# Team 13


| Jie Wang | Yuhong He | Yunni Xiao | Yu Bai |
|---|---|---|---|
| 23205138 | 19326053 | 21212287 | 22212818 |


## Synopsis

This application is a UCD parcel delivery system, aiming to build an **informative** delivery system for UCD staff. This application is a frontend and backend separated system and mocks the real logistics system at UCD. According to the relevant research, the UCD post system is a central-dispense endpoint in the logistics system. This system contains logistics modules for each apartment including “Estate Service”, “Broker”, “Postman”, “Merville” and “Receiver”, as well as other modules including “Email”, “Database” and “Register”, which aims to assist the logistics ones’ process.

This application provides full process track as soon as the parcel arrives and mitigates the possibility of the loss of the package. Each role in the system has a clear view of the relevant information. The application provides strong scalability, maintainability, and fault tolerance by combining various distributed system technologies.

The application is abstracted as 3 layers, the application layer, the backend and frontend layer, and the Delivery system layer. System administrators can operate the application layer. Frontend and backend layer interacts with each other. At last, the delivery layer residents within the backend layer and is unseeable to the frontend layer. We used various distributed system techniques to implement the system.

Application website link: https://www.ucdparcel.ie/

## Technology Stack

### Distribution Technologies

1. Spring Boot 3
   The basic framework of the code base. The entire backend system is divided into eight Spring Boot projects, each responsible for different tasks.
2. Amazon MQ – RabbitMQ
   The project encompasses multiple services like parcel tracking, user notifications, data processing, and more. RabbitMQ enables these services to communicate via message queues, avoiding direct dependencies.
3. AKKA Actors
   Actors interact through message passing, simplifying communication and data sharing in complex systems.

### DevOps Technologies

1. SpringDoc - Swagger
   Automatically generates interactive API documentation. It plays a crucial role in the integration test because of its straightforward request tests functionality.
   E.g. Entry server swagger URL: https://api.ucdparcel.ie/swagger-ui/index.html
2. Jenkins
   Used to enable one-click deployment to simplify the cooperative integration testing during the release stage. And during the development phase, we continuously deploy the frontend to facilitate better system testing for team members, allowing them to effectively utilize the frontend. URL: http://162.62.224.31:8080/ (Non-login read allowed)

3. Nginx
   Utilized for reverse proxying and load balancing. It directs requests for www.ucdparcel.ie to the system's frontend path and proxies' requests for api.ucdparcel.ie to the server ports. To enhance system performance, we've deployed the backend of the project in two locations, employing Nginx for load balancing, and evenly distributing incoming requests.
4. Postman
   Used for integration test for each module during the dev stage.
5. Git
   Used for code version management.
   Repository link:
   Backend:https://github.com/Yuhong-He/ucd_parcel_backend.git
   Frontend: https://github.com/Yuhong-He/ucd_parcel_frontend.git

