TRƯỜNG ĐẠI HỌC FPT

Capstone Project Final Report

Admission Supporting Chatbot

FUTO Team	
Group Members	Nguyễn Văn Du – SE4312 Đỗ Thị Thu Quỳnh – SE04133 Dương Việt Anh - SE04154 Đào Mạnh Tuấn – SE04002
Supervisor	Lecturer: Nguyễn Tất Trung
Capstone Project code	ASC

Hanoi, August 28th, 2018

Table of contents

Acknowledgements

We wish to express our deepest gratitude to our supervisor Mr. Nguyen Tat Trung for his continuously sharing and motivating throughout the project. Under the instructions of Mr. Trung, the project team has always felt comfortable and confident. Apart from all the technologies and methodologies, Mr. Trung shared with us the philosophy of keeping the right attitude towards problems. We believe that was the most important factor that led to the success of this project.

We would also like to thank the instructors at FPT University for all the classes. We hope you will find this project as a reflection of the knowledge and experiences you have given us during this period of four years.

Finally, we truly appreciate the random fellows on the internet who has discussed with us. Without them, it would have likely been a real struggle to solve every tiny problem.

Definitions and Acronyms

Terminologies

Terminology	Definition
Chatbot (also Bots)	Software that is used to interact between a computer and a human in natural language.
Broadcast	A message proactively to users.
Nodes	A node is where different points of a dialog or workflow intersect
Utterance	Anything someone using chatbot says to it.
Intent	The few essential words that describe what the user wants the chatbot to do.
Entity	Entities are the fields, data, or words the developer designates are necessary for a chatbot to complete the user's request. An entity could be a date, a time, a location, a description or any number of things.
Chat logs	Histories of all recorded human-to-bot interactions.

Acronyms

Acronym	Definition
API	Application Programming Interface
BDD	Behavior-Driven Development
CPU	Central Processing Unit
NLP	Natural Language Processing
NLU	Natural Language Understanding
NER	Named Entity Recognition
BoW	Bag-of-Words
SVM	Support Vector Machine
TF-IDF	Term Frequency- Inverse Document Frequency
DBMS	Database Management System
DOM	Document Object Model
GUI	Graphical User Interface
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
UML	Unified Modeling Language
N/A	Not Available
OS	Operating System
PM	Project manager
REST	Representational State Transfer
SRS	Software Requirement Specification
TDD	Test-Driven Development

UC	Use Case
UI	User Interface
UX	User Experience
URL	Uniform Resource Locator
FAQ	Frequently Asked Questions
PSID	Page-Scoped ID
Fb	Facebook
FU	FPT University
JWT	JSON Web Token
ASC	Admission Supporting Chatbot

Chapter 1 : Introduction

1.1 Purpose

This chapter provides an overview of the project including background information, a literature review of the existing system and raising a proposal for ideas of improvement.

1.2 Project Information

- Project name: Admission Supporting Chatbot
- Project code: **ASC**
- Project group name: **FUTO**
- Product type: **Chatbot Application**
- Timeline: **May 14th 2018 - August 28th 2018**

1.3 The people

1.3.1 Supervisor

	Full name	**Phone**	**E-Mail**	**Title**
Supervisor	Nguyễn Tất Trung	0904399139	trungnt77@fe.edu.vn	Lecturer

Table 1-1: Supervisor's information

1.3.2 Team members

	Full name	**Student code**	**Phone**	**E-mail**	**Role in Group**
1	Nguyễn Văn Du	SE04312	0904423162	dunvse04312@fpt.edu.vn	Leader
2	Đỗ Thị Thu Quỳnh	SE04133	0966343635	quynhdttse04133@fpt.edu.vn	Member
3	Dương Việt Anh	SE04154	01645500597	anhdvse04154@fpt.edu.vn	Member
4	Đào Mạnh Tuấn	SE04002	0974481558	tuandmse04002@fpt.edu.vn	Member

Table 1-2: Team member's information

1.4 Background

In early 2016, the admission office of FPT University in Hanoi created a Facebook page named "Đại học FPT Hà Nội"[1] to connect with students and their family. According to statistics of the admissions counselor[2], so far, the page has nearly 84,000 followers, an average of 40 people inbox a day, and the peak period of enrollment is up to 60 people inbox a day.

[1] "Đại học FPT Hà Nội" Facebook page: https://www.facebook.com/DaihocFPTHaNoi/

[2] The admissions counselor 's Facebook page: https://www.facebook.com/laiiihonganh (Lại Hồng Anh)

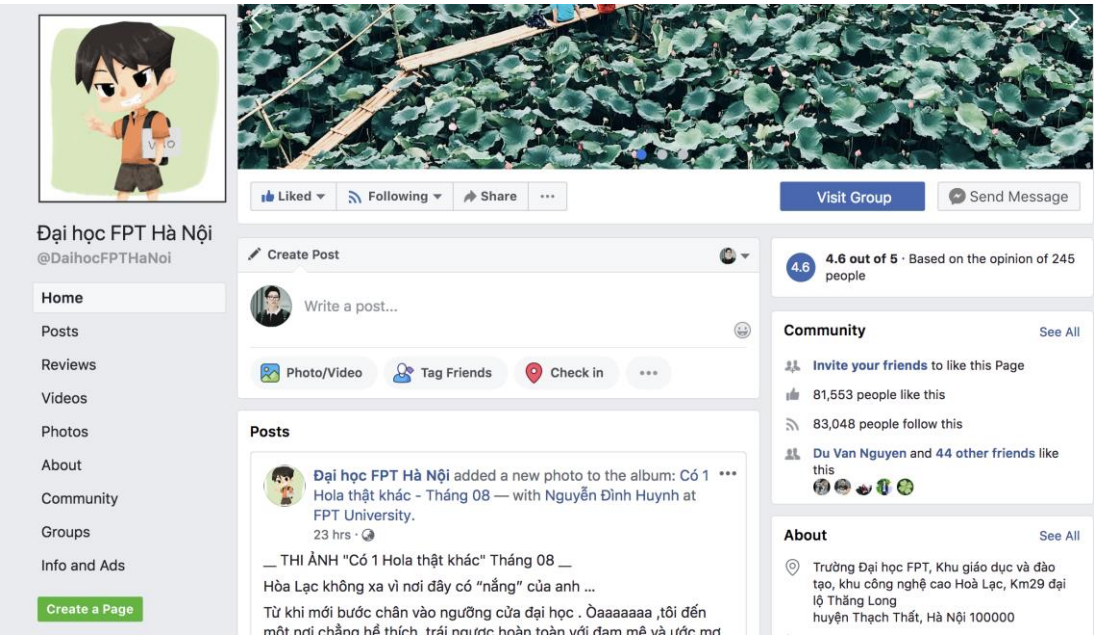


Figure 1-1: FU Facebook page

The communication of potential students with FU offices is performed manually and it is a very time-consuming procedure. The communication will require a staff to expend several hours to find suitable answers and contact each student. It would be useful to reduce costs and time. Addressing this issue, the project aims to reduce the burden on the head of admissions, and potentially other users, by developing a convincing chatbot. The main goal of such a system is to conveniently retrieve information without having to look or browse several web pages to fetch answers to frequently asked questions.

1.5 The Existing System

While there are already chatbots that allow users to communicate, they all have certain problems. Below, we will go over a few chatbots application and their problems.

Features	**Ana**[1]	**Harafunel**[2]
Support Vietnamese chatting.		✓
Serve in specific admission domain.	✓	
Throw random greeting responses.		
The accuracy of a domain specific bot can further improve.		
Having the flow of conversations and context.		
Allow the end user to leave personal information.	✓	✓
Allow the end user to rate and review.		✓

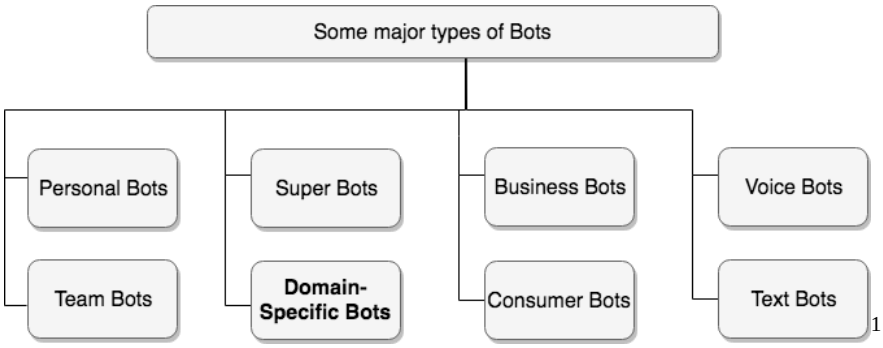
[1] Ana Facebook page: https://www.facebook.com/Study.at.US/
[2] Harafunel website: https://harafunnel.com/

Having dashboard administration.	✓	✓
Allow the admin to view statistics.		✓
Allow the admin to manage chat logs.	✓	✓
Allow the admin to send messages to all end users at time.		✓
Allow admin to categorize end users into different groups.		✓

Table 1-3: The existing system information

1.6 The Proposal of system

As we seek that not all bots are born the same. Bots differ from one another in many aspects.



[1]

Figure 1-2: Major types of Bots

Furthermore, it is very important to know how to choose the use cases where bot can be utilized.

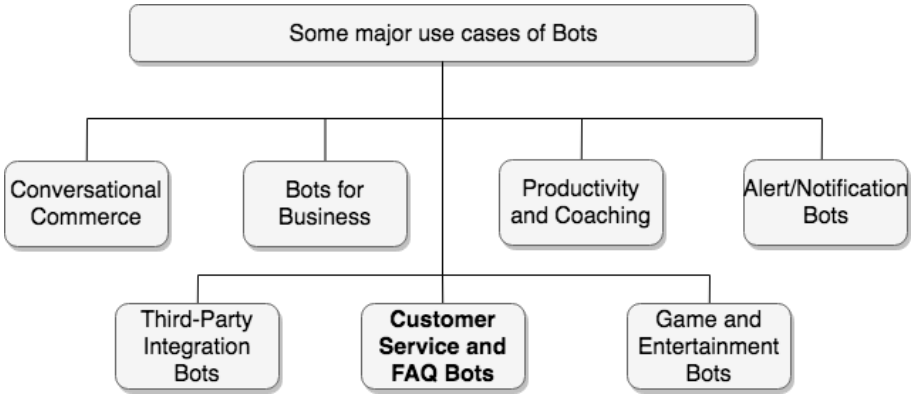


Figure 1-3: Major use cases of Bots

Considering the admission problems of FPT University, the proposal investigates the implementation of ASC system as a domain-specific chatterbox named FAQ bot, our work will show

[1] Designing Bots Book: https://www.amazon.com/Designing-Bots-Creating-Conversational-Experiences/dp/1491974826

how a chatbot can work as domain specific information system and experiments on how the system's accuracy could be improved based on a specific domain.

Figure 1-4: Our idea

ASC is a chatbot designed for Vietnamese. Unlike existing applications, all basic functions will be usable for free and there have a delightful conversation. In addition, the application was evaluated by keeping logs of questions and answers and by feedback received by potential end users that used it.

1.6.1 System functions

- Allow the end user to chat in private.
- Allow the end user to rate and feedback.
- Allow the staff to login by ASC account.
- Allow the staff to view and update personal profile.
- Allow the staff to view some statistics.
- Allow the staff to search for account and response of the chatbot.
- Allow the staff to view facebook information, feedback and chat logs of the end user.
- Allow the staff to request to edit data.
- Allow the staff to receive notifications related to his/her activities.
- Allow the admin to manage data.
- Allow the admin to add tags or remove tags from end user.
- Allow the admin to manage ASC accounts including change role, status or create a new account.
- Allow the admin to collect end user's personal information, then export to a file.
- Allow the admin to send messages to all potential end users with different labelling tags.

1.6.2 Out of scope functions

Because of the time limitation, we will not implement these following functions in the ASC initial version 1.0.0 release of the project but not permanently excluded, and will be developed in future version 2.0.0, although we are aware that they so also important:

- ASC chatbot will analyze end user's information to recommend what should to do.
- ASC chatbot will be able to self-study, without manual training.
- ASC chatbot will be integrated into website besides Facebook Messenger.
- ASC chatbot will support English language.
- Staff can edit chatbot scenario to response the end user.

Finally, we use some new approaches to solve the stated problems. Having awareness of the importance of new technologies, we aim at applying them to our project.

1.6.3 Special approaches

- For backend system
    - Using HTTP Methods & API Routes for building a Node.js RESTful API.
    - Using HTTP Status Codes Correctly if something goes wrong while serving a request, we must set the correct status code for that in the response.
    - Using HTTP headers to Send Metadata.
    - Using Express Framework for Node.js REST API, to create browser applications, and as such, it supports templating and rendering.
    - Using Microsoft Azure, Linux Virtual Machine for storage, hosting and deployment.
    - Using real-time WebSocket technology with Socket.IO library.
    - Using Elasticsearch for searching and filtering.
    - Using Memcached in-memory data store for caching.
- For frontend system:
    - Using ReactJS for web component rendering.
    - Using Redux for managing application state.

Chapter 2 : Project Plan

2.1 Purpose

This chapter provides an overview of the project plan including project organization and project management plan.

2.1.1 Software Process Model

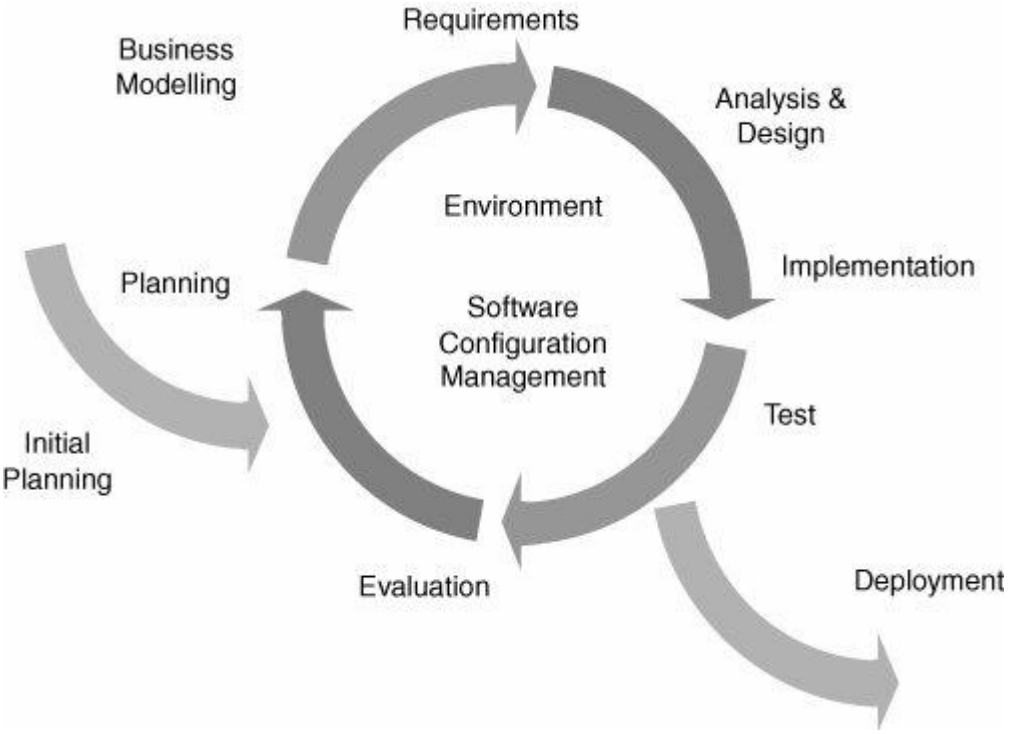


Figure 2-1: Iterative and Incremental Software Process Model

ASC project uses the Iterative and Incremental Software Process Model.

The Iterative and Incremental Software Process Model is mostly used when the scope of the project is big, the major requirements are defined clearly, some more details will be added later in software development. By using this software process model, we break down the developing system task into series of smaller tasks which will be completed separately, allowing us to take advantage of what was learned during development of earlier parts of the system. In addition, the iterative model is easier than other models when the issues are discovered. They are fed back to the team, and solutions will be found while the project is still in development.

2.1.2 Roles and Responsibilities

2.1.2.1 Organization Structure

Role	Responsibility
Project Manager	Planning, developing schedules, coordinating communication, generally responsible for keeping the team's focus on the main goal.
Technical Leader	Responsible for choosing and deciding what technologies should be used, as well as for overseeing the work being done by other developers.
Quality Assurance Manager	Ensuring the product meets the certain standards of quality from requirements.
Test Leader	Responsible for test execution, including test set-up and test run, evaluation of test run and error recovery, defect logging and test results recording.
Developer	Involve to code the product and review code of other developers.
Designer	Involve to design product's user interface.
Tester	Involve to test the product.
Business Analyst	Analyzes an organization or business domain and documents its business or processes or systems.

Table 2-1: Project Structure

2.1.2.2 Project Team Member

Team Member	Role
DuNV	Project Manager, Developer, Business Analyst, Designer.
QuynhDTT	Developer, Quality Assurance Manager, Business Analyst, Test Leader.
AnhDV	Technical Leader, Developer, Tester.
TuanDM	Developer, Tester.

Table 2-2: Project Team Member

2.1.3 Tools and Techniques

Programming languages	NodeJS, JavaScript
Framework	ExpressJS, Socket.IO, Elasticsearch
API	Wit.ai
DBMS	MongoDB
IDEs/Editors	Visual Studio Code, Atom
UML tools	Visual Paradigm, Hackolade, Microsoft Visio, Astah
Version Control	Gitlab
Deployment server	Microsoft Azure, Linux Virtual Machines
Project management tool	Microsoft Project, Trello, Backlog
Process model	Iterative and Incremental Software Process Model
Development process	Test-driven development, Behavior Driven Development

Table 2-3: Project Team Member

2.2 Project Management Plan

2.2.1 Tasks

Refers to “ASC_ProjectManagement.mpp” file.

Task Name	Duration	Start	Finish	Predecessors
ASC	77 days	Mon 5/14/18	Tue 8/28/18	
Initation	4 days	Mon 5/14/18	Thu 5/17/18	
First Meeting	1 day	Mon 5/14/18	Mon 5/14/18	
Define Project	3 days	Tue 5/15/18	Thu 5/17/18	
Compare existing system	1 day	Tue 5/15/18	Tue 5/15/18	
Define System function	1 day	Wed 5/16/18	Wed 5/16/18	
Create Proposal Document	1 day	Thu 5/17/18	Thu 5/17/18	7
Complete Proposal Document	0 days	Thu 5/17/18	Thu 5/17/18	8
Solution	9 days	Mon 5/21/18	Thu 5/31/18	
Technologies Research	8 days	Mon 5/21/18	Wed 5/30/18	
React Research	4 days	Mon 5/21/18	Thu 5/24/18	9
Node JS Research	4 days	Thu 5/24/18	Tue 5/29/18	12
MongoDB Research	1 day	Thu 5/24/18	Thu 5/24/18	9
Wit.AI Research	4 days	Fri 5/25/18	Wed 5/30/18	14
Analyst and Requirement	1 day	Thu 5/31/18	Thu 5/31/18	
Executation	49 days	Mon 6/4/18	Thu 8/9/18	
Initial Documentation	14 days	Mon 6/4/18	Thu 6/21/18	
Implementation	38 days	Tue 6/19/18	Thu 8/9/18	
Phase 1	15 days	Tue 6/19/18	Mon 7/9/18	
Controlling	15 days	Tue 6/19/18	Mon 7/9/18	
Traing Data	15 days	Tue 6/19/18	Mon 7/9/18	
Coding	14 days	Tue 6/19/18	Fri 7/6/18	
Testing	15 days	Tue 6/19/18	Mon 7/9/18	
Phase 2	23 days	Tue 7/10/18	Thu 8/9/18	
Controlling	23 days	Tue 7/10/18	Thu 8/9/18	
Training	23 days	Tue 7/10/18	Thu 8/9/18	
Coding	20 days	Tue 7/10/18	Mon 8/6/18	
Testing	23 days	Tue 7/10/18	Thu 8/9/18	
Termination	13 days	Fri 8/10/18	Tue 8/28/18	

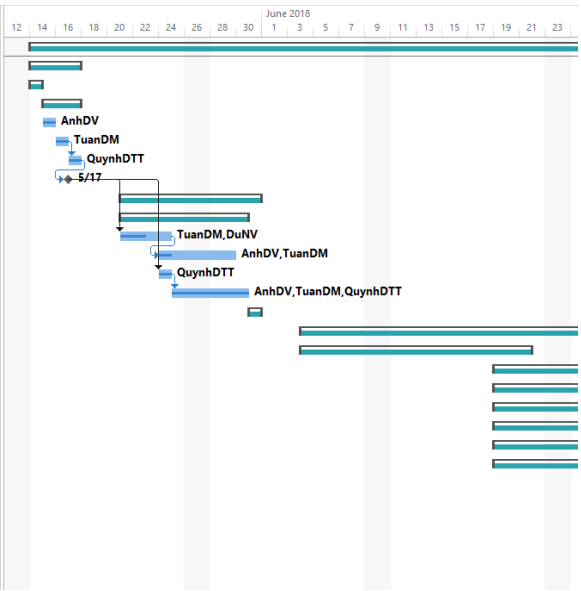


Figure 2-2: ASC Project Management file

2.2.2 Meeting Minutes

All meeting minutes will be written following this template:

Meeting/Project Name:	*ASC*		
Date of Meeting:	*18/05/2018*	***Time: (Type)***	*1 hours (Face-to-face)*
Meeting Called by:	*DuNV*	***Location:***	*FPT University – P205*
Note Taker:	*QuynhDTT*	***Time Keeper:***	*DuNV*

1. Meeting Objective

1. Assign new task for member
2. Do report No.2

2. Attendance

Name	***Roles***	***E-mail***	***Phone***
Nguyen Van Du	Project Manager	dunvse04312@fpt.edu.vn	0904423162
Do Thi Thu Quynh	Developer, Tester Leader	quynhdttse04133@fpt.edu.vn	0966343635
Duong Viet Anh	Developer, Tester	anhdvse04154@fpt.edu.vn	01645500597
Dao Manh Tuan	Developer, Tester	tuandmse04002@fpt.edu.vn	0974481558

3. Done task

1. Build project structure
2. Configure git repository with security
3. Do report No.1

4. New task

1. Backend
   a. Training Data with Wit.ai
   b. Setup Environment
   c. Code Modules Integration
2. Frontend
   a. Setup environment
   b. Code Dashboard Homepage
   c. Code Login page
3. Do report No.2

5. Risk & Difficulty

N/A

Table 2-4: Meeting Minutes Template

2.2.3 Coding Conventions

We strictly follow Eslint JavaScript Style Guide.

Please refer to **JavaScript Style Guide - Eslint.pdf** file or the official website at https://www.npmjs.com/package/eslint

2.2.4 Risk Management Plan

No	Description	Avoidance plan	Contingency plan	Status
R1	Data loss	- Use GitLab for version control. - Teach members how to use Git and resolve conflicts. - Always have important backups.	- Restore backed up data from GitLab.	Closed
R2	Illness or absence of team members	- Provide schedules in advance. - For long periods of absence, members should notify the group in advance.	- Assign the tasks of absent member to other members. - Work overtime if necessary.	Closed
R3	Misunderstanding of requirements	- Discuss requirements carefully with the customer. - Always ask for clarification if requirement is unclear. - Comment need to meet reality and possibility.	- Make sure idea's business logic is carefully analyzed.	Closed
R4	Requirement changed	- Every new update of requirement has to be reviewed by all team members and supervisor. - Team member has to analyze requirement carefully before raise up to team.	- If requirement has new update, all members have to join the meeting to aware and make decision.	Closed
R5	Conflict between team members	- Everything must be documented. - Every team member has to express clearly and carefully.	- Make sure any miscommunication has to be resolved.	Closed
R6	Failure to meet deadline	- Plan and develop schedule carefully - Assign tasks carefully - Define punishment for team members who neglect work	- Find the root cause of the problem - Reassign tasks - Focus on important functions first	Closed

Table 2-5: Risk Management

2.2.5 Communication Plan

Weekly meeting schedule: We use Iterative and Incremental Process Model, then we divide the system into two sub-systems (ASC Backend and ASC Frontend), each sub-system is divided into a series of small tasks. Each task is logged to TRELLO then estimated depending on difficulty and the amount of work by the whole team, after that the task will be assigned to team members by the Team Leader and depending on difficulty the Technical Leader will assign deadlines for each task. We will have a meeting every Sunday to inform to all team about what each member finished last week, the status (fast, on time or slow), the issues met and how to solve them. If any member raises any issue, the whole team will help to find out a solution together. After that, the team will define detailed stories for next week tasks.

Daily meeting schedule: Each sub-system has one development team with different schedule. When starting work-day, each team will have a stand-up meeting to inform to others: "What did I do yesterday?", "What will I do today?" and "Is there any difficulty?". By focusing on what each person accomplished yesterday and will accomplish today, the team gains an excellent understanding of what has been done and what remains.

Unscheduled meeting: If someone has an important problem that he wants to solve immediately, we will have a meeting for discussion, usually via some online channel: Slack, Facebook or Phone.

Communication channel: Our main communication channel is Slack. On the other hand, we use face-to-face meeting, Facebook group and comment on TRELLO issues. However, we sometimes make a phone call or instant message if someone has a problem.

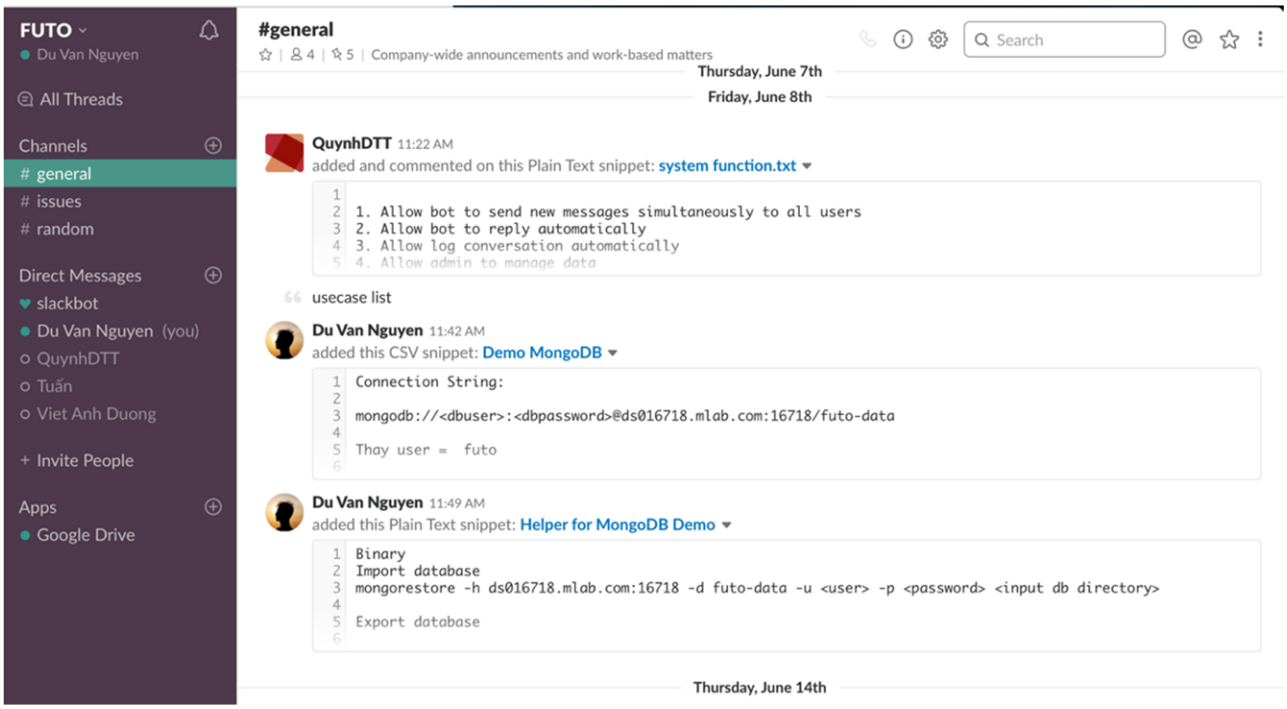


Figure 2-3: Communication via Slack

Chapter 3 : Software Requirement Specification

3.1 Purpose

This chapter outlines functional and non-functional requirements of our website. It also provides some format constraints in common requirements and project success criteria. All members will work based on the information provided in this chapter.

3.2 Functional Requirements

3.2.1 Use Case Diagram

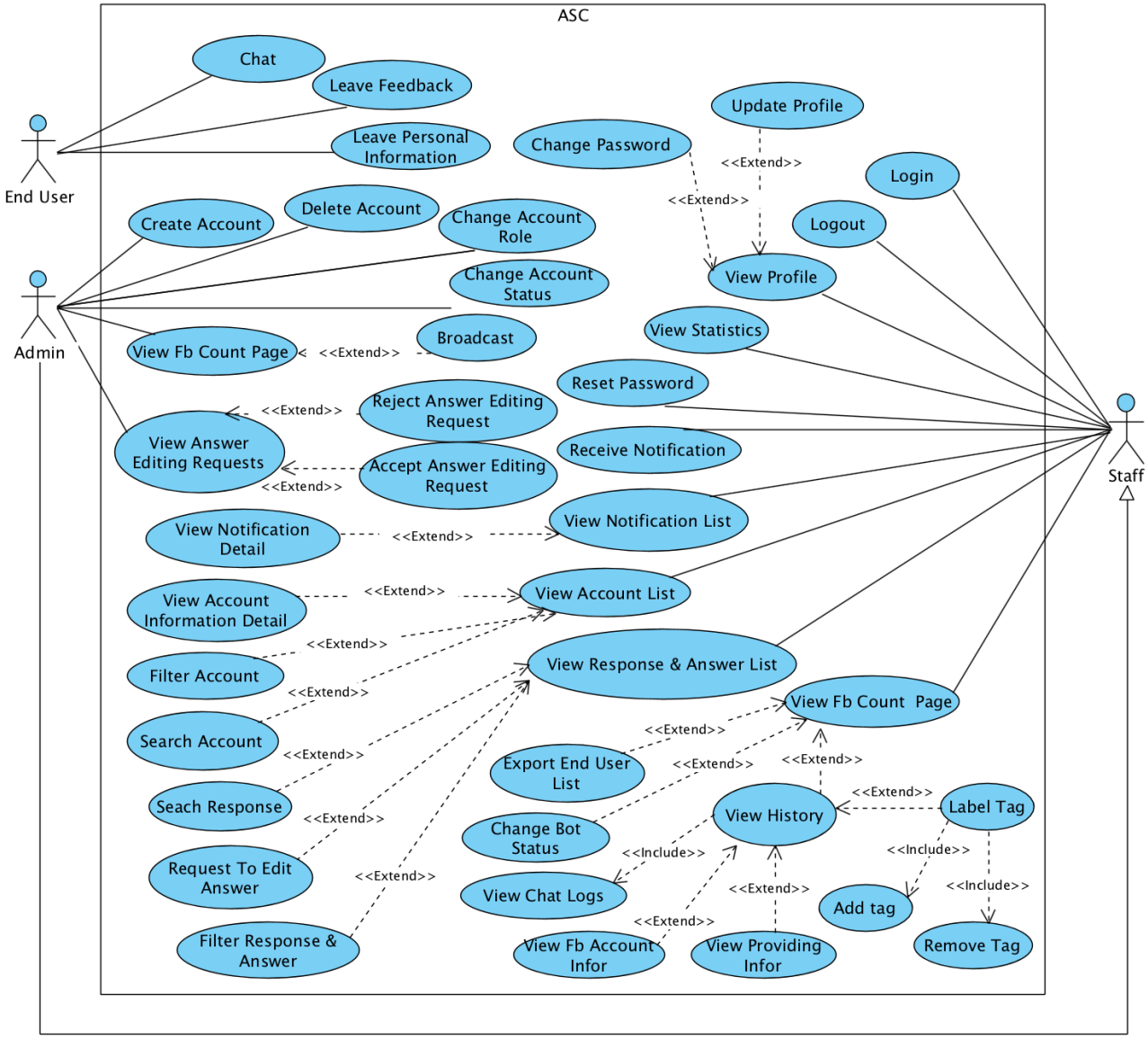


Figure 3-1:Use case diagram of ASC system

3.2.2 Business Rules

No	Description
B1	Username must not be empty.
B2	Username must be unique.

B3	User's full name must not be empty.
B4	User's address must not be empty.
B5	The email must not be empty.
B6	Email address must be valid.
B7	Each email address may be used for only one account.
B8	Re-enter password must match with password.
B9	Password must not be empty.
B10	Password reset token expires after 24 hours.
B11	The password length must not be less than 6.
B12	Deactivation must note reasons.
B13	Change deactivated status to active status must delete note and deactivated time.
B14	Phone number must be valid.
B15	Phone number must be numeric string only.
B16	Exported files must be Microsoft Excel files with valid format.
B17	The admin must fill out username and email field.
B18	Every user accessing dashboard has a role: staff or admin.
B19	The administrator is able to: active/ deactivate account, change account role, change request state.
B20	Staff can request to edit chatbot answer.
B21	An answer editing request has 3 states: "PENDING", "REJECTED", "ACCEPTED"
B22	Answer editing request form must have: old answer, new answer, created date, creator.
B23	Only "PENDING" editing answer can be accepted by the admin, after that editing answer's state is "ACCEPTED".
B24	Only "PENDING" editing answer can be rejected by the admin, after that editing answer's state is "REJECTED".
B25	Answer editing request status only can be changed from "PENDING" to "ACCEPTED" or "REJECTED".
B26	To review an answer request, the staff must have at least an editing answer.
B27	The chatbot always has a state: "ON" or "OFF".
B28	The chatbot will not automatically reply in "OFF" state.
B29	The chatbot will automatically reply in "ON" state.
B30	When admin deactivates account. System logs that account out of ASC system.
B31	Rating must be 1 in 5 states: "EXCELLENT", "VERY GOOD", "GOOD", "BAD", "VERY BAD".
B32	When no reply after 10 minutes, the chatbot auto finishes the conversation.

Table 3-1: Business rules

3.2.3 Use Cases

Actor	Description
End User	Someone who uses the application to engage in a conversation.
Staff	Everyone who has a ASC Account such as FU staffs.
Admin	Administrator is a staff, has the highest permission level and is responsible for managing administrative tasks.

Table 3-2: Actor description

ID	Actor	Name	Description
UC-1.0	End User	Chat	End User asks a question and chatbot replies automatically.
UC-2.0	End User	Leave Feedback	End User is able to leave feedback, which is comprised of a text message and a rating.
UC-3.0	End User	Leave Personal Information	End User leaves personal information.
UC-4.0	Staff, Admin	Reset Password	Reset a forgotten password using email.
UC-5.0	Staff, Admin	Login	Login to dashboard in ASC account.
UC-6.0	Staff, Admin	View Statistics	View feedbacks, number of end user and related admission information statistics.
UC-7.0	Staff, Admin	View Profile	View personal profile.
UC-8.0	Staff, Admin	Update Profile	Update personal profile.
UC-9.0	Staff, Admin	Change Password	Change password of account.
UC-10.0	Staff, Admin	View Account List	View a list of accounts in the system.
UC-11.0	Staff, Admin	View Account Information Detail	View detailed information regarding specific account.
UC-12.0	Staff, Admin	Search Account	Search for account by username or full name.
UC-13.0	Staff, Admin	Filter Account	Filter account by role or status.
UC-14.0	Staff, Admin	View Response & Answer List	View chatbot responses regarding answers.
UC-15.0	Staff, Admin	Filter Response & Answer	Filter chatbot response and answer by intent.
UC-16.0	Staff, Admin	Search Response	Search for chatbot response.

