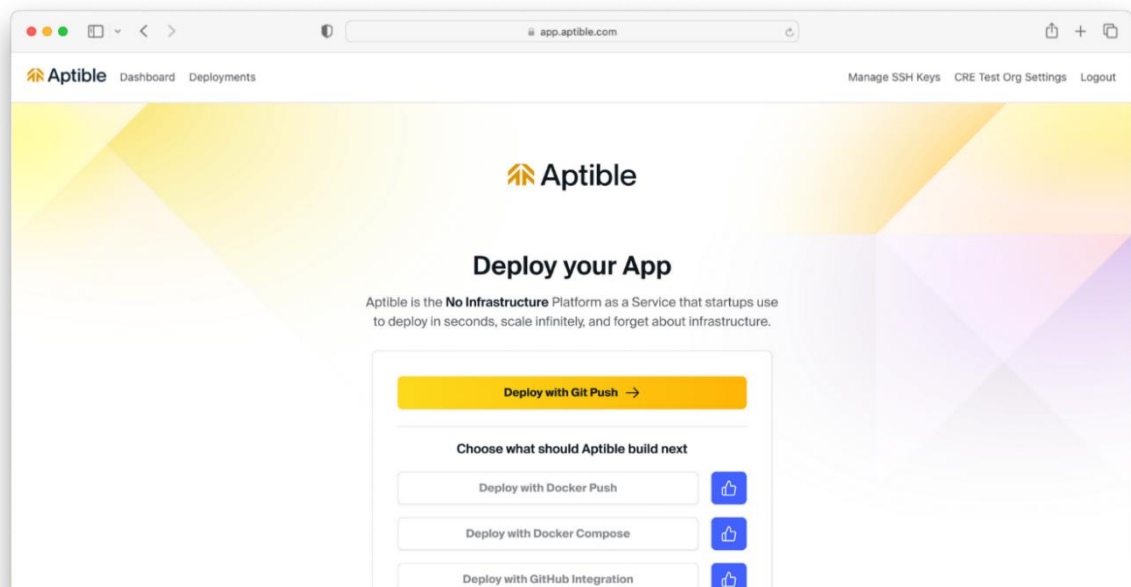


Figure 4.10 Deploy to Aptible with git push

Step 01: Deploy with Git Push.

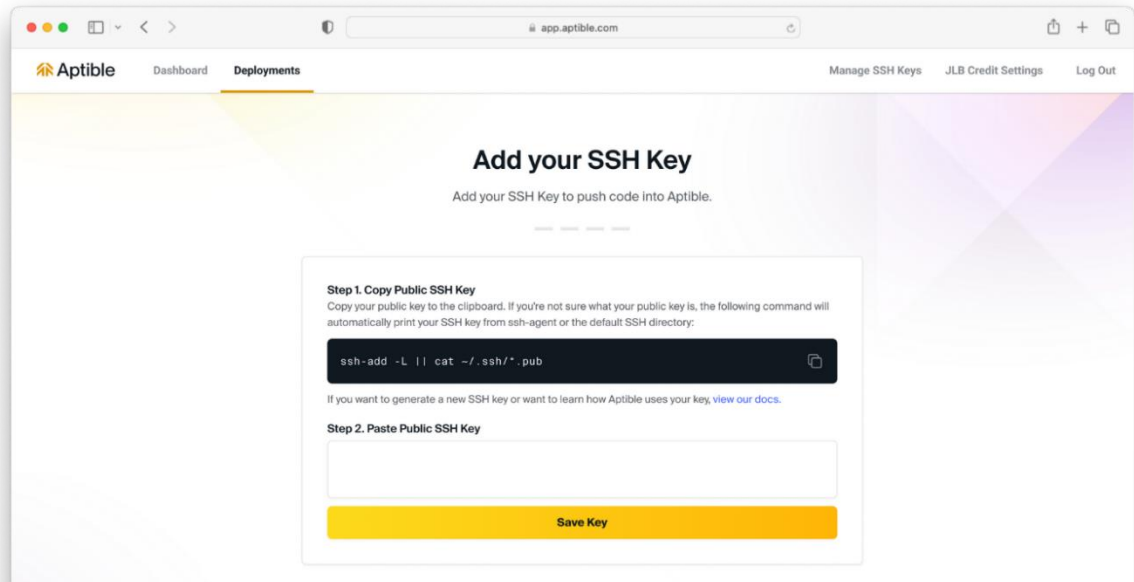


Figure 4.11 Deploy to Aptible – add ssh key

Step 02: Add an SSH key.

Step 03: Environment Setup.

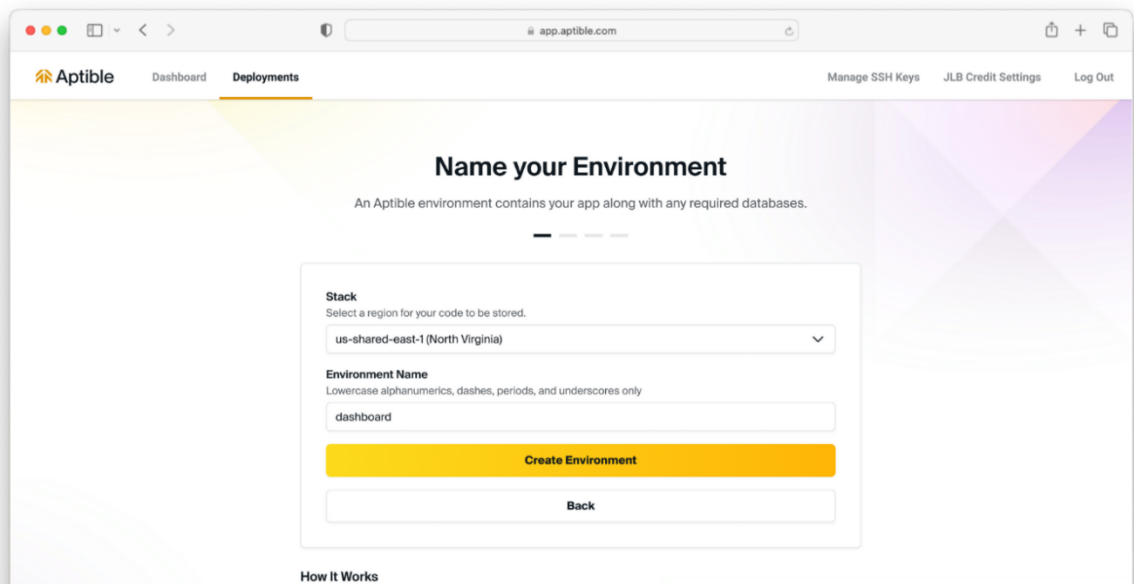


Figure 4.12 Deploy to Aptible - environment

Select stack to deploy resources. This will determine what region resources are deployed to. Then, name the environment resources will be grouped into.

## Step 04: Prepare the template

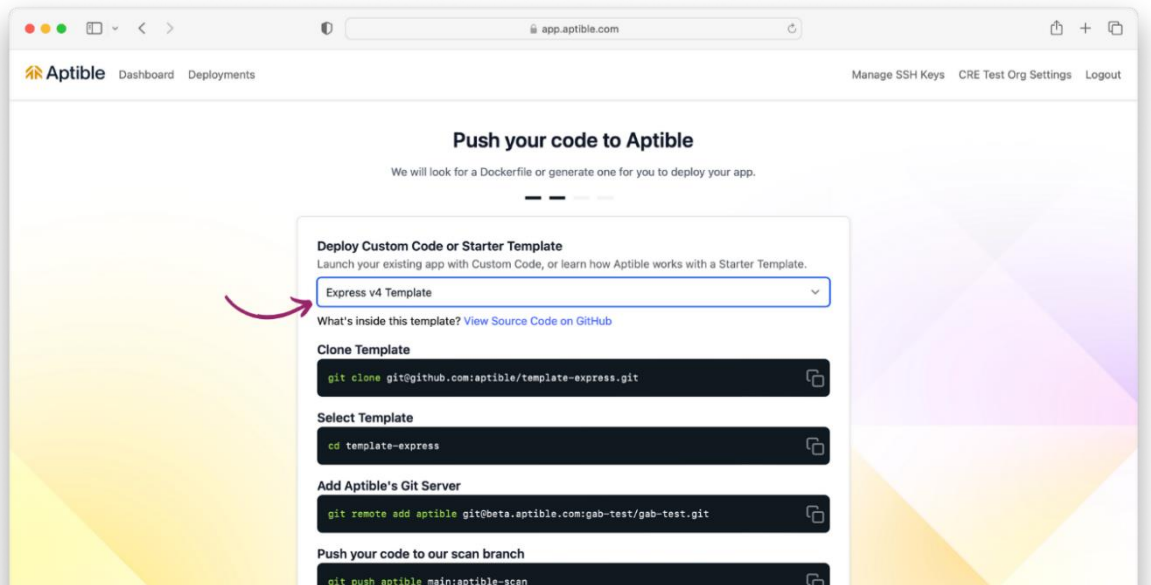


Figure 4.13 Deploy to Aptible – prepare the template

Select Express Template for deployment and follow command-line instructions.