## Data Storage

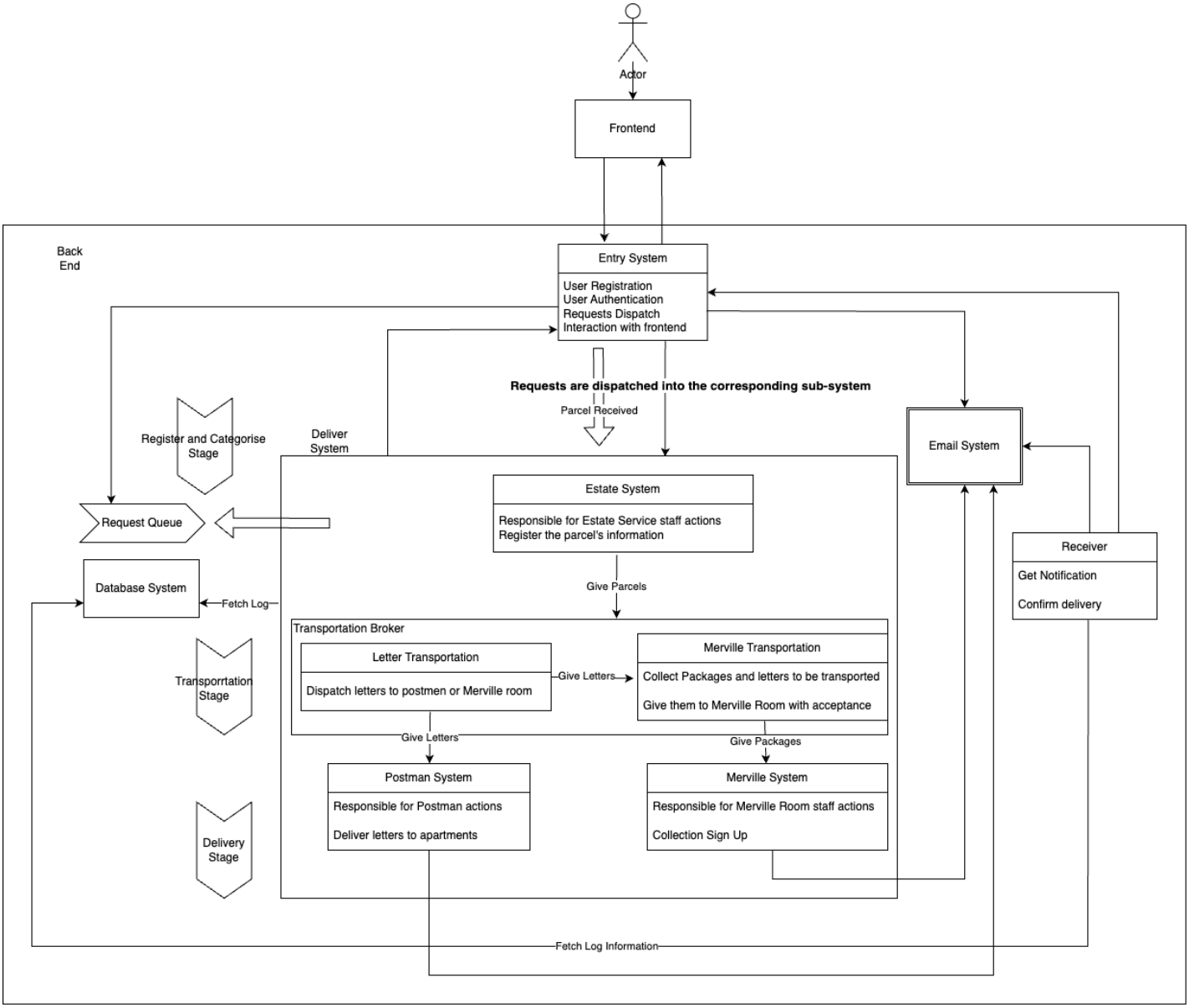
1. MongoDB
   Used to store parcel information due to its documental data structure, containing a list of parcel track.
2. AWS RDS– MySQL
   Used to store user information. It is highly reliable and stable in handling structured data, providing robust data consistency and integrity, ensuring the accuracy and completeness of user information.
3. Redis
   Used to store user registration verification codes. This is because verification codes are temporary data that can effectively leverage Redis' fast storage and automatic expiration capabilities, aiding in providing swift and reliable services in registration verification scenarios.
4. ORM Frameworks
   a. Spring Data MongoDB, responsible for MongoDB data operations
   b. MyBatis-Plus, responsible for MySQL data operations
   c. Spring Data Redis, responsible for Redis data operations