UC-17.0	Staff, Admin	Request to Edit Answer	Request to edit a chatbot answer.
UC-18.0	Staff, Admin	Receive Notification	Notification of changes to events automatically sent by the system as a result of an edit.
UC-19.0	Staff, Admin	View Notification List	View a list of notifications.
UC-20.0	Staff, Admin	View Notification Detail	View some details of the notification.
UC-21.0	Staff, Admin	View Fb Count Page	View all End User interacting with the chatbot.
UC-22.0	Staff, Admin	View Chat Logs	View a conversation history between End User and the chatbot.
UC-23.0	Staff, Admin	View Fb Account Infor	View information of Fb Account and his/her satisfaction ratings and reviews.
UC-24.0	Staff, Admin	View Providing Infor	View providing information of End User.
UC-25.0	Staff, Admin	Add Tag	Add a tag to categorize and filter End User.
UC-26.0	Staff, Admin	Remove Tag	Remove a tag from End User.
UC-27.0	Staff, Admin	Export End User List	Export a list of End Users into Excel file.
UC-28.0	Staff, Admin	Change Bot Status	Turn chatbot on or off.
UC-29.0	Staff, Admin	Logout	Log out of the ASC dashboard.
UC-30.0	Admin	View Answer Editing Requests	View a list of answers edited by staff.
UC-31.0	Admin	Accept Answer Editing Request	Accept an answer editing request with status Pending.
UC-32.0	Admin	Reject Answer Editing Request	Reject an answer editing request with status Pending.
UC-33.0	Admin	Create Account	Create a new ASC account.
UC-34.0	Admin	Delete Account	Delete an existing account.
UC-35.0	Admin	Change Account Role	Change role of account: Admin or Staff.
UC-36.0	Admin	Change Account Status	Change status to be a deactivated account or active account.

UC-37.0	Admin	Broadcast	Send a single message to all of the chatbot end users.

Table 3-3: Use Case list

3.2.3.1 End User

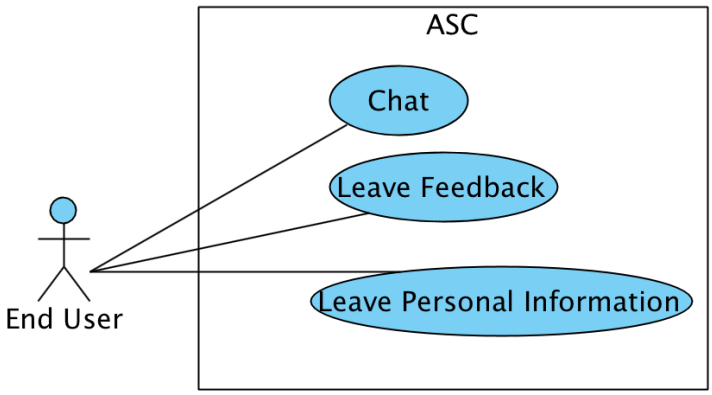


Figure 3-2:Use case diagram of End User actor

3.2.3.1.1 Chat

3.2.3.1.1.1 Training the Chatbot

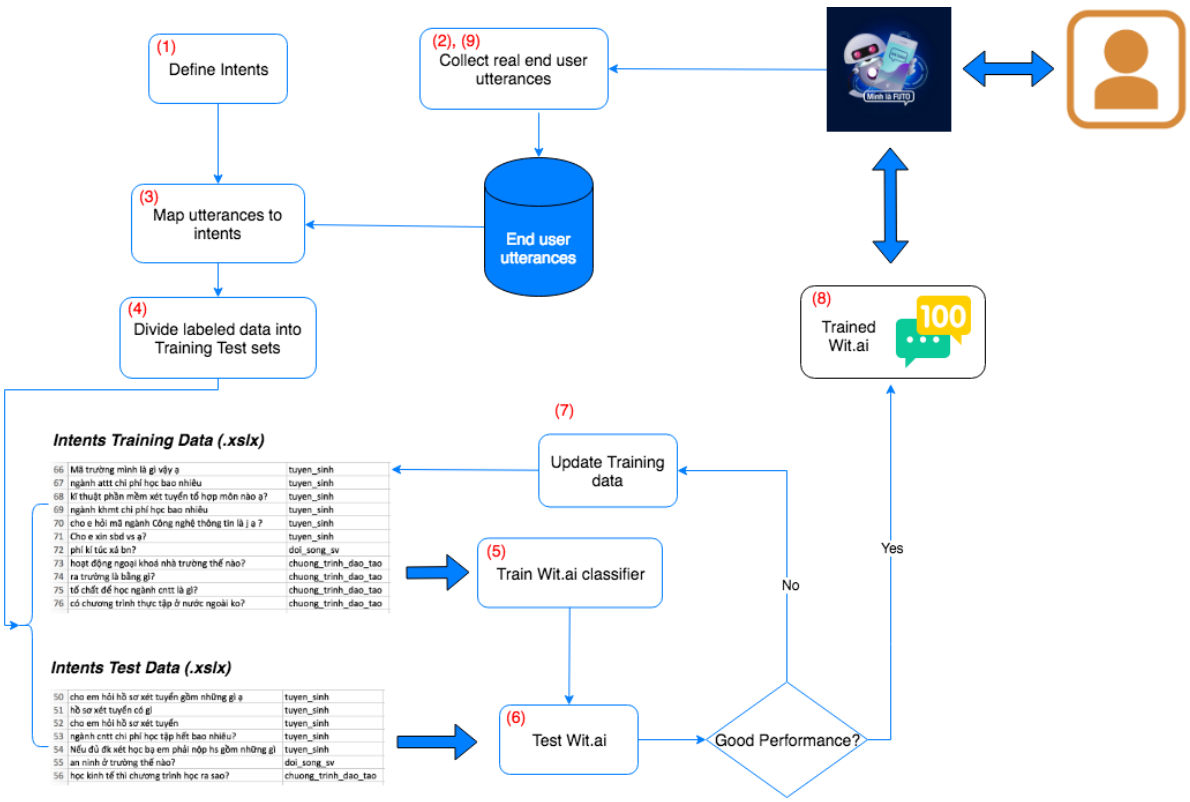


Figure 3-3: Training the Chatbot activity diagram

Explanation:

(1): Define the intents chatbot to extract from natural language utterances.

(2): Collect real end user utterances that the chatbot to map to intents.

(3): Assign the utterances collected to the different intents.

(4): Randomly divide the utterances into two sets, a training set and a test set. A 70% training and 30% test is a typical split.

(5): Train the chatbot using the training set.

(6): Once training is complete, run the test set against the trained classifier and collect performance metrics such as accuracy, precision, and recall.

(7): Perform Error Analysis: review the results in previous step to understand why the classifier missed certain utterances. Update training data accordingly and go back to step (5)

(8): If satisfied with the results produced by the trained system, the system is now ready to be released.

(9): When the chatbot is in use, continue to collect end user utterances. Then map results collected to new training/test data. Go back to step (4) and iterate.

Detail Specification:

I. Define intent

An intent represents the purpose of an end user's input.

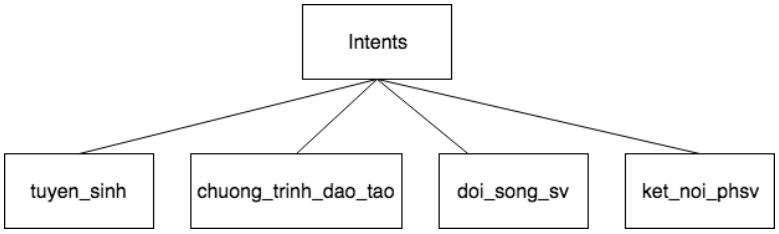
To define intent , we go to my customer FPT Admission Office to see what they think are the top reasons end users are contacting them. The typical support material that the reps use we will get closer to understanding the end user intent is website of FU[1]. It is largely divided into "Question" and "Answer". Data named "Answer" are normal article data from the website with section, title, body attributes. Data named "Question" are the end user's comments.

Figure 3-4: Answer data

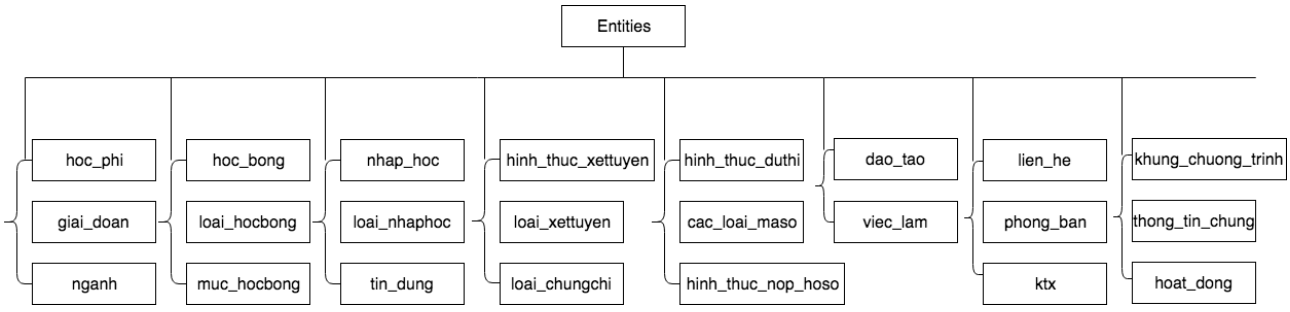
[1] FU website: http://daihoc.fpt.edu.vn/

Figure 3-5: Question Data

Since the raw data had too many topics such as admission procedures, tuition fees, scholarships, and other classes making the end user is interested in so much, we grouped them into larger classes, also called intents.



Besides the intent, we create a lot of entities. An entity represents a term or object in the end user's input that provides clarification or specific context for a particular intent.



II. Collect training data

To get all real comments of potential end user from the "dai.hoc.fpt.edu.vn" website for training , we must create a crawler tool to accquire data.

```
Input: "http:daihoc.fpt.edu.vn/tuyen-sinh/"
Output: data.json
```

A sample of raw data is provided below:

```
E sinh năm 1996 e muốn xét tuyển ngành ngôn ngữ anh.. Em có cơ hội không ạ..tư vấn giúp em với..
E ở tuyên quang nếu muốn đk xét tuyển thì thi ở hà nội hay tại tuyên quang luôn vậy ạ ?
Điều kiện để xét tuyển là gì ạ ?
em muốn nhập học FPT ngành công nghệ thông tin thì phải đăng kí như thế nào ạ? cụ thể học phí cùng học bổng thig như thế nà
Cho em hỏi nếu muốn đăng kí FPT thì chỉ cần đăng kí vào phần nguyện vọng thôi ạ có phải đăng kí ở trên ko, với lại mã trườn
Em đã nộp hồ sơ thi đh rồi ạ, nhưng bây h em muốn đăng kí FPT có còn được ko ạ
Mã trường mình là gì vậy ạ
Mã trường là j ạ
Cho em hỏi , xét học bạ . về Thiết Kế đồ họa là xét tổ hợp môn nào ạ . có cần thi năng khiếu Vẽ không ?
trường có thể cho e biết chi tiết hơn về cách thi và xét tuyển của trường ko ạ? e xin cảm ơn ạ
Cho em hỏi năm nay trường có nghành ngôn ngữ nhật không ?nếu có cho em xin mã nghành mã trường
Thay co cho em hoi: khi nao thi het hang dang ki xet tuyen hoc ba, khi co diem thi THPT QG roi co con dang ki xet tuyen hoc
```

III. Clean and assign data collected to the different intent.

To get high quality labeled training data, we must clean the misunderstanding and ambigious utterances. Only keeping the utterance having an intent and relating our model domain.

To label training data, we attach each of utterance into diffrent intent and entity.

```
doi_tuong
1.đối tượng tuyển sinh của trường FPT là gì?
2.Trường FPT tuyển sinh đối tượng như thế nào?
3.điều kiện cho những đối tượng muốn tham gia tuyển sinh là gì?


cac_loai_maso-mã trường
1. Mã trường mình là gì vậy ạ
2. Mã trường là j ạ
3. Cho em xin mã trường với ạ
4. cho em hỏi mã trường FPT là j ạ

```

3.2.3.1.1.2 Chat process

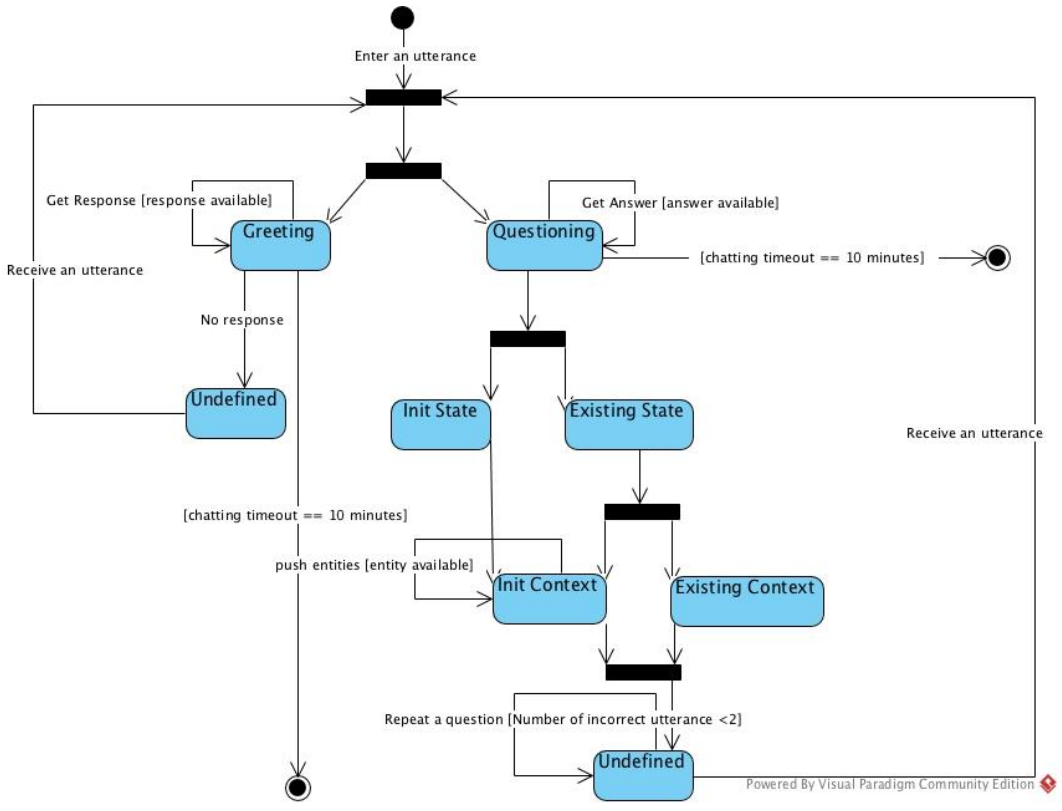


Figure 3-6: Chat state machine diagram

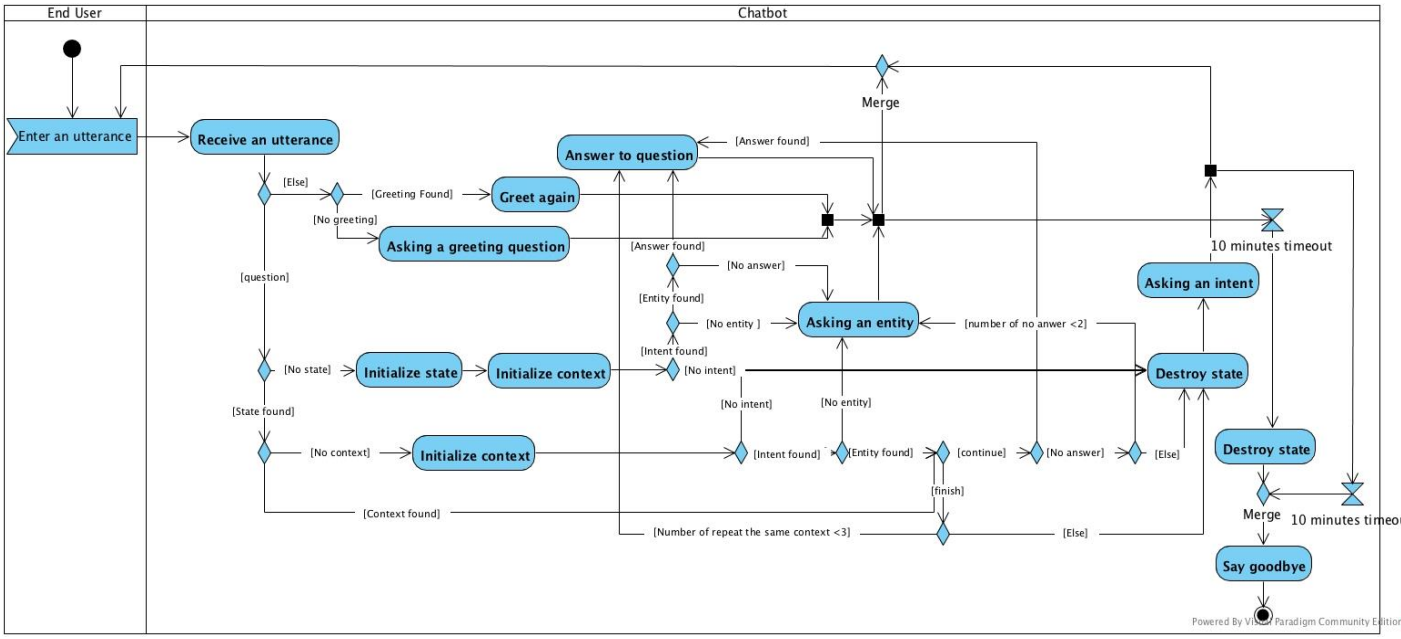


Figure 3-7: Chat activity diagram

Use Case Specification

UC ID and Name:	**UC-1.0 Chat**		
Created by:	**QuynhDTT**	Date Created:	**24/05/2018**
Primary Actor:	**End User**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow End User to send a question and receive the answer to their question immediately.		
Preconditions:	**PRE-1.1.** The end user starts the Facebook Messenger application.		
Post conditions:	**POST-1.1.** The answer to the end user's question is sent.		
Normal Flow:	**1.0 Chat** 1. End User enters a question then clicks on the send button. 2. System generates an answer displayed in the chat window to reply his/her question.		
Alternative Flows:	**1.0 End User violates some business rule in step 1** 1. System sends a goodbye message to finish the conversation.		
Exceptions:	**1.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		

Frequency of Use:	High
Business Rules:	**B32**
Other Information:	N/A
Assumptions:	N/A

3.2.3.1.2 Leave Feedback

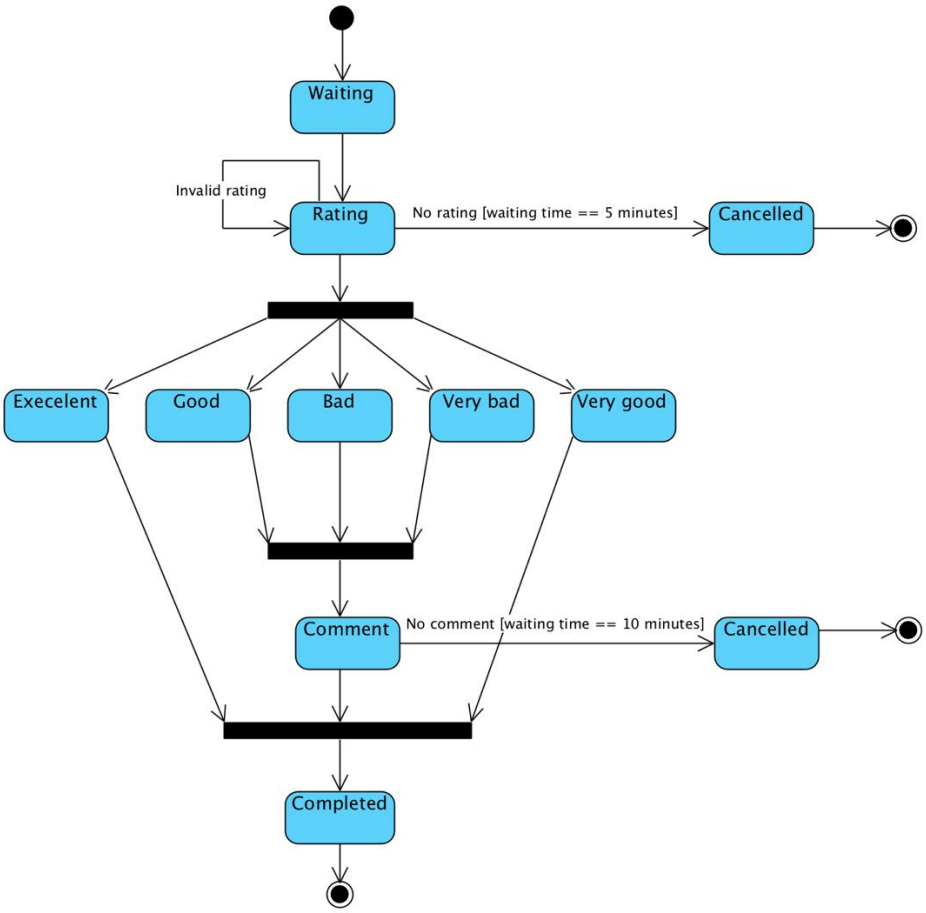


Figure 3-8: Feedback state machine diagram

Use Case Specification

UC ID and Name:	**UC-2.0 Leave Feedback**		
Created by:	**QuynhDTT**	Date Created:	**24/05/2018**
Primary Actor:	**End User**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow End User to rate and review for chatbot in the end of conversation.		

Preconditions:	**PRE-2.1.** End User has already chat. **PRE-2.2.** End User has not sent any message for 10 minutes.
Post conditions:	**POST-2.1.** System saves new feedback to database.
Normal Flow:	**2.0 Leave Feedback** 1. After 10 minutes finishing conversation, system sends a message with 5 rating quick replies. 2. End User chooses one of them. 3. System asks for reasons that they chose it. 4. End User gives their review for chatbot. 5. System saves his/her feedback to database.
Alternative Flows:	**2.0 End User violates some business rule in step 2** 1. System sends the help message" (Fb frirst name) vui lòng chọn 1 trong số các lựa chọn bên dưới giúp Futo." **2.0 End User violates some business rule in step 4** 2. System sends a goodbye message to finish the conversation.
Exceptions:	**2.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	High
Frequency of Use:	Medium
Business Rules:	**B31, B32**
Other Information:	N/A
Assumptions:	N/A

3.2.3.1.3 Leave Personal Information

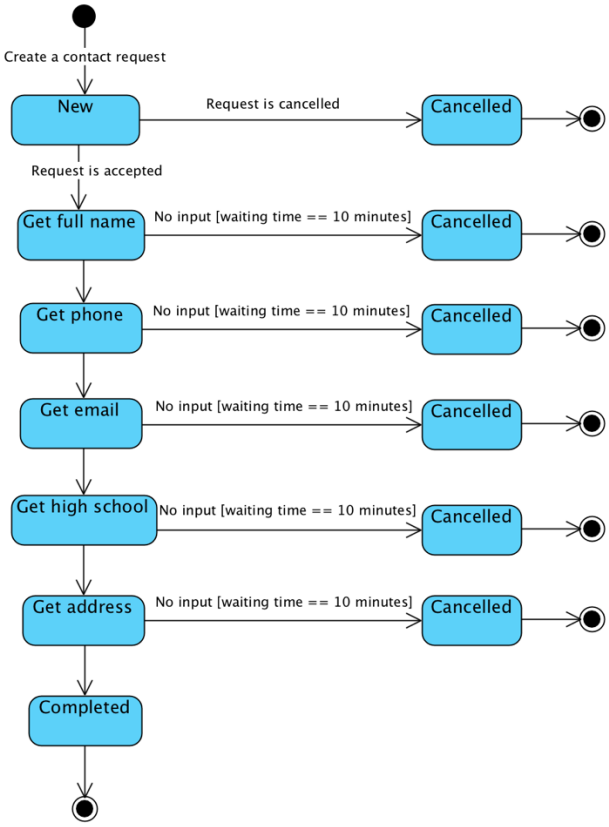


Figure 3-9: Leave Personal Information state machine diagram

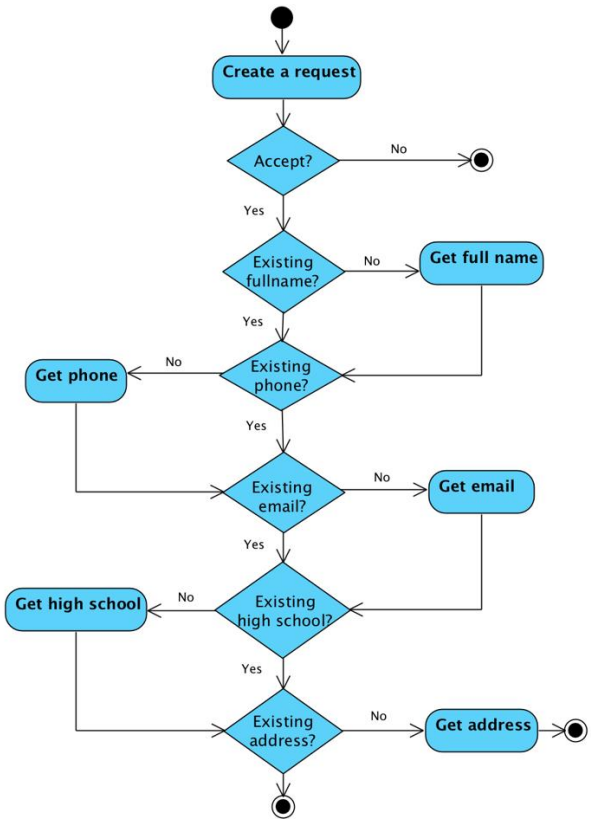


Figure 3-10: Leave Personal Information activity diagram

Use Case Specification

UC ID and Name:	**UC-3.0 Leave Personal Information**		
Created by:	**QuynhDTT**	Date Created:	**24/05/2018**
Primary Actor:	**End User**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow End User to leave personal information to contact.		
Preconditions:	**PRE-3.1.** End User sends a message to want to talk to some real person.		
Post conditions:	**POST-3.1.** System saves end user's personal information to database.		
Normal Flow:	**3.0 Leave Personal Information** 1. System asks some questions about getting end user's personal information. 2. End User replies to each of them. 3. System saves information that end user provides.		
Alternative Flows:	**3.0 End User violates some business rule in step 2** 1. System sends a goodbye message to finish the conversation.		
Exceptions:	**3.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	**B32**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2 Staff

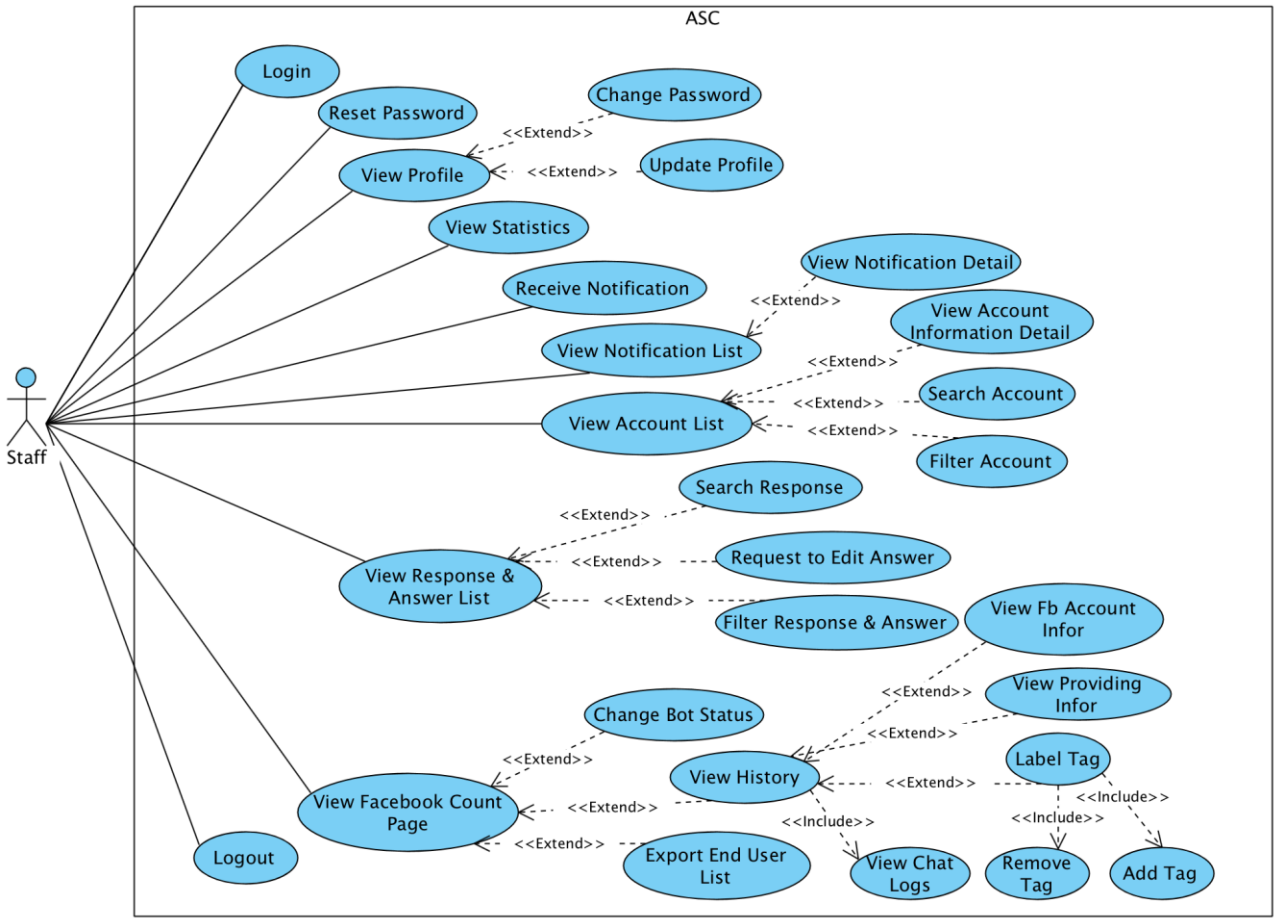


Figure 3-11: Use case diagram of Staff actor

3.2.3.2.1 Reset password

Use Case Specification

UC ID and Name:	**UC-4.0 Reset password**		
Created by:	**AnhDV**	Date Created:	**24/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to reset a forgotten password.		
Preconditions:	**PRE-4.1**. Staff account has been set up and username and password has been generated.		
Post conditions:	**POST-4.1**. New password is reset and sent to staff via his/her email.		

Normal Flow:	**4.0 Reset password** 1. Staff visits the dashboard page. 2. System displays Login page. 3. Staff clicks “Quên mật khẩu”. 4. System displays Forgot Password page. 5. Staff enters his/her email, then clicks “Xin cấp lại mật khẩu”. 6. System displays a message that asks the staff to check email and follow instruction. 7. Staff clicks on the link in email. 8. System displays Reset Password page. 9. Staff is required to input password twice, then clicks “Xác nhận”. 10. System displays message “Thay đổi mật khẩu thành công”. 11. System displays Login page.
Alternative Flows:	**4.0 User violates some business rule in step 5** 1. System displays help message popup. 2. Staff enters email address complying the business rules then clicks “Xin cấp lại mật khẩu”. 3. Go to step 6 in normal flow. **4.0 User violates some business rule in step 9** 1. System displays help message popup. 2. Staff enters password complying the business rules then clicks “Xác nhận”. 3. Go to step 10 in normal flow.
Exceptions:	**4.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	High
Frequency of Use:	Low
Business Rules:	**B5, B6, B8, B10, B11**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.2 Login

Use Case Specification

UC ID and Name:	**UC-5.0 Login**

Created by:	**TuanDM**	Date Created:	**24/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff who has ASC account to login.		
Preconditions:	**PRE-5.1**. Staff account has been set up and username and password has been generated.		
Post conditions:	**POST-5.1**. The staff is logged into ASC successfully. **POST-5.2**. System displays Dashboard Home Page.		
Normal Flow:	**5.0 Login** 1. Staff visits the website. 2. Staff fills valid username and password then clicks "Đăng nhập". 3. System displays Dashboard Home page.		
Alternative Flows:	**5.0 Staff violates some business rule in step 2** 1. System displays help message popup. 2. Staff enters username and password complying the business rules then clicks "Đăng nhập". 3. Go to step 3 in normal flow.		
Exceptions:	**5.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	High		
Business Rules:	**B1, B9**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.3 View Statistics

Use Case Specification

UC ID and Name:	**UC-6.0 View Statistics**		
Created by:	**AnhDV**	Date Created:	**24/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	

Trigger:	N/A
Description:	Allow Staff to view statistics about majors, feedbacks, number of end users.
Preconditions:	**PRE-6.1.** Staff can access the system. **PRE-6.2.** Staff logins successfully.
Post conditions:	**POST-6.1.** System displays statistics.
Normal Flow:	**6.0 View Statistics** 1. From index page, system displays statistics using 4 cards and 2 charts: 2. Card about the number of all end users asking. 3. The number of new end users asking today. 4. The number of old end users asking today. 5. The number of end users proving personal information. 6. Pie chart about FU academic majors. 7. Bar chart about the number of end user's feedback.
Alternative Flows:	N/A
Exceptions:	**6.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	High
Frequency of Use:	Medium
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.4 View Profile

Use Case Specification

UC ID and Name:	**UC-7.0 View Profile**		
Created by:	**DuNV**	Date Created:	**24/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		

Description:	Allow all staffs to view his/her personal profile.
Preconditions:	**PRE-7.1.** Staff can access the system. **PRE-7.2.** Staff logins successfully.
Post conditions:	**POST-7.1.** System displays the staff profile page.
Normal Flow:	**7.0 View Profile** 1. From the website header, staff clicks on avatar icon in the top right corner. 2. Staff clicks "Trang cá nhân". 3. System displays the staff profile page with following tab: o Thông tin tài khoản o Tên tài khoản o Vai trò o Thông tin liên hệ o Họ và tên o Email o Số điện thoại o Địa chỉ
Alternative Flows:	N/A
Exceptions:	**7.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.5 Update Profile

Use Case Specification

UC ID and Name:	**UC-8.0 Update Profile**		
Created by:	**QuynhDTT**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		

Description:	Allow Staff to update his/her basic information after signing in.
Preconditions:	**PRE-8.1.** Staff can access the system. **PRE-8.2.** Staff logins successfully.
Post conditions:	**POST-8.1.** New staff profile is saved into database. **POST-8.2.** New updated profile is displayed in profile page.
Normal Flow:	**8.0 Update Profile** 1. From the website header, staff clicks on avatar icon in the top right corner. 2. Staff clicks "Trang cá nhân". 3. System displays the Staff profile. 4. Staff can edit information about fullname, address, phone, email. 5. Staff clicks "Lưu thông tin cá nhân" to save the changed inforamtion. 6. System saves changed into database then displays updated profile information.
Alternative Flows:	**8.0 Staff violates some business rule in step 4** 1. System displays help message under each input area. 2. Staff enters fullname, email, phone and address complying the business rules then clicks "Lưu thông tin cá nhân". 3. Go to step 6 in normal flow.
Exceptions:	**8.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**B3, B4, B5, B6, B7, B14, B15**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.6 Change Password

Use Case Specification

UC ID and Name:	**UC-9.0 Change password**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**

Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to change his/her account password.		
Preconditions:	**PRE-9.1.** Staff can access the system. **PRE-9.2.** Staff logins successfully.		
Post conditions:	**POST-9.1.** New password is saved into database. **POST-9.2.** Staff logins successfully with new password.		
Normal Flow:	**9.0 Change Password** 1. From the website header, staff clicks on avatar icon in the top right corner. 2. Staff clicks "Thay đổi mật khẩu" 3. System displays the Change Password page. 4. Staff enters current password, new password and re-enter password in the corresponding text boxes. 5. Staff clicks "Đổi mật khẩu". **6.** System saves new data into database.		
Alternative Flows:	N/A		
Exceptions:	**9.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	Low		
Business Rules:	**B8, B9, B11**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.7 View Account List

Use Case Specification

UC ID and Name:	**UC-10.0 View Account List**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		

Description:	Allow Staff to view the list of accounts in the system.
Preconditions:	**PRE-10.1.** Staff can access the system. **PRE-10.2.** Staff logins successfully. **PRE-10.3.** Account data has been added to the system by the admin.
Post conditions:	System displays the account list screen with necessary elements.
Normal Flow:	**10.0 View Account List** 1. Staff clicks on "Tất cả tài khoản" on menu in the left side bar of the screen. 2. System displays list of all accounts.
Alternative Flows:	N/A
Exceptions:	**10.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.8 View Account Information Detail

Use Case Specification

UC ID and Name:	**UC-11.0 View Account Information Detail**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to view detail information of an account.		
Preconditions:	**PRE-11.1.** Staff can access the system. **PRE-11.2.** Staff logins successfully. **PRE-11.3.** Account data has been added to the system by the admin.		
Post conditions:	System displays a model with chosen account information.		