## Step 05: Fill environment variables and deploy

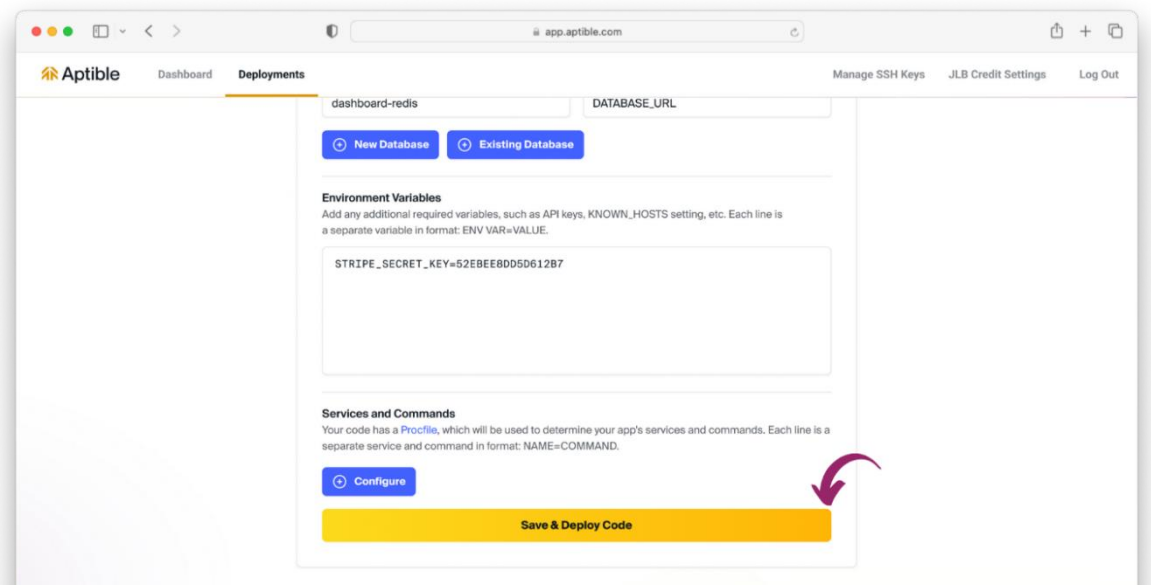


Figure 4.14 Deploy to Aptible - environment variables

Aptible will automatically fill this template's required databases, services, and app's configuration with environment variable keys to fill with values. Once complete, save and deploy the code.

### Step 06: View logs in real time

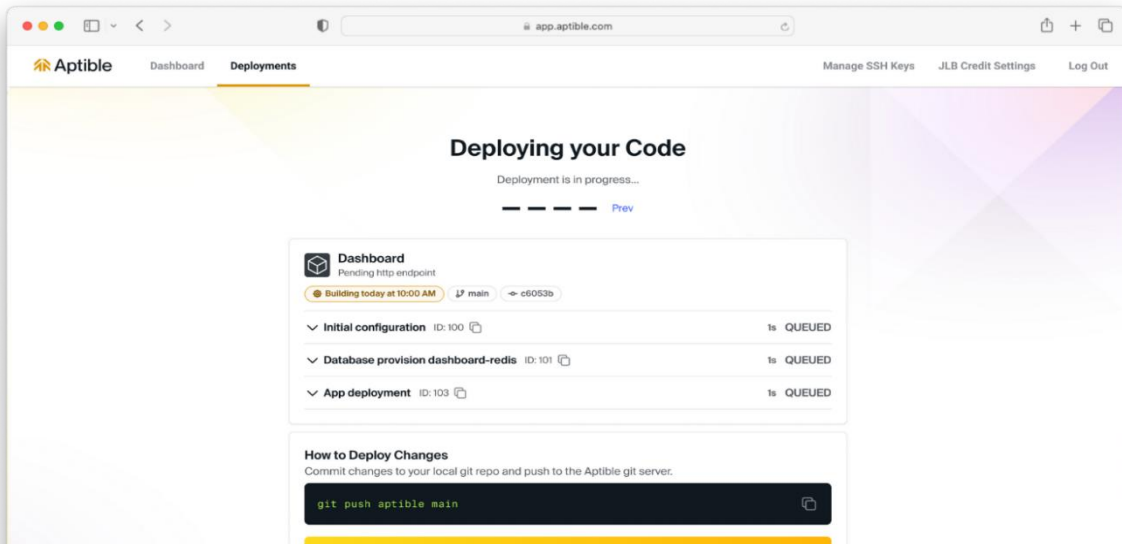


Figure 4.15 Deploy to Aptible - view logs in real time

### Step 07: Expose app to the internet

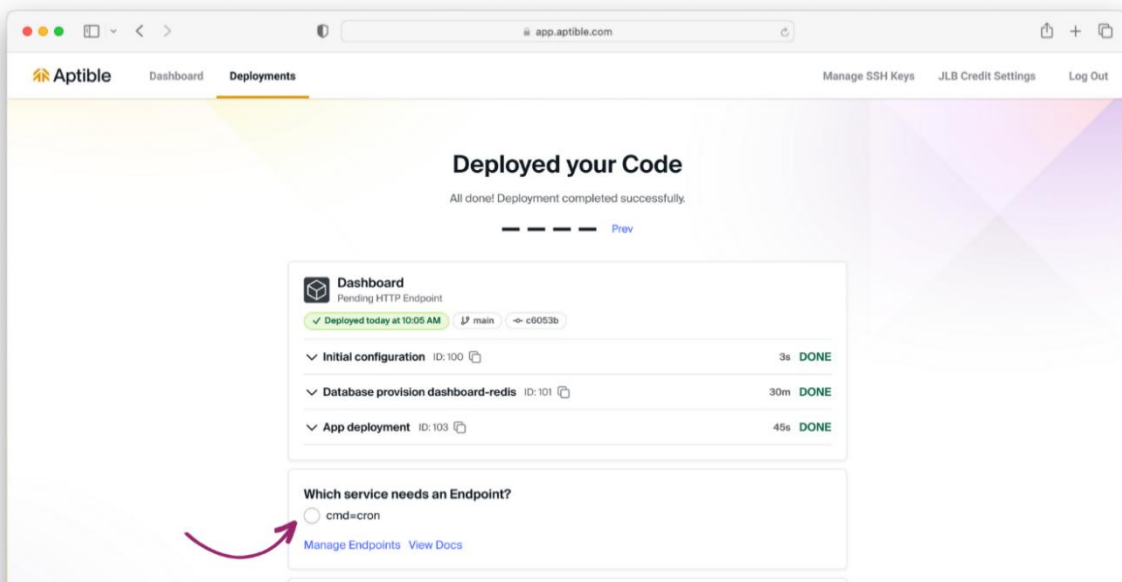


Figure 4.16 Deploy to Aptible - expose app

Now that code is deployed, it is time to expose the app to the internet. Select the service that needs an endpoint, and Aptible will automatically provision a managed endpoint.

#### Step 08: View deployed app

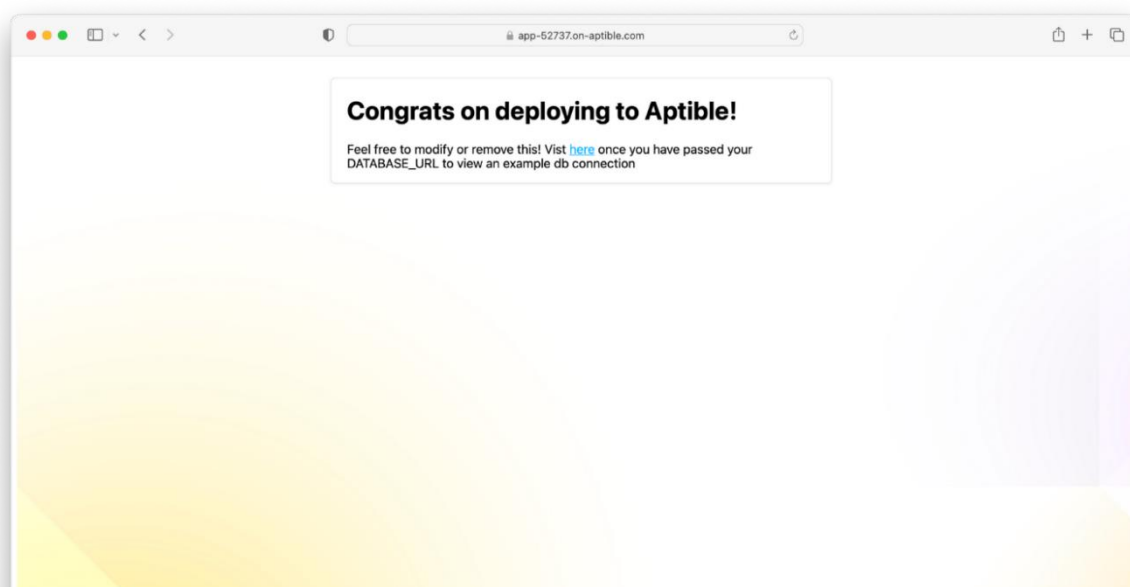


Figure 4.17 Deploy to Aptible successfully

So, we have successfully deploy backend with Aptible at url <https://app-74434.on-aptible.com/>

## 4.1.5 Built-in APIs

Have built a website system for manager, medical staff and civilian, 1 server provides business APIs:

POST	/api/signup	Đăng ký tài khoản		{ * phoneNumber: so dien thoai * password: mat khau * type: chon chuc vu: 0- người dân, 1- nhân viên y tế, 2- admin * unitCode: mã unicode của đơn vị * }	{ * message: "Signup successful", * userid: result._id, * }
POST	/api/login	Đăng nhập tài khoản		{ * phoneNumber: so dien thoai * password: mat khau * } body	{ * token, * type, * unitCode, * UnitDetail * userid: loadedUser._id.toString(), * }
POST	/api/Person	Thêm thông tin cá nhân cho người dân		{ * fullName: Họ và tên * dateOfBirth: Ngày sinh * Sex: Giới tính * addressCode: Mã địa chỉ cư trú * address: Địa chỉ mức nhỏ hơn phường * nationality: Quốc tịch * diseaseStatus: Trạng thái (F0, F1, F2) không bị gì thì không cần cập nhật * }	{ * message: "Successful registration information", * link: process.env.SERVER_URL + "/personinfo/getperson", * }
PUT	/api/editPerson	Cập nhật thông tin cá nhân cho người dân		{ * fullName: Họ và tên * dateOfBirth: Ngày sinh * Sex: Giới tính * addressCode: Mã địa chỉ cư trú * address: Địa chỉ mức nhỏ hơn phường * nationality: Quốc tịch * diseaseStatus: Trạng thái (F0, F1, F2) không bị gì thì không cần cập nhật * }	{ * message: "Update information successful", * link: process.env.SERVER_URL + "/personinfo/getperson", * }
GET	/api/getPerson	Lấy thông tin cho người dân			{ message: "fetching user info successfully", data: { - userid, phoneNumber, fullName, dateOfBirth, Sex, addressCode, address, nationality, diseaseStatus, }, }, }
DELETE	/api/deletePerson	Xóa thông tin người dân			{ message: "Delete information successful" }
GET	/api/getListMedical	Lấy về danh sách tài khoản nhân viên y tế cần duyệt	{ token : "String" }		{ message: "Successfully", data: [ { userid, phoneNumber, unitCode, unitDetail, },... ], }
GET	/api/getListAdmin	Lấy về danh sách tài khoản quản lý cần duyệt	{ token : "String" }		{ message: "Successfully", data: [ { userid, phoneNumber, unitCode, unitDetail, },... ], }
PUT	/api/accountbrowsing	Duyệt tài khoản	{ token : "String" }		{ message: "Successfully", }

Figure 4.18 Build in APIs (1)



POST	/api/admin	Thêm thông tin cá nhân cho admin		{ <ul style="list-style-type: none"> <li>* fullName: Họ và tên /String</li> <li>* dateOfBirth: Ngày sinh /Date</li> <li>* Sex: Giới tính /String</li> <li>* addressCode: Mã địa chỉ cư trú /String</li> <li>* address: Địa chỉ mức nhỏ hơn phường /String</li> <li>* nationality: Quốc tịch /String</li> <li>* diseaseStatus: Trạng thái (F0, F1, F2) không bị gì thì không cần cập nhật</li> </ul> /String * }	{ <ul style="list-style-type: none"> <li>* message: "Successful registration information",</li> <li>* link: process.env.SERVER_URL + "/admininfo/getadmin",</li> </ul> * }
PUT	/api/editAdmin	Cập nhật thông tin cá nhân cho admin		{ <ul style="list-style-type: none"> <li>* fullName: Họ và tên</li> <li>* dateOfBirth: Ngày sinh</li> <li>* Sex: Giới tính</li> <li>* addressCode: Mã địa chỉ cư trú</li> <li>* address: Địa chỉ mức nhỏ hơn phường</li> <li>* nationality: Quốc tịch</li> <li>* diseaseStatus: Trạng thái (F0, F1, F2) không bị gì thì không cần cập nhật</li> </ul> * }	{ <ul style="list-style-type: none"> <li>* message: "Update information successful",</li> <li>* link: process.env.SERVER_URL + "/admininfo/getadmin",</li> </ul> * }
GET	/api/getAdmin	Lấy thông tin cho admin			{ <ul style="list-style-type: none"> <li>message: "fetching user info successfully",</li> <li>data: { <ul style="list-style-type: none"> <li>userId,</li> <li>phoneNumber,</li> <li>fullName,</li> <li>dateOfBirth,</li> <li>Sex,</li> <li>addressCode,</li> <li>address,</li> <li>nationality,</li> <li>diseaseStatus,</li> </ul> }, </li> </ul> }
DELETE	/api/deletePerson	Xóa thông tin admin			{ <ul style="list-style-type: none"> <li>message: "Delete information successful"</li> </ul> }
GET	/api/getMedical	Lấy thông tin cho nhân viên y tế			{ <ul style="list-style-type: none"> <li>message: "fetching user info successfully",</li> <li>data: { <ul style="list-style-type: none"> <li>userId,</li> <li>phoneNumber,</li> <li>fullName,</li> <li>dateOfBirth,</li> <li>Sex,</li> <li>addressCode,</li> <li>address,</li> <li>nationality,</li> <li>diseaseStatus,</li> </ul> }, </li> </ul> }
DELETE	/api/deleteMedical	Xóa thông tin người dân			{ <ul style="list-style-type: none"> <li>message: "Delete information successful"</li> </ul> }
POST	/api/Medical	Thêm thông tin cá nhân cho nhân viên y tế		{ <ul style="list-style-type: none"> <li>* fullName: Họ và tên</li> <li>* dateOfBirth: Ngày sinh</li> <li>* Sex: Giới tính</li> <li>* addressCode: Mã địa chỉ cư trú</li> <li>* address: Địa chỉ mức nhỏ hơn phường</li> <li>* nationality: Quốc tịch</li> <li>* diseaseStatus: Trạng thái (F0, F1, F2) không bị gì thì không cần cập nhật</li> </ul> * }	{ <ul style="list-style-type: none"> <li>* message: "Successful registration information",</li> <li>* link: process.env.SERVER_URL + "/personinfo/getperson",</li> </ul> * }
PUT	/api/editMedical	Cập nhật thông tin cá nhân cho nhân viên y tế		{ <ul style="list-style-type: none"> <li>* fullName: Họ và tên</li> <li>* dateOfBirth: Ngày sinh</li> <li>* Sex: Giới tính</li> <li>* addressCode: Mã địa chỉ cư trú</li> <li>* address: Địa chỉ mức nhỏ hơn phường</li> <li>* nationality: Quốc tịch</li> <li>* diseaseStatus: Trạng thái (F0, F1, F2) không bị gì thì không cần cập nhật</li> </ul> * }	{ <ul style="list-style-type: none"> <li>* message: "Update information successful",</li> <li>* link: process.env.SERVER_URL + "/personinfo/getperson",</li> </ul> * }

Figure 4.19 Build in APIs (2)

POST	/api/adddomesticguests	Tạo 1 bản khai y tế toàn dân	{ token : "String" }	{ <ul style="list-style-type: none"> <li>* ismovingThroughTerritory: Có đi qua vùng bệnh không /Boolean</li> <li>* nCoVSignal: Có dấu hiệu mắc Covid không /Boolean</li> <li>* patientContact: Có tiếp xúc với người bệnh hoặc nghi ngờ không /Boolean</li> <li>* nCoVConPCountry: Có tiếp xúc với người từ nước có Covid không /Boolean</li> <li>* nCoVConPSignal: Có tiếp xúc với người có dấu hiệu mắc nCoV /Boolean</li> </ul> }	{ <ul style="list-style-type: none"> <li>* message: "Update a domestic guests successfully",</li> <li>* link: process.env.SERVER_URL + "/domesticguests/getdomesticguests"</li> </ul> }
GET	/api/getdomesticquest	Lấy bản khai y tế	{ token : "String" }		{ <ul style="list-style-type: none"> <li>message: "Get infomation domestic guests successful",</li> <li>data: { <ul style="list-style-type: none"> <li>userId,</li> <li>ismovingThroughTerritory,</li> <li>nCoVSignal,</li> <li>patientContact</li> <li>nCoVConPCountry</li> <li>nCoVConPSignal</li> <li>declarationDate,</li> </ul> </li> </ul> }
DELETE	/api/deletedomesticquest	Xóa bản khai y tế toàn dân	{ token : "String" }		{ <ul style="list-style-type: none"> <li>message: "Delete infomation domestic guests successful",</li> </ul> }
GET	/api/getentrydeclaration	Lấy bản khai nhập cảnh	{ token : "String" }		{ <ul style="list-style-type: none"> <li>* message: "Get infomation entry declaration successful",</li> <li>*data: { <ul style="list-style-type: none"> <li>userId,</li> <li>object,</li> <li>gate,</li> <li>vehicle,</li> <li>vehicleNumber,</li> <li>chairNumber,</li> <li>departureDay,</li> <li>entryDate,</li> <li>departureCountry,</li> <li>departureCity,</li> <li>destinationCountry,</li> <li>passingCountry,</li> <li>addressAfterQuarantine,</li> <li>fever,</li> <li>cough,</li> <li>stuffy,</li> <li>soreThroat,</li> <li>nausea,</li> <li>diarrhea,</li> <li>hemorrhage,</li> <li>rash,</li> <li>vaccinesUsed,</li> <li>animalContact,</li> <li>nCoVPContact,</li> <li>isolationFacility,</li> </ul> </li> </ul> }
DELETE	/api/deleteentrydeclaration	Xóa bản khai nhập cảnh	{ token : "String" }		{ <ul style="list-style-type: none"> <li>* message: "Delete infomation entry declaration successful",</li> </ul> }
POST	/api/addmovedeclaration	Tạo 1 bản khai y tế di chuyển nội địa	{ token : "String" }	{ <ul style="list-style-type: none"> <li>* vehicle: Phương tiện /String</li> <li>* vehicleNumber: Số hiệu phương tiện /String</li> <li>* chairNumber: Số ghế /Number</li> <li>* departureDay: Ngày khởi hành /Date</li> <li>* departureAddress: Địa chỉ xuất phát /String</li> <li>* arrivalAddress: Địa chỉ đến /String</li> <li>* ismovingThroughTerritory: Có di chuyển qua quốc gia lãnh thổ nào /Boolean</li> <li>* nCoVSignal: Có dấu hiệu mắc nCoV /Boolean</li> <li>* patientContact: Có tiếp xúc với người bệnh hoặc nghi ngờ /Boolean</li> <li>* nCoVConPCountry: Có tiếp xúc với người từ nước có nCoV /Boolean</li> <li>* nCoVConPSignal: Có tiếp xúc với người có dấu hiệu mắc nCoV /Boolean</li> </ul> }	{ <ul style="list-style-type: none"> <li>* message: "Update a move delclaration successfully",</li> <li>* link: process.env.SERVER_URL + "/movedeclaration/getmovedeclaration"</li> </ul> }
GET	/api/getmovedeclaration	Lấy bản khai y tế di chuyển nội địa	{ token : "String" }		{ <ul style="list-style-type: none"> <li>* message: "Get infomation entry declaration successful",</li> <li>*data: { <ul style="list-style-type: none"> <li>userId,</li> <li>vehicle,</li> <li>vehicleNumber,</li> <li>chairNumber,</li> <li>departureDay,</li> <li>departureAddress,</li> <li>arrivalAddress,</li> <li>ismovingThroughTerritory,</li> <li>nCoVSignal,</li> <li>patientContact,</li> <li>nCoVConPCountry,</li> <li>nCoVConPSignal,</li> <li>declarationDate,</li> </ul> </li> </ul> }