## Others

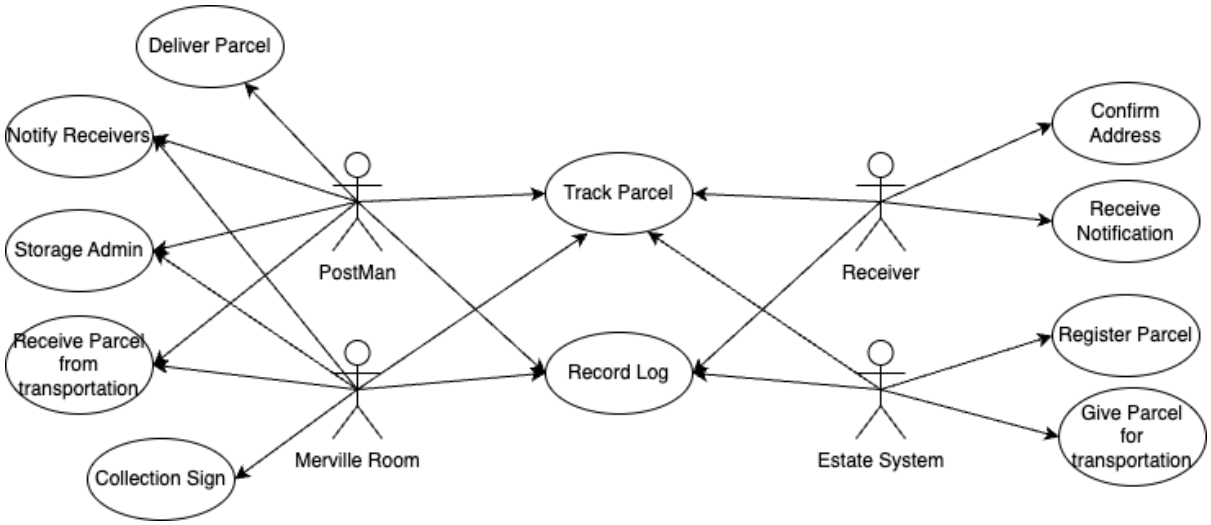
1. AWS SES
   AWS SES is utilized for sending emails to users across numerous scenarios within the system. It provides stable and efficient email services, ensuring that the dispatched emails have a high deliverability rate, minimizing the likelihood of them being flagged as spam or ending up in the recipients' spam folders.
2. Vue.js
   Vue.js serves as the frontend framework responsible for constructing the user interface. Additionally, it incorporates complementary packages such as Vue Router, Vuex, Axios, and utilizes the Element-UI framework for the user interface components.