Normal Flow:	**11.0 View Account Information** 1. Staff clicks on "Tất cả tài khoản" on the left side of the screen. 2. System displays list of all accounts. 3. Staff clicks "Xem" button on each row in table. 4. System displays account information.
Alternative Flows:	N/A
Exceptions:	**11.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.9 Search Account

Use Case Specification

3.2.3.2.9.1 Search Account by Username

UC ID and Name:	**UC-12.0 Search Account by Username**		
Created by:	**TuanDM**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to search account(s) belong to username.		
Preconditions:	**PRE-12.1.** Staff can access the system. **PRE-12.2.** Staff logins successfully. **PRE-12.3.** Account data has been added to the system by the admin.		
Post conditions:	N/A		
Normal Flow:	**12.0 Search Account by Username** 1. Staff clicks on "Tất cả tài khoản" on the left side of the screen. 2. System displays table of all accounts. 3. In the "Tài khoản" text box, staff enters text.		

	4. System displays account(s) that related to search keyword.
Alternative Flows:	N/A
Exceptions:	**12.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Medium
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.9.2 Search Account by Fullname

UC ID and Name:	**UC-12.0 Search Account by Fullname**		
Created by:	**TuanDM**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to search account by fullname.		
Preconditions:	**PRE-12.1.** Staff can access the system. **PRE-12.2.** Staff logins successfully. **PRE-12.3.** Account data has been added to the system by the admin.		
Post conditions:	N/A		
Normal Flow:	**12.0 Search Account by Fullname** 1. Staff clicks on "Tất cả tài khoản" on the left side of the screen. 2. System displays account list. 3. In the "Họ và tên" text box, staff enters text. 4. System displays account list that related to search keyword.		
Alternative Flows:	N/A		
Exceptions:	**12.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		

Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.10 Filter Account

Use Case Specification

3.2.3.2.10.1 Filter Account by Role

UC ID and Name:	**UC-13.0 Filter Account by Role**		
Created by:	**TuanDM**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to filter accounts by role.		
Preconditions:	**PRE-13.1.** Staff can access the system. **PRE-13.2.** Staff logins successfully. **PRE-13.3.** Account data has been added to the system by the admin.		
Post conditions:	N/A		
Normal Flow:	**13.0 Filter Account by Role** 1. Staff clicks role dropdown button 2. System displays role list in dropdown list. 3. Staff selects a specific role. 4. System hided role list and displays selected role and accounts list related to selected role.		
Alternative Flows:	N/A		
Exceptions:	**13.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**N/A**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.10.2 Filter Account by Status

UC ID and Name:	**UC-13.0 Filter Account by Status**		
Created by:	**TuanDM**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to filter account by status.		
Preconditions:	**PRE-13.1.** Staff can access the system. **PRE-13.2.** Staff logins successfully. **PRE-13.3.** Account data has been added to the system by the admin.		
Post conditions:	N/A		
Normal Flow:	**13.0 Filter Account by Status** 1. Staff clicks on status dropdown button. 2. System displays status list in dropdown list. 3. Staff selects a specific status. 4. System hided status list and displays selected state and accounts list related to selected status.		
Alternative Flows:	N/A		
Exceptions:	**13.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**N/A**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.11 View Response & Answer List

Use Case Specification

UC ID and Name:	**UC-14.0 View Response & Answer List**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	

Trigger:	N/A
Description:	Allow Staff to view list of responses regarding answers.
Preconditions:	**PRE-14.1.** Staff can access the system. **PRE-14.2.** Staff logins successfully.
Post conditions:	System displays response and answer.
Normal Flow:	**14.0 View Response & Answer List** 1. Staff clicks on "Quản lí dữ liệu của Chatbot" on the left side of the screen. 2. System displays list of responses regarding answers.
Alternative Flows:	N/A
Exceptions:	**14.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.12 Filter Response & Answer

Use Case Specification

UC ID and Name:	**UC-15.0 Filter Response & Answer**		
Created by:	**QuynhDTT**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to filter response and answer.		
Preconditions:	**PRE-15.1.** Staff can access the system. **PRE-15.2.** Staff logins successfully. **PRE-15.3.** Response and Answer data has been added to the system by the admin.		
Post conditions:	**POST-15.1.** System displays filtered response and answer.		

Normal Flow:	**15.0 Filter Response & Answer.** 1. Staff clicks on intent dropdown butoon. 2. System displays intent list in dropdown list. 3. Staff selects a specific intent. 4. System hided intent list and displays selected intent and responses regarding answers list related to selected intent.
Alternative Flows:	N/A
Exceptions:	**15.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.13 Search Response

Use Case Specification

UC ID and Name:	**UC-16.0 Search Response**		
Created by:	**QuynhDTT**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to search chatbot response(s).		
Preconditions:	**PRE-16.1.** Staff can access the system. **PRE-16.2.** Staff logins successfully. **PRE-16.3.** Response data has been added to the system by the admin.		
Post conditions:	**POST-16.1.** System displays searched responses regarding answers.		
Normal Flow:	**16.0 Search Response** 1. Staff clicks on “Quản lí dữ liệu của Chatbot” on the left side of the screen. 2. System displays responses regarding answers list. 3. In the response textbox, Staff enters text.		

	4. System displays response regarding answers list related to search keyword.
Alternative Flows:	N/A
Exceptions:	**12.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.14 Request to Edit Answer

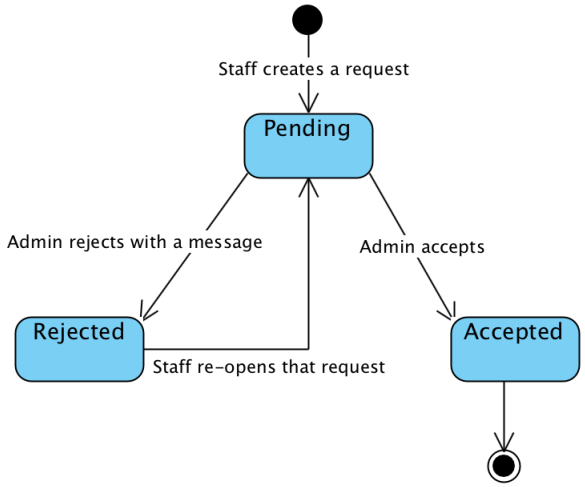


Figure 3-12: Answer Editing Request state machine diagram

Use Case Specification

UC ID and Name:	**UC-17.0 Request to Edit Answer**		
Created by:	**QuynhDTT**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to request to edit an answer of chatbot.		
Preconditions:	**PRE-17.1.** Staff can access the system. **PRE-17.2.** Staff logins successfully.		

	PRE-17.3. Answer data has been added to the system by the admin.
Post conditions:	**POST-17.1.** An answer editing request is successfully submitted and noticed to admin.
Normal Flow:	**17.0 Edit Answer** 1. Staff clicks "Quản lí dữ liệu của Chatbot" on the left side of the screen. 2. System displays response regarding answers list. 3. In each row of answer, Staff clicks on edit button. 4. Staff enters new answer. 5. Staff clicks "Lưu". 6. System saves new answer editing request into database with pending status.
Alternative Flows:	N/A
Exceptions:	**17.0-E1 – Cannot communicate with server** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**B20, B21, B22**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.15 Receive notification

Use Case Specification

UC ID and Name:	**UC-18.0 Receive notification**		
Created by:	**TuanDM**	Date Created:	**08/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	An chatbot editing answer, end user's request or changing account detail from admin, which is related to Staff, is created.		
Description:	Allow Staff to receive notification.		
Preconditions:	**PRE-18.1.** Staff can access the system. **PRE-18.2.** Staff logins successfully.		

	PRE-18.3. Admin deactivates account successfully. **PRE-18.4.** Admin changes role of account. **PRE-18.5.** Staff request to edit chatbot answer. **PRE-18.6.** Admin rejects answer editing request. **PRE-18.7.** Admin accepts answer editing request. **PRE-18.7.** End User requests to contact.
Post conditions:	**POST-18.1.** Staff receives notification.
Normal Flow:	**18.0 Account is deactivated.** 1. Admin deactivates staff's account. 2. Staff receives notification. **18.1 Role of account is changed.** 1. Admin changes staff's role. 2. Staff receives notification. **18.2 Staff request to edit Answer.** 1. Staff submits a request for editing answers. 2. Admin receives notification. **18.3 Answer is rejected.** 1. Admin rejects answer editing request. 2. Staff receives notification that your request has been rejected. **18.4 Answer is accepted.** 1. Admin accepts answer editing request. 2. Staff receives notification that your request has been accepted. **18.5 End User requests to contact.** 1. End User send a call back or chatting request with staff . 2. Admin receives notifications.
Alternative Flows:	N/A
Exceptions:	**18.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	High
Frequency of Use:	High
Business Rules:	**N/A**
Other Information:	N/A

Assumptions:	N/A

3.2.3.2.16 View notification list

Use Case Specification

UC ID and Name:	**UC-19.0 View notification list**		
Created by:	**TuanDM**	Date Created:	**08/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to view notifications about account status, answer editing request status, end user's contact request.		
Preconditions:	**PRE-19.1.** Staff can access the system. **PRE-19.2.** Staff logins successfully. **PRE-19.3.** Staff has an action history.		
Post conditions:	**POST-19.1.** System resets notification count to zero.		
Normal Flow:	**19.0 View notification list** 1. On top right of navigation bar, Staff clicks on the notification icon. 2. System displays dropdown menu to show all notifications. 3. Staff scrolls down to view more notifications. 4. System changes status of all notifications to read.		
Alternative Flows:	N/A		
Exceptions:	**19.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	**N/A**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.17 View notification detail

Use Case Specification

UC ID and Name:	**UC-20.0 View notification detail**		
Created by:	**TuanDM**	Date Created:	**08/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	Mark selected notification as read.		
Description:	Allow Staff to view specific content to the notification.		
Preconditions:	**PRE-20.1.** Staff can access the system. **PRE-20.2.** Staff logins successfully. **PRE-20.3.** Staff has notification(s).		
Post conditions:	**POST-20.1.** Redirect to specific content correspond to each notification.		
Normal Flow:	**20.0 View notification detail** 1. Staff clicks on the notification icon on top right of navigation bar. 2. Staff clicks on a notification in notification list.		
Alternative Flows:	N/A		
Exceptions:	**20.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	**N/A**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.18 View Facebook Count Page

Use Case Specification

UC ID and Name:	**UC-21.0 View Facebook Count Page**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to view list of end user’s information.		
Preconditions:	**PRE-21.1.** Staff can access the system.		

	PRE-21.2. Staff logins successfully.
Post conditions:	System displays all end users interacting with the chatbot.
Normal Flow:	**21.0 View Facebook Count Page** 1. Staff clicks "Quản lí dữ liệu từ Facebook" on the left side of the screen. 2. System displays a table of all Facebook accounts having a conversation with the chatbot. Some information is displayed with each of account: 3. Fb name 4. Count of his/her messages 5. Current Bot status 6. Labeling tag
Alternative Flows:	N/A
Exceptions:	**21.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Medium
Frequency of Use:	Low
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.19 View Chat Logs

Use Case Specification

UC ID and Name:	**UC-22.0 View Chat Logs**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to view conversation history between specific end user and the chatbot.		
Preconditions:	**PRE-22.1.** Staff can access the system. **PRE-22.2.** Staff logged in.		
Post conditions:	System displays conversation history.		

Normal Flow:	**22.0 View Chat Logs** 1. Staff clicks “Lịch sử hội thoại” on each row of facebook account table. 2. System displays the history of all messages from end user. 3. Staff clicks “Xem thêm” to view more messages.
Alternative Flows:	N/A
Exceptions:	**22.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Low
Frequency of Use:	Medium
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.20 View Fb Account Infor

Use Case Specification

UC ID and Name:	**UC-23.0 View Fb Account Infor**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to view a specific facebook account information detail corresponding with his/her feedback(s).		
Preconditions:	**PRE-23.1.** Staff can access the system. **PRE-23.2.** Staff logged in.		
Post conditions:	System displays information about facebook account.		
Normal Flow:	**23.0 View Fb Account Infor** 1. Staff clicks “Lịch sử hội thoại” on each row of facebook account table. 2. System displays some tabs. 3. Clicks “Thông tin Facebook” tab 4. System displays facebook name, gender and his/her feedbacks.		

Alternative Flows:	N/A
Exceptions:	**23.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Low
Frequency of Use:	Medium
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.21 View Providing Infor

Use Case Specification

UC ID and Name:	**UC-24.0 View Providing Infor**		
Created by:	**AnhDV**	Date Created:	**26/05/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to view providing personal information of end user.		
Preconditions:	**PRE-24.1.** Staff can access the system. **PRE-24.2.** Staff logged in.		
Post conditions:	System displays the providing information of end user.		
Normal Flow:	**24.0 View Providing Infor** 1. Staff clicks “Lịch sử hội thoại” on each row of facebook account table. 2. System displays some tabs. 3. Staff clicks “Thông tin đã cung cấp” tab. 4. System displays end user’s providing information.		
Alternative Flows:	N/A		
Exceptions:	**24.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	High		

Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.22 Add Tag

Use Case Specification

UC ID and Name:	**UC-25.0 Add Tag**		
Created by:	**QuynhDTT**	Date Created:	**01/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to add item tags to end user.		
Preconditions:	**PRE-25.1.** Staff can access the system. **PRE-25.2.** Staff logged in.		
Post conditions:	**POST-25.1.** End User is tagged with some label.		
Normal Flow:	**25.0 Add Tag** 1. Staff clicks "Lịch sử hội thoại" on each row of facebook account table. 2. System displays some tabs. 3. Staff clicks "Gãn nhãn phân loại". 4. System displays all labelling tag with the end user. 5. Staff clicks inside add tags field and choose an existing tag from the dropdown list or type a new tag then choose recently generated tag from the dropdown list. 6. Staff clicks "Lưu" button. 7. System saves changed data into database.		
Alternative Flows:	N/A		
Exceptions:	**25.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**N/A**		

Other Information:	N/A
Assumptions:	N/A

3.2.3.2.23 Remove Tag

Use Case Specification

UC ID and Name:	**UC-26.0 Remove Tag**		
Created by:	**QuynhDTT**	Date Created:	**01/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to remove a tag from the end user.		
Preconditions:	**PRE-26.1.** Staff can access the system. **PRE-26.2.** Staff logged in.		
Post conditions:	**POST-26.1.** Tag is removed successfully.		
Normal Flow:	**26.0 Remove Tag** 1. Staff clicks "Lịch sử hội thoại" on each row of facebook account table. 2. System displays some tabs. 3. Staff clicks "Gán nhãn phân loại" tab. 4. System displays all tags labeling with the end user. 5. Staff selects a tag, then clicks "x". 6. Staff clicks "Lưu" button. 7. System saves changed to database.		
Alternative Flows:	N/A		
Exceptions:	**26.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**N/A**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.24 Export End User List

Use Case Specification

UC ID and Name:	**UC-27.0 Export End User List**		
Created by:	**QuynhDTT**	Date Created:	**01/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to export list of all End Users in the system.		
Preconditions:	**PRE-27.1.** Staff can access the system. **PRE-27.2.** Staff logged in.		
Post conditions:	**POST-27.1.** Excel file is generated.		
Normal Flow:	**27.0 Export End User List** 1. Staff clicks "Quản lí dữ liệu từ Facebook" on the left side of the screen. 2. Staff clicks "Tải xuống dữ liệu" button on top right of the screen. 3. System auto generates end user's information list into excel file.		
Alternative Flows:	N/A		
Exceptions:	**27.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	**B16**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.2.25 Change Bot Status

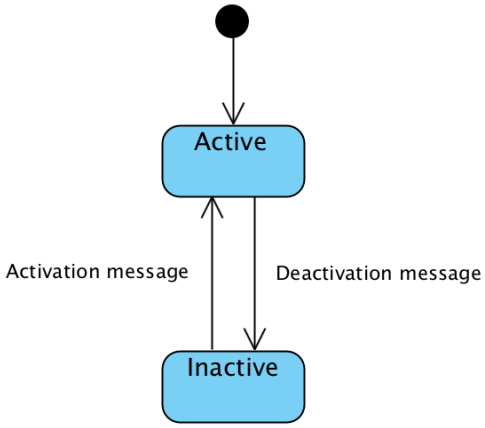


Figure 3-13: Bot state machine diagram

Use Case Specification

UC ID and Name:	**UC-28.0 Change Bot Status**		
Created by:	**AnhDV**	Date Created:	**01/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to turn on/off chatbot.		
Preconditions:	**PRE-28.1.** Staff can access the system. **PRE-28.1.** Staff logged in.		
Post conditions:	**POST-28.1.** System turns chatbot on/off corresponding with staff request.		
Normal Flow:	**28.0 Turn on chatbot** 1. Staff chooses a specific end user in facebook account table, then clicks "ON" toggle button. 2. Whenever the end user asking, chatbot will automatically reply. **28.0 Turn off chatbot** 1. Staff chooses a specific end user in facebook account table, then clicks "OFF" toggle button. 2. Whenever the end user asking, chatbot will not automatically reply.		
Alternative Flows:	N/A		
Exceptions:	**28.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		

Frequency of Use:	Medium
Business Rules:	**B27, B28, B29**
Other Information:	N/A
Assumptions:	N/A

3.2.3.2.26 Logout

Use Case Specification

UC ID and Name:	**UC-29.0 Logout**		
Created by:	**AnhDV**	Date Created:	**01/06/2018**
Primary Actor:	**Staff**	Secondary Actors:	
Trigger:	N/A		
Description:	Allow Staff to log out of ASC system.		
Preconditions:	**PRE-29.1.** Staf can access the system. **PRE-29.1.** Staff logged in.		
Post conditions:	**POST-29.1.** Staff logs out of system.		
Normal Flow:	**29.0 Logout** 1. On Navigation Bar, Staff clicks on profile avatar. 2. Staff clicks "Đăng xuất". 3. System logs Staff out of system and redirects to Login page.		
Alternative Flows:	N/A		
Exceptions:	**29.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	High		
Frequency of Use:	Medium		
Business Rules:	**N/A**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.3 Admin

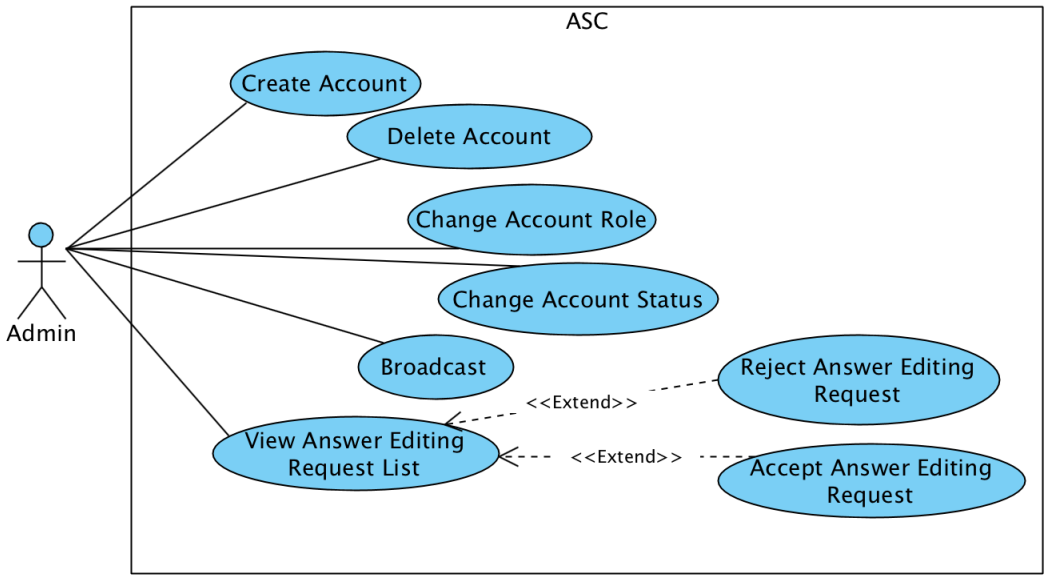


Figure 3-14: Use case diagram of Admin actor

3.2.3.3.1 View Answer Editing Requests

Use Case Specification

UC ID and Name:	**UC-30.0 View Answer Editing Requests**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**
Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Admin views a list of all chatbot answer editing requests.		
Preconditions:	**PRE-30.1**. Admin logins into ASC administration successfully.		
Post conditions:	**POST-30.1**. System displays a list of requests.		
Normal Flow:	**30.0 View Answer Editing Request List** 1. Admin clicks “Kiểm duyệt dữ liệu Chatbot” tab on the left side in the window screen. 2. System displays all requests as table.		
Alternative Flows:	N/A		
Exceptions:	**30.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		

Business Rules:	**B19, B26**
Other Information:	N/A
Assumptions:	N/A

3.2.3.3.2 Accept Answer Editing Request

Use Case Specification

UC ID and Name:	**UC-31.0 Accept Answer Editing Request**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**
Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Admin accepts an answer editing request.		
Preconditions:	**PRE-31.1**. Admin logins into ASC administration successfully. **PRE-31.2**. Answer editing request status is pending.		
Post conditions:	**POST-31.1**. Answer editing request status is changed to accepted. **POST-31.2**. System pushes new status to Websocket server to notify staff.		
Normal Flow:	**31.0 Accept an answer opening request** 1. Admin navigates to Request Management. 2. System displays a list of answer editing request. 3. Admin chooses an answer editing request from list. 4. Admin chooses option "Chấp nhận". 5. System changes chosen answer editing request to accepted. 6. System sends a notification about his/her answer editing request has been accepted.		
Alternative Flows:	N/A		
Exceptions:	**31.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Low		
Business Rules:	**B23, B25, B26**		
Other Information:	N/A		

Assumptions:	N/A

3.2.3.3.3 Reject Answer Editing Request

UC ID and Name:	**UC-31.0 Reject Answer Editing Request**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**
Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Admin rejects an answer editing request.		
Preconditions:	**PRE-31.1**. Admin logins into ASC administration successfully. **PRE-31.2**. Answer editing request status is pending.		
Post conditions:	**POST-31.1**. Answer editing request status is changed to rejected. **POST-31.2**. System pushes new status to Websocket server to notify staff.		
Normal Flow:	**31.0 Reject answer editing request** 1. Admin navigates to Request Management. 2. System displays a list of answer editing request. 3. Admin chooses an answer editing request from list. 4. Admin chooses option "Từ chối". 5. System displays admin message textbox. 6. Admin fills in message. 7. Admin clicks "Gửi phản hồi". 8. System changes chosen answer editing request to rejected. 9. System sends a notification about his/her answer editing request has been rejected.		
Alternative Flows:	N/A		
Exceptions:	**31.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Low		
Business Rules:	**B24, B25, B26**		
Other Information:	N/A		

Assumptions:	N/A

3.2.3.3.4 Create Account

Use Case Specification

UC ID and Name:	**UC-32.0 Create Account**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**
Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Admin creates a new account.		
Preconditions:	**PRE-32.1**. Admin logins into ASC administration successfully		
Post conditions:	**POST-32.1**. System displays Add Account page. **POST-32.2**. A new account is created.		
Normal Flow:	**32.0 Create Account** 1. Admin clicks "Thêm mới tài khoản" tab on the left side in the window screen. 2. Add Account page is displayed. 3. Admin enters username, email, fullname and role, then clicks "Tạo tài khoản". 4. System saves the new account with his/her information into database and sends password to his/her email.		
Alternative Flows:	**32.0 Admin violates some business rule in step 3** 1. System displays help message under each input area. 2. Admin enters usename, email and fullname complying business rules, then clicks "Tạo tài khoản". 3. Go to step 4 in normal flow.		
Exceptions:	**32.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**B1, B2, B5, B6, B7, B17**		
Other Information:	N/A		

Assumptions:	N/A

3.2.3.3.5 Delete Account

Use Case Specification

UC ID and Name:	**UC-33.0 Delete Account**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**
Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Account is deleted from the system.		
Preconditions:	**PRE-33.1**. Admin logins into ASC administration successfully.		
Post conditions:	**POST-33.1**. Selecting account is deleted.		
Normal Flow:	**33.0 Delete Account** 1. Admin clicks "Tất cả tài khoản" tab on the left side in the window screen. 2. Accounts is displayed as table. 3. Admin clicks on "Xoá" button on each row. 4. System deletes the selected account from system.		
Alternative Flows:	N/A		
Exceptions:	**33.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**N/A**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.3.6 Change Account Role

Use Case Specification

UC ID and Name:	**UC-34.0 Change Account Role**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**

Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Admin changes role of account.		
Preconditions:	**PRE-34.1**. Admin logins into ASC administration successfully.		
Post conditions:	**POST-34.1**. Role of his/her account is changed.		
Normal Flow:	**34.0 Change Account Role** 1. Admin clicks "Tất cả tài khoản" tab on the left side in the window screen. 2. Accounts is displayed as table. 3. Admin changes specific role from role dropdown list. 4. System saves the new updated role.		
Alternative Flows:	N/A		
Exceptions:	**34.0-E1 – Cannot communicate with server.** 1. ASC displays error message.		
Priority:	Medium		
Frequency of Use:	Medium		
Business Rules:	**B18, B19**		
Other Information:	N/A		
Assumptions:	N/A		

3.2.3.3.7 Change Account Status

Use Case Specification

UC ID and Name:	**UC-35.0 Change Account Status**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**
Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Admin changes status of account.		
Preconditions:	**PRE-35.1**. Admin logins into ASC administration successfully		
Post conditions:	**POST-35.1**. Account status is changed		

Normal Flow:	**35.0 Change Account Status** 1. Admin clicks “Tất cả tài khoản” tab on the left side in the window screen. 2. Accounts is displayed as table. 3. Admin select a specific user click “Hoạt động” | “Ngừng” depends on account current status. 4. Admin fill reason that causes admin action. 5. Status of account is updated.
Alternative Flows:	N/A
Exceptions:	**35.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	Low
Frequency of Use:	Low
Business Rules:	**B12, B13, B19, B30**
Other Information:	N/A
Assumptions:	N/A

3.2.3.3.8 Broadcast

Use Case Specification

UC ID and Name:	**UC-36.0 Broadcast**		
Created by:	**QuynhDTT**	Date Created:	**30/05/2018**
Primary Actor:	Admin	Secondary Actors:	
Trigger:	N/A		
Description:	Admin sends information to all end user in the system.		
Preconditions:	**PRE-36.1**. Admin logins into ASC administration successfully.		
Post conditions:	**POST-36.1**. A single message is sent to all end users in the system.		
Normal Flow:	**36.0 Broadcast** 1. Admin navigates to Facebook Data Management. 2. System displays broadcast form. 3. Admin inputs broadcast content, selects appreciate label, then clicks “Gửi”.		

	4. System sends a single message to all chosen end users.
Alternative Flows:	N/A
Exceptions:	**36.0-E1 – Cannot communicate with server.** 1. ASC displays error message.
Priority:	High
Frequency of Use:	High
Business Rules:	**N/A**
Other Information:	N/A
Assumptions:	N/A

3.3 Non-functional Requirement

3.3.1 Security

- ✓ Token-Based Authentication, relies on a signed token that is sent to the server on each request.
- ✓ Apply SSL encryption to restrict user access and prevent man-in-the-middle attacking.
- ✓ All passwords are encrypted by MD5 algorithm.
- ✓ The security matrix is as the following table:

Function	End User	Staff	Admin
Chat	✓		
Leave Feedback	✓		
Leave Personal Information	✓		
Reset Password		✓	✓
Login		✓	✓
View Statistics		✓	✓
View Profile		✓	✓
Update Profile		✓	✓
Change Password		✓	✓
View Account List		✓	✓
Search Account		✓	✓
View Account Information Detail		✓	✓

Filter Account		✓	✓
View Response & Answer List		✓	✓
Filter Response & Answer		✓	✓
Search Response		✓	✓
Request to Edit Answer		✓	✓
Receive Notification		✓	✓
View Notification List		✓	✓
View Notification Detail		✓	✓
View Facebook Count Page		✓	✓
View Chat Logs		✓	✓
View Fb Account Infor		✓	✓
View Providing Infor		✓	✓
Add Tag		✓	✓
Remove Tag		✓	✓
Export End User List		✓	✓
Change Bot Status		✓	✓
Logout		✓	✓
View Answer Editing Requests			✓
Reject Answer Editing Request			✓
Accept Answer Editing Request			✓
Create Account			✓
Delete Account			✓
Change Account Role			✓
Change Account Status			✓
Broadcast			✓

Table 3-4: Security matrix

3.3.2 Accuracy

- ✓ The overall accuracy is calculated by diving total number of correct answers by the number of questions asked.

3.3.3 Maintainability & Extensibility

- ✓ Follow the coding convention of Airbnb JavaScript Style Guide to help improve readability of source code and make the website more maintainable.
- ✓ Client-side web application uses Redux architecture. It complements React's compostable view components by utilizing a unidirectional data flow.

3.3.4 Availability and Scalability

- ✓ Using Azure Load Balancer for high availability.
- ✓ Using Microsoft Azure Virtual Machines for automatic horizontal scalability.
- ✓ Using Azure Monitor to monitor all services and alert when something wrong happened.

3.3.5 Performance

- ✓ Using HTTP/2, which uses a single, multiplexed connection to allow multiple requests to be sent on the same connection instead of establishing new connection for each request, increase speed of request-response.
- ✓ Web frontend uses React which uses several clever techniques to minimize the number of costly DOM operations required to update the UI.
- ✓ Web frontend is a Single Page Application with the goal of providing a more fluid user experience.
- ✓ Node.js brings event-driven programming to web servers, enables development of fast web servers in JavaScript.
- ✓ Using Microsoft Azure, Linux Virtual Machine for storage, hosting and deployment.

3.3.6 Usability

- ✓ The dashboard interface should be elegant, simple: links and buttons look like they are clickable, and are easily clickable, search box is wide enough for users to see what they have typed, interface does not break at various zoom levels.
- ✓ Chatbot avatar design is distinct, colorful and sets of words chatbot using are simple, friendly to create a friendly feel for the conversation and give the chatbot a bit of a persona.
- ✓ Having human in the loop to resolve ambiguity and provide response supervision.
- ✓ Having response buttons, rating buttons and clickable menus to help guide end users to their answer when needed.
- ✓ Using typing indicators allow the chatbot to interact with an end user in a more natural conversation.

3.4 Database document diagram

enduser_informations	
_id	
fullname	
phone	
email	
school	
schoolLocation	
facebookProfile	
	firstName
	lastName
	middleName
	gender
	profilePicture
tags[]	
	tagName
feedback	
	feedbackDone
	isLongTimeNoSee
	informationDone
	psid
	ignore
	feedbacks[]
	comment
	rating
	createdDate

scenarios	
_id	
intentName	
question	
valueId	
nodes[]	
	nodeId
	nodeName
	question
	values[]
	valueId
	valueName
	responseId

answers	
_id	
answers[]	
	content
	createdBy
	createdDate
	reviewdDate
	reviewer

responses	
_id	
intent	
response	
answerId	
entityValue[]	
	entity
	value

requests	
_id	
answerId	
editedContent	
createdDate	
createdBy	
state	
reviewer	
reviewerDate	

accounts	
_id	
username	
email	
password	
role	
fullname	
phone	
address	
active	
visible	
deactivatedTime	
deactivateReason	
resetPasswordExpires	
resetPasswordToken	

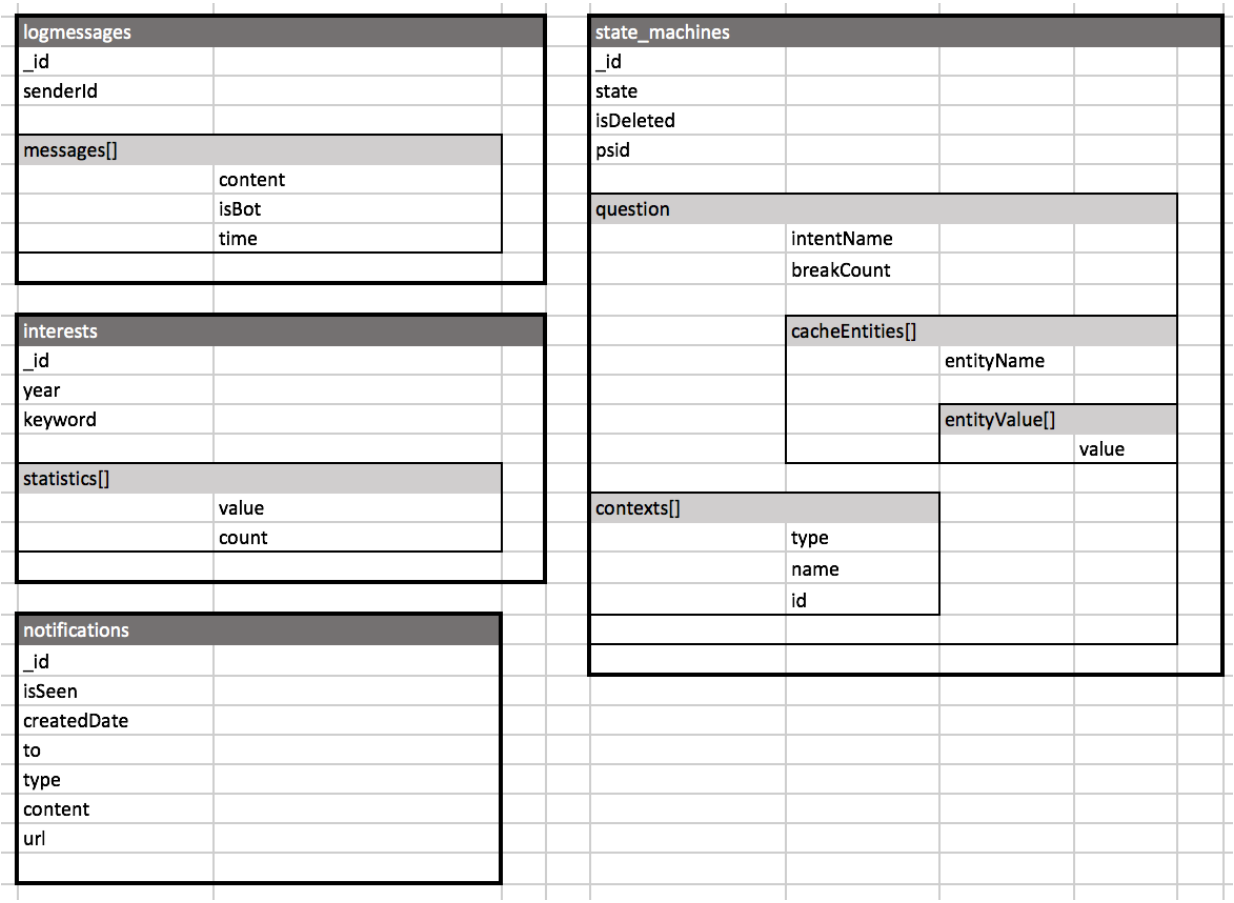


Figure 3-15: ASC database document diagram

Chapter 4 : Software Design

4.1 Purpose

This chapter is to give the developer team the overview and detailed design of what the system's architecture is, and how they should be implemented. This chapter consists of:

- ✓ Architecture overview
- ✓ Component diagram
- ✓ Detailed design
- ✓ Detailed description of components
- ✓ Database design

4.2 Architecture Overview

4.2.1 System Architecture

4.2.1.1 Diagram

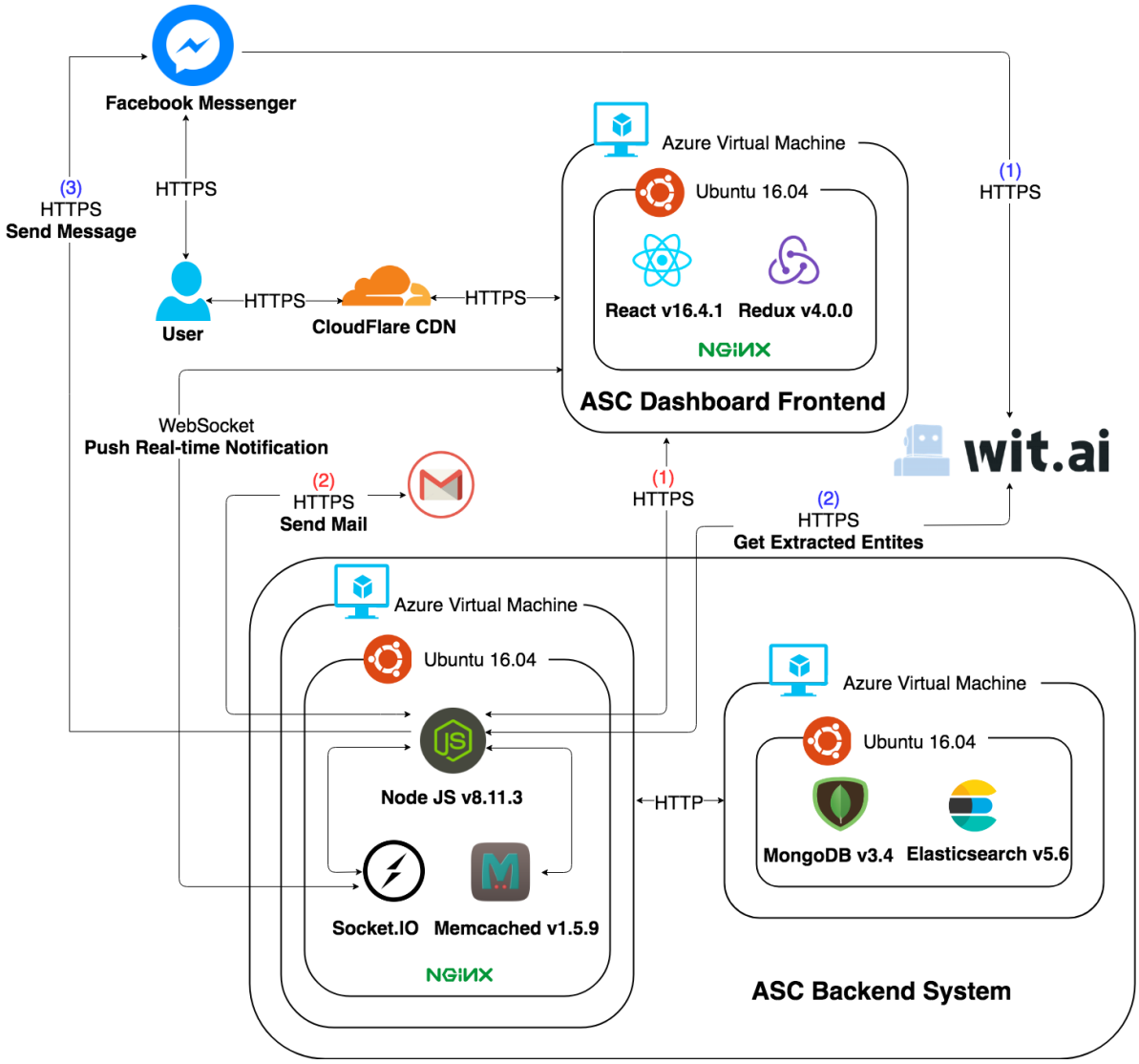


Figure 4-1: ASC System Architecture

4.2.1.2 Explanation

(1) ASC Frontend sends a request to ASC Backend.