Figure 4.20 Build in APIs (3)

POST	/api/addentrydeclaration	Tạo 1 bản khai y tế nhập cảnh	{ token : "String" }	<pre> {   * object: Đối tượng (nước ngoài hay Vn, ...) /String   * gate: Cửa khẩu /String   * vehicle: Phương tiện /String   * vehicleNumber: Số hiệu phương tiện /String   * chairNumber: Số ghế /Number   * departureDay: Ngày khởi hành /Date   * entryDate: Ngày nhập cảnh /Date   * departureCountry: Quốc gia khởi hành /String   * departureCity: Thành phố khởi hành /String   * destinationCountry: Quốc gia cần đến /String   * passingCountry: Đã đi qua quốc gia nào /String   * addressAfterQuarantine: Địa chỉ lưu trú sau cách ly tập trung /String   * fever: Ho /Boolean   * cough: Sốt /Boolean   * stuffy: Khó thở /Boolean   * soreThroat: Đau họng /Boolean   * nausea: nôn /Boolean   * diarrhea: Tiêu chảy /Boolean   * hemorrhage: Xuất huyết /Boolean   * rash: Nổi ban /Boolean   * vaccinesUsed: Loại Vacxin đã sử dụng /String   * animalContact: Có tiếp xúc với động vật hay cơ sở giết mổ /Boolean   * nCoVPCContact: Có tiếp xúc với người mắc nCoV /Boolean   * isolationFacility: Cơ sở cách ly /String   * negativeConfirmation: xác nhận âm tính   * } </pre>	<pre> {   * message: "Update a domestic entry declaration successfully",   * link: process.env.SERVER_URL + "/entrydeclaration/getmovedeclaration"   * } </pre>
DELETE	/api/deletemovedeclaration	Xóa bản khai y tế di chuyển nội địa	{ token : "String" }		<pre> {   * message: "Delete information move declaration successful",   * } </pre>

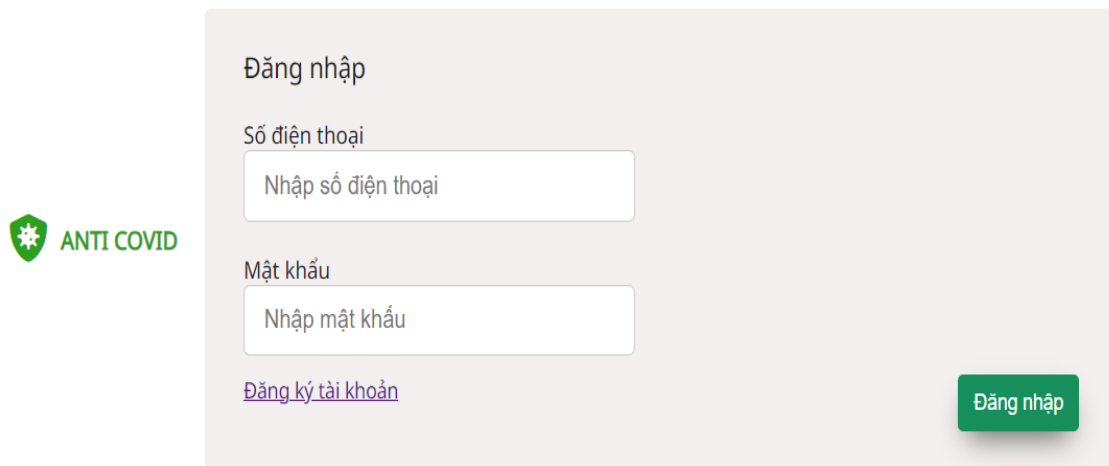
Figure 4.21 Build in APIs (4)

Build storage for media: photos, etc. using firebase storage.

## 4.2 System interface design

Deploy with Netlify at url <https://pandemic-management.netlify.app/>

### Login



Đăng nhập

Số điện thoại

Nhập số điện thoại

Mật khẩu

Nhập mật khẩu

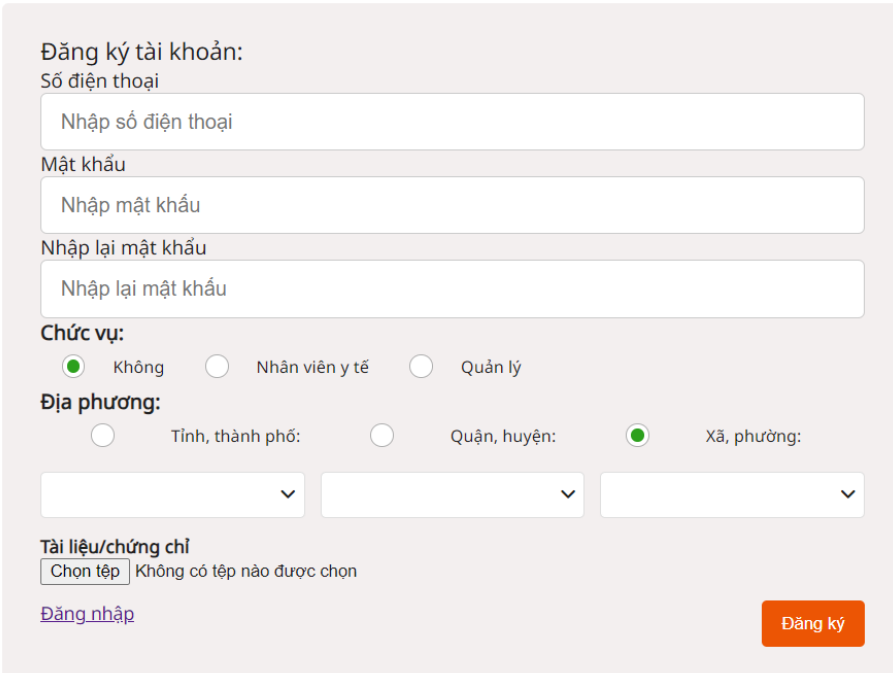
[Đăng ký tài khoản](#)

Đăng nhập

Figure 4.22 Login interface

The login home page is the first page visitors see when accessing the epidemiological management website. This is where users use their existing account to log in to use the system or, if they do not have an account, they need to register to use it.

## Register



**Đăng ký tài khoản:**

Số điện thoại

Mật khẩu

Nhập lại mật khẩu

**Chức vụ:**

Không  Nhân viên y tế  Quản lý

**Địa phương:**

Tỉnh, thành phố:  Quận, huyện:  Xã, phường:

**Tài liệu/chứng chỉ**

Không có tệp nào được chọn

[Đăng nhập](#)

Figure 4.23 Register interface

Users can register user accounts for three types of users according to usage needs: Civilian, Medical Staff, Administrators.

- **Civilian:** For this user, you need to use a phone number and password to register. Click to select the "None" function button for Civilian users. Select specific address information in order from Province, City - District - Commune, Ward. The request will be automatically approved by the system and can be used immediately.
- **Medical Staff:** For this user, you need to use a phone number and password to register. Click to select the "Medical Staff" function button for Medical Staff users. Select specific working units by Province, City - District - Commune, Ward. Upload relevant documents and certificates used to

authenticate your identity. After clicking register, the system will automatically record the results and need to wait for approval from the system administrator before you can use it.

- Administrators: For this user, you need to use a phone number and password to register. Click to select the " Administrator" function button for Administrators users. Select specific working units by Province, City - District - Commune, Ward. Upload relevant documents and certificates used to authenticate your identity. After pressing register, the system will automatically record the results and need to wait for approval from a system administrator with higher authority before you can use it.