# System Overview

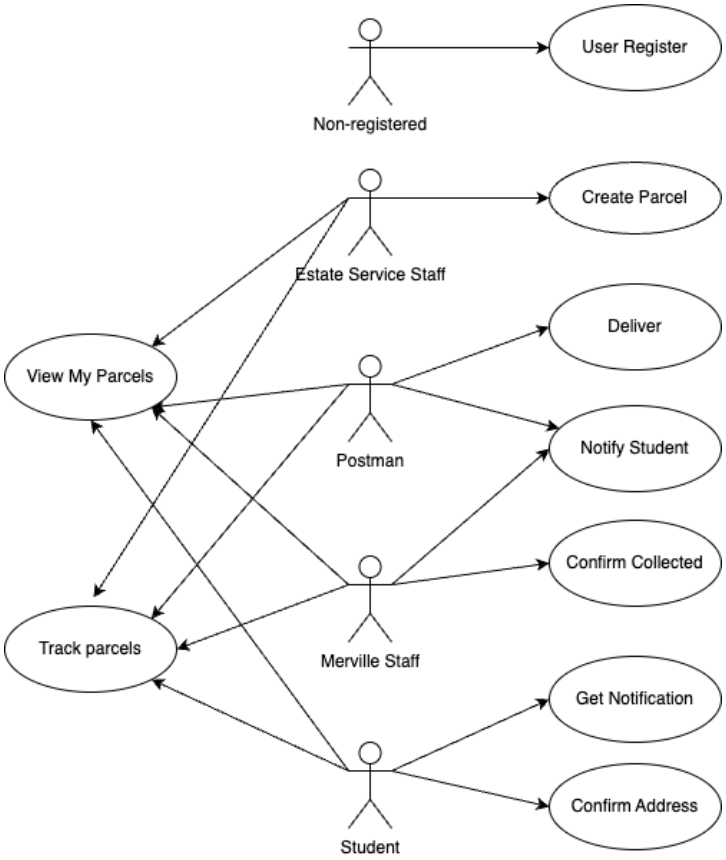
## Architecture Figures



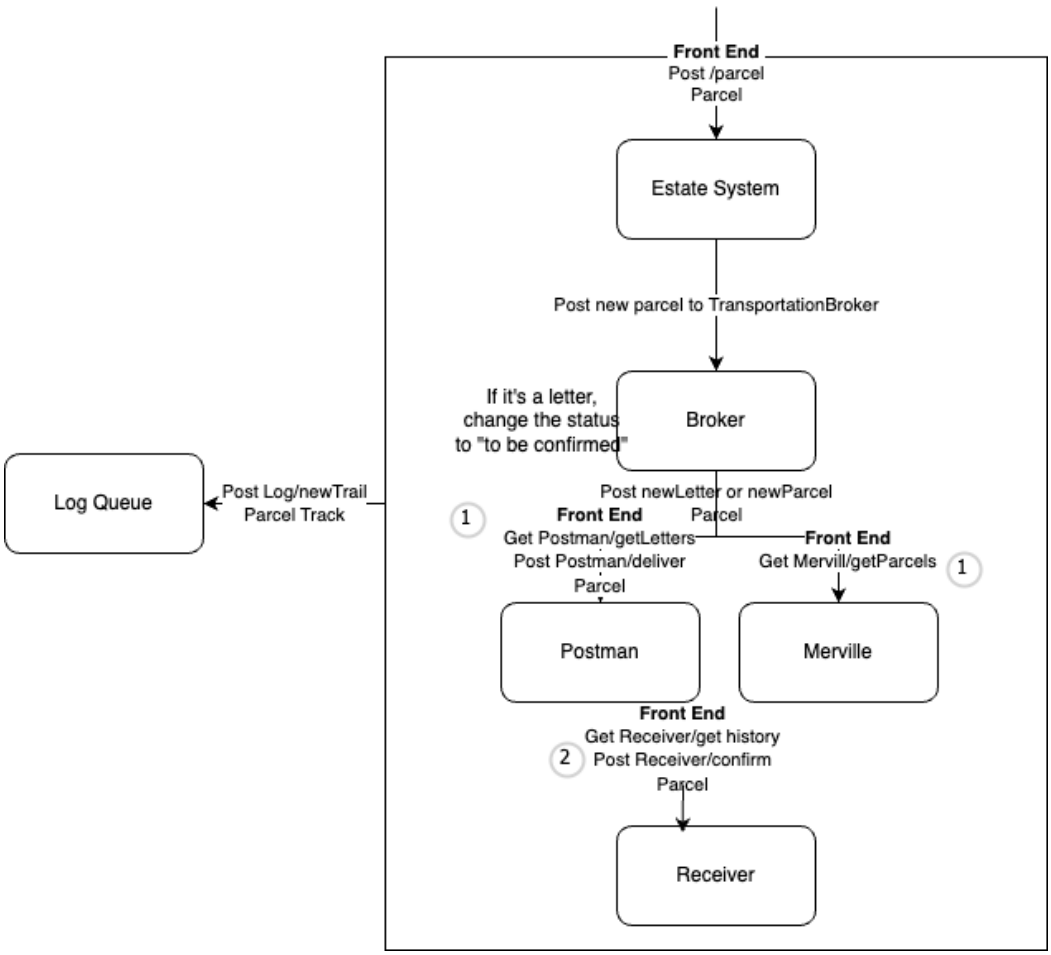

System Design




Backend Use Case




Frontend Use Case



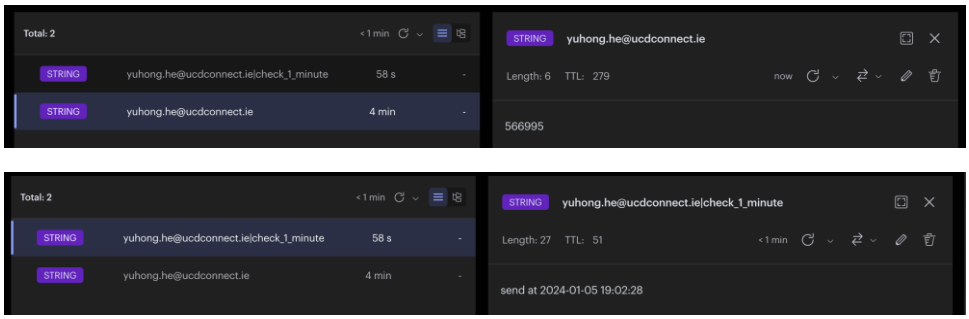

The Use Case of a New Parcel

```
_id: "0b9d9e5b-f868-488f-ac66-6744f047afc5"
type: 1
address1: "Ashfield"
address2: "Mercury 11"
student: 4
tracks: Array (2)
  0: Object
      description: "Estate Service created parcel label"
      postman: -1
      merville_room: false
      create_by: 1
      create_at: "2024-01-05 03:30:58"
  1: Object
      description: "Broker send parcel to Merville Room"
      postman: -1
      merville_room: true
      create_by: -1
      create_at: "2024-01-05 03:31:00"
_class: "com.example.database_system.MongoDB.Parcel"
```


*MongoDB Parcel Data Schema*

| id | username | password | email | type |
|---|---|---|---|---|
| 1 | Yuhong He | $2a$10$TlStZS.XTU1t3XA0lIGMMOI0iWnL0h0EcW… | heyuhong1273@gmail.com | 4 |
| 2 | Yuhong He | $2a$10$zYeScVV2sjA0UkczFLKmWeugOb4YiKg.9/… | heyuhong1273@gmail.com | 3 |
| 3 | Yuhong He | $2a$10$DJaZ28JEyZ9qFhFc/QngHOOR0OfwXI9AgI… | heyuhong1273@gmail.com | 2 |
| 4 | Yuhong He | $2a$10$TFy2fBa7XHGpinSb296bweLv1.gti4WneK… | heyuhong1273@gmail.com | 1 |