Implementation

```
Request URL: https://futo-dashboard.luzotech.com/data/facebook
Request Method: GET
Status Code: 200
```

(2) ASC Backend requests to send mail using library.

(1) Messenger's built-in NLP automatically detects meaning and intent in every text message.

(2) ASC Backend requests to Wit.AI API to get all entities detected in the message.

(3) ASC Backend sends messages using Facebook Webhook.

Implementation

```
Request URL: https://graph.facebook.com/v3.0/me/messages?access_token=
Request Method: POST
```

4.2.2 System Architecture Explanation

The entire project will be deployed on Microsoft Azure. We aim at delivering a secured, responsive, and highly available system. In the following section, we will explain the function and mechanism of each unit in the system architecture design.

4.2.2.1 CloudFlare CDN

Figure 4-2: CloudFlare

CloudFlare provides a content delivery network and distributed domain name server (DNS) services, sitting between the visitor and the CloudFlare user's hosting provider, acting as a reverse proxy for websites**. We use it to provide addition layer of protection for ASC via Cloud Flare's SSL and resolve domain name to IP address point to server.**

4.2.2.2 Nginx

Figure 4-3: Nginx

Nginx is a web server. It can act as a reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer and an HTTP cache. **We use Nginx as a web server to serve static assets of ASC Frontend System.**

4.2.2.3 React

Figure 4-4: React

React is an open-source JavaScript library for building user interfaces, allows developers to create large web applications that use data which can change over time, without reloading the page. **We use it to build entire website view instead of plain HTML/JavaScript.**

4.2.2.4 Redux

Figure 4-5: Redux

Redux is a predictable state container for JavaScript apps. **Redux** evolves the ideas of **Flux** what is the application architecture that Facebook uses for building client-side web applications. It complements **React** view components by utilizing a unidirectional data flow. **We use Redux to take advantages of better separation of concerns, less time debugging.**

4.2.2.5 MongoDB

Figure 4-6: MongoDB

MongoDB is a free and open-source cross-platform document-oriented database program. **We use it to manage ASC operational data.**

4.2.2.6 Elasticsearch

Figure 4-7: Elasticsearch

Elasticsearch is a search engine that provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. **We use it to implement full-text search for end user's name and name of school.**

4.2.2.7 Node.js

Figure 4-8: Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side.

4.2.2.8 Socket.IO

Figure 4-9: Socket.IO

Socket.IO is a JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for node.js**. We use Socket.IO to push real-time updates to our Web application when there are changes of answer information, when dashboard user's status is updated, etc....**

4.3 Design of ASC System

4.3.1 Architecture Layers Design

4.3.1.1 ASC– API Layer Design

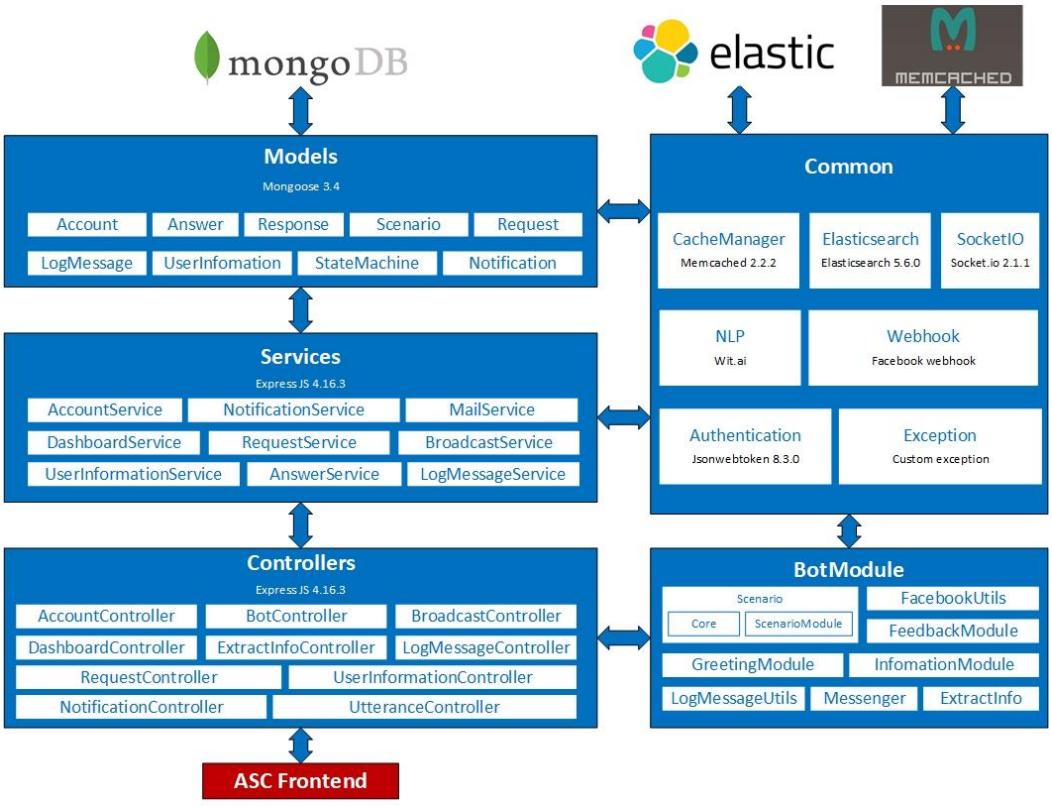


Figure 4-10: ASC - API Layer Design

No	Directory/File	Description	Convention
1	app.js	Entry point for the application. Everything in the app gets triggered through this file.	N/A

2	config/	This folder contains additional configuration files.	XXXConfig.js
3	controllers/	Contain Authentication, Account, Request, LogMessage, UserInformation, Utterance, Bot.	XXXController.js
4	constans/	Contain the constants.	N/A
5	bot_modules/	Contain modules related to bot.	N/A
6	doc/	Contains documentation that uses the API.	N/A
7	models/	Contain all Mongoose models.	N/A
8	services/	Contains files that directly interact with database like AccountService.js, RequestService.js.	XXXService.js
9	views/	Contain all ejs files.	N/A
10	routes/	Contain all route logic for RESTful API endpoint.	N/A
11	templates/	Contain all template.	N/A
12	utilities/	Contain helper files.	N/A
13	test/	Contain unit tests, integration tests, fixtures, and others test apparatus.	N/A
14	package.json	Contain meta data about application: the list of dependencies, npm run script.	N/A
15	.eslintrc.json	Contain rules for ESLint, which a set of rules to keep our project's code clean and well conventional.	N/A

Table 4-1: ASC-API Layer Detail

4.3.1.2 ASC – Frontend Design

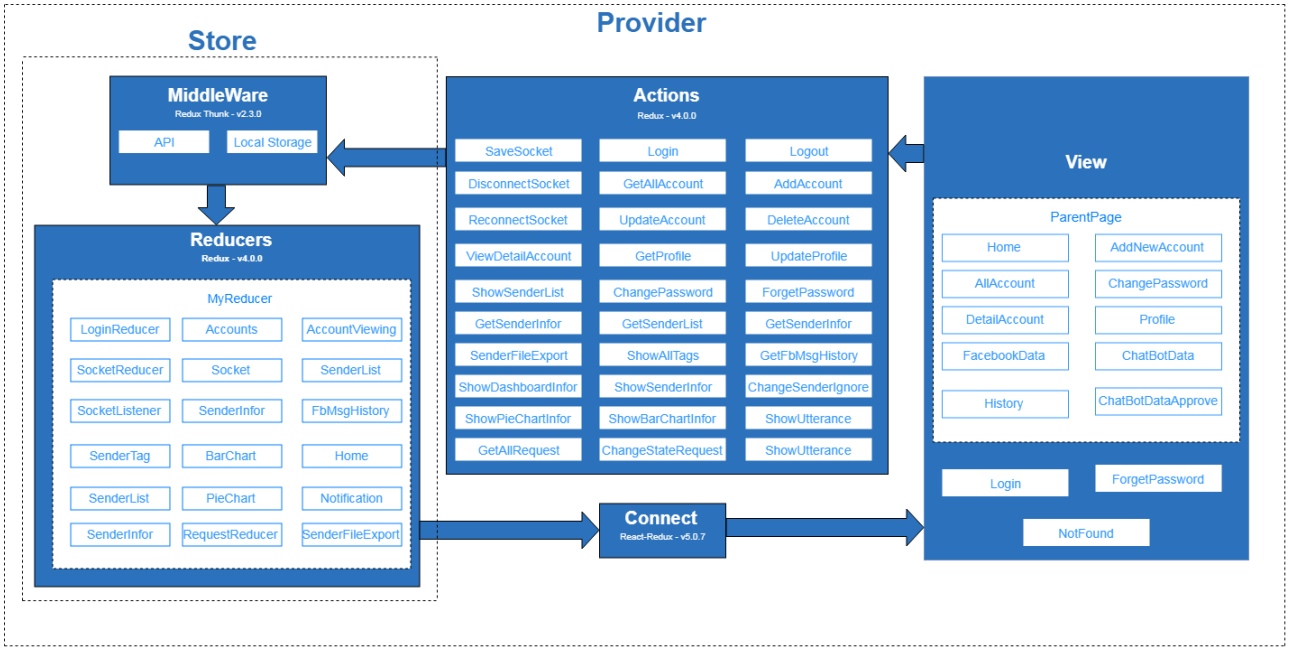


Figure 4-11: ASC – Frontend Design

No	Directory/File	Description	Convention
1	node_modules/	This folder contains all library of node modules	N/A
2	public/	This folder contains resource about html, css, javascript	N/A
3	src/	This folder contains all source code of project	N/A
4	src/actions/	This folder contains all Redux actions which components dispath	N/A
5	src/components/	This folder contains some components which is common UI for whole app	N/A
6	src/constants/	This folder contains all constants variable of project	N/A
7	src/img/	This folder contains some image	N/A
8	src/pages/	This folder contains all pages of application, each page map with a route	N/A
9	src/reducers/	This folder contains all Redux Reducer of projects	N/A
10	src/utils/	This folder contains some file to connect with api	N/A

11	src/App.css	The file contains some css of App component	N/A
12	src/App.js	This root component of application. All others components are show through this component	N/A
13	src/index.js	Entry point for the application. Everything in the app gets triggered through this file.	N/A
14	src/ registerServiceWorker.js	This file to register service with opera system to help app always run.	N/A
15	src/routes.js	This file contains route matching configurations.	N/A
16	.gitignore	Specifies intentionally untracked files to ignore.	N/A
17	package.json	This file contains information of project and all library with it is version of node modules	N/A
18	package-lock.json	The file describes the exacted node modules tree that was generated	N/A

Table 4-2: ASC- Frontend Layer Detail

4.3.2 Database Design

4.3.2.1 Explanation of database decision

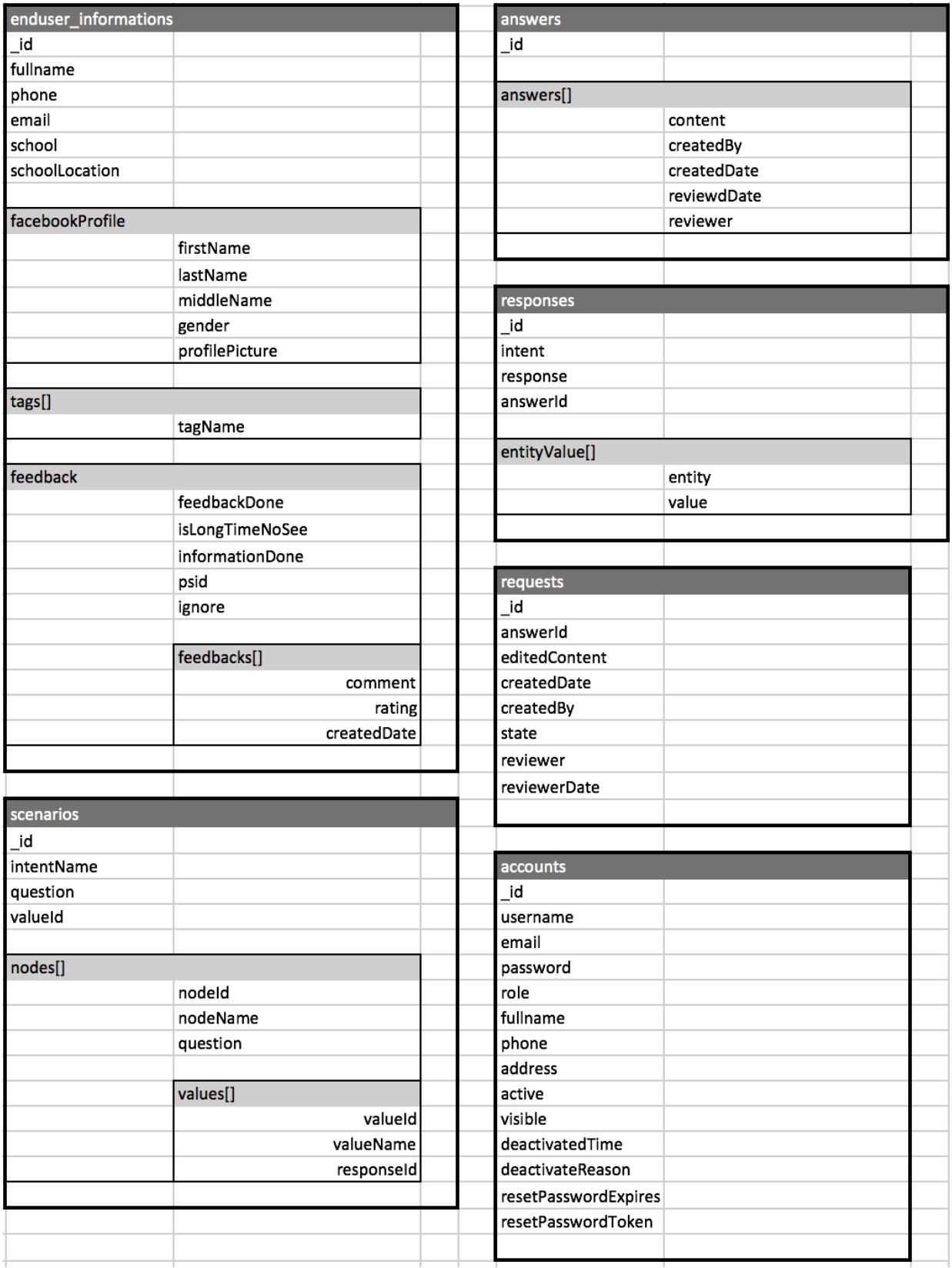
As our system have following characteristic:

- Users manage data frequently on dashboard.
- Flexible data model.
- System may need to scale.
- Provide real-time experience for statistics.

So, we chose MongoDB because:

- MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.
- MongoDB is free and open-source.
- MongoDB enables scaling by shards (partitions data by key ranges and distributes the data among two or more database instances).
- MongoDB allows stores sub-document inside document avoiding join multiple tables, therefore providing faster write and read time.

4.3.2.2 Database Diagram



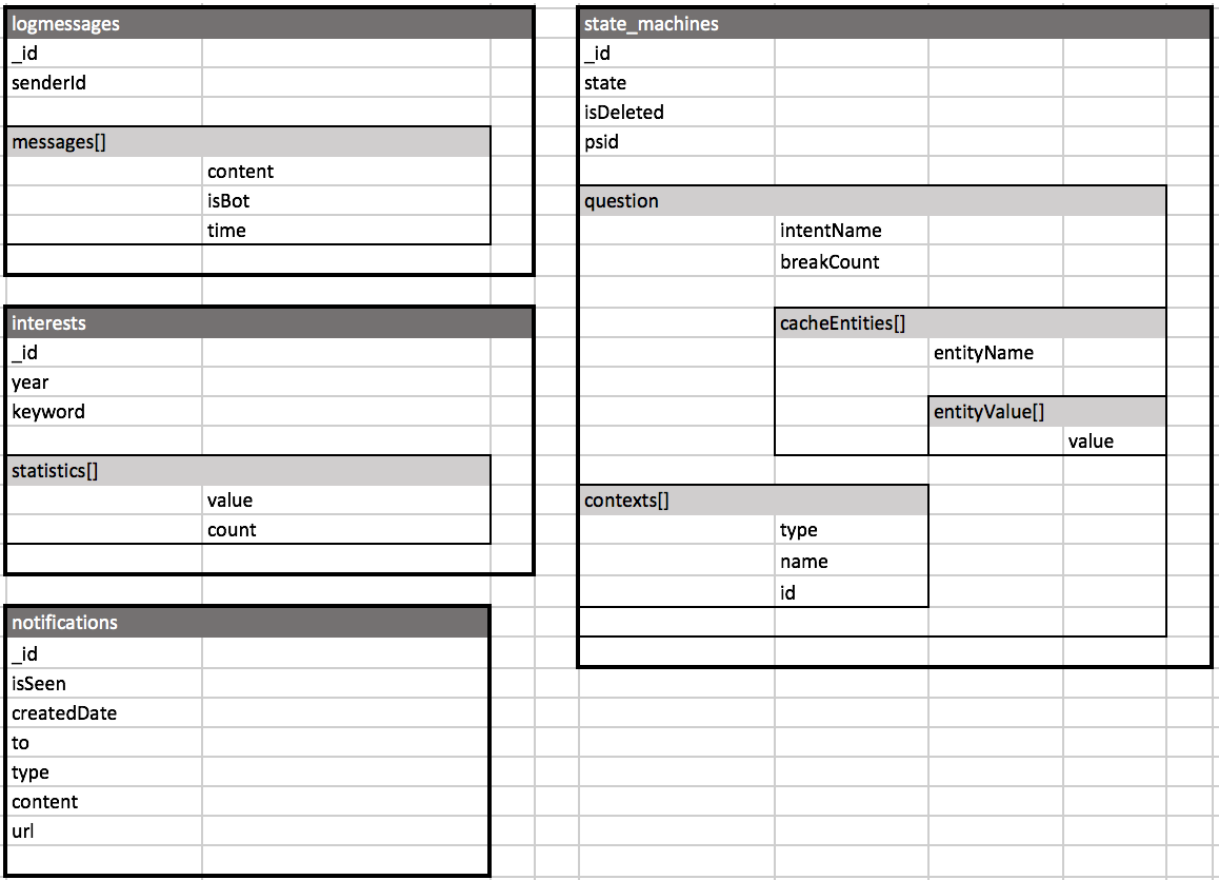


Figure 4-12: ASC database document diagram

4.3.2.3 Data Dictionary

Collection	Field	Type	Description
accounts	_id	ObjectId	MongoDB ObjectId, Unique for each document
	username	String	User's name
	email	String	User's email
	password	String	User's password
	role	String	Role of user
	fullname	String	Fullname of user
	phone	String	Contact phone of user
	address	String	Address of user
	active	Boolean	Status of user: active or not

	visible		Boolean	Status of user: visible or not
	deactivatedTime		Date	Time that user is deactivated
	deactivatedReason		String	Reason if user is deactivated
	resetPasswordExpires		Date	Time that mark all token create after this time is valid
	resetPasswordToken		String	Password reset token
requests	_id		ObjectId	MongoDB ObjectId, Unique for each document
	answerId		String	Id of answer that user edited
	editedContent		String	Content of answer that user edited
	createdDate		Date	Time when request is created
	createdBy		String	User created request
	state		String	State of request: pending, rejected or approved
	reviewer		String	User that reviewed request
	reviewedDate		Date	Time when request is reviewed
answers	_id		ObjectId	MongoDB ObjectId, Unique for each document
	answers		ArrayString	List of answers
		content	String	Content of answer
		createdBy	String	User when created answer

		createdDate	Date	Time when created answer
		reviewedDate	Date	Time when reviewed answer
		reviewer	String	User when reviewed answer
logmessages	_id		ObjectId	MongoDB ObjectId, Unique for each document
	senderId		String	psid of user sending messages
	messages		BsonArray	List of messages
		content	String	Text content in message
		isBot	Boolean	Whether or not the message is sent by bot.
		time	Date	Time when sent message
responses	_id		ObjectId	MongoDB ObjectId, Unique for each document
	intent		String	Intent of end user's message
	response		String	Response of chatbot
	entityValue		BsonArray	List of entities and values
		entity	String	Entity that extracted in message
		value	String	Value of each entity
	_id		ObjectId	MongoDB ObjectId, Unique for each document

enduser_informations	fullname		String	Fullname of end user
	phone		String	Phone of end user
	email		String	Email of end user
	school		String	Scholl of end user
	schoolLocation		String	Location of school
	facebookProfile		BsonObject	Fb profile of end user
		firstName	String	Fb first name of end user
		lastName	String	Fb last name of end user
		middleName	String	Fb middle name of end user
		gender	String	Fb gender of end user
		profilePicture	String	Fb profile picture of end user
	tags		ArrayString	List of tags attached to end user's interest
	feedback		BsonObject	Feedback of end user
		feedbackDone	Boolean	Status of getting feedback: done or not
		isLongTimeNoSee	Boolean	Status of end user that has not seen
		informationDone	Boolean	Status of getting end user's information
		psid	String	Id of end user's Fb page
		ignore	Boolean	Whether or not the end user is ignored.

					If set to false, chatbot is not auto reply with them.
		feedbacks		BsonArray	List of feedbacks
			comment	String	Content for report
			rating	String	Status of rating: Excellent, very good, good, bad, very bad
			createdDate	Date	Time when feedback is created
scenarios	_id			ObjectId	MongoDB ObjectId, Unique for each document
	intentName			String	Name of intent
	question			String	Question to ask end user
	valueId			String	Id of extracted value
	nodes			BsonArray	List of nodes
		nodeId		String	Id of node
		nodeName		String	Name of node
		question		String	Question to ask end user
		values		BsonArray	List of values
			valueId	String	Id of value
			valueName	String	Name of value
			responseId	String	Id of response
interests	_id			ObjectId	MongoDB ObjectId, Unique for each document
	year			String	Admission year

	keyword		String	Intent end user interested in.
	statistics		BsonArray	List of statistics
		value	String	Value end user interested in.
		count	Number	Number of value is interested in.
notifications	_id		ObjectId	MongoDB ObjectID, Unique for each document.
	isSeen		Boolean	Notification status: seen or not.
	createdDate		Date	Time when notification is created.
	to		String	Person is received notification.
	type		String	Type of notification.
	content		String	Content for notification.
	url		String	Reference to specific url.
state_machines	_id		ObjectId	MongoDB ObjectID, Unique for each document.
	state		String	Current state in the conversation.
	psid		String	Id of end user's Fb page.
	question		BsonObject	Question to ask end user.
		intentName	String	Intent of end user.
		breakCount	Number	Count to break state.

	contexts		BsonObject	Context of utterance.
		type	String	Type of context.
		name	String	Name of context.
		id	String	Id of context.

Table 4.2 ASC's data dictionary

4.3.3 Common Design

4.3.3.1 Natural Language Processing

Natural language processing (NLP) employs computational techniques for the purpose of **learning, understanding, and producing human language content**.

A more advanced application of NLP is Natural language understanding (NLU), **genuinely understanding** what the text says.

Depending on our project demand integrating with Facebook Messenger and supporting the primary Vietnamese language, we find out the 2 most appropriate NLU platforms Wit.AI and FPT.AI.

Description	**Wit.AI[1]**	**FPT.AI[2]**
Accuracy[3]	0.83419	0.85424
Vietnamese Support	Trial version	Version 2.0
Integrating tools into bots for Fb Messenger	Using built-in NLP easily and quickly	No support for built-in NLP
Service charge	Completely free	Free 25 entities, 25 intents. Having an upgrade fee.
Community	Large	Small
Docs	Full detail	Empty

Table 4-3:NLP comparison

After having a comparison of two of these commercial platforms, we chose Wit.ai for applying NLP. With Wit.ai, we simply train my bot with sample utterances that get parsed into intents and entities. Moreover, we can define our own intents and entities, or use those that have already been built into Wit.ai.

The pros and cons of using NLP with Wit.ai:

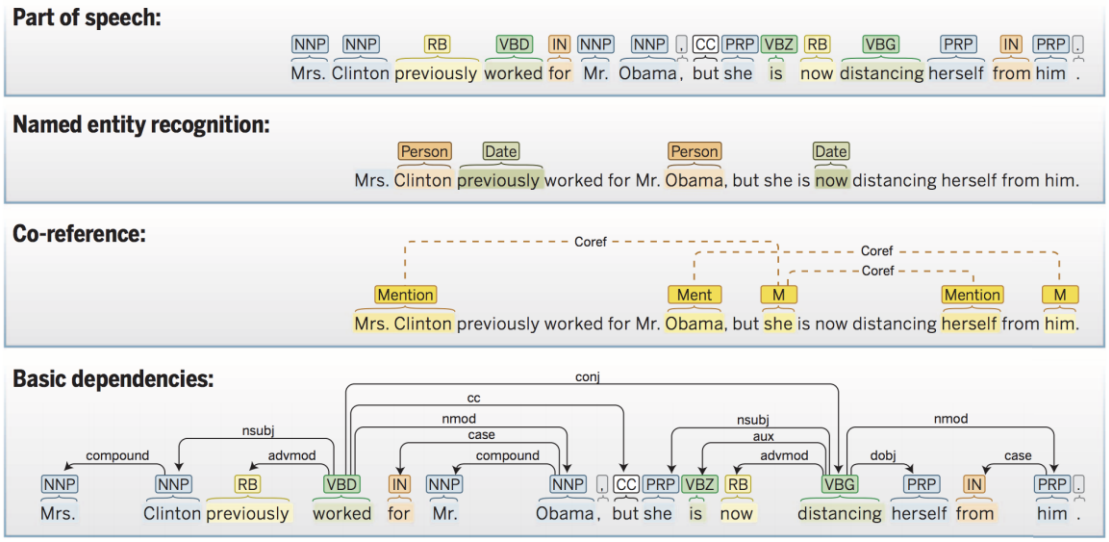
[1] Wit.ai website: https://wit.ai/
[2] FPT.ai website: https://fpt.ai/
[3] Accuracy on FTel test data set: https://www.slideshare.net/minhpqn/cc-bi-ton-x-l-ngn-ng-t-nhin-trong-pht-trin-h-thng-chatbot

- ✓ Lots of built-in intents and entities.
- ✓ Easy to use bot creation tool.
- ✓ Extensive language support (including Vietnamese).
- ✓ An open source model for sharing apps.
- ✓ An inbox tool to classify utterances from a running bot.
- ✓ Free.
- ✓ Multiple ways to parse utterances (**trait, free-text, keywords, free-text and keywords**).
- – No support for bot dialogs - responses are built and maintained outside of Wit.ai.
- – It's unclear how Wit.ai scales.

4.3.3.1.1 Data preprocessing

One example is Stanford CoreNLP, which provides a standard NLP preprocessing pipeline that includes:

- POS tagging (with tags such as noun, verb, and preposition).
- Identification of named entities, such as people, places, and organizations.
- Parsing of sentences into their grammatical structures.
- Identifying co-references between noun phrase mentions.



[1]

Table 4-4: Many language technology tools start by doing linguistic structure analysis

As shown from top to bottom, this tool determines the parts of speech of each word, tags various words or phrases as semantic named entities of various sorts, determines which entity mentions co-refer to the same person or organization, and then works out the syntactic structure of each sentence, using a dependency grammar analysis.

With Wit.ai, it uses duckling library to extract entities. Duckling is a Haskell library that parses text into structured data. Example of text extraction using duckling is provided below.

```
"the first Tuesday of October"
=> {"value":"2017-10-03T00:00:00.000-07:00","grain":"day"}
```

[1] Advances in natural language processing page: https://cs224d.stanford.edu/papers/advances.pdf

4.3.3.1.2 Extraction process

Wit.ai has predefined entities developers need to extract temporal expressions, numbers, locations, temperatures, etc. One example Wit.ai auto extracts an entity "wit/amount_of_money" after inputing a sentence.

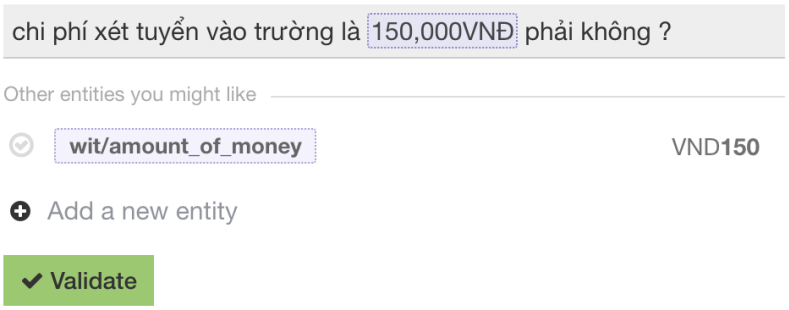


Figure 4-13: Built-In Entity Extraction

If the entity we need is not built-in, we can create our own. We will have to choose a Lookup Strategy for our entity:

Lookup Strategy	**Use Case**
Trait	There is no obvious association between certain words in the sentence and the value of the entity, but rather need the sentence as a whole to determine the value.
Free Text	When need to extract a substring of the message, and this substring does not belong to a predefined list of possible values.
Keywords	When the entity value belongs to a predefined list, and just need substring matching to look it up in the sentence.

Table 4-5: Wit.ai Lookup Strategy

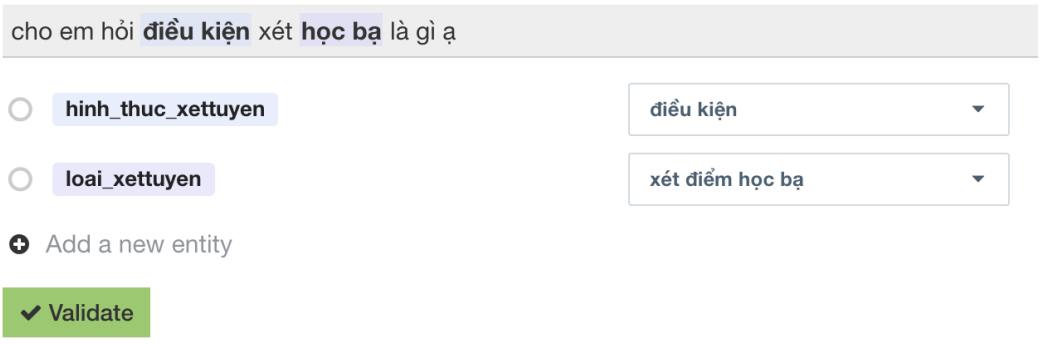


Figure 4-14: Custom Entities Labeling

4.3.3.1.3 Classification process

Intent is categorized using text classification methods that are based on Bag of Words or more advanced word embedding methods like Support Vector Machine.

A **bag-of-words (BoW)** is a representation of text that describes the occurrence of words within a document. A simple illustration of bag-of-words model is provided below:

[1]

Table 4-6: Bag-of-words model

Input	điều kiện	xét học bạ	là	gì	lịch trình	đến	khi	nào	cần	liên lạc	với	phòng	tuyển sinh	thế	Output
"điều kiện cần xét học bạ là gì"	1	1	1	1	0	0	0	0	1	0	0	0	0	0	[1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
"lịch trình xét học bạ đến khi nào nào"	0	1	0	0	1	1	1	2	0	0	0	0	0	0	[0, 1, 0, 0, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0]
"xét học bạ cần điều kiện gì"	1	1	0	1	0	0	0	0	1	0	0	0	0	0	[1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
"liên lạc với phòng tuyển sinh thế nào"	0	0	0	0	0	0	0	1	0	1	1	1	1	1	[0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1]

Figure 4-15: BoW Example

Once a vocabulary has been chosen, the occurrence of words in documents needs to be scored. A problem with scoring word frequency is that highly frequent words may not contain as much "informational content" to the model as rarer but perhaps domain specific words.

One approach is to rescale the problem called **Term Frequency – Inverse Document Frequency (TF-IDF)[2].** The scores have the effect of highlighting words that are distinct (contain useful information) in a given document.