## Account approval

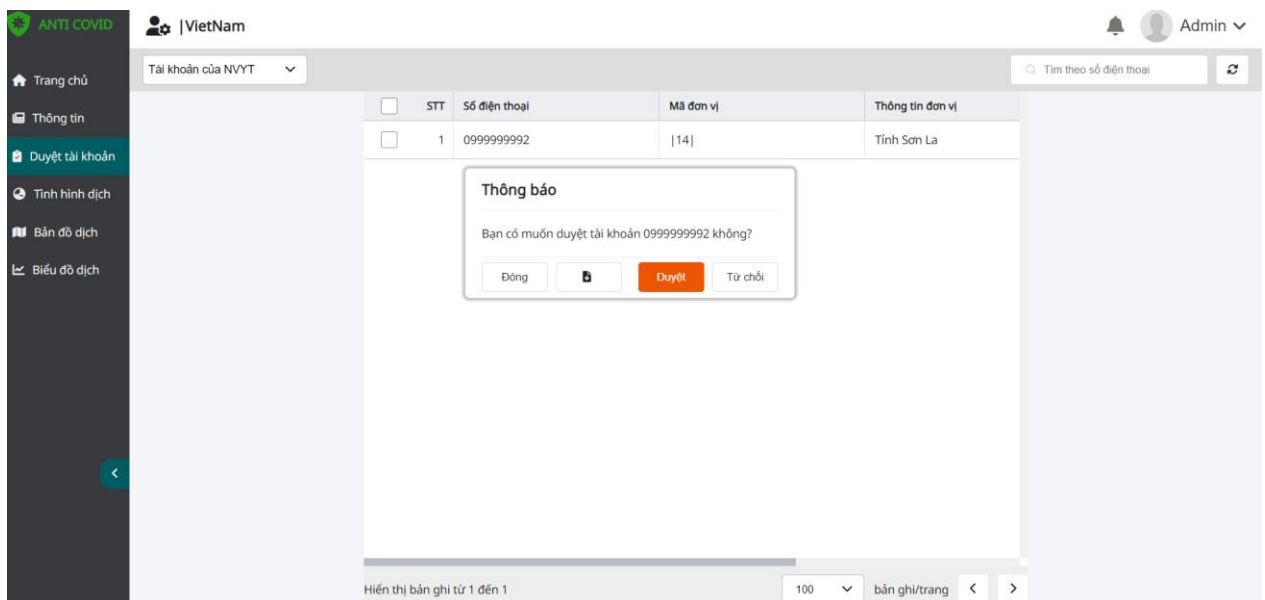


Figure 4.24 Account approval interface

Use the Administrator account to track account approval requests from medical staff and administrators.

Double-click on the user you want to approve. Select close to cancel the operation, select the document icon to check the certificate, select approve to approve the account, select reject to reject the request.

## Personal information

The screenshot shows the 'Thông tin cá nhân' (Personal Information) form. The form is titled 'Thông tin cá nhân' and has a close button (X) in the top right corner. The form fields are as follows:

- Họ và tên: [Text input field]
- Ngày sinh: [Date picker (dd/mm/yyyy)]
- Giới tính:  Nam  Nữ  Không
- Quốc tịch: [Text input field]
- Thông tin dịch tễ (F0, F1, F2):  F0  F1  F2  Không
- Tỉnh/Thành phố: [Dropdown menu]
- Quận/Huyện: [Dropdown menu]
- Phường/xã: [Dropdown menu]
- Địa chỉ cụ thể: [Text area]

At the bottom of the form, there is a blue button with a hand icon. At the bottom right of the modal, there is an orange button labeled 'Cập nhật thông tin'.

Figure 4.25 Personal information interface

Users need to fill in personal information including Full name, date of birth, gender, nationality, click on epidemiological information F0, F1, F2, None. Fill in address information and note the specific address. Then press the "Update information" button for the system to automatically record.

Filling in personal information completely and accurately is very important. It makes statistical work and information verification easy, saving time for medical staff and senior managers.

## Announcement/Post

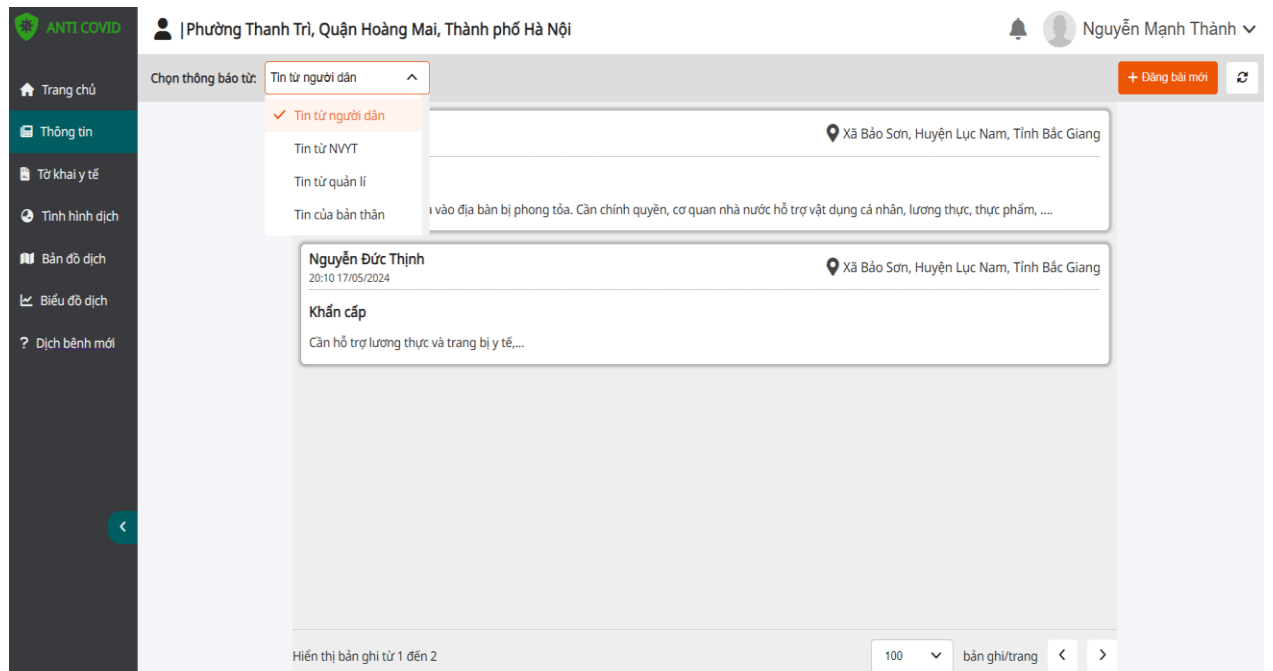


Figure 4.26 Announcement/Post interface

Displays announcements and posts from medical staff, managers and residents. Allows you to review your posts. An announcement, post includes a title and content.

After the system records post requests from Civilian users, Medical Staff users, and Administrators users, the administrator account needs to approve them before the users can view them.



## Approve announcement/post

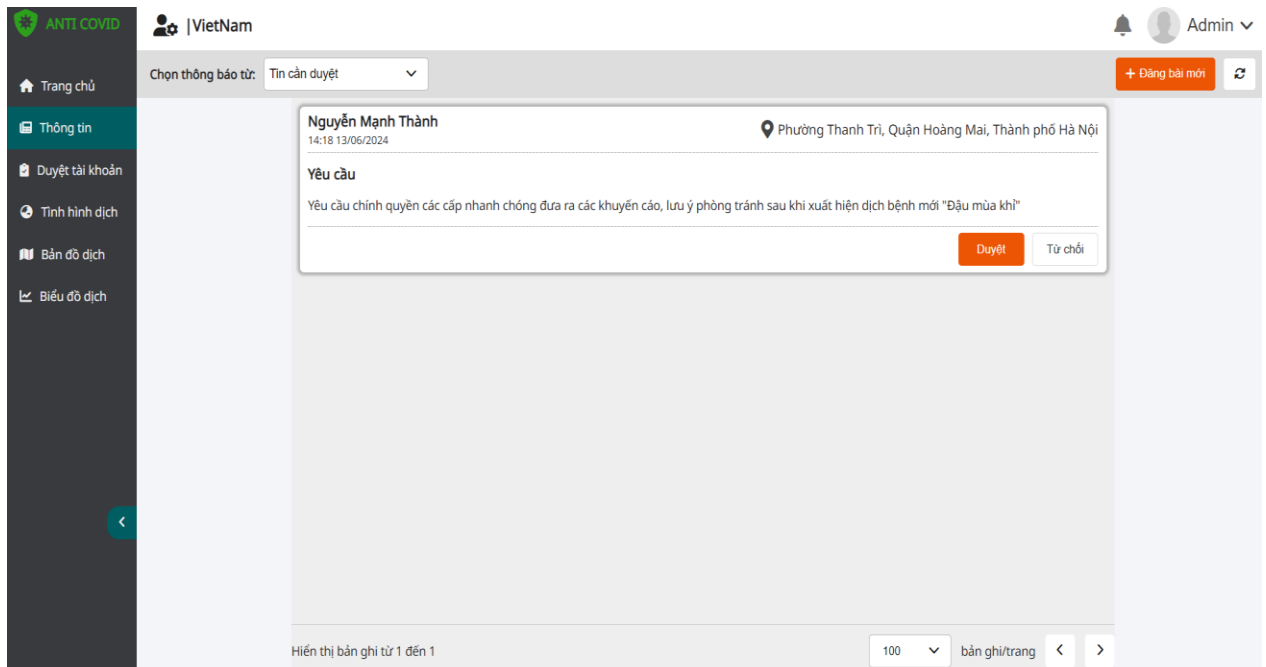


Figure 4.27 Approve announcement/post interface

Use the Administrator account to approve announcements/post from Civilian users, Medical Staff users and Administrators users.

Check the information and authenticity of the announcement/post before clicking approve to agree to post the announcement/post or click decline to refuse to approve the announcement/post.