| 5 | Yuhong He UCD | $2a$10$VdkqtMhpsEsYjULWUPQBH.HmQGRJZKjl63… | yuhong.he@ucdconnect.ie | 1 |
| 6 | SP | $2a$10$AK2OtK2F/o62LerH1N0iD.JJLhfGEqeFts… | yunni.xiao@ucdconnect.ie | 4 |
| 7 | SP | $2a$10$TtTgsQWybWLdTvTmOsqnfeeSeV/QaZmNlj… | yunni.xiao@ucdconnect.ie | 1 |
| 8 | SP | $2a$10$jg/EauvhCXM2d4oOntdsMuuRmXZayi60l0… | yunni.xiao@ucdconnect.ie | 2 |
| 9 | SP | $2a$10$wj8TH6TNFJ6/CHrnbWlvUucPqMKqtvMK2d… | yunni.xiao@ucdconnect.ie | 3 |

*MySQL User Data Schema*




*Redis verification code record, verification code has 5 minutes expiration*
*Along with a "|check_1_minute" key with 1 minute expiration avoid sending verification codes repeatedly*

**a.** Entry System: Spring Boot, Spring Security, JWT, MongoDB, MySQL, Redis
   a. Interaction with frontend, convert the data received from other systems to appropriate format and return to frontend.
   b. User registration: verification code functions, register and login functions.
   c. User authentication: handle JWT, responsible for system security, validate user permissions for an endpoint.
   d. Distribute the request to different system based on actions and user rights.

**b.** Email System: Spring Boot, AWS SES

    a. Receive an encrypted RESTful request from other system and send the email.