TF-IDF is computed for each term in each document following the formula:

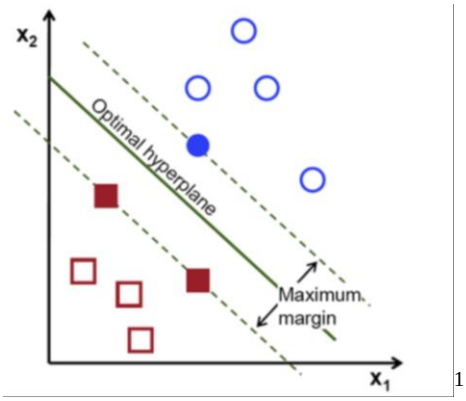
```
TF(t) = (Number of times term t appears in a document) / (Total number of terms in
the document)

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Value = TF * IDF
```

Doc	Input	điều kiện	xét học bạ	là	gì	lịch trình	đến	khi	nào	cần	liên lạc	với	phòng	tuyển sinh	thế	Output
d1	"điều kiện cần xét học bạ là gì"	1	1	1	1	0	0	0	0	1	0	0	0	0	0	
d2	"lịch trình xét học bạ đến khi nào nào"	0	1	0	0	1	1	1	2	0	0	0	0	0	0	W(lịch trình,d2) = 1/5 * log(4/1) = 0.12 W(nào,d2) = 2/5 * log(4/3) = 0.05
d3	"xét học bạ cần điều kiện nào"	1	1	0	0	0	0	0	1	1	0	0	0	0	0	
d4	"liên lạc với phòng tuyển sinh thế nào"	0	0	0	0	0	0	0	1	0	1	1	1	1	1	

Figure 4-16: TF-IDF Example

Furthermore, the other algorithm can be used for classification is **Support Vector Machine (SVM).** With given labeled training data (*supervised learning*), the algorithm outputs the hyper-plane that differentiate the two classes very well so the misclassification is reduced when training new data set.

[1] Bag-of-words page: https://machinelearningmastery.com/gentle-introduction-bag-words-model/
[2] TF-IDF page: http://www.tfidf.com/



[1]

Table 4-7: SVM algorithm

4.3.3.2 Elasticsearch on ASC Backend

We use Elasticsearch database to support ASC back-end search by full-text search helping backend search for name and school.

- Initialization

Implementation:

```
var elasticsearch = require("elasticsearch")
const ElasticConfig = require ("../config/elasticConfig")
var client = new elasticsearch.Client
       host: ElasticConfi.host
       log: ElasticConfig.log
```

- Searching with elasticsearch

Implementation:

```
searchName()
   body = index: ElasticConfig.index
          type: "names"
          body: query: match: name: query
```

4.3.3.3 Real-time features with Socket.IO

Real-time applications

Socket.IO is a JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers.

Socket.IO has two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API.

Socket.IO primarily uses the WebSocket protocol with polling as a fallback option, while providing the same interface.

4.3.3.4 Socket.IO on ASC Frontend

[1] SVM page: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

We use **socket.io-client** library on NPM to implement it on ASC Frontend.

- Initialization

Implementation:

```
const socket = socketIOClient (config.SOCKET_URL)
secure = true
```

- Receive Socket.IO event from server as actions in Redux using socketIoMiddleware from **socket.io-client**.

Implementation:

```
const store = createStore (myReducer, window._REDUX_DEVTOOLS_EXTENSION_ &&
window._REDUX_DEVTOOLS_EXTENSION(), applyMiddleware(thunk))
```

4.3.3.5 Socket.IO on ASC Backend

We use **socket.io** library on NPM to implement it on ASC Backend.

- Initialization

Implementation:

```
var io = require ("socket.io”)(http)
```

- Socket.IO connection

Implementation:

```
io.on()
    App.request.socketId = socket.id
    Socket.on ()
            Var socketIds = global.socketIds
            If socketIds === null
                    Return
            For each user in socketId list
                    If socketId of user includes socket.id in socketId list
                            For each index in socketId of user list
                                    If socketId === socket.id
                                    Break
                    If socketId of user === null
                            delete socketId of user
                    global.socketIds = socketIds
```

4.3.3.6 Using MongoDB with Mongoose

Mongoose is a library on NPM for Mongo, written in Node.js. The document of Mongoose is very clear and well updated. Every model in our project will be wrapped inside Mongoose Model. Thus, every model by default will have a unique **_id.**

4.3.3.7 Class diagram

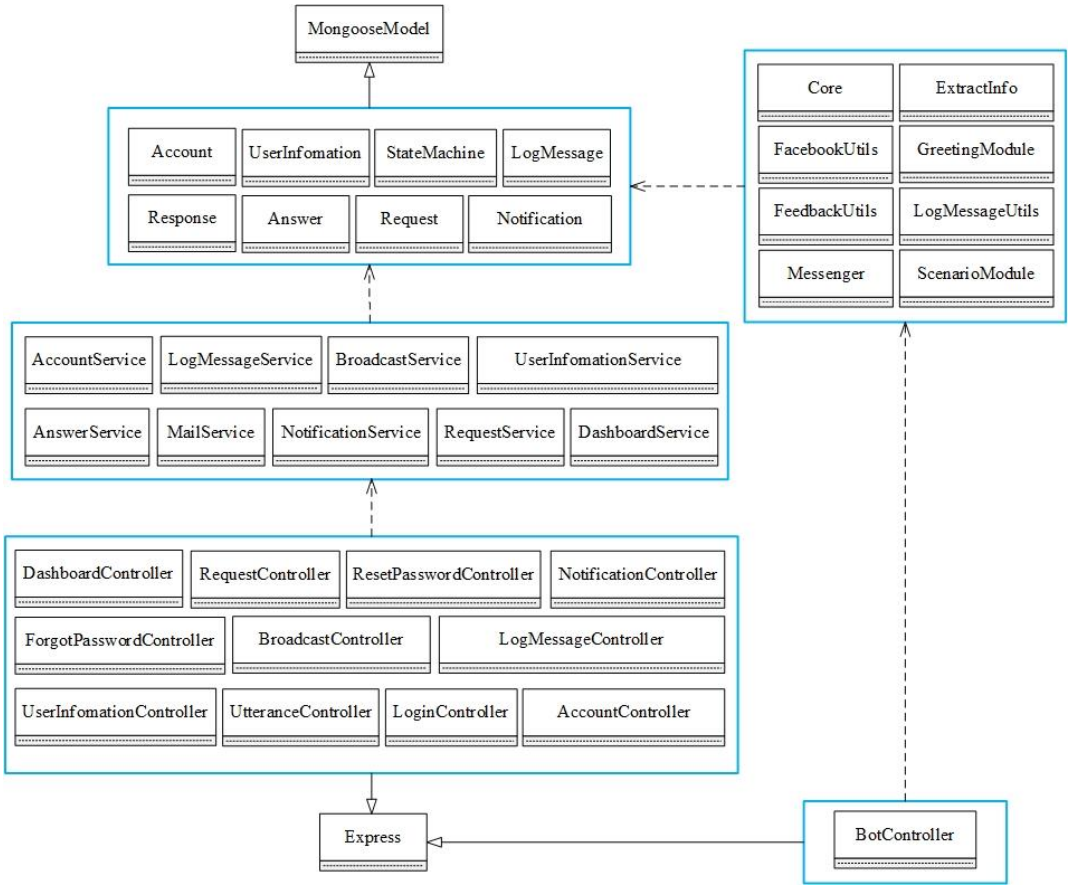


Figure 4-17: ASC class diagram

4.3.4 Detail Design

4.3.4.1 End user chats

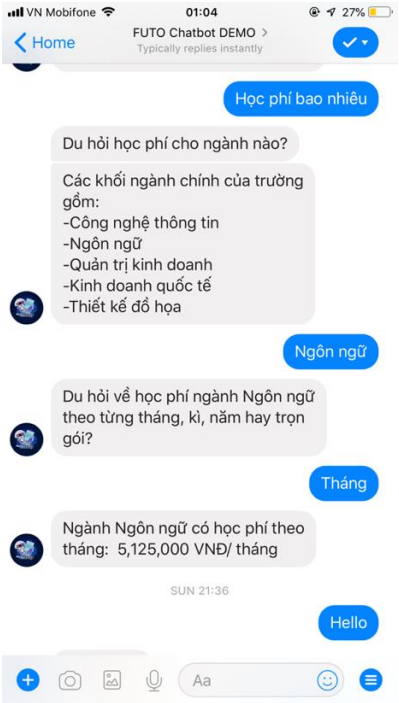


Figure 4-18: Chat Conversation design

4.3.4.1.1 Training the chatbot using Wit.ai

Collect real end user utterances on website

Implementation

```
var data = request.request("http://daihoc.fpt.edu.vn/tuyen-sinh/")
    auto_load_more: true
io.writeAsync()
    write data into file
    path: "./"
    file_name: "data.json"
```

Define intents and entities

With my project, we choose 2 types of Lookup Strategy including Trait for intents and Free Text and Keywords for entities.

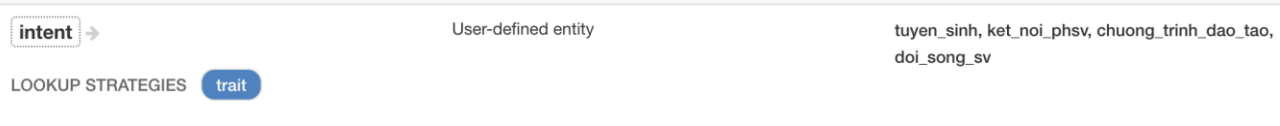


Figure 4-19: Intent list

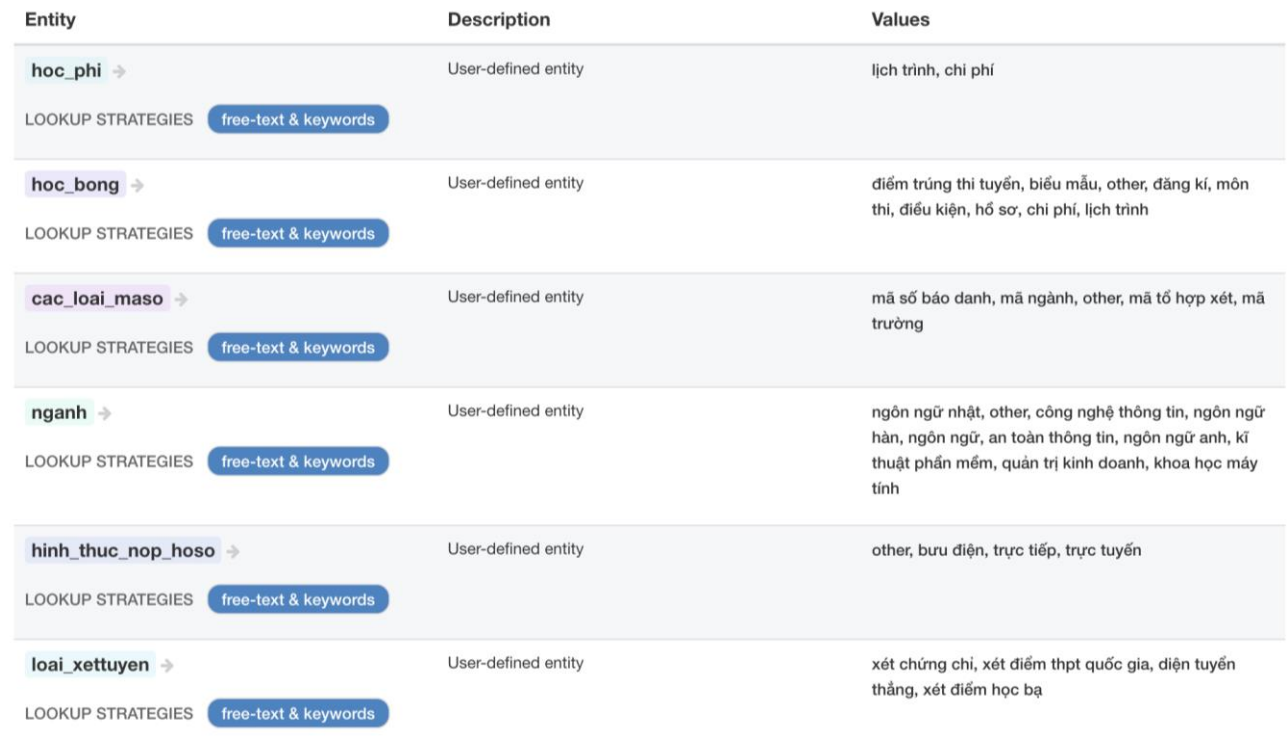


Figure 4-20: Entity list

Categorize the end user intent and extract entities

Using labelling training data set, we train all questions mapped to the different intent. If a sentence containing one of the keywords or substring of the message, Wit.ai will extract all its. To push all questions into Wit instead of typing manually, we must create a tool to read Excel file.

Implementation

```
ReadExcel = require("./ReadExcel")
Wit = require("./Wit")
var Data = readExcel.process
    input path
Wit.insertManyAsync(entities)

→ OUTPUT:
    doc: cho mình hỏi thông tin tuyển sinh của nhà trường với
    id: tuyen_sinh
    status: success
```

After importing training data successfully, we check again intent regarding to entities Wit.ai extracted and validate manually to train Wit again.

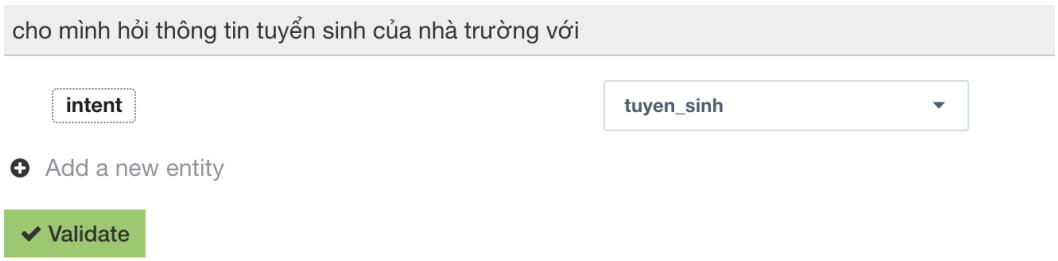


Figure 4-21: Intent Classification

4.3.4.1.2 Conversation Design

4.3.4.1.2.1 Dialogues by finite state method

In order to model dialogues, we made so many states and defined lots of transitions using Wit intent model. When our system initialized, it enters initiation state. A response is selected among the candidate sentences based on the transition triggered by an end user query and by the explicit state which stores context information.

The dialogues managed by finite state method has two types: multi-round and one-round dialogue.

4.3.4.1.2.1.1 Multi-round dialogues

We designed 4 multi-round dialogues: Admission, Major, Accommodation and Contact. These dialogues are triggered when the end user asks about the topics.

- Admission dialogue provides a general information about the admission that end user asked about and provides about scholarships, costs and fees, entry requirements, how to apply.
- Major dialogue provides a general information about degree programs, short courses.
- Accommodation dialogue provides information about dormitory, services, sports, security.
- Contact dialogue provides information about receive counseling indirectly or directly.

To break down a complex end user's utterance we use decision tree, a map of the possible outcomes of related choices. A decision tree typically starts with a single node, which branches into possible outcomes. Each of those outcomes leads to additional nodes, which branch off into other possibilities. This gives it a treelike shape. We design 4 decision trees corresponding with 4 multi-round dialogues.

Now we describe our decision trees for making state machine transitions in multi-rounds conversations.