## General declaration

ANTICOVID | Phường Thanh Trì, Quận Hoàng Mai, Thành phố Hà Nội

Nguyễn Mạnh Thành

Chọn loại khai báo: Khai báo toàn dân

### Khai báo toàn dân

Có đi qua vùng bệnh không:  
 Không  Có

Có dấu hiệu mắc Covid không:  
 Không  Có

Có tiếp xúc với người bệnh hoặc nghi ngờ không:  
 Không  Có

Có tiếp xúc với người từ nước có Covid không:  
 Không  Có

Có tiếp xúc với người có dấu hiệu mắc nCoV:  
 Không  Có

**TIME IS NOW**

Khai báo

Figure 4.28 General declaration interface

Use the “yes” or “no” quick choice form. The quick declaration list helps medical staff quickly compile data on individuals at risk of infection.

## General declaration list

ANTICOVID | Thành phố Hà Nội

undefined

Khai báo toàn dân

Tim theo số điện thoại

STT	Họ và tên	Ngày sinh	Số điện thoại	Quốc tịch	Thuộc đơn vị quản lý
1	Nguyễn Mạnh Thành	01/02/2001	9876543210	Việt Nam	Phường Thanh Trì, Quận Hoàng

ANTICOVID | Thành phố Hà Nội

undefined

Khai báo toàn dân

Tim theo số điện thoại

Có đi qua vùng bệnh	Có đi dấu hiệu mắc covid	Có tiếp xúc với người bệnh, nghi ngờ	Có tiếp xúc với người từ nước có Covid không	Có tiếp xúc với người có dấu hiệu mắc nCoV
●	■	●	●	■

Figure 4.29 General declaration list interface

Medical staff compile statistics on users at risk of epidemic and contact them according to the information on the form.

After the system records the results of people's declarations, medical staff can check the declarations within the scope of management. Accurate statistics and data help update epidemic information in the region.

## Entry declaration

The screenshot shows the 'Khai báo nhập cảnh' (Entry Declaration) form. The form is titled 'Khai báo nhập cảnh' and contains the following fields:

- Đối tượng: (Dropdown menu)
- Cửa khẩu: (Text input)
- Số hiệu phương tiện: (Text input)
- Số ghế: (Text input)
- Ngày khởi hành: (Date input, format: dd/mm/yyyy)
- Ngày nhập cảnh: (Date input, format: dd/mm/yyyy)
- Quốc gia khởi hành: (Text input)
- Thành phố khởi hành: (Text input)
- Quốc gia cần đến: (Text input)
- Đã đi qua quốc gia nào: (Text input)
- Địa chỉ lưu trú sau cách ly tập trung: (Text input)
- Họ: (Radio buttons: Không, Có)
- Sốt: (Radio buttons: Không, Có)
- Khó thở: (Radio buttons: Không, Có)
- Đau họng: (Radio buttons: Không, Có)
- Nôn: (Radio buttons: Không, Có)
- Tiêu chảy: (Radio buttons: Không, Có)
- Xuất huyết: (Radio buttons: Không, Có)
- Nổi ban: (Radio buttons: Không, Có)
- Có tiếp xúc với động vật hay cơ sở giết mổ: (Radio buttons: Không, Có)
- Có tiếp xúc với người mắc nCoV: (Radio buttons: Không, Có)
- Loại Vaccin đã sử dụng: (Text input)
- Xác nhận âm tính: (Radio buttons: Không, Có)
- Cơ sở cách ly: (Text input)

The form is displayed in a modal window over a dashboard background. The dashboard includes a sidebar with navigation options: Trang chủ, Thông tin, Tô khai y tế, Tình hình dịch, Bản đồ dịch, Biểu đồ dịch, and Dịch bệnh mới. The top right corner shows the user's name: Nguyễn Mạnh Thành.

Figure 4.30 Entry declaration interface

Civilian users declare entry information according to the form available on the form.

## Entry declaration list

The screenshot shows the 'Entry declaration list' interface. The interface includes a sidebar with navigation options: Trang chủ, Thông tin, Khai báo y tế, Tình hình dịch, Bản đồ dịch, Biểu đồ dịch, and Dịch bệnh mới. The top right corner shows the user's name: ADM Hà Nội. The main content area displays a table with the following columns:

trở sau cách ly tập trung	Loại vaccin đã sử dụng	Cơ sở cách ly	Họ	Sốt	Khó thở	Đau họng	Nôn	Tiêu chảy	Xuất huyết	Nổi ban	Có tiếp xúc với động vật hay cơ sở giết mổ	Có tiếp xúc với người mắc nCoV	Xác nhận âm tính

The table is currently empty, and the interface includes a search bar and a sidebar with navigation options.

Figure 4.31 Entry declaration list interface

Similar to the declaration above.

## Move declaration

Phường Thanh Trì, Quận Hoàng Mai, Thành phố Hà Nội

Chọn loại khai báo: Khai báo di chuyển

### Khai báo di chuyển

Phương tiện: Hyundai Santafe Số hiệu phương tiện: 99A44803

Số ghế: 7 Ngày khởi hành: 18/06/2024

Địa chỉ xuất phát: Hà Nội Địa chỉ đến: Quảng Ninh

Có đi chuyển qua quốc gia lãnh thổ nào:  
 Không  Có

Có dấu hiệu mắc nCoV:  
 Không  Có

Có tiếp xúc với người bệnh hoặc nghi ngờ:  
 Không  Có

Có tiếp xúc với người từ nước có nCoV:  
 Không  Có

Có tiếp xúc với người có dấu hiệu mắc nCoV:  
 Không  Có

Khai báo

Figure 4.32 Move declaration interface

Similar to the declaration above.

## Move declaration list

Khai báo di chuyển

Tìm theo số điện thoại

STT	Họ và tên	Ngày sinh	Số điện thoại	Quốc tịch	Thuộc đơn vị quản lý	Phương tiện	Số hiệu phụ
1	Nguyễn Mạnh Thành	01/02/2001	9876543210	Việt Nam	Phường Thanh Trì, Quận Hoàng Mai, Thành phố Hà Nội	Hyundai Santafe	99A44803

Tìm theo số điện thoại

Địa chỉ xuất phát	Địa chỉ đến	Có đi chuyển qua quốc gia lãnh thổ nào	Có dấu hiệu mắc covid	Có tiếp xúc với người bệnh/nghi ngờ	Có tiếp xúc với người từ nước có Covid không	Có tiếp xúc với người có dấu hiệu mắc nCoV
Hà Nội	Quảng Ninh	0	✓	0	0	0

Figure 4.33 Move declaration list interface

Similar to the declaration above.

## Situation reports pandemic

STT	Tên đơn vị	Mã đơn vị	Mức báo động	Tổng số ca nhiễm	Tổng số ca tử vong	Tổng số ca phi
1	Thành phố Hà Nội	[1]	3	141	63	
2	Tỉnh Hà Giang	[2]	0	0	0	
3	Tỉnh Sơn La	[14]	0	0	0	
4	Tỉnh Hoà Bình	[17]	0	0	0	
5	Tỉnh Hải Dương	[30]	0	0	0	
6	Tỉnh Thái Bình	[34]	0	0	0	
7	Tỉnh Nam Định	[36]	3	90	22	
8	Tỉnh Ninh Bình	[37]	0	0	0	
9	Tỉnh Hà Tĩnh	[42]	1	5	0	
10	Tỉnh Quảng Nam	[49]	0	0	0	
11	Tỉnh Khánh Hòa	[56]	0	0	0	

Figure 4.34 Situation reports pandemic interface (1)

The city unit displays the epidemic situation. Including unit code, alert level, total number of cases, total number of deaths, total number of recoveries, number of new cases, number of new deaths, number of new recoveries.

STT	Tên đơn vị	Mã đơn vị	Mức báo động	Tổng số ca nhiễm	Tổng số ca tử vong	Tổng số ca phi
1	Thành phố Hà Nội	[1]	3	141	63	
2	Tỉnh Hà Giang	[2]	0	0	0	
3	Tỉnh Sơn La	[14]	0	0	0	
4	Tỉnh Hoà Bình	[17]	0	0	0	
5	Tỉnh Hải Dương	[30]	0	0	0	
6	Tỉnh Thái Bình	[34]	0	0	0	
7	Tỉnh Nam Định	[36]	3	90	22	
8	Tỉnh Ninh Bình	[37]	0	0	0	
9	Tỉnh Hà Tĩnh	[42]	1	5	0	
10	Tỉnh Quảng Nam	[49]	0	0	0	
11	Tỉnh Khánh Hòa	[56]	0	0	0	

Figure 4.35 Situation reports pandemic interface (2)

Double-click on the province/city unit to select to view district-level Pandemic information. Similarly, double-click on the district-level unit to see the epidemic in the commune/ward.

## Update situation pandemic

The screenshot displays the ANTI COVID web application interface. At the top, the user is logged in as 'Xã Yên Viên, Huyện Gia Lâm, Thành phố Hà Nội'. The main content area shows a table titled 'Trong nước' (In the country) with columns for 'STT' (Serial Number), 'Tên đơn vị' (Unit Name), 'Mã đơn vị' (Unit Code), 'Mức báo động' (Alert Level), 'Tổng số ca nhiễm' (Total number of cases), 'Tổng số ca tử vong' (Total number of deaths), and 'Tổng số ca ph' (Total number of recoveries). The table lists 11 units, with the first one being 'Thành phố Hà Nội' (Hanoi City) with 141 cases and 63 deaths. A modal window titled 'Cập nhật tình hình đơn vị (cấp xã)' (Update unit status (district level)) is open, allowing the user to update the information for the selected unit. The modal includes a text field for 'Thông tin đơn vị' (Unit information) with the value 'Xã Yên Viên, Huyện Gia Lâm, Thành phố Hà Nội'. It also has three input fields for 'Số ca mắc mới' (New cases), 'Số ca tử vong mới' (New deaths), and 'Số ca phục hồi mới' (New recoveries). A date field for 'Ngày:(để trống ương ứng ngày hiện tại)' (Date: (leave blank for current date)) is also present. An orange 'Cập nhật' (Update) button is at the bottom right of the modal. The interface also features a sidebar with navigation options like 'Trang chủ', 'Thông tin', 'Duyệt tài khoản', 'Tình hình dịch', 'Bản đồ dịch', and 'Biểu đồ dịch'. At the bottom, there are pagination controls showing '100' items per page and 'bản ghi/trang' (records/page).