**c.** Estate System: Spring Boot
    a. Handle Estate Service staff actions.
    b. Register a new parcel with Database System.
    c. Send the parcel to Broker System.

**d.** Broker System: Spring Boot, RabbitMQ, AKKA
    a. Dispatch parcel to Postman System or Merville System based on the type of the parcel.
    b. Send email to receiver, either request receiver to confirm delivery address, or notify receiver comes to Merville Room to collect the parcel.

**e.** Postman System: Spring Boot, RabbitMQ
    a. Handle postman actions.
    b. Deliver letters.

**f.** Merville System: Spring Boot, RabbitMQ
    a. Handle Merville Room staff actions.
    b. Monitor the status of parcels, if receiver come to collect them, should be confirmed in the system.

**g.** Receiver System: Spring Boot, RabbitMQ
    a. Handle receiver (student) actions.
    b. Confirm the delivery address before the postman delivers and send a notification email to the postman.

**h.** Database System: Spring Boot, RabbitMQ, MongoDB, MySQL
    a. Act as a central database interaction system, other systems need to interact with the database through this system.
    b. Return parcels and history tracks required from other systems' RESTful requests.
    c. Whenever other systems require information from the MySQL database, query it and return the requested data to them.
    d. Receive messages from RabbitMQ and append parcel tracks from other modules to the Database.

## Scalability and fault tolerance.

Functionality of the system is well split into modules and interaction between modules is simplified using AKKA and RabbitMQ. The operations to the databases are implemented with database sub-system so that newly added modules don't have to implement the ORM or JDBC but focuses its business logic. The entry module handles all the requests from the browser and dispatches them into modules, simplifying the user permission access administration.

MongoDB, RabbitMQ and Email services are deployed on the cloud service platform like AWS and MongoDB, which provide strong fault tolerance and robustness.

By setting Jenkins build hooks, we provide CI/CD services to make deployments of new modules easier. Moreover, the Swagger docs are well written, providing clear API documentation for further development.

## Contributions

Jie Wang

- System design.
- Project management and development documentation.
- Implementation of Database System and Postman System.
- Performing integration tests for all systems.

- Application deployment with Jenkins and Nginx.

Yuhong He

- Participate in system design.
- Development, cloud and deployment configuration (AWS RDS MySQL, Redis, MongoDB, Amazon MQ RabbitMQ, Jenkins, AWS SES, domain name registration and DNS management, Nginx).
- Implementation of Entry System, Email System, Estate System, Merville System.
- Implementation of frontend.
- Integrating all systems together.
- Performing system tests.
- Application deployment with Jenkins and Nginx.

Yunni Xiao

- Participate in system design.
- Implementation of Receiver System.
- Performing system tests.

Yu Bai

- Participate in system design.
- Implementation of Broker System.
- Performing system tests.

## Reflections

### What were the key challenges you have faced in completing the project? How did you overcome them?

In the area of technology, we used things like RabbitMQ, Akka and Spring Boot. Though these tools made the whole system work better, their use and complexity were problems for the group. We spent a lot of time learning about these new ideas and using the powers of these tools.

For working together on a team, locations far apart and time differences caused problems. To fix this problem, along with regular meeting, we defined standard workflow including Git, Google doc and Jenkins to make everyone just in the right place of the codebase and continue contributing.

Although we used the Git, the configuration files which store connection sentences to the paid services are not included in it. Keeping them synced between the team member dev environment, servers and Jenkins was a hassle. We used Jenkins to store the latest version of them however it was inconvenient to fetch them.

As the project developed, we found some problems in its initial design circle. To make the system smarter, we chose to move from MySQL to MongoDB for storing data. We also added a Database module so we can aggregate the wide used operations on data storage. But this choice affected ready-made parts called Receiver and Broker, which deal with data management. This required changes to these pieces so they can work right again. One the other hand, different teams or modules implemented the data model based on varying interpretations, resulting in incompatible interfaces. To overcome this issue, we've enhanced communication, established unified data models, and standardized interface protocols, ensuring clarity for all team members.