Admission dialogue

- Intent Node: tuyen_sinh

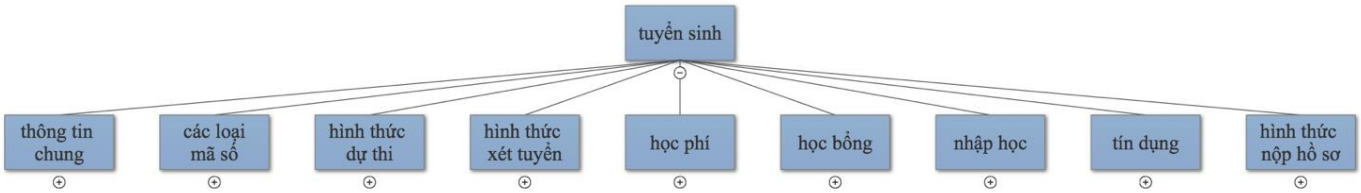


Figure 4-22: tuyen_sinh node

- Entity Node: thong_tin_chung

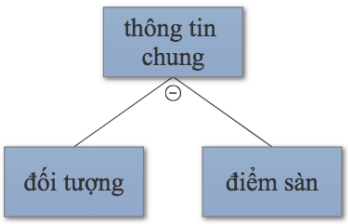


Figure 4-23: thong_tin_chung node

- Entity Node: cac_loai_maso

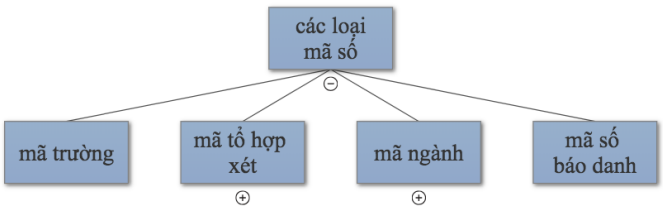


Figure 4-24: cac_loai_maso node

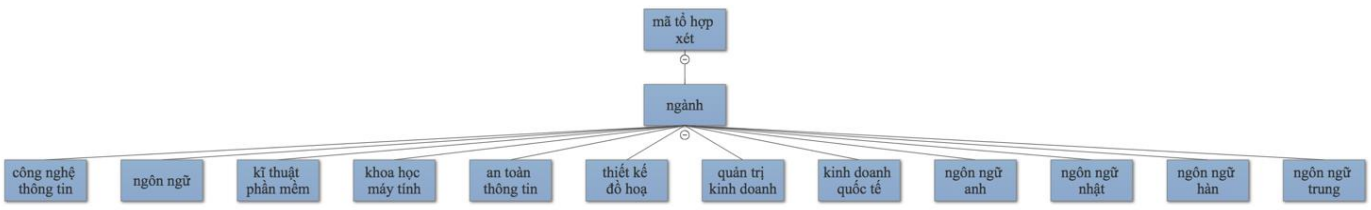


Figure 4-25: mã tổ hợp xét node

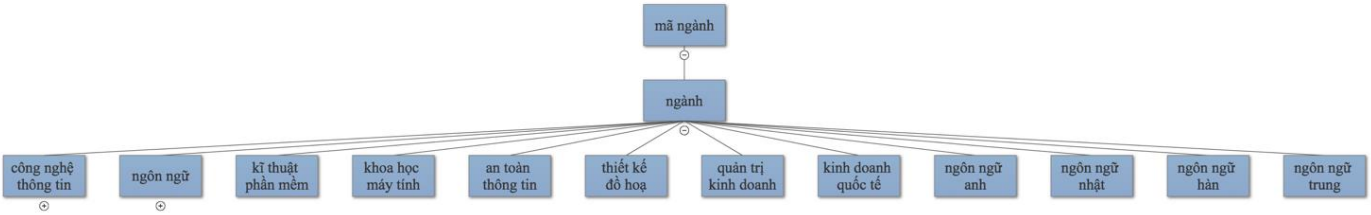


Figure 4-26: mã ngành node

- Entity Node: hinh_thuc_duthi

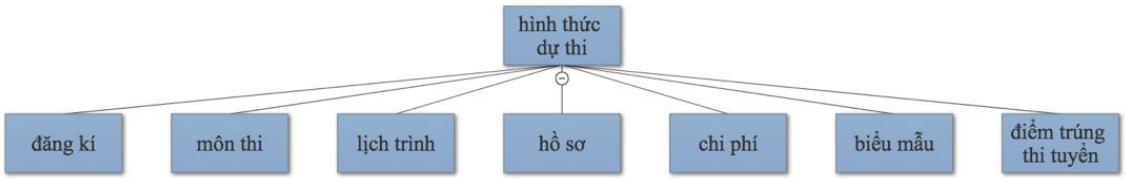


Figure 4-27: hinh_thuc_duthi node

- Entity Node: hinh_thuc_xettuyen

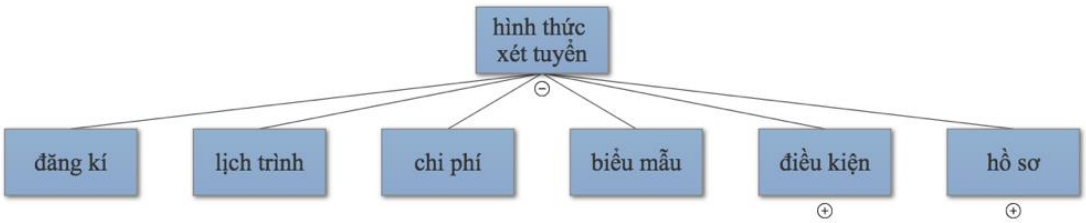


Figure 4-28: hinh_thuc_xettuyen node

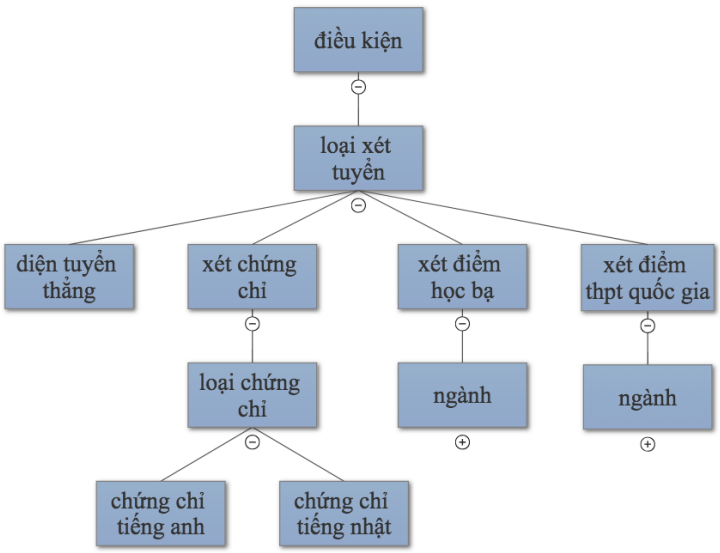


Figure 4-29: điều kiện node

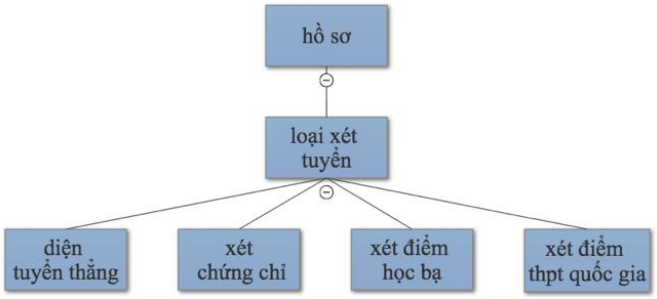


Figure 4-30: hồ sơ node

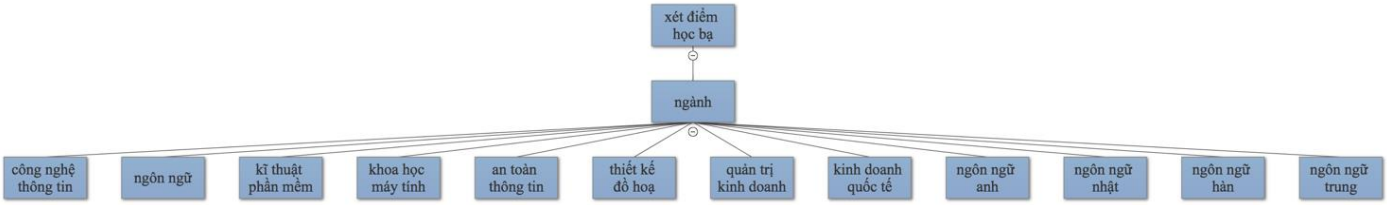


Figure 4-31: xét điểm học bạ node

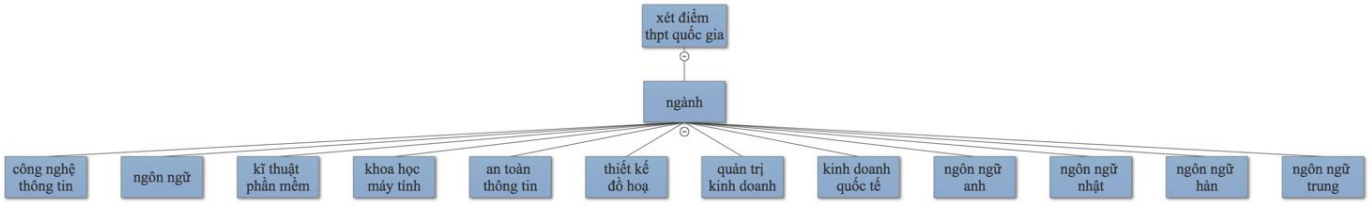


Figure 4-32: xét điểm thpt quốc gia node

- Entity Node: hoc_phi

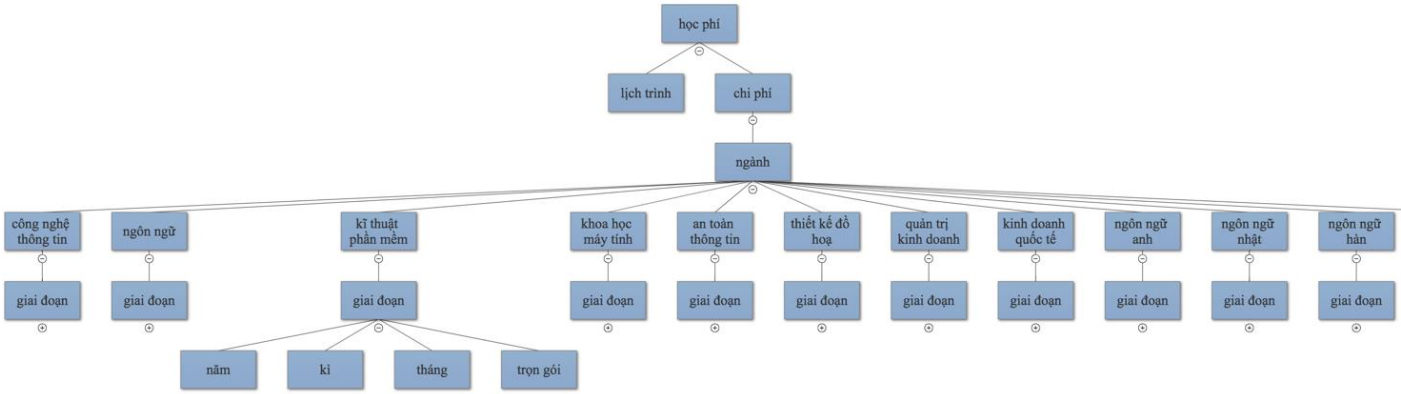


Figure 4-33: hoc_phi node

- Entity Node: hoc_bong

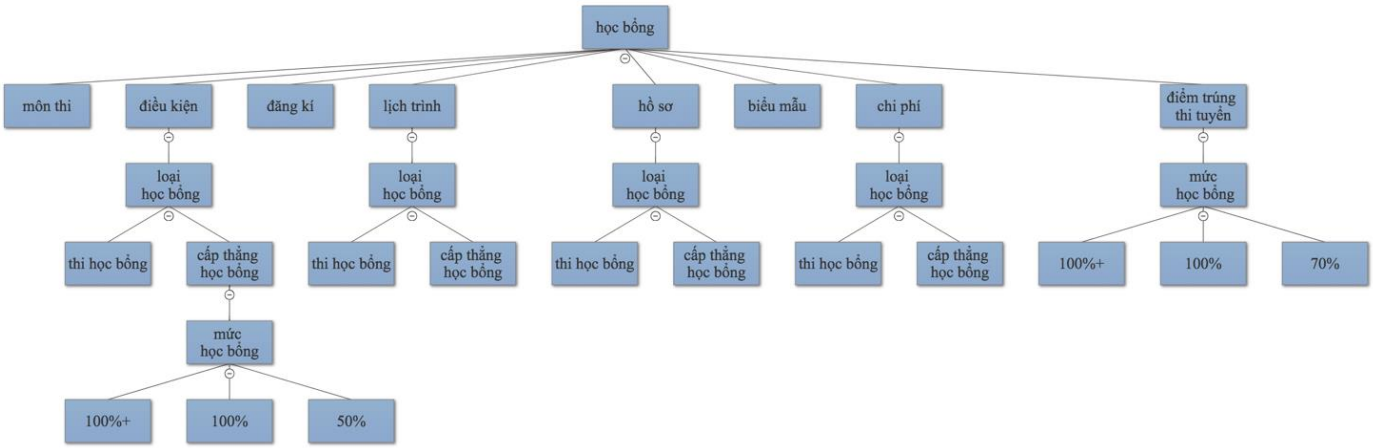


Figure 4-34: hoc_bong node

- Entity Node: nhap_hoc

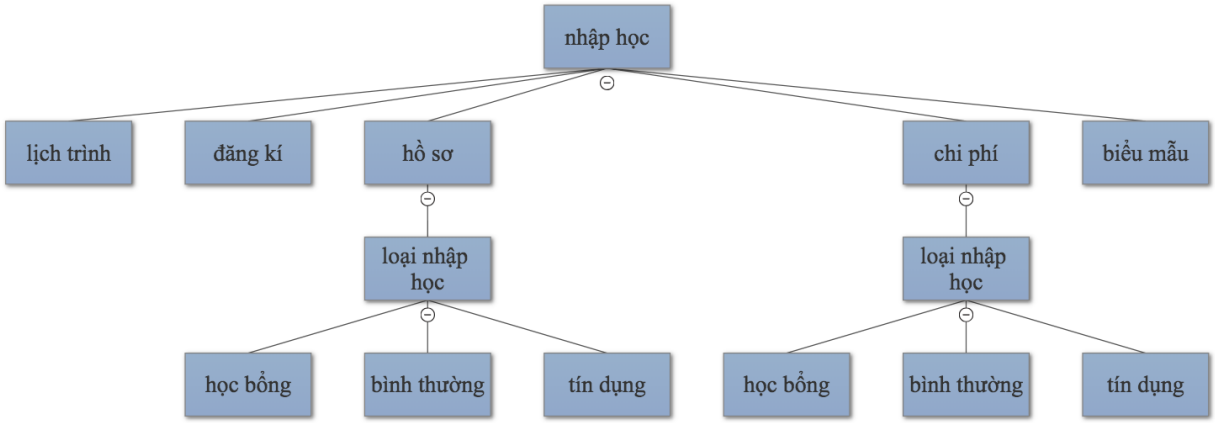


Figure 4-35: nhap_hoc node

- Entity Node: tin_dung

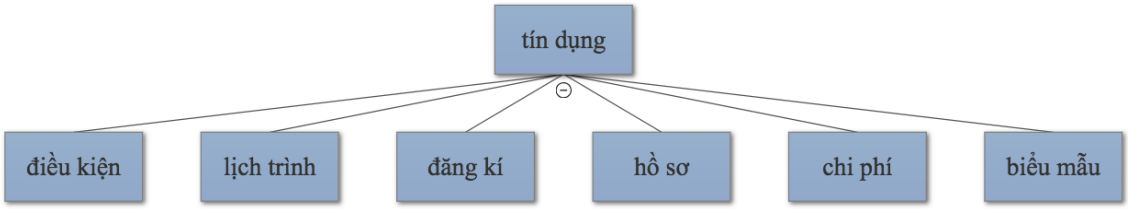


Figure 4-36: tin_dung node

- Entity Node: hinh_thuc_nop_hoso

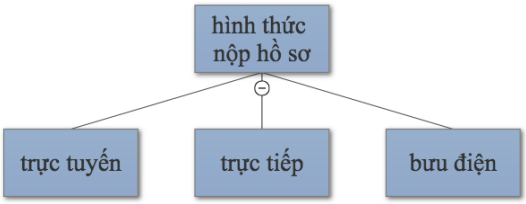


Figure 4-37: hinh_thuc_nop_hoso node

Major Dialogue

- Intent Node: chuong_trinh_dao_tao

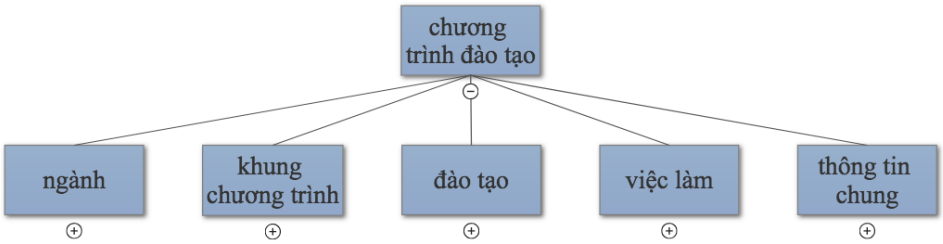


Figure 4-38: chuong_trinh_dao_tao node

- Entity Node: nganh

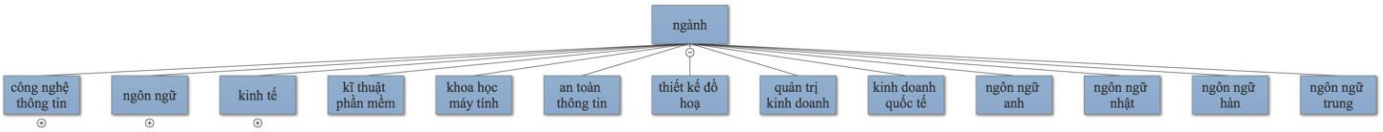


Figure 4-39: nganh node

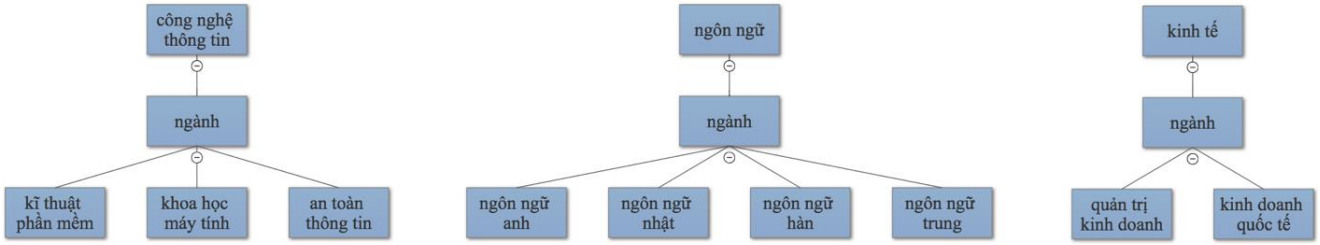


Figure 4-40: công nghệ thông tin, ngôn ngữ, kinh tế node

- Entity Node: khung_chuong_trinh

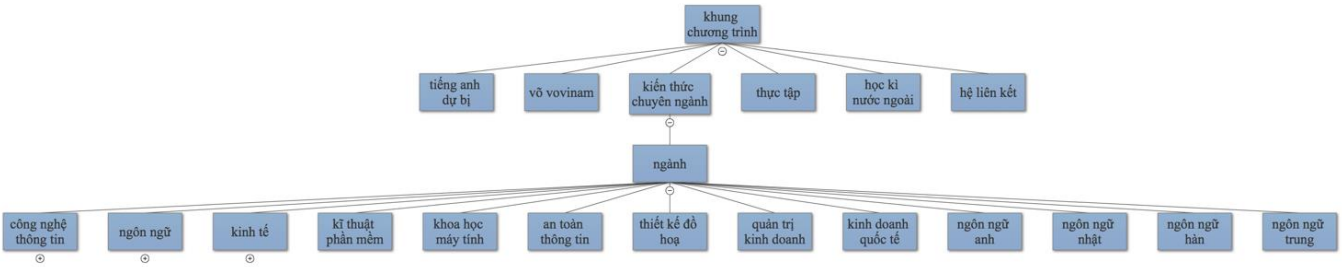


Figure 4-41: khung_chuong_trinh node

- Entity Node: dao_tao

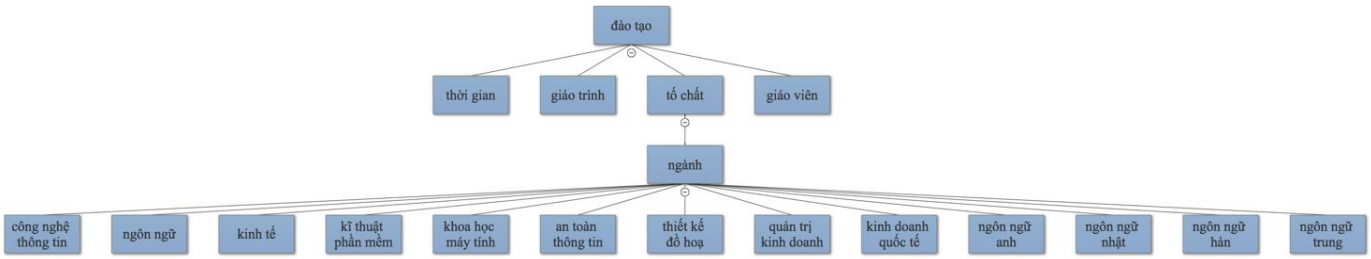


Figure 4-42: dao_tao node

- Entity Node: viec_lam

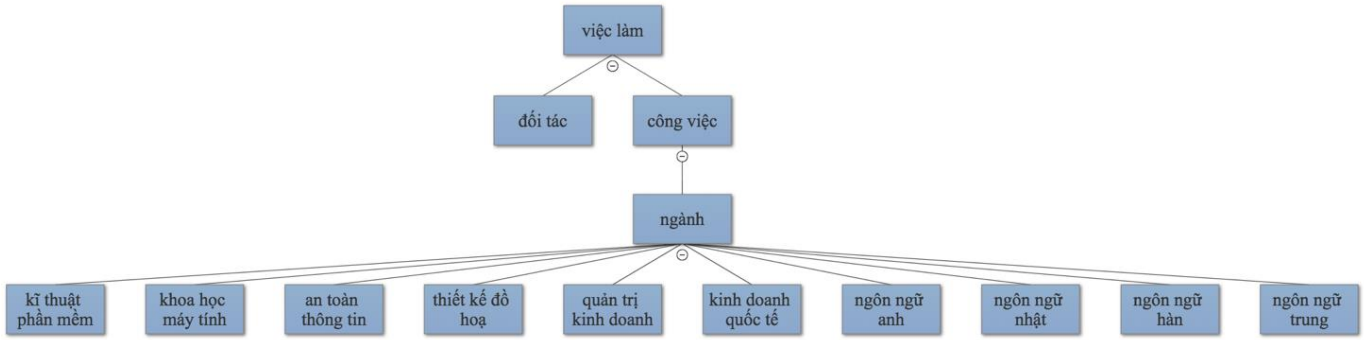


Figure 4-43: viec_lam node

- Entity Node: thong_tin_chung

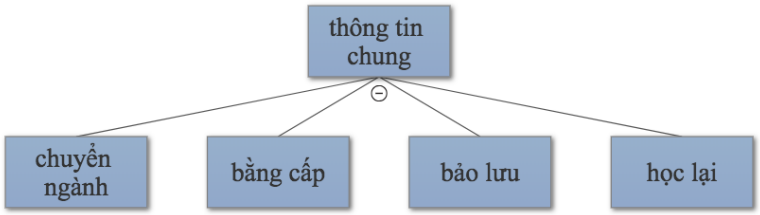


Figure 4-44: thong_tin_chung node

Accommodation Dialogue

- Intent Node: doi_song_sv

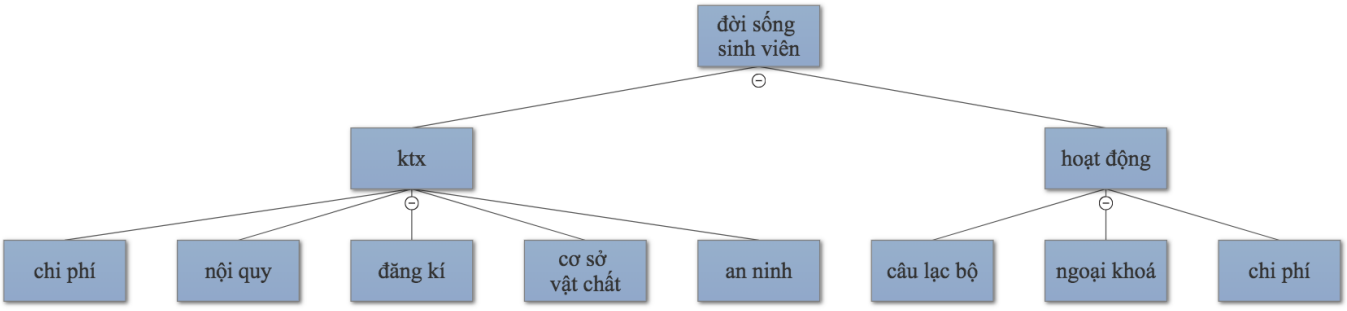


Figure 4-45: doi_song_sv node

Contact Dialogue

- Intent Node: ket_noi_phsv

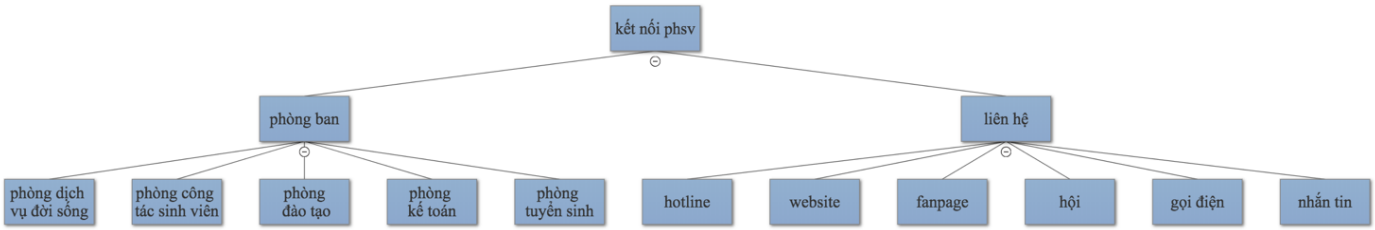


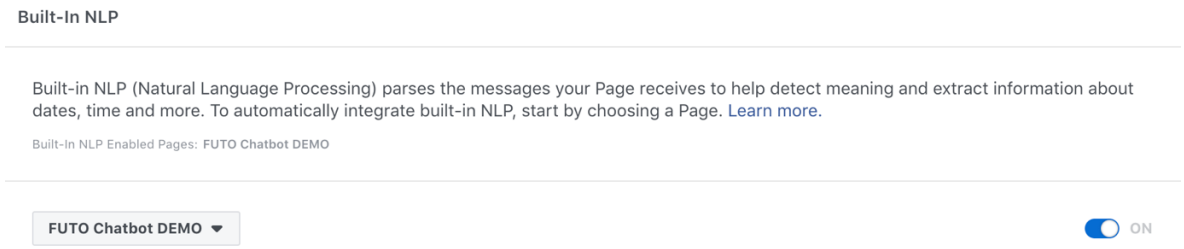
Figure 4-46: ket_noi_phsv node

4.3.4.1.2.1.2 One-round dialogues

We also defined intents for one-round conversations. These intents return appropriate utterance and change state to Default. First, we made an intent handling greetings from end users. Moreover, to handle end user's questions about the system's personal information such as age, name, sex, address, maker we made 5 different intents for each of them. Also we desinged intents to respond to ethical issues using bad words.

4.3.4.1.3 Integrate Wit.ai with Facebook Messenger

By choosing a page in https://developers.facebook.com/apps , Built-In NLP is automatically integrated on it.



4.3.4.1.4 Class diagram

One-round dialogue

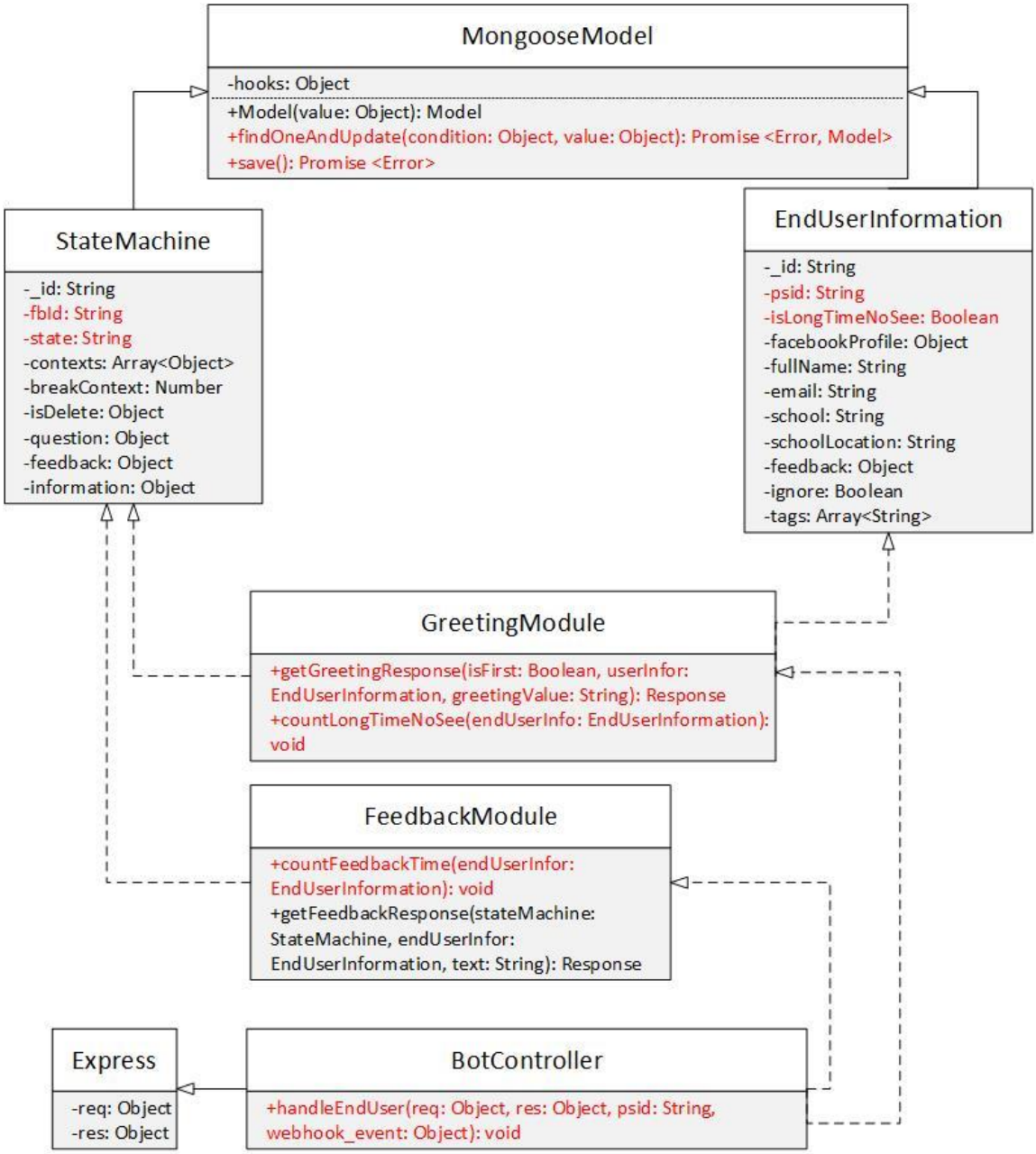


Figure 4-47: One-round dialogue class diagram

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend/models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
2	isLongTimeNoSee	Boolean	
Operation			

StateMachine

StateMachine

Physical address	Backend/models/StateMachine.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	fbId	String	
2	state	Boolean	
Operation			

GreetingModule

GreetingModule			
Physical address	Backend/controllers/GreetingModule.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getGreetingResponse			
Return Type	Response		
Parameters	Name	Type	Description
	isFirst	Boolean	
	userInfor	EndUserInformation	
	greetingValue	String	
countLongTimeNoSee			
Return Type	void		
Parameters	Name	Type	Description
	endUserInfor	EndUserInformation	

FeedbackModule

FeedbackModule			
Physical address	Backend/controllers/FeedbackModule.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
countFeedbackTime			
Return Type	void		
Parameters	Name	Type	Description
	endUserInfor	EndUserInformation	

BotController

BotController			
Physical address	Backend/controllers/BotController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
handleEndUser			
Return Type	void		

Parameters	Name	Type	Description
	req	object	
	res	object	
	psid	String	
	webhook_event	object	

Multi-round dialog

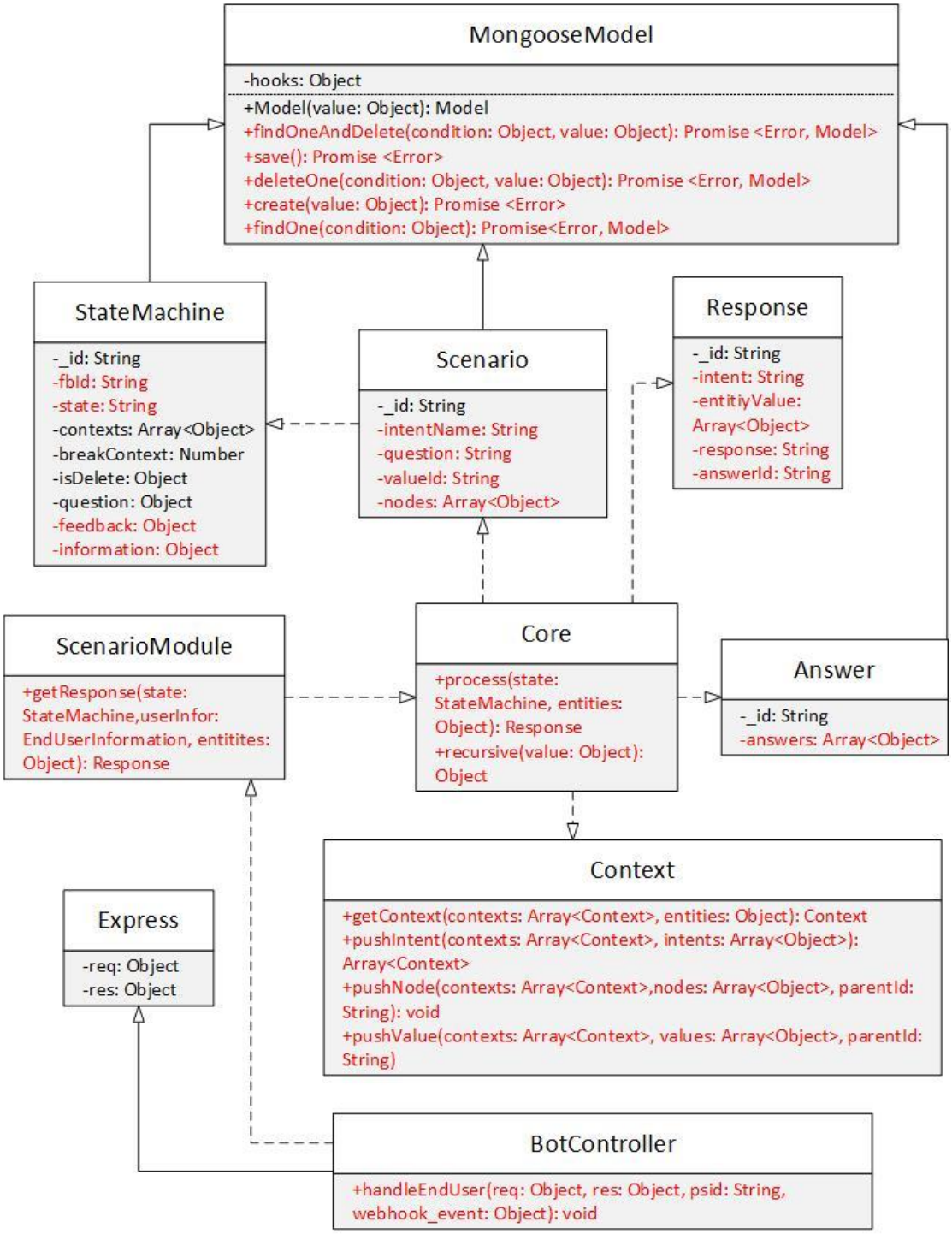


Figure 4-48: Multi-round dialogue class diagram

Class Specification

Scenario

Scenario			
Physical address	Backend/models/Scenario.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	intentName	String	
2	question	Boolean	
3	valueId	String	
4	nodes	Array<Object>	
Operation			

StateMachine

StateMachine			
Physical address	Backend/models/StateMachine.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	fbId	String	
2	state	Boolean	
Operation			

Response

Response			
Physical address	Backend/models/Response.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	intent	String	
2	entityValue	Array<Object>	
3	response	String	
4	answerId	String	
Operation			

Answer

Answer			
Physical address	Backend/models/Answer.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	answers	Array<Object>	
Operation			

ScenarioModule

ScenarioModule			
Physical address	Backend/controllers/ScenarioModule.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A

Operation			
getResonse			
Return Type	Response		
Parameters	Name	Type	Description
	state	StateMachine	
	userInfor	EndUserInformation	
	entities	Object	

Core

Core			
Physical address	Backend/controllers/Core.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
process			
Return Type	void		
Parameters	Name	Type	Description
	state	StateMachine	
	entities	Object	

Context

Context			
Physical address	Backend/controllers/Context.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getContext			
Return Type	Context		
Parameters	Name	Type	Description
	contexts	Array<Context>	
	entities	Object	
pushIntent			
Return Type	Array<Context>		
Parameters	Name	Type	Description
	contexts	Array<Context>	
	intents	Array<Object>	
pushNode			
Return Type	void		
Parameters	Name	Type	Description
	contexts	Array<Context>	
	nodes	Array<Object>	
	parentId	String	
pushValue			
Return Type	void		
Parameters	Name	Type	Description
	contexts	Array<Context>	

	values	Array<Object>	
	parentId	String	

BotController

BotController			
Physical address	Backend/controllers/BotController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
handleEndUser			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	psid	String	
	webhook_event	object	

4.3.4.1.5 Sequence diagram

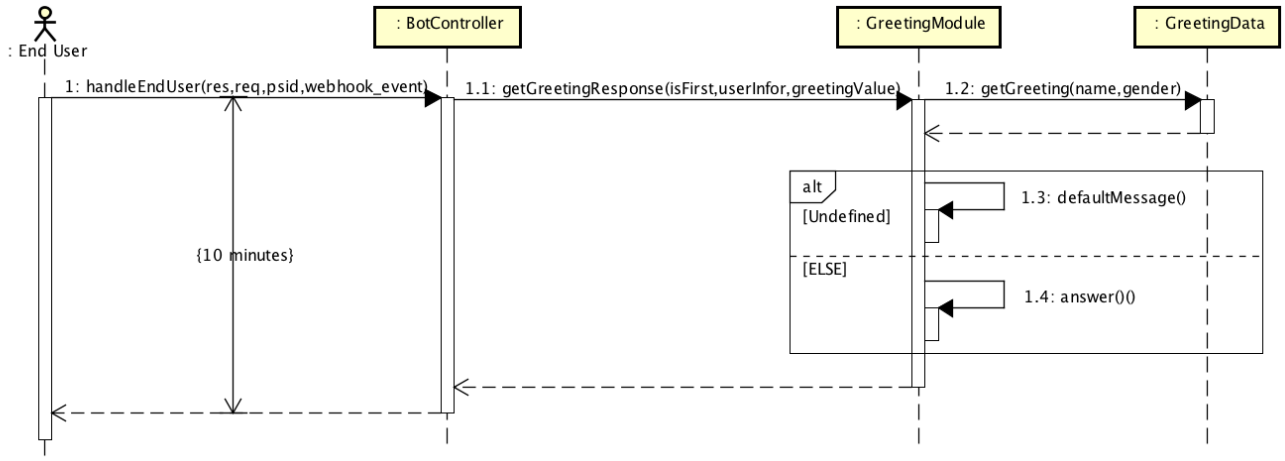


Figure 4-49: One-round dialog sequence diagram

Implementation

Pseudo code

```
var getFinalGreetingResponse ()
    for each greeting intent list
        greeting = greetingData at each index
        if greeting.intentName === intentName
            if intentName === "bat_dau"
                if isFirstInbox
                    return greeting.firstInboxResponse
                else if isLongTimeNoSee
                    return greeting.responseAfterLongTime
            return randomResponse(greeting.responses)
```

```
    return undefined


var randomResponse ()
    index = Math.floor(Math.random() * responses.length)
    if index < length of responses
          return responses at index
    return undefined
```

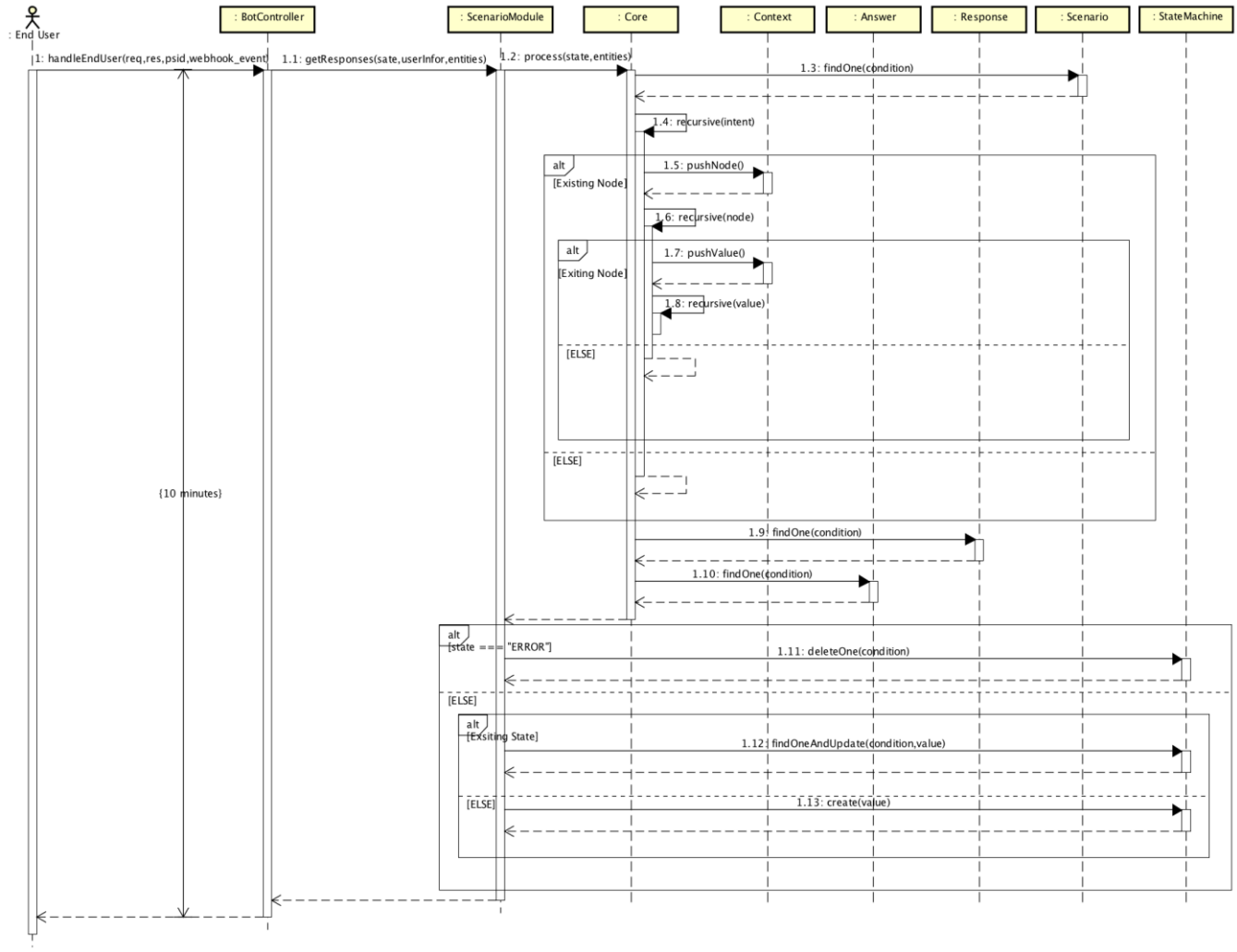


Figure 4-50: Multi-round dialog sequence diagram

Implementation

Pseudo code

```
if stateMachine not exist
    init stateMachine


if context exist
    map entities with context
    if entities map with context
        if stateMachine is continue but currentNode can go award
            use context to continue
```

```
        else
            if use context less than 3 time
                use context to continue
            else
                recreate stateMachine and recreate context with intent
    else if entities do not map with context and stateMachine not continue
        recreate stateMachine and recreate context with intent
else
    create context with intent

if stateMachine continue
    if continue at a value
        recursive at this value to find answer
    else if continue at node
        if entities map with a value of this node
            recursive at this value to find answer
        else
            return stateMachine with continue true at this node
    else
        handleError
else
    if intent and entities valid
        recursive at root to find answer
    else
        return stateMachine with continue true at root
```

4.3.4.2 End user leaves feedback

Conversation Design

Figure 4-51: Leave Feedback Conversation Design

Class diagram

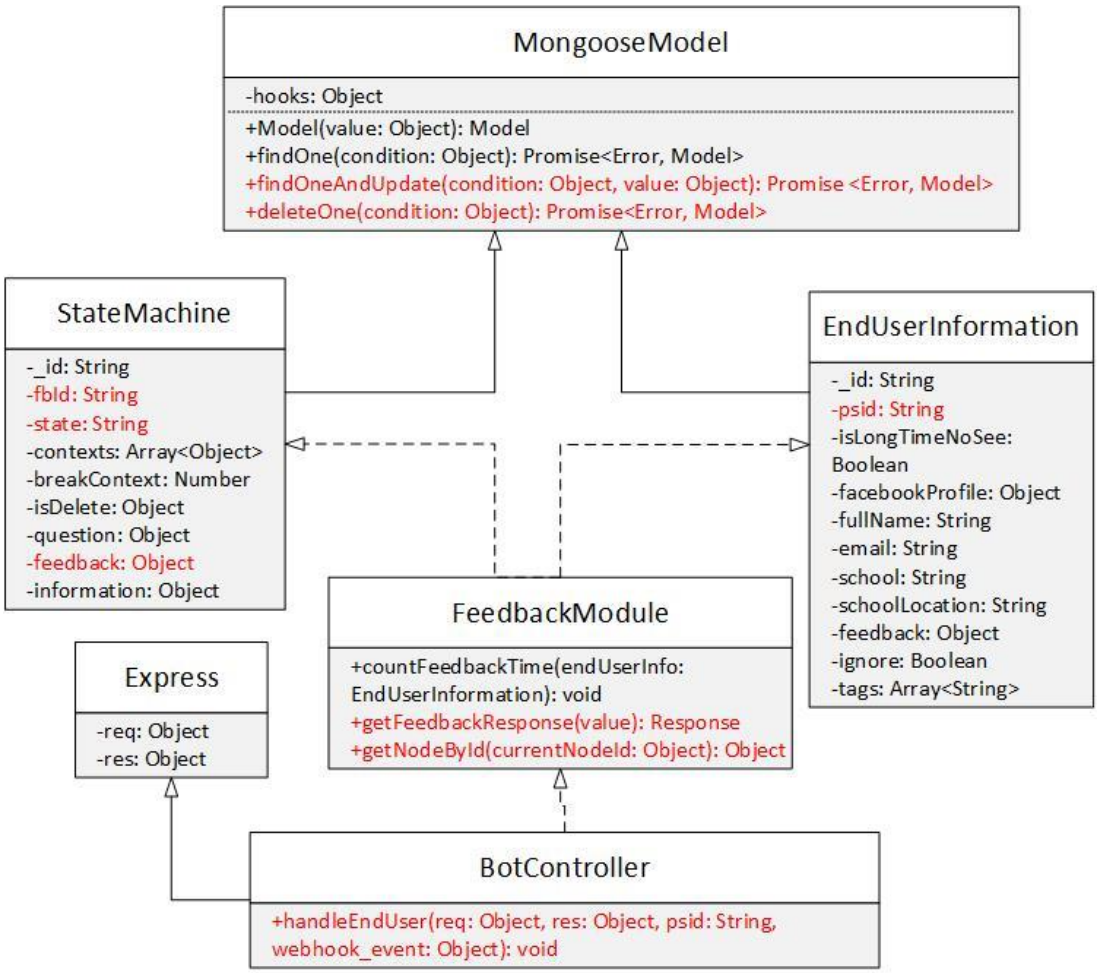


Figure 4-52: Leave feedback class diagram

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend/models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
Operation			

StateMachine

StateMachine			
Physical address	Backend/models/StateMachine.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	fbId	String	
2	state	Boolean	
3	feedback	Object	
Operation			

FeedbackModule

FeedbackModule			
Physical address	Backend/controllers/FeedbackModule.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getFeedbackResponse			
Return Type	Response		
Parameters	Name	Type	Description
	value	Object	
getNodeById			
Return Type	Object		
Parameters	Name	Type	Description
	currentNodeId	Object	

BotController

BotController			
Physical address	Backend/controllers/BotController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
handleEndUser			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	psid	String	
	webhook_event	object	

Sequence diagram

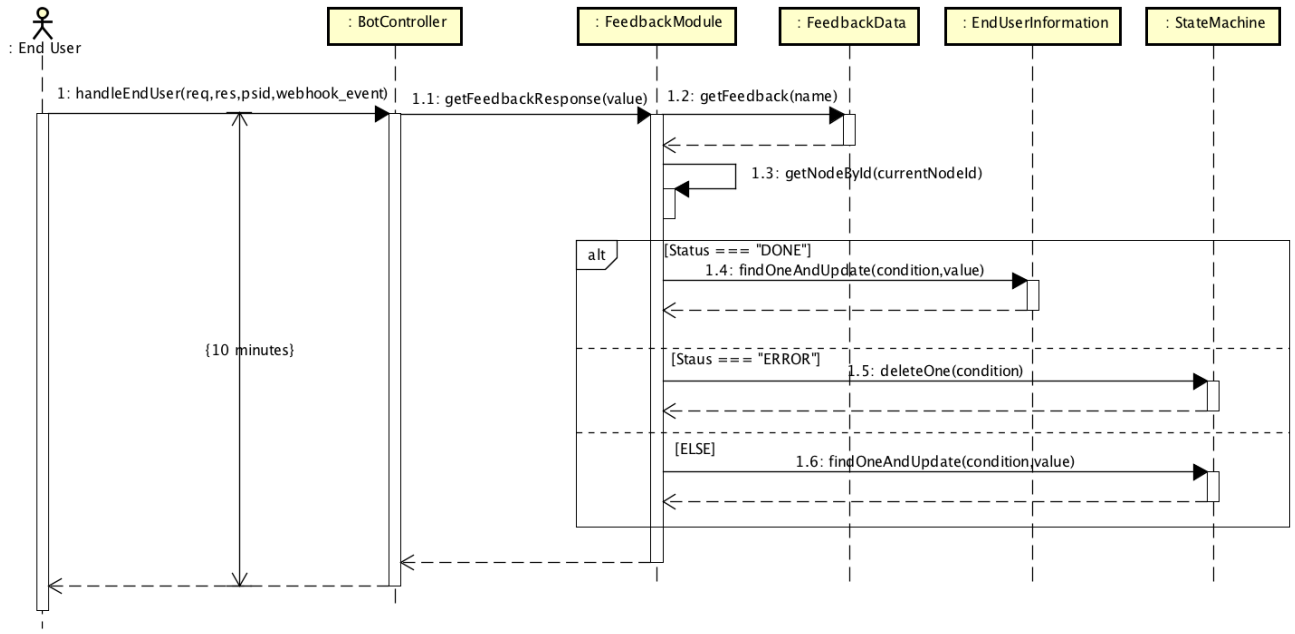


Figure 4-53: Leave Feedback sequence diagram

4.3.4.3 End user leaves personal information

Conversation Design

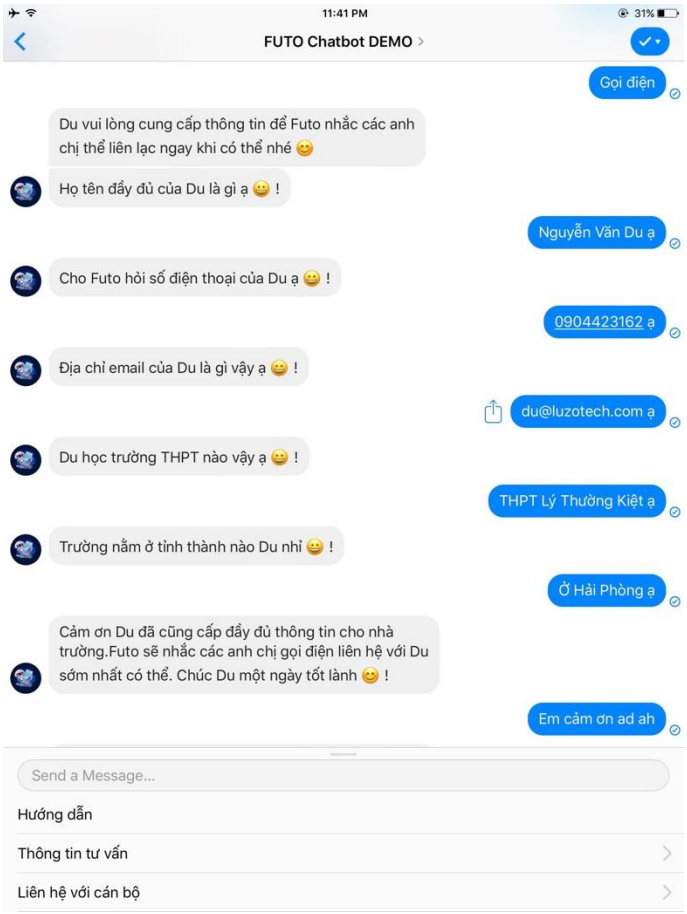


Figure 4-54: Leave Personal Information Conversation

Class Diagram

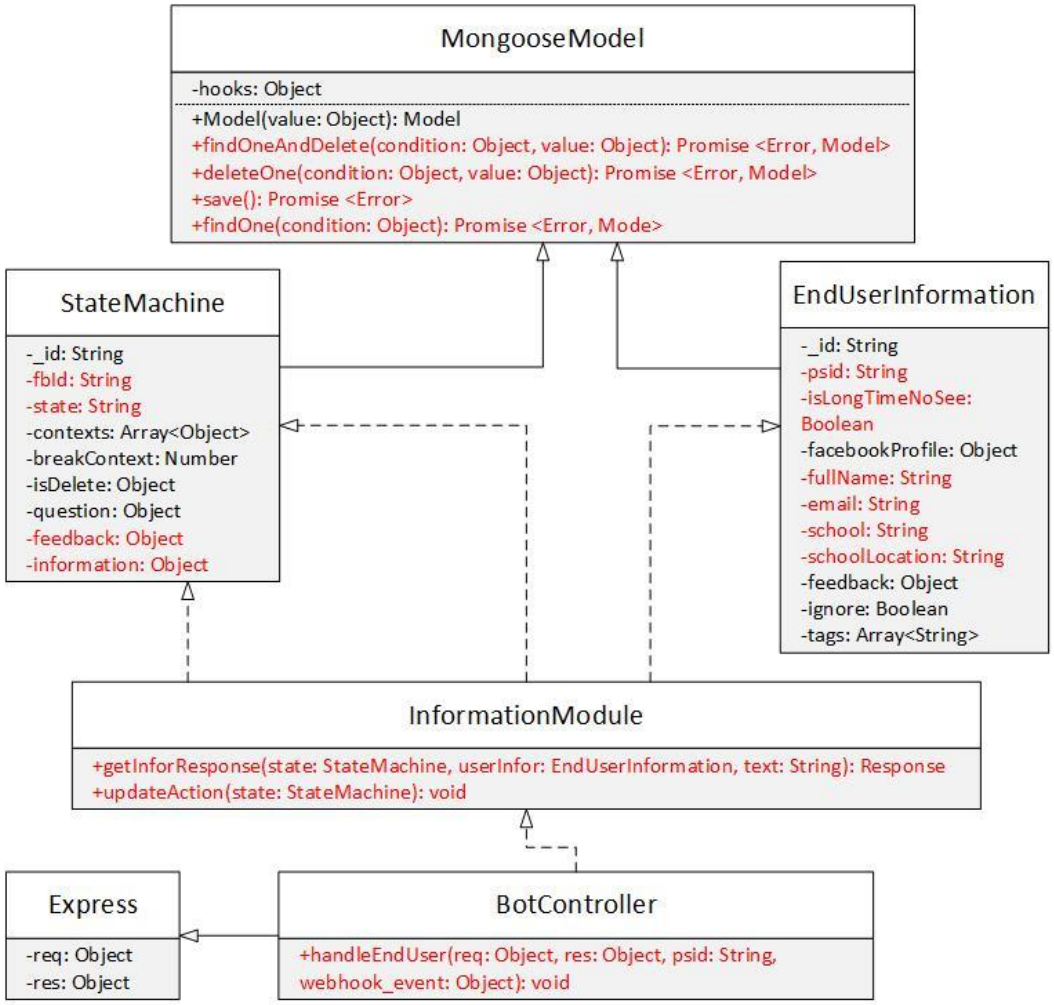


Figure 4-55: Leave information class diagram

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend/models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
2	isLongTimeNoSee	Boolean	
3	fullName	String	
4	email	String	
5	school	String	
6	schoolLocation	String	
Operation			