STT	Tên đơn vị	Mã đơn vị	Mức báo động	Tổng số ca nhiễm	Tổng số ca tử vong	Tổng số ca ph
1	Thành phố Hà Nội	1	3	141	63	
2	Tỉnh Hà Giang	2	0	0	0	
3	Tỉnh Sơn La	14	0	0	0	
4	Tỉnh Hoà Bình					
5	Tỉnh Hải Dương					
6	Tỉnh Thái Bình					
7	Tỉnh Nam Định				22	
8	Tỉnh Ninh Bình				0	
9	Tỉnh Hà Tĩnh				0	
10	Tỉnh Quảng Nam				0	
11	Tỉnh Khánh Hòa				0	

Figure 4.36 Update situation pandemic interface

To be able to update the Pandemic situation, you need to use a Commune/Ward administrator account. Including the number of new cases, the number of new deaths and the number of new recoveries. If you do not enter a date, the system will automatically select the current date to retrieve data. After clicking update, the system will automatically add District units - Province/City units.

## Pandemic map

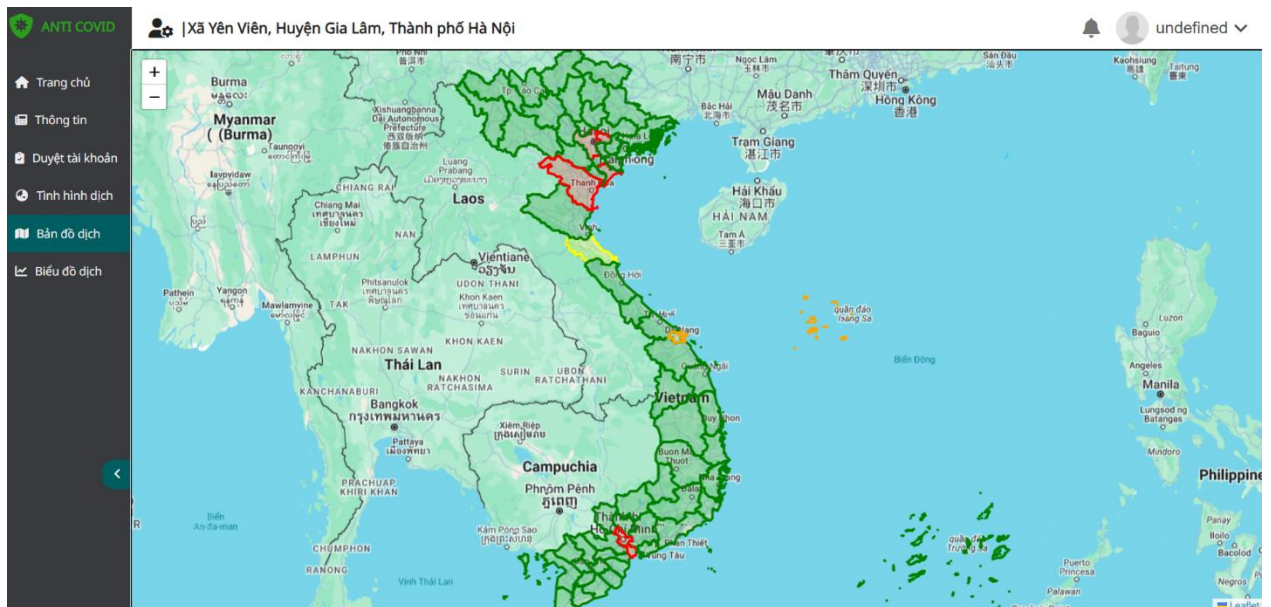


Figure 4.37 Pandemic map interface (1)

The Pandemic map takes data directly from the epidemic situation function. Allows users to view areas on the map and the epidemic situation of each area.

In particular, the Pandemic situation of the regions is expressed according to indicators such as: number of cases, number of recovered cases, number of deaths, etc.

In addition, depending on the level of epidemic in that area, there will be different colors. Includes 4 translation levels corresponding to 4 colors:

- Level 0 – Green
- Level 1 – Yellow
- Level 2 – Orange
- Level 3 – Red

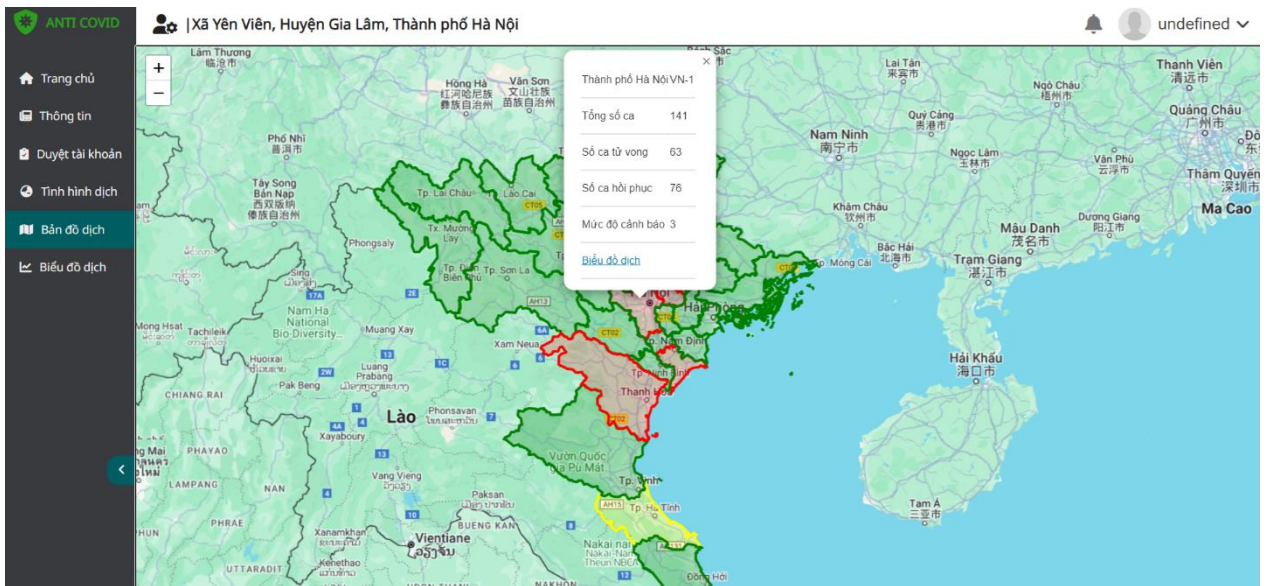


Figure 4.38 Pandemic map interface (2)

Use the mouse wheel to zoom out or zoom in on the map. Click on the area you want to view to see the epidemic situation displayed on the map. Click on the pandemic chart to see the epidemic chart in that province/city.

### Pandemic chart



Figure 4.39 Pandemic chart interface (1)

The chart shows the pandemic situation of the province.





Figure 4.40 Pandemic chart interface (2)

The chart shows the pandemic situation in the country and based on updated data from commune/ward administrators including the number of new cases, the number of new deaths, the number of new recoveries and dates.

The colors of the chart are displayed as follows:

- The number of new cases: Yellow
- The number of new deaths: Red
- The number of new recoveries: Green

## New pandemic

The screenshot shows a web interface for reporting a new pandemic. The main form is titled "Đậu mùa khí" and is divided into two sections. The first section, "Thông tin người khai báo", contains three input fields: "Họ tên", "CCCD", and "Địa chỉ". The second section, "Danh sách câu hỏi", contains three questions: "1. Triệu chứng", "2. Đã từng sử dụng các loại vắc xin MVA-BN - còn gọi là Imvamune, Imvanex hoặc Jynneos", and "3. Có tiếp xúc với các cá thể f0, f1...". A "Lưu" button is located at the bottom right of the form.

Figure 4.41 New pandemic interface

When clicking the "action" button on the list of new epidemics, the Civilian User fills in personal information and answers the questions available in the form. After click "save" the system will automatically record the results.

## New pandemic list

The screenshot shows a web interface for viewing a list of new epidemics. The main table is titled "Đậu mùa khí" and has six columns: "Họ tên", "CCCD", "Địa chỉ", "Triệu chứng", "Đã từng sử dụng các loại vắc xin MVA-BN - còn gọi là Imvamune, Imvanex hoặc Jynneos", and "Có tiếp xúc với các cá thể f0, f1...". The first row contains the following data: "Nguyễn Đức Thịnh", "024201024102", "Hải Dương", "Ho, Sốt, Phát ban, ...", "Chưa", and "Không".

Họ tên	CCCD	Địa chỉ	Triệu chứng	Đã từng sử dụng các loại vắc xin MVA-BN - còn gọi là Imvamune, Imvanex hoặc Jynneos	Có tiếp xúc với các cá thể f0, f1...
Nguyễn Đức Thịnh	024201024102	Hải Dương	Ho, Sốt, Phát ban, ...	Chưa	Không

Figure 4.42 New pandemic list interface

After the system records the results, medical staff can view civilians' declarations through the "action" button.