## What would you have done differently if you could start again?

On the design part, I would spend more time researching and understanding the needs at the start of a project. This makes sure we know how the system works and what it needs to run well, so there will be fewer changes or readjustments needed later. For example, we should concentrate on things like how data is stored and what needs the program must meet. When making and building the system, I would first focus on figuring out what's needed for databases and core parts to handle data management. Then I could move forward with creating other parts of it. Meanwhile, before the project begins, we will spend more time on standardizing and meticulously planning the data structure and interface design. This ensures that prior to the start of coding, all team members have a clear understanding of the data models and interaction protocols being used. This prevents interface mismatches or data processing errors among different modules and team members due to inconsistent data structures.

On the technical side, I would review more closely how familiar team members are with technology and consider the existing technological environment. This will help make sure that selected technologies better match project needs. When we start using new tools, I would give the team more time to learn and change fast. This will keep projects on track with less disruption. Moreover, we would set a gitlab repo to build a more functional version control environment during development stage.

## What have you learnt about the technologies you have used? Limitations? Benefits?

### Module Framework:

SpringBoot：

Benefits： Spring Boot uses the idea of "convention over configuration" by giving many default settings. This helps developers work faster and makes their task easier. It uses different parts from the Spring family of tools, helping developers easily add features such as safety, getting data and sending messages. Spring Boot brings in an automatic setup system that sets up most parts of the app based on what it needs. This makes setting up easier and we need less help from people. These things all together make it easy and quick for people who write computer programs to build web apps. They have everything they need, making their work better and faster too.

RabbitMQ：

Benefits： On one side, RabbitMQ makes it possible for different parts to talk with each other without waiting. This helps them work better and respond faster. However, RabbitMQ gives message queues to hold and handle messages. This makes sure that reliable delivery of messages happens even if there are failures.

Limitation： Setting up and running RabbitMQ is really hard, so we spent much time learning how it works and putting it in place.

## Database

MongoDB：

Benefits： MongoDB can hold datas with various shapes. It easily changes to fit changing needs for data storage. It's really good at handling more and bigger data smoothly as it gets larger. By using strong search languages with options like range searches and matching regular patterns, getting data becomes easy. MongoDB brings these tools together to give a flexible and big-growing way for handling many kinds of changing information.

Limitation： Sometimes, MongoDB's ability to handle transactions may not be as good as that of some other databases. This can make it less useful for situations where many important actions need fast and reliable processing. For example, in the part of the system that deals with users, information about them like usernames and passwords stays saved in a MySQL database. This is because relational databases provide better help for dealing with tasks that need strong transactions.

Redis：

Benefits： Redis keeps data in memory, allowing fast read and write actions. This makes it good for situations that need quick turnaround times. In our system, Redis is used to handle the number of confirmation codes during user sign-up. Using Redis's automatic delete function, we make sure verification code data becomes useless after a given time. This works well with the business needs and meets them properly.

Limitation：Redis keeps data in memory, and how much it can hold is restricted by the actual amount of memory we have. As the amount of information gets bigger, it can cause more strain on computer systems.

## Devops Tools:

Jenkins：

Benefits： Jenkins is a free open-source tool that helps with continuous integration and delivery (CI/CD). It's available for everyone. It has an easy-to-use setup and arrangement process, as well as lots of helpful plugins. This lets groups connect Jenkins easily and start builds fast. Also, they can deploy apps quickly too. Furthermore, the many plugin options help meet different needs. The connection helps make the creation and release of new projects more professional and quicker.

Swagger：

Benefits： Swagger makes it easy to make API papers based on code notes and rules for APIs. This helps keep things equal and standard while also making manual paperwork easier, so people don't have to do as much of it by hand. Swagger is good for testing. It has a simple way to test APIs called Swagger UI, so you don't need other tools or steps. Using Swagger makes the API paperwork and testing look more professional.

Limitation： But, keep in mind that extra work may be needed because special needs might require more setup.