StateMachine

StateMachine

Physical address	Backend/models/StateMachine.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	fbId	String	
2	state	Boolean	
3	information	Object	
Operation			

InformationModule

InformationModule			
Physical address	Backend/controllers/InformationModule.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getInforResponse			
Return Type	Response		
Parameters	Name	Type	Description
	state	StateMachine	
	userInfor	EndUserInformation	
	text	String	
updateAction			
Return Type	void		
Parameters	Name	Type	Description
	state	StateMachine	

BotController

BotController			
Physical address	Backend/controllers/BotController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
handleEndUser			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	psid	String	
	webhook_event	object	

Sequence Diagram

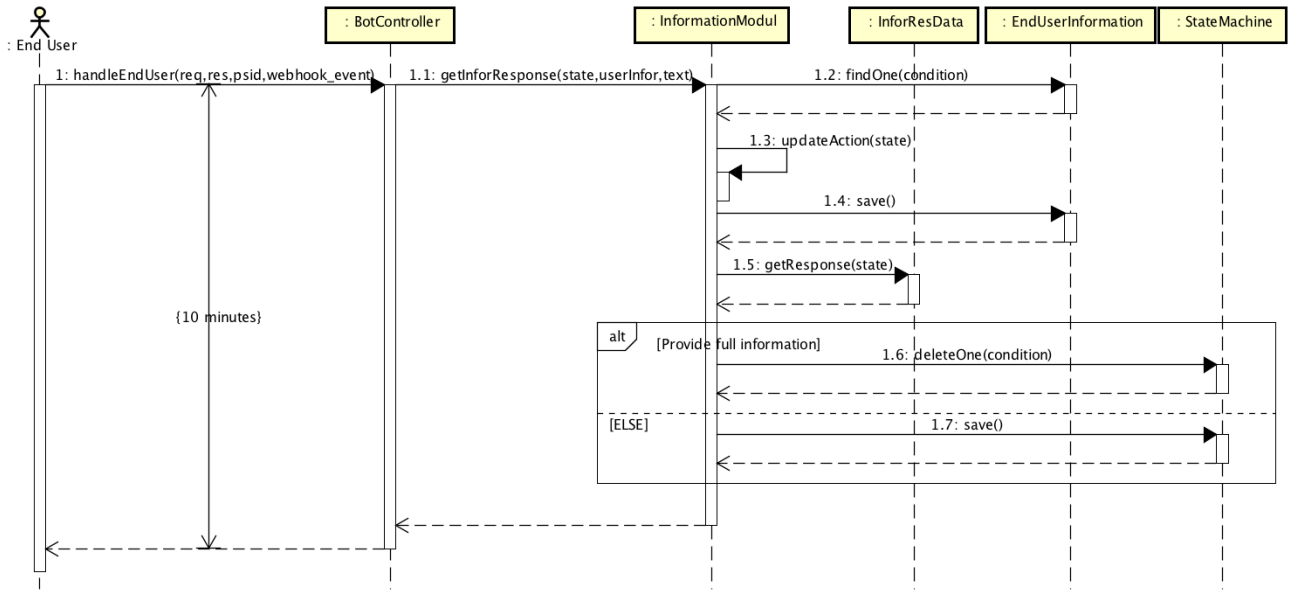


Figure 4-56: Leave personal information sequence diagram

4.3.4.4 Staff resets password

Screen Design

Figure 4-57: Staff resets password screen design

Class Diagram

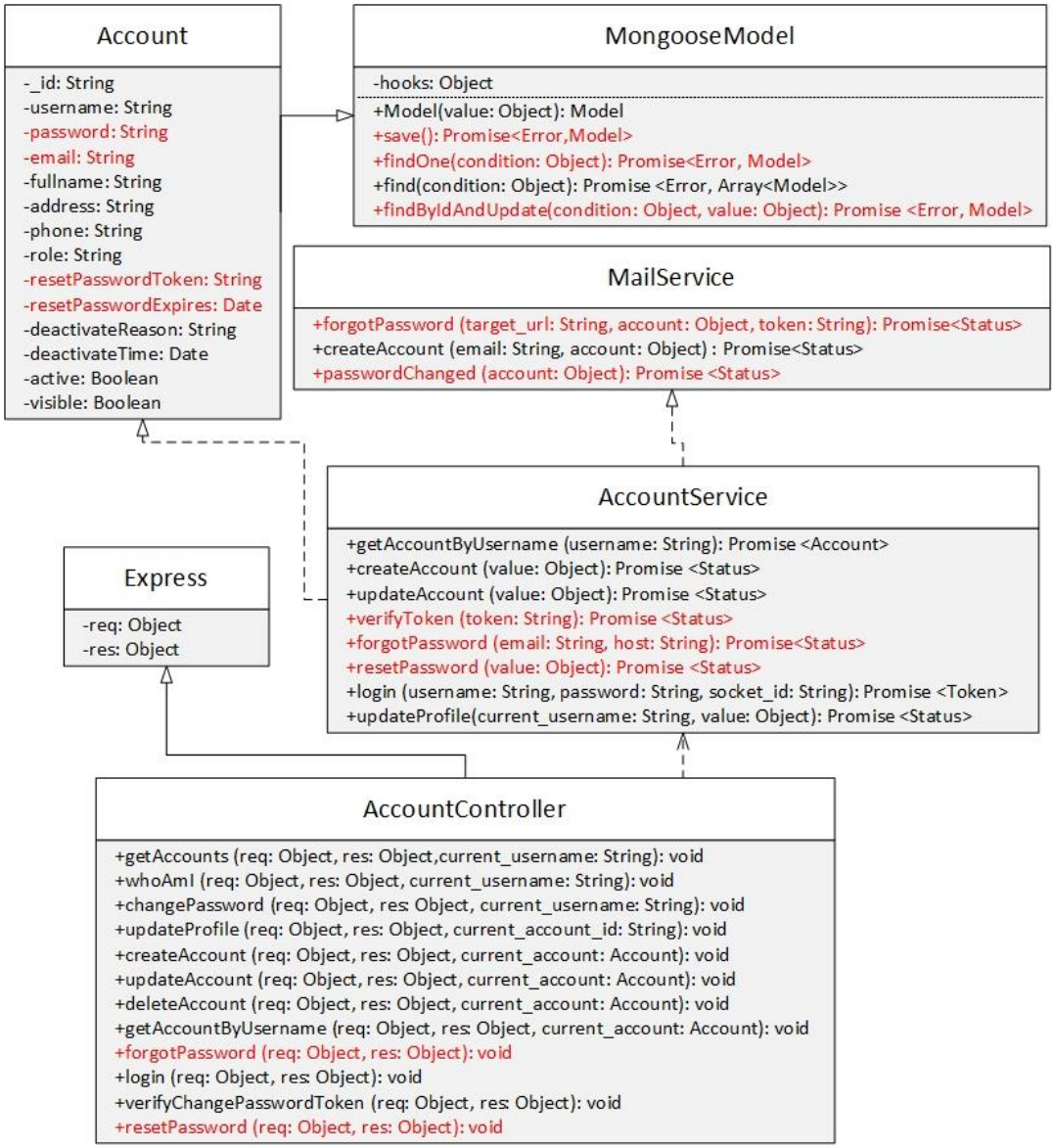


Figure 4-58: Staff resets password class diagram

Class Specification

Account

Account			
Physical address	Backend/models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	email	String	
2	password	String	
3	resetPasswordToken	String	
4	resetPasswordExpires	Date	
Operation			

AccountService

AccountService

Physical address	Backend/controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
forgotPassword			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	email	String	
	host	String	
resetPassword			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	value	object	

MailService

MailService			
Physical address	Backend/controllers/MailService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
passwordChanged			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	account	object	
forgotPassword			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	target_url	String	
	account	object	
	token	String	

AccountController

AccountController			
Physical address	Backend/controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
resetPassword			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
forgotPassword			
Return Type	void		

Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

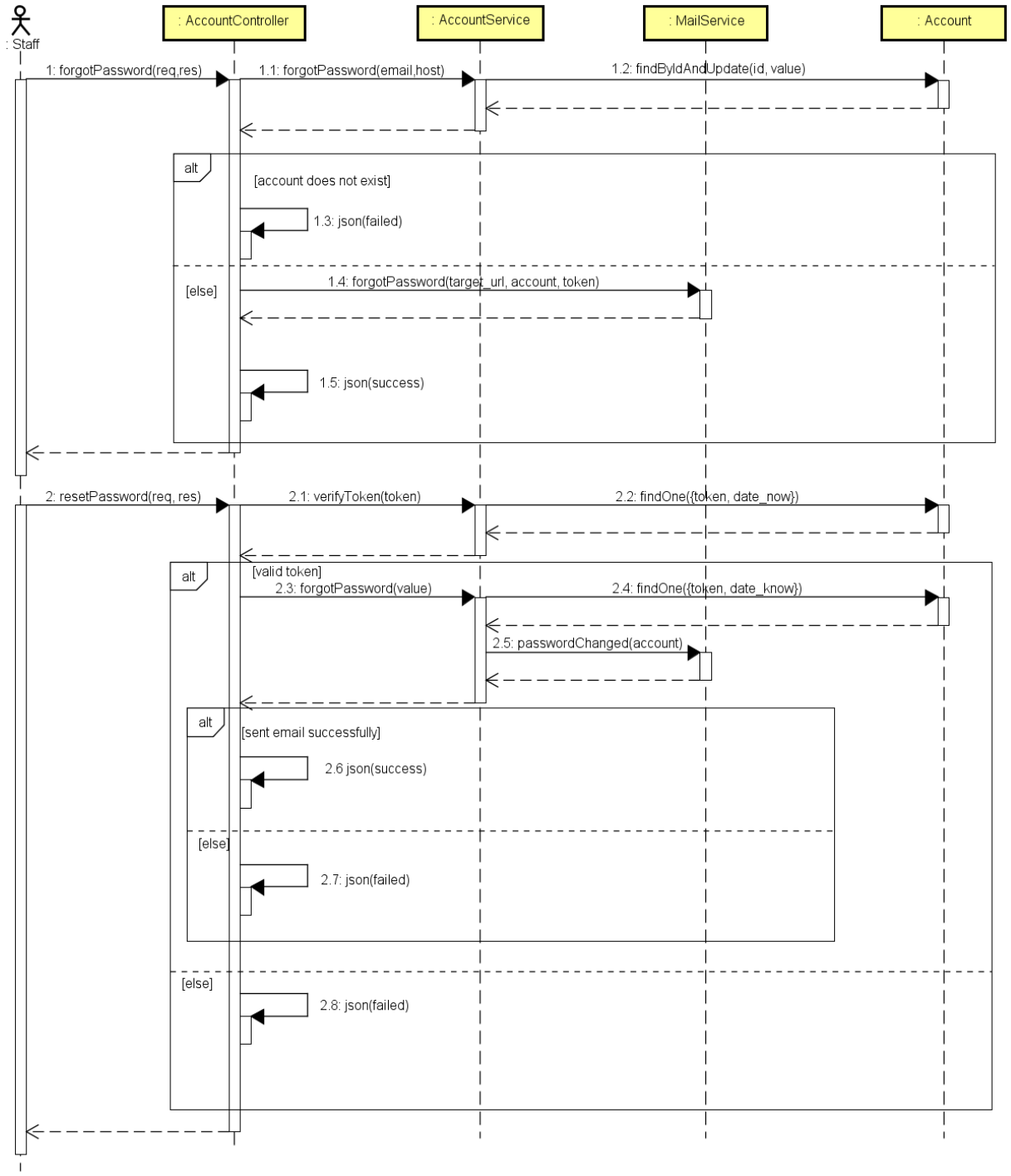


Figure 4-59: Staff resets password sequence diagram

4.3.4.5 Staff logins by ASC account

Screen Design

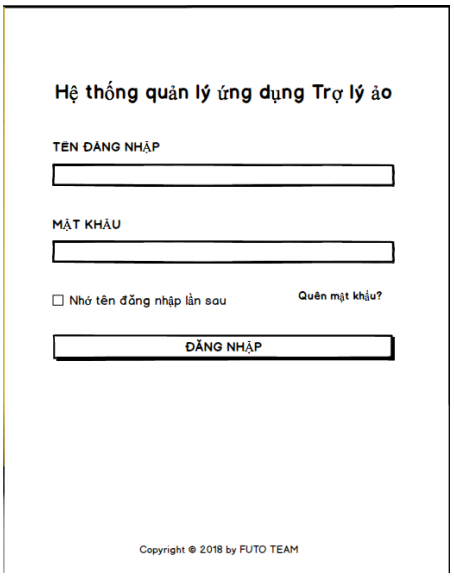


Figure 4-60: Staff logins by ASC account screen design

Class Diagram

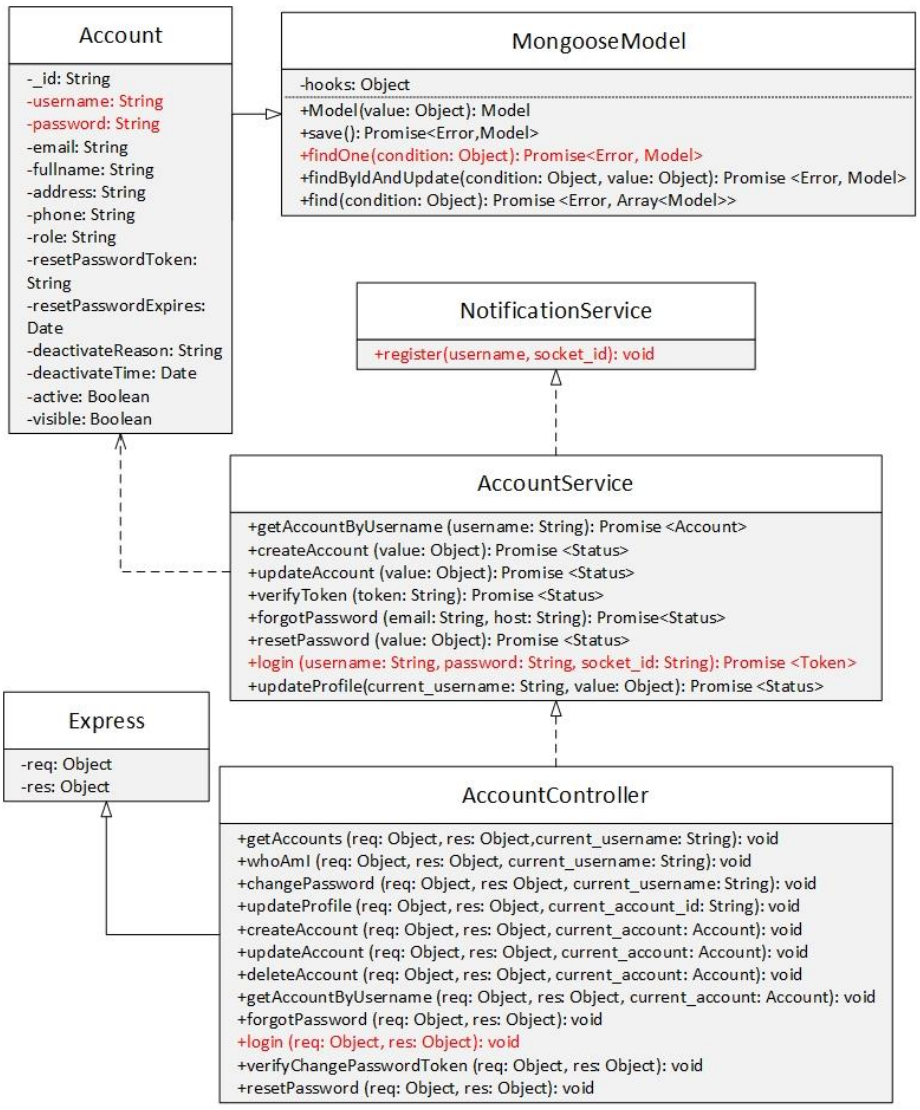


Figure 4-61: Staff logins by ASC account class diagram.

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	username	String	
2	password	String	
Operation			
N/A	N/A		

AccountService

AccountService			
Physical address	Backend/controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
login			
Return Type	Promise<Token>		
Parameters	Name	Type	Description
	username	String	
	password	String	
	socket_id	String	

AccountController

AccountController			
Physical address	Backend/controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
login			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

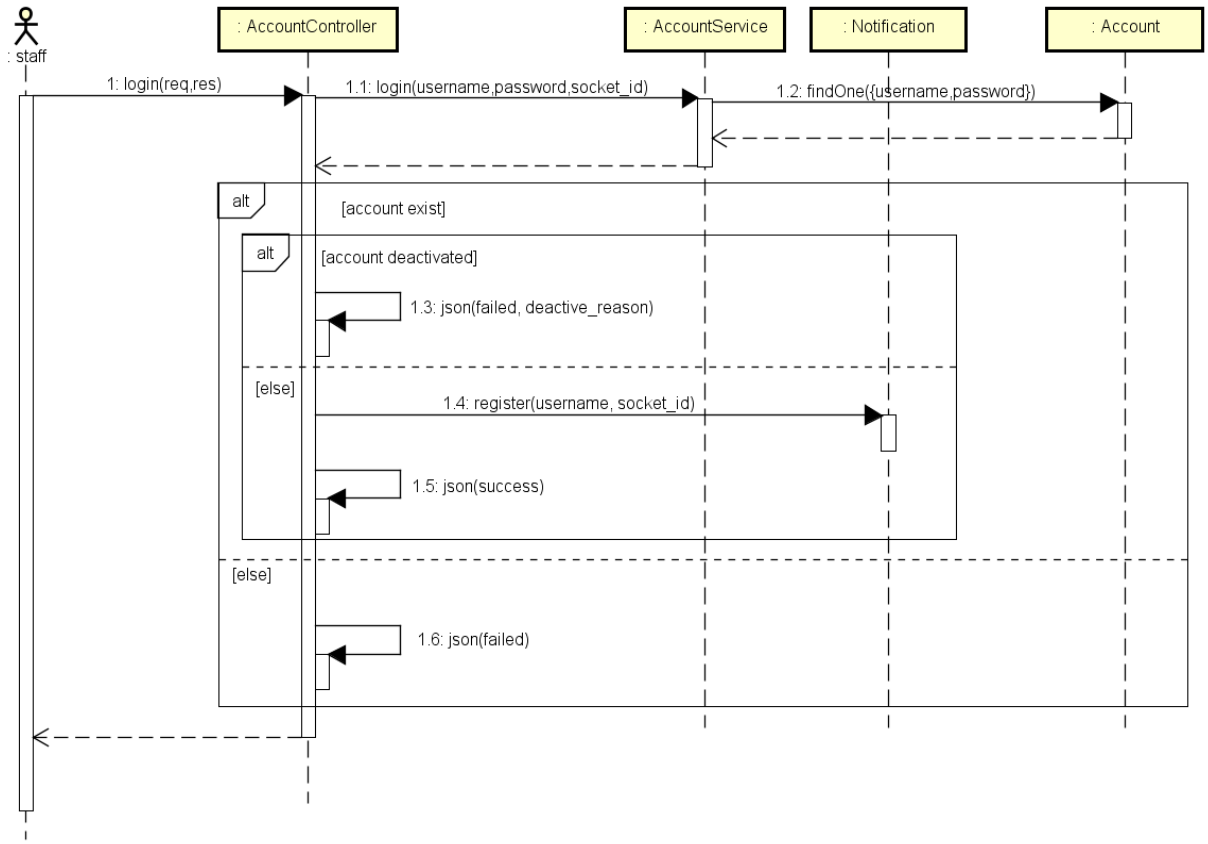


Figure 4-62: Staff logins by ASC account sequence diagram

4.3.4.6 Staff views statistics

Screen Design

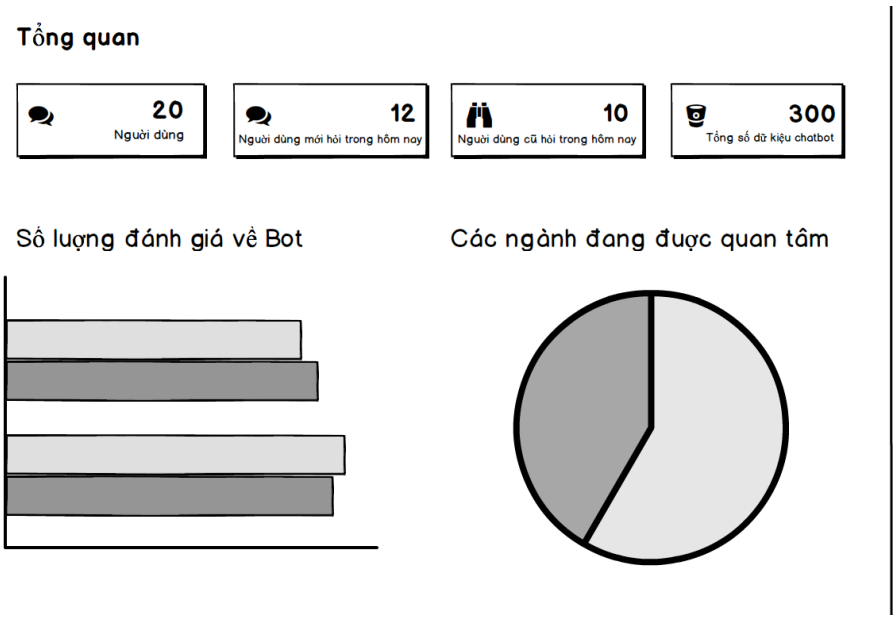


Figure 4-63: Staff views statistics screen design

Class Diagram

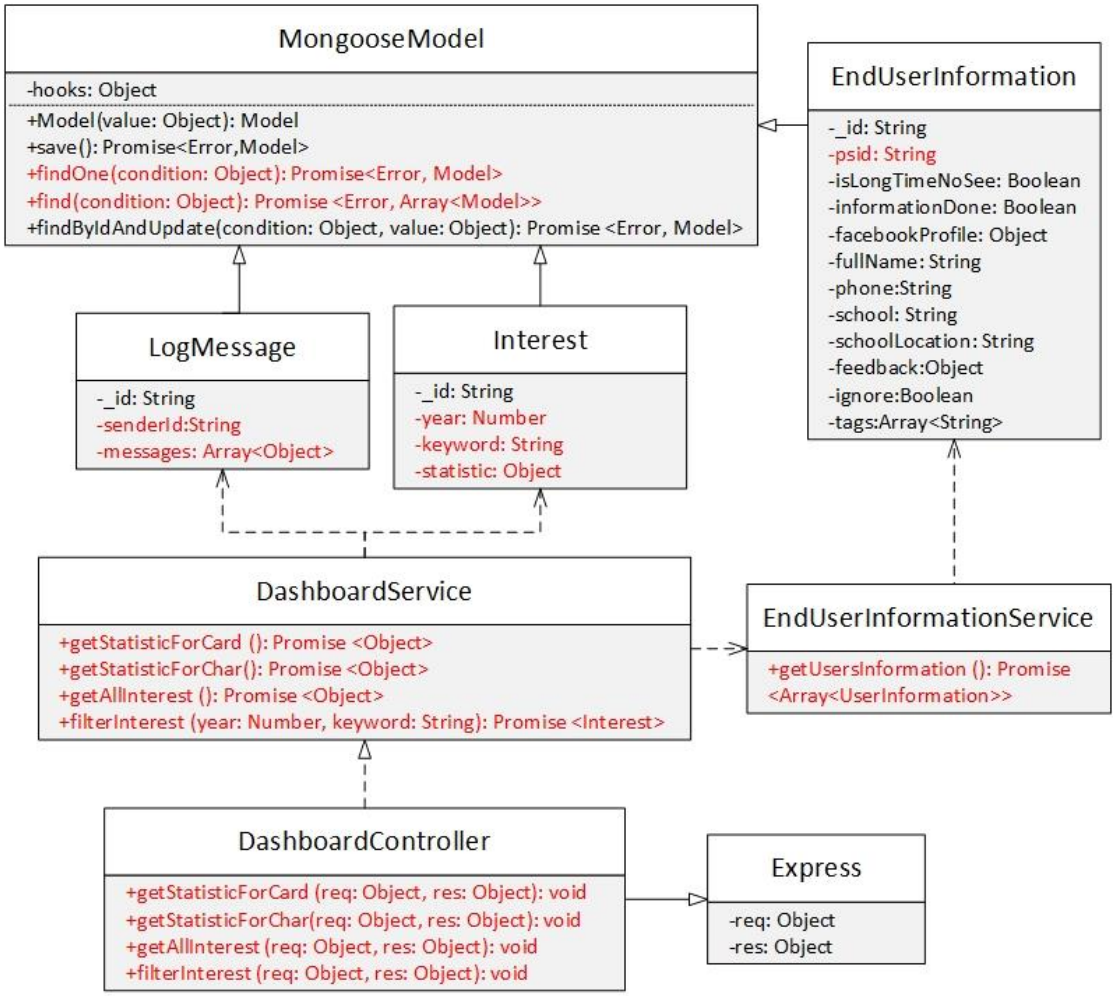


Figure 4-64: View statistics class diagram

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend /models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
Operation			
N/A	N/A		

LogMessage

LogMessage			
Physical address	Backend /models/LogMessage.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	_id	object	
2	senderId	String	

3	messages	Array<object>	
Operation			
N/A	N/A		

Interests

Interests			
Physical address	Backend /models/Interests.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	year	number	
2	keyword	String	
3	statistics	object	
Operation			
N/A	N/A		

DashboardService

DashboardService			
Physical address	Backend /controllers/DashboardService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getStatisticForCard			
Return Type	Promise<object>		
Parameters	Name	Type	Description
	N/A	N/A	N/A
getStatisticForChart			
Return Type	Promise<object>		
Parameters	Name	Type	Description
	N/A	N/A	N/A
getAllInterest			
Return Type	Promise<object>		
Parameters	Name	Type	Description
	N/A	N/A	N/A
filterInterest			
Return Type	Promise<Interest>		
Parameters	Name	Type	Description
	year	Number	
	keyword	String	

EndUserInformationService

EndUserInformationService			
Physical address	Backend /controllers/EndUserInformationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getUserInformation			
Return Type	Promise<Array<EndUserInformation>>		

Parameters	Name	Type	Description
	N/A	N/A	N/A

DashboardController

DashboardController			
Physical address	Backend /controllers/DashboardController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getStatisticForCard			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
getStatisticForChart			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
getAllInterest			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
filterInterest			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

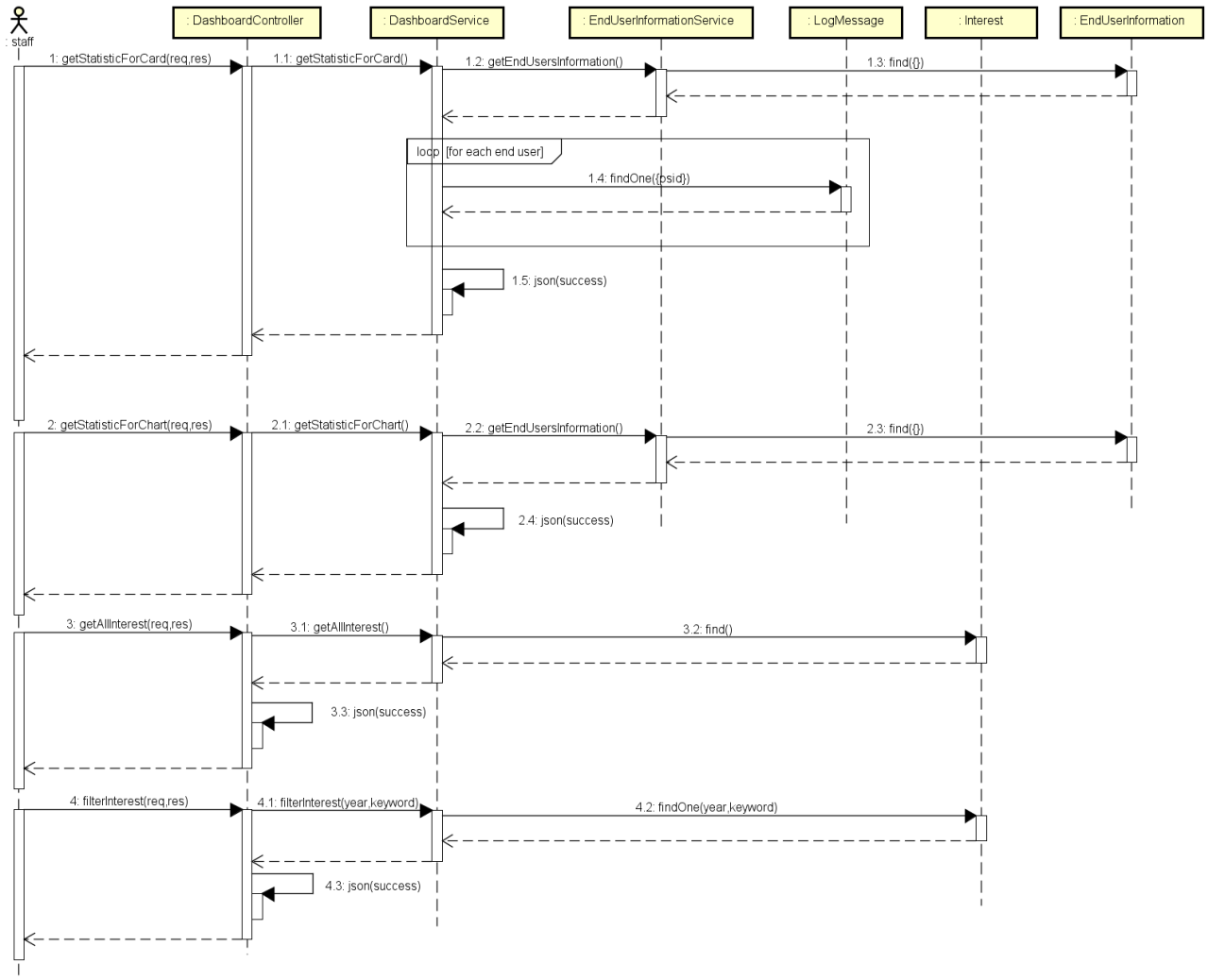


Figure 4-65: View statistics sequence diagram

4.3.4.7 Staff views profile

Screen Design

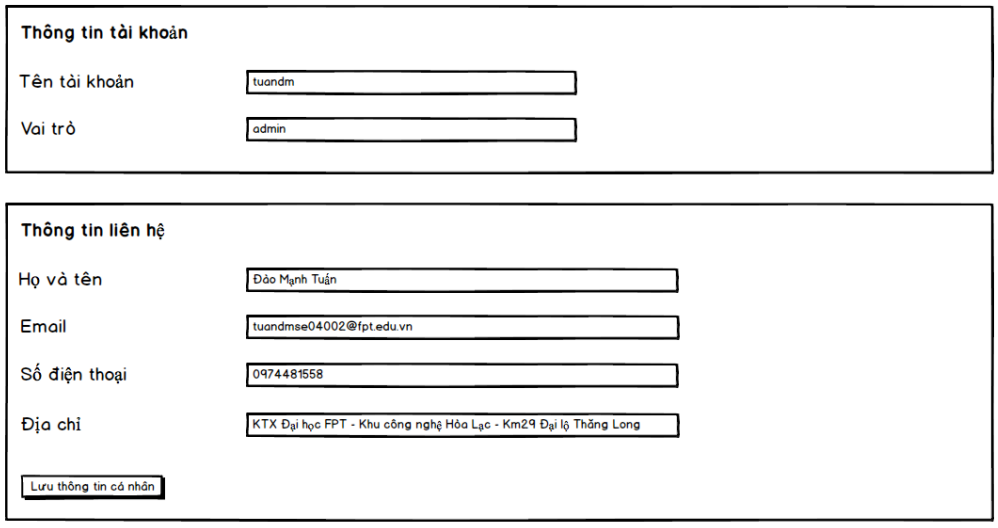


Figure 4-66: Staff views profile screen design

Class Diagram

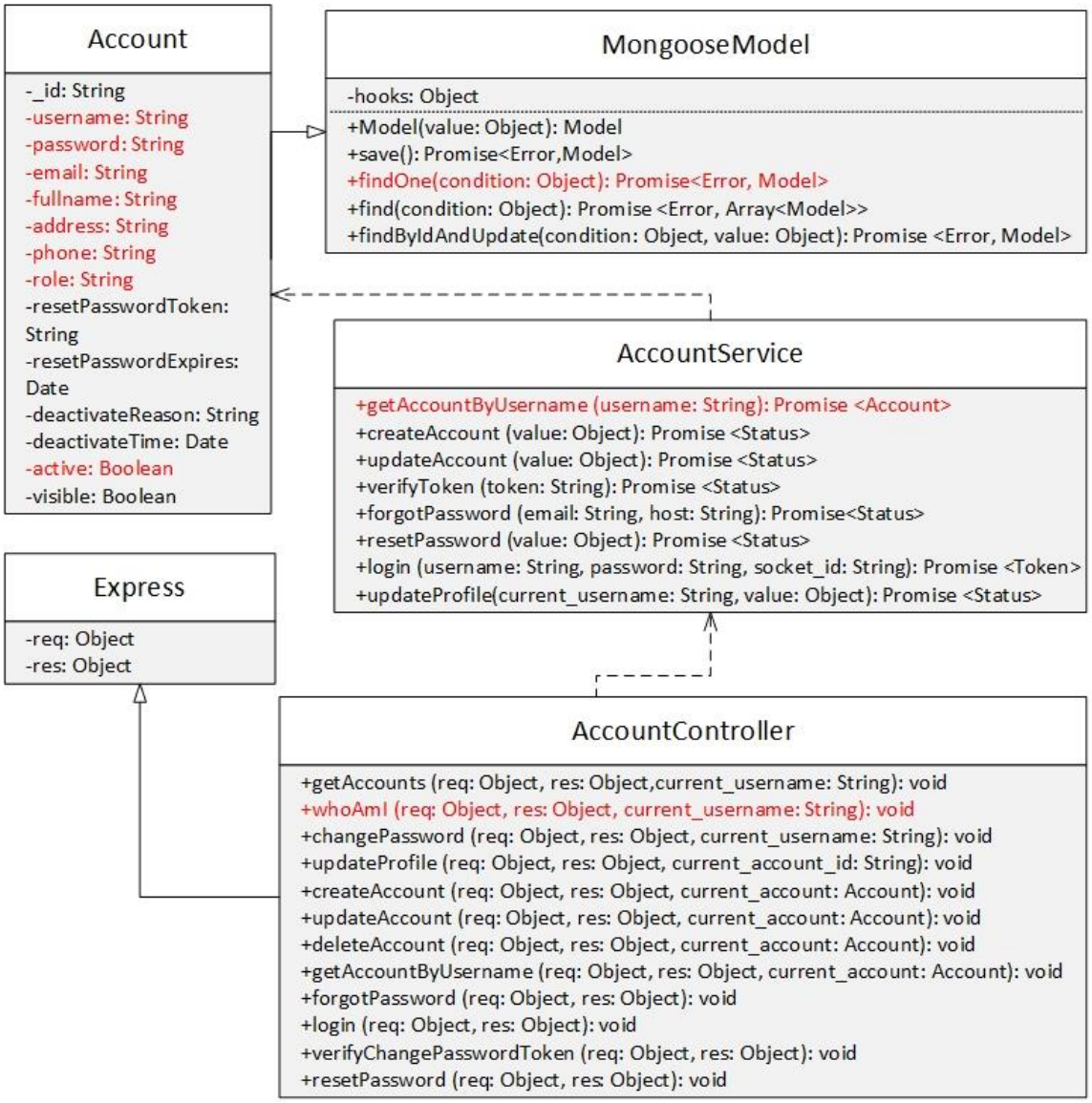


Figure 4-67: Staff views profile class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	username	String	
2	email	String	
3	fullname	String	
4	address	String	
5	phone	String	
6	role	String	
Operation			
N/A	N/A		

AccountService

AccountService			
Physical address	Backend /controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getAccountByUsername			
Return Type	Promise<Account>		
Parameters	Name	Type	Description
	username	String	

AccountController

AccountController			
Physical address	Backend /controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
whoAmI			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	current_username	String	

Sequence Diagram

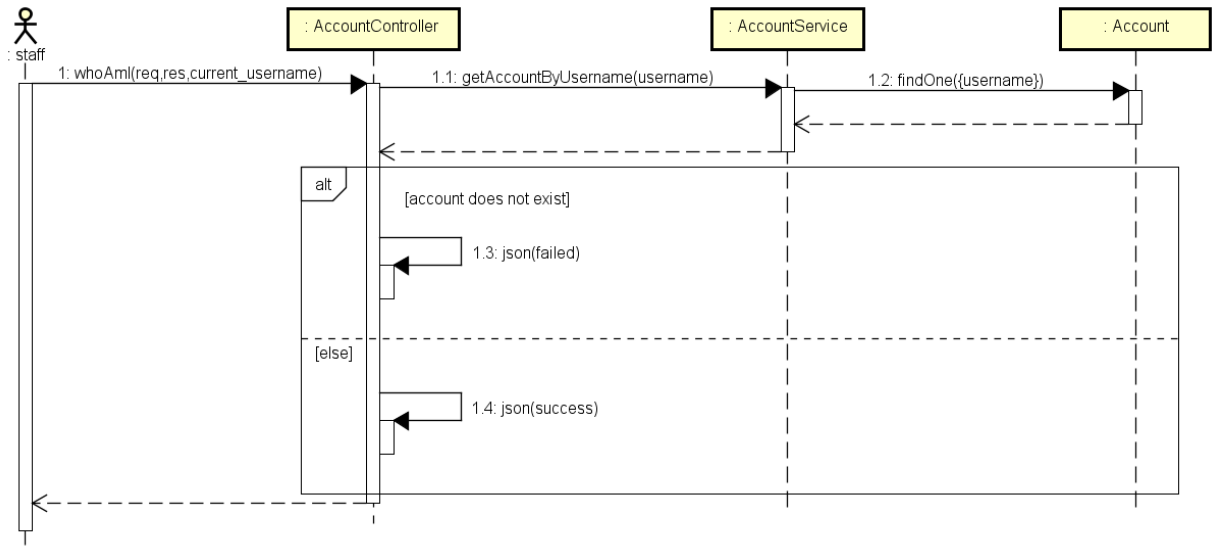


Figure 4-68: Staff views profile sequence diagram

4.3.4.8 Staff updates profile

Screen Design

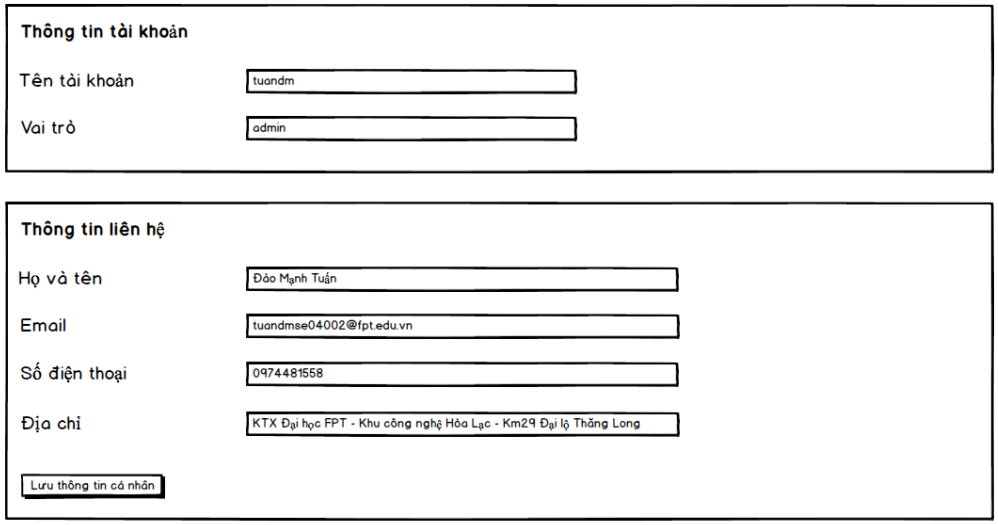


Figure 4-69: Staff updates profile screen design

Class Diagram

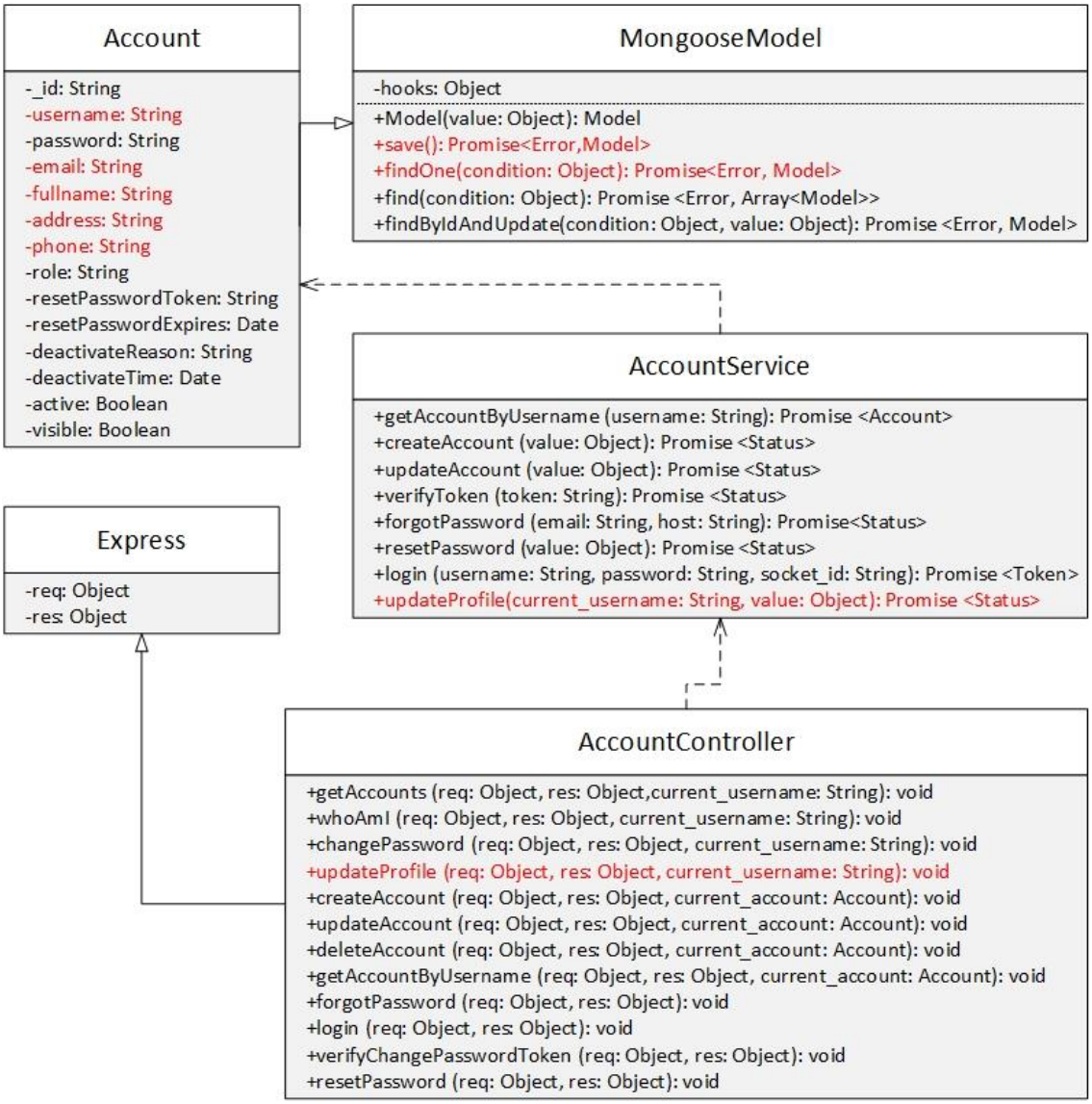


Figure 4-70: Staff updates profile class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	fullname	String	
2	email	String	
3	phone	String	
4	address	String	
Operation			

AccountService

AccountService	
Physical address	Backend /controllers/AccountService.js

Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getAccountByUsername			
Return Type	Promise<Account>		
Parameters	Name	Type	Description
	username	String	
updateProfile			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	current_username	String	
	value	object	

AccountController

AccountController			
Physical address	Backend /controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
updateProfile			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	current_account_id	String	

Sequence Diagram

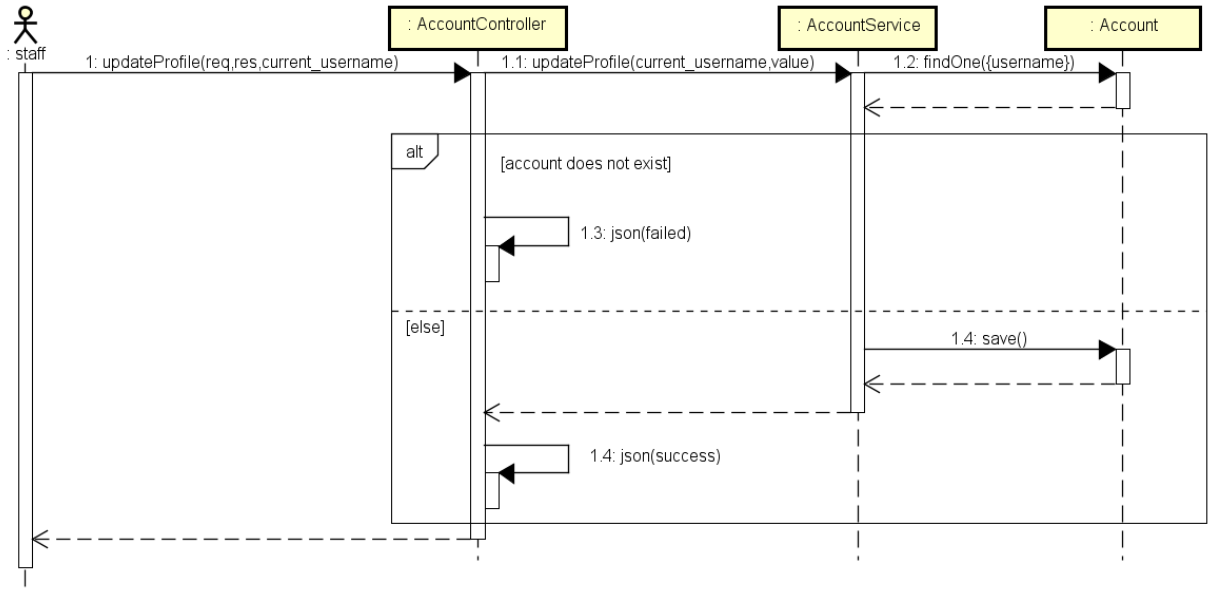


Figure 4-71: Staff updates profile sequence diagram

4.3.4.9 Staff changes password

Screen Design

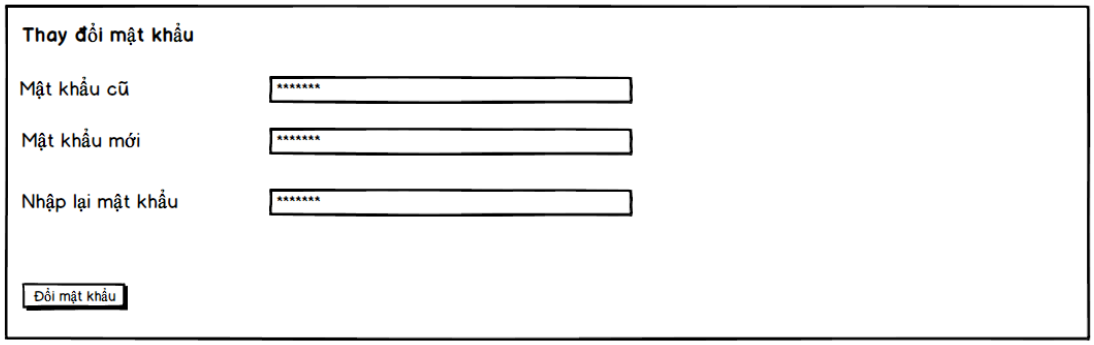


Figure 4-72: Staff changes password screen design

Class Diagram

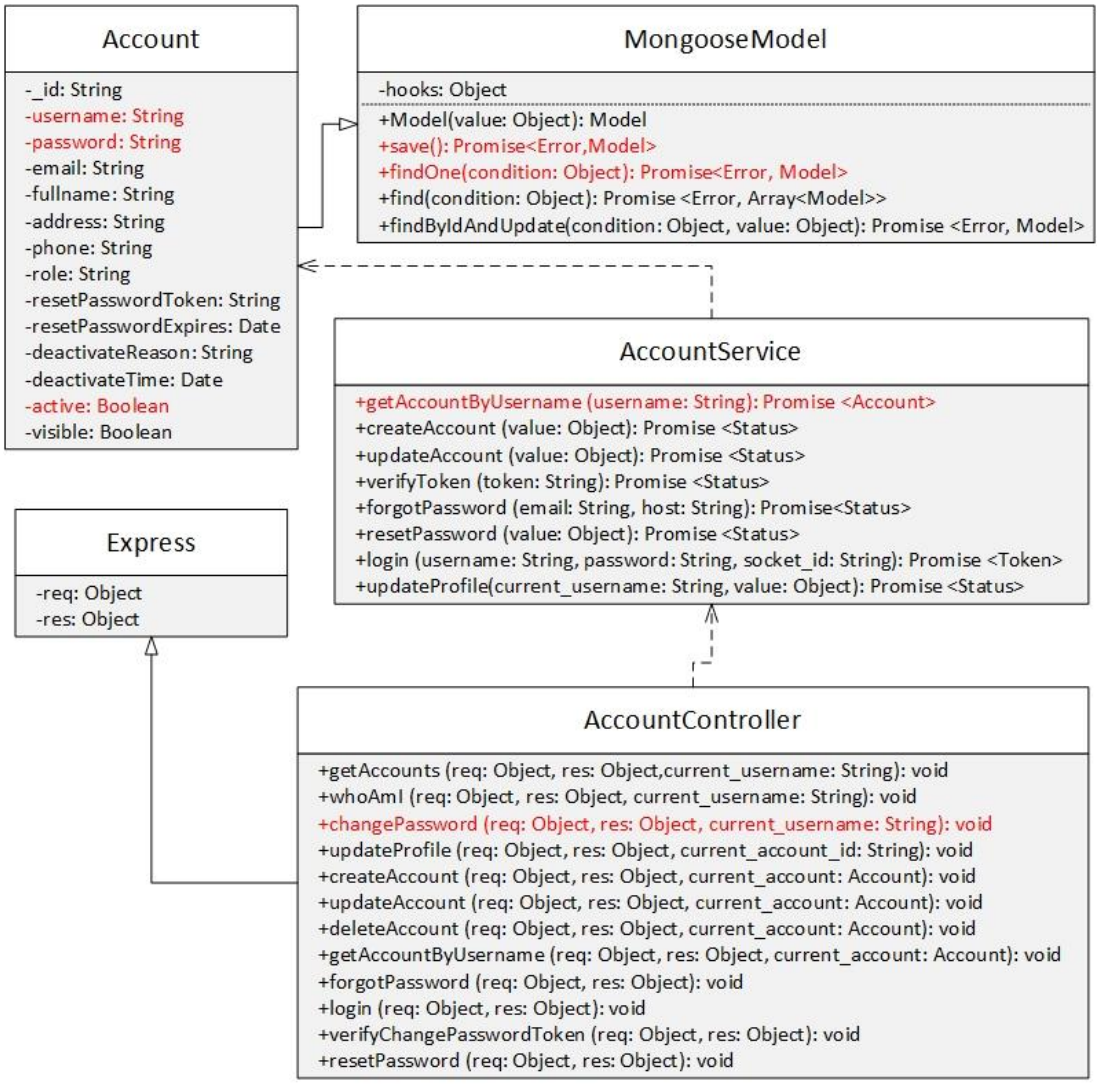


Figure 4-73: Staff changes password class diagram

Class Specification

Account

Account

Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	username	String	
2	password	String	
Operation			
N/A	N/A		

AccountService

AccountService			
Physical address	Backend /controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getAccountByUsername			
Return Type	Promise<Account>		
Parameters	Name	Type	Description
	username	String	

AccountController

AccountController			
Physical address	Backend /controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
changePassword			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	current_username	String	

Sequence Diagram

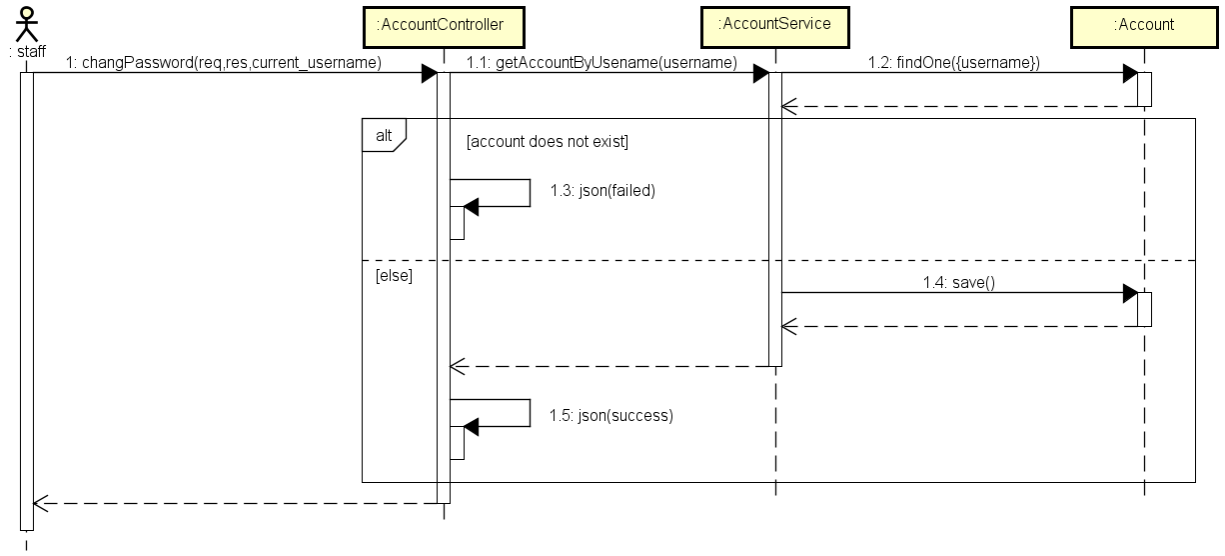


Figure 4-74: Staff changes password sequence diagram

4.3.4.10 Staff views account list

Screen Design

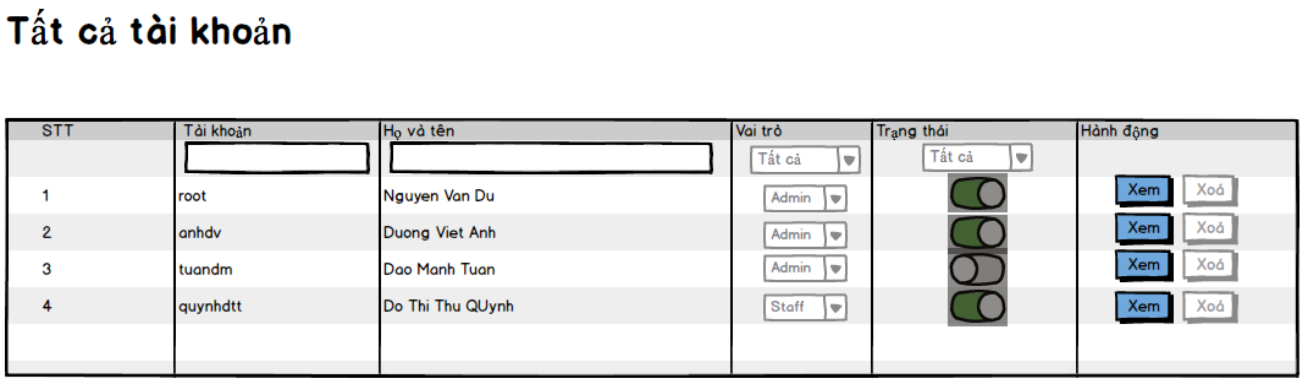


Figure 4-75: Staff views account list

Class Diagram

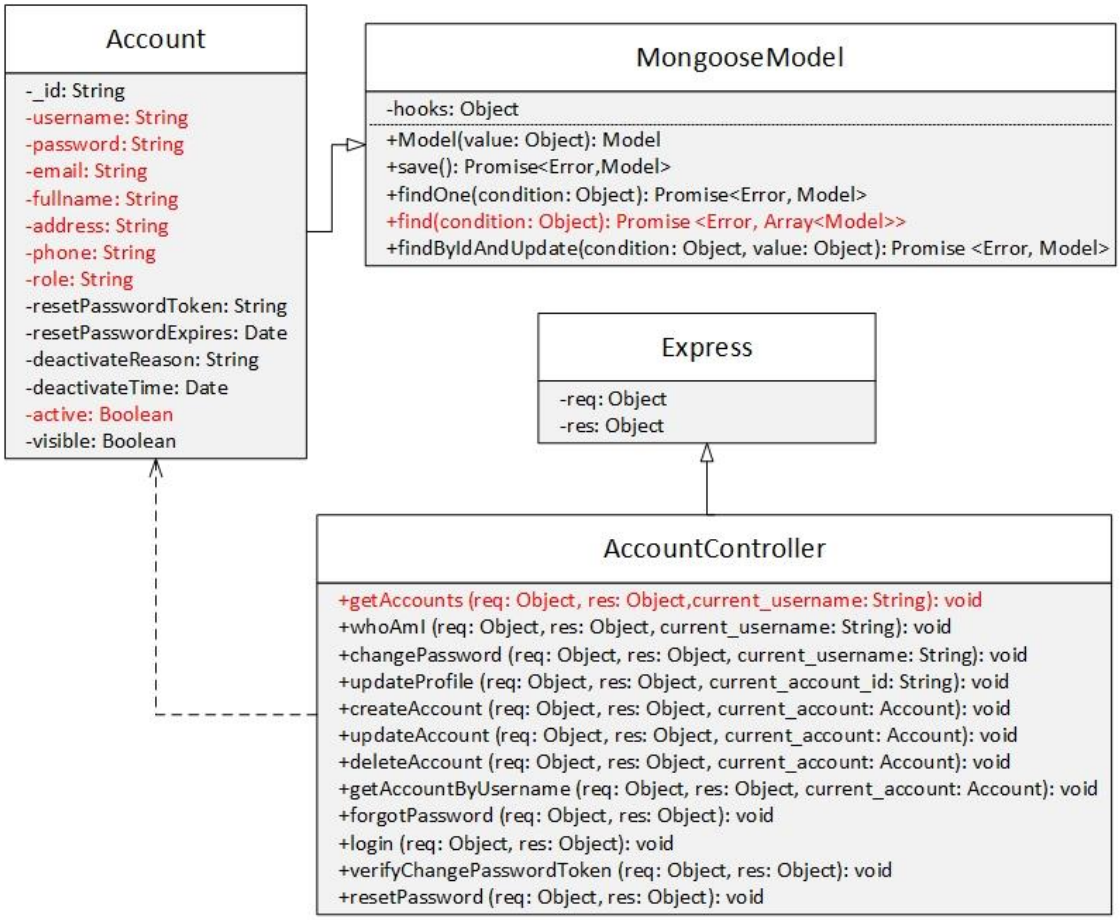


Figure 4-76: Staff views account list class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	username	String	
2	email	String	
3	fullname	String	
4	address	String	
5	phone	String	
6	role	String	
7	active	Boolean	
Operation			
N/A	N/A		

AccountController

AccountController	
Physical address	Backend /controllers/AccountController.js

Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getAccounts			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	current_username	String	

Sequence Diagram

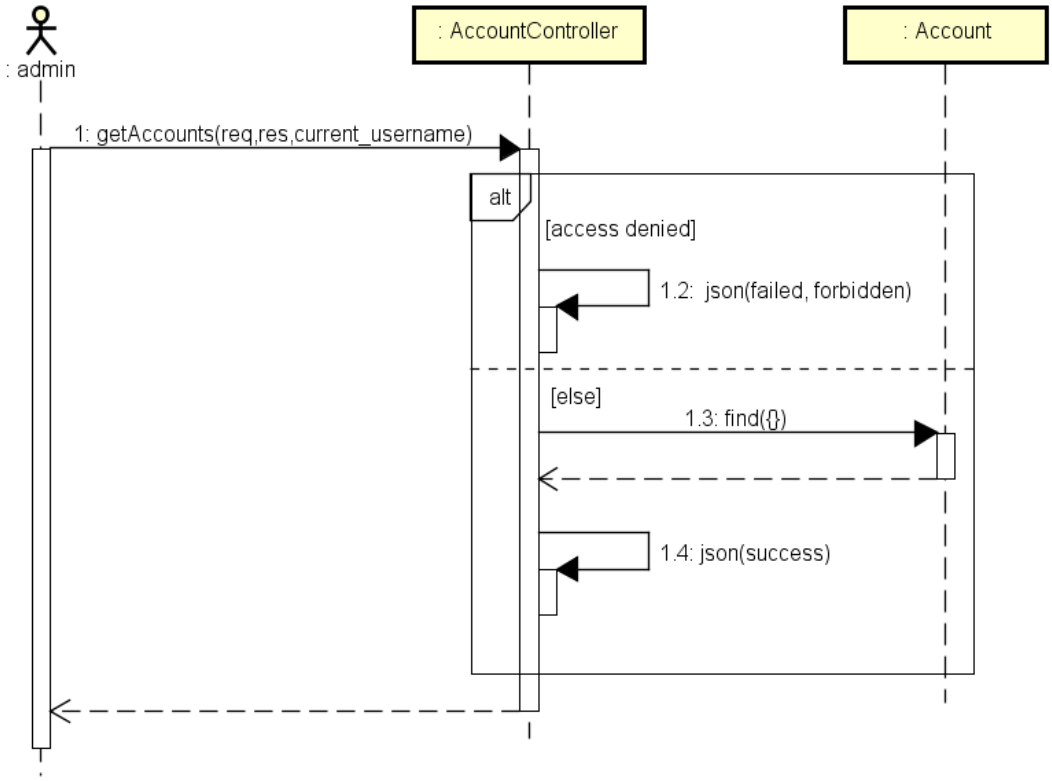


Figure 4-77: Staff views account list sequence diagram

4.3.4.11 Staff views account information detail

Screen Design

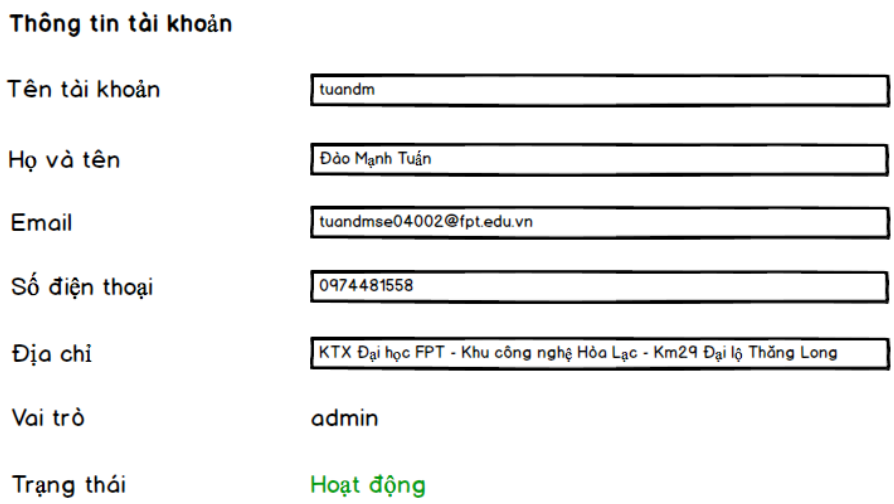


Figure 4-78: Staff views account information detail screen design

Class Diagram

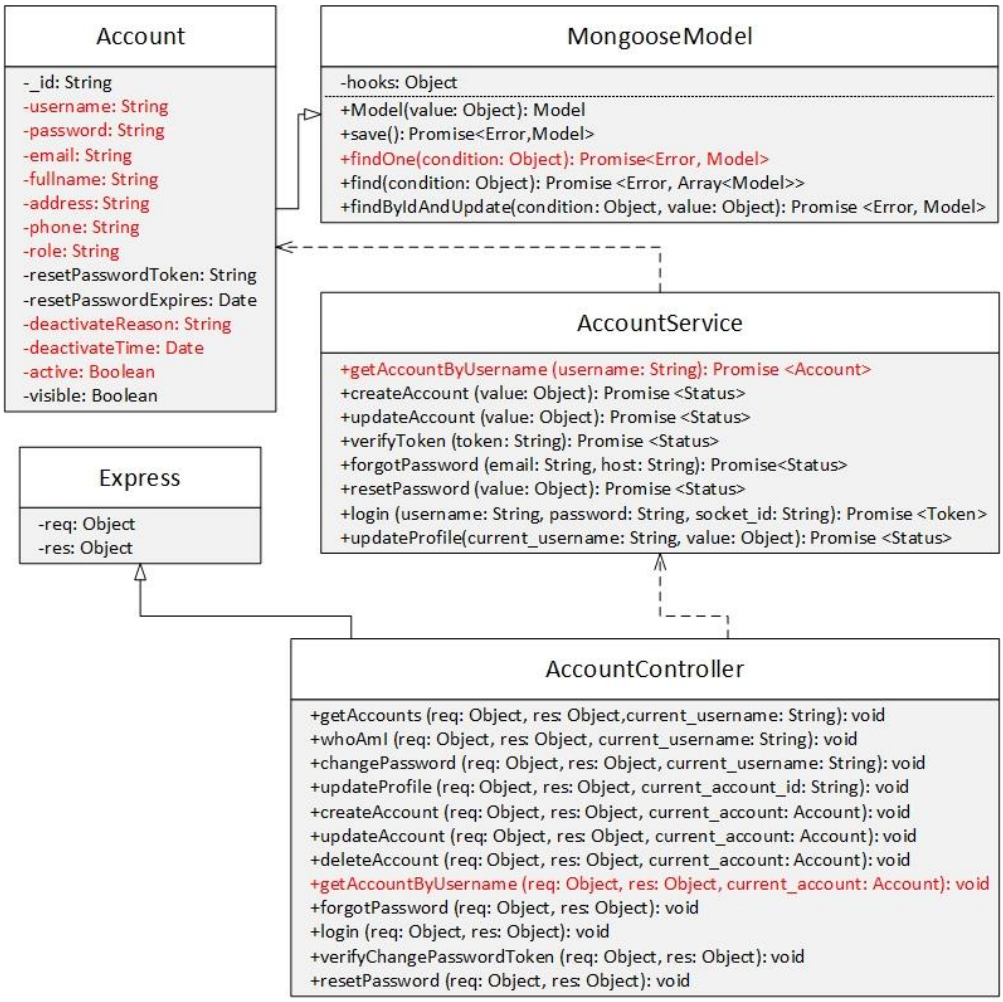


Figure 4-57: Staff view account information detail class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	username	String	
2	email	String	
3	fullname	String	
4	address	String	
5	phone	String	
6	role	String	
7	active	Boolean	
8	deactivatedReason	String	
9	deactiveTime	Date	
Operation			
N/A	N/A		

AccountService

AccountService			
Physical address	Backend /controllers/AccountService.js		
Base class			
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getAccountByUsername			
Return Type	Promise<Account>		
Parameters	Name	Type	Description
	username	String	

AccountController

AccountController			
Physical address	Backend /controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getAccountByUsername			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	
	current_account	Account	

Sequence Diagram

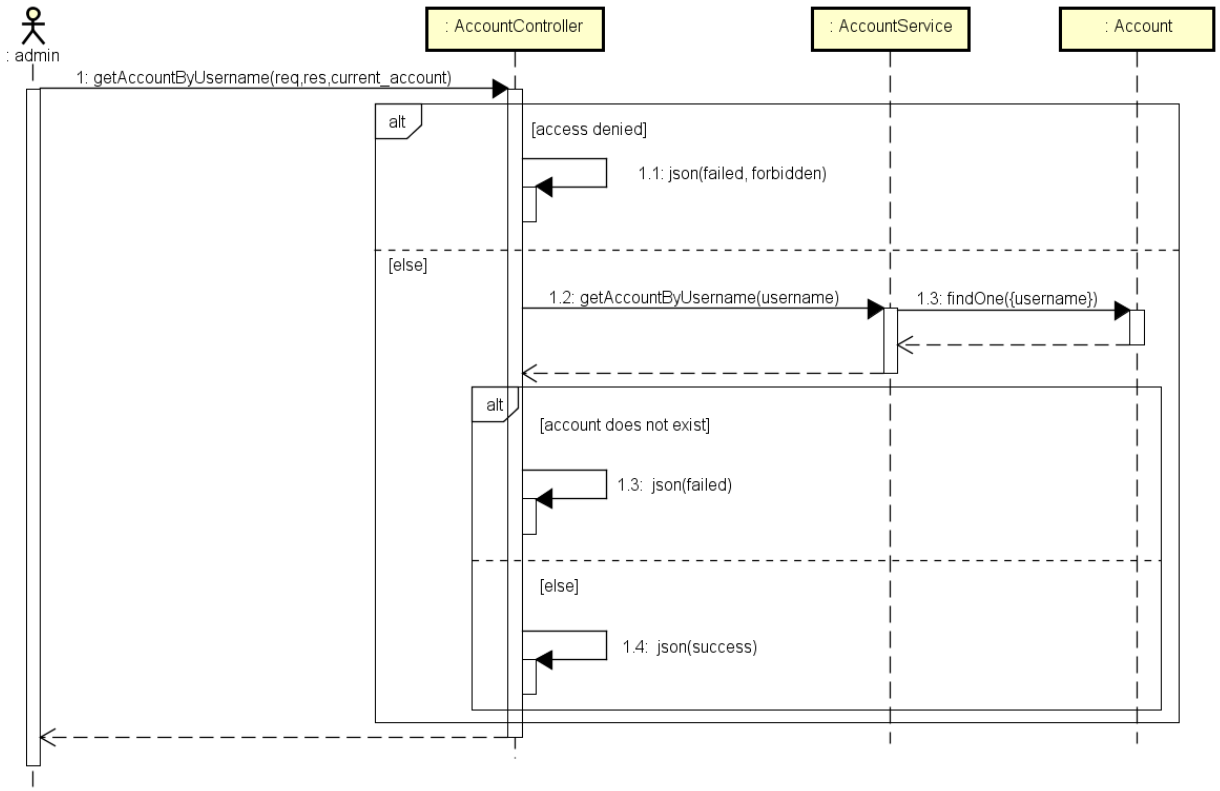


Figure 4-79: Staff views account information detail sequence diagram

4.3.4.12 Staff searches account

Screen Design

STT	Tài khoản	Họ và tên
1	root	Nguyen Van Du
2	anhdv	Duong Viet Anh
3	tuandm	Dao Manh Tuan
4	quynhdtt	Do Thi Thu QUynh

Figure 4-80: Staff searches account screen design

Class Diagram