## Add new pandemic

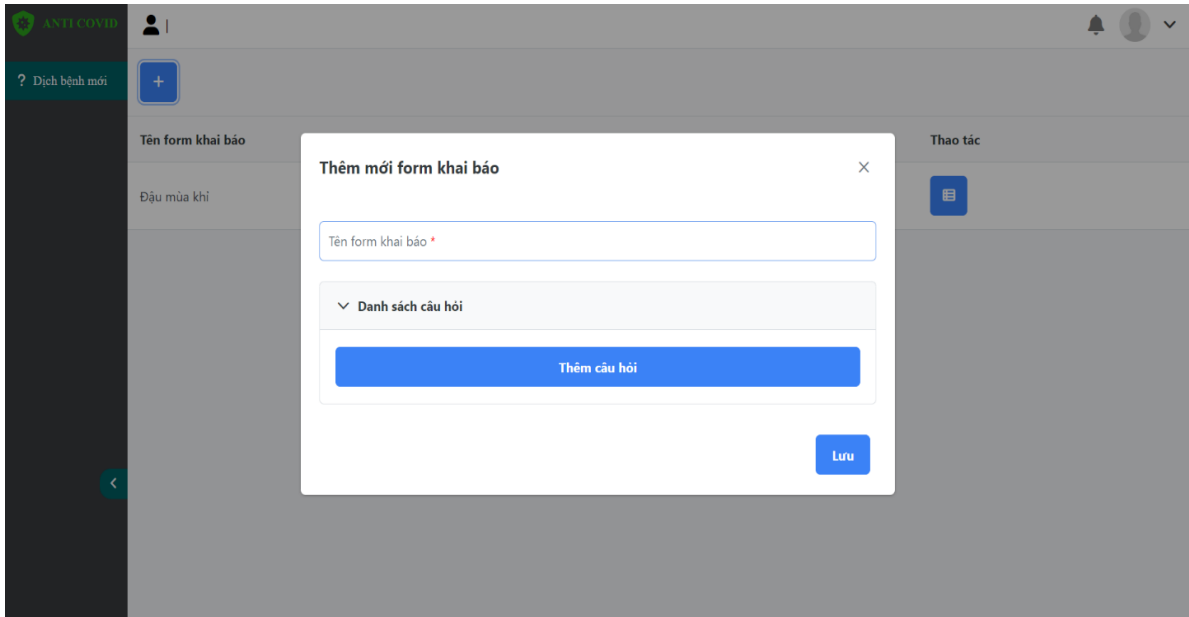


Figure 4.43 Add new pandemic interface

Medical staff can create new pandemic declaration forms by pressing the "add question" button to create and fill in the desired question forms. Click "save" to create the template.

### **4.3 Conclusion**

The development and deployment of the application has been completed, integrating necessary functions and ensuring system stability and security. The system is now ready for operation, meeting the set requirements and addressing practical issues in managing the pandemic. Next, we will summarize the entire process, evaluate the system's effectiveness, and propose future development directions, thereby opening new potentials and continually improving the system.

## CHAPTER V: CONCLUSION

### 5.1 Results and evaluation

#### 5.1.1 Results

“Building a website for epidemiology management using mongodb database” project designed for three main stakeholders: administrators, medical staff, and civilian, has been successfully completed with numerous essential features aimed at supporting effective epidemic management and prevention.

The result of the project is a complete epidemiology management system that meets the set requirements and brings practical benefits to epidemic prevention and control efforts. The system not only aids administrators and medical staff in working more efficiently but also provides the public with a valuable tool to protect their own health and that of the community.

The main functions of the system for civilians: register, login, personal information, posting, viewing notifications from administrators and medical staff, declaration, epidemic situation, epidemic map, epidemic chart, declaration of new epidemics.

The main functions of the system for medical staff: register, login, personal information, posting notices to civilians, viewing information requesting support from civilians and notifications from administrators. Manage and track declarations in the area, view epidemic situations, epidemic maps, epidemic charts, create new epidemic declaration forms.

The main functions of the system for administrators: register, login, personal information, browsing accounts of medical staff and administrators in the area, browsing posting, notifications from civilians, medical staff, administrators in the area, updated epidemic situation, epidemic map, epidemic chart.

### **5.1.2 Evaluation**

The project execution involved various stages including information collection, system design, technology research, programming, and testing to ensure the system operates stably and efficiently.

In the initial phase, I gathered information on different epidemics, epidemiological conditions across various regions, the needs of the public, and specific requirements from medical staff. The programming and testing phases were conducted rigorously to ensure the system's stability, and efficiency. Functional modules were developed and tested independently before being integrated and subjected to system-wide testing. I also conducted performance tests to ensure the system could handle large volumes of data and simultaneous access by multiple users.

In summary, the development of the epidemiology management system has successfully as the initial goals with the need for surveys and solutions, bringing practical benefits to epidemic prevention and control efforts. It not only helps administrators and medical staff work more effectively, but also provides civilians with a valuable system to protect their own health and that of the community.

### **5.2 Limitation**

Although, I have successfully built an epidemiology management system website. But due to poor knowledge and limited implementation time, the website still has many problems in research, design and implementation. This system is still rudimentary, has poor functionality and has not achieved high accuracy. Some incomplete functions sometimes cause errors. The built database for testing is still small with a few hundred cases and with limitations in scenarios. Weak security, vulnerable to data exploitation. Appropriate web technologies have not been deployed and applied, leading to system construction that is still sketchy, the general interface is not innovative, and system optimization is not effective. Poor survey to propose a new system and improve the system. Using mongodb database requires high memory to store data. During the project

implementation, I realized the necessity and importance of learning and self-development. I hope to be able to develop the project more effectively and apply it into practice in the future.

### **5.3 Future development**

As we look to the future, I want to implement an epidemiology management system to make it more robust, efficient, and capable of addressing a broader range of health challenges.

One significant upgrade involves transitioning from traditional web technologies to ReactJS for the front-end development. ReactJS, with its component-based architecture and efficient rendering, will allow us to build a more responsive and interactive user interface.

Another development will be the implementation of a real-time epidemic information system. Utilizing WebSocket or similar real-time communication technologies. Real-time data will be crucial in managing epidemic outbreaks more effectively, allowing for quicker response times and more accurate situational awareness.

Furthermore, I want to combine advanced analytics and machine learning algorithms to better understand epidemic patterns and predict future outbreaks. By leveraging big data and AI, the application can identify trends and anomalies that human analysts may not immediately notice. These insights will help health authorities make informed decisions and proactively implement preventative measures.

These advances will significantly improve the app's ability to manage and respond to many health challenges, ultimately contributing to more effective epidemic prevention and control.

## REFERENCES

- [1] Dùng Firebase Storage như backend lưu trữ dữ liệu cho ứng dụng Phạm Xuân Lu  
<https://viblo.asia/p/dung-firebase-storage-nhu-backend-luu-tru-du-lieu-cho-ung-dung-android-ZDEvLYAzGJb>
- [2] Use Case Diagram iviettech.vn <https://iviettech.vn/blog/543-ban-ve-use-case-use-case-diagram.html>
- [3] [https://www.linkedin.com/posts/akash-satpute31\\_mongodb-is-a-popular-open-source-nosql-database-activity-7181241114153164801-j0Cb](https://www.linkedin.com/posts/akash-satpute31_mongodb-is-a-popular-open-source-nosql-database-activity-7181241114153164801-j0Cb)
- [4] <https://www.imensosoftware.com/blog/why-nodejs-is-the-perfect-choice-for-building-scalable-and-high-performance-applications/>
- [5] <https://geobgu.xyz/web-mapping/leaflet.html>
- [6] <https://www.npmjs.com/package/@canvasjs/charts>
- [7] <https://www.apptible.com/docs/getting-started/deploy-starter-template/node-js>
- [8] <https://support.planet.com/hc/en-us/articles/360016337117-Creating-a-GeoJSON-file>
- [9] <https://www.sentrient.com.au/blog/covid-19-management-system>
- [10] [https://www.researchgate.net/figure/Promise-of-MS-C-therapies-for-COVID-19-A-Rapid-global-spread-of-severe-acute\\_fig1\\_341496418](https://www.researchgate.net/figure/Promise-of-MS-C-therapies-for-COVID-19-A-Rapid-global-spread-of-severe-acute_fig1_341496418)
- [11] <https://www.semanticscholar.org/paper/COVID-19%3A-Epidemiology%2C-Evolution%2C-and-Perspectives-Sun-He/2f547947bf87380c7fab13ba2c663bbbe9e643ec/figure/0>
- [12] <https://bloganchoi.com/mang-xa-hoi-anh-huong-suc-khoe-tam-than-trong-dai-dich/>

- [13] <https://www.path.org/our-impact/articles/open-source-software-tool-helps-governments-monitor-covid-19/>
- [14] <https://wallhere.com/vi/wallpaper/1333085>
- [15] <https://appsbd.com/how-to-create-map-using-leaflet-js-best-way-to-figure/>
- [16] <https://support.planet.com/hc/en-us/articles/360016337117-Creating-a-GeoJSON-file>
- [17] <https://www.npmjs.com/package/@canvasjs/charts>
- [18] <https://www.mongodb.com/products/tools>
- [19] <https://www.geeksforgeeks.org/how-to-add-collaborators-to-a-firebase-app/>
- [20] <https://medium.com/scalp/how-to-back-up-firebase-firestore-and-firebase-storage-bucket-b6d8dbc0cd7c>
- [21] <https://cloudsundial.com/salesforce-server-access-oauth-flows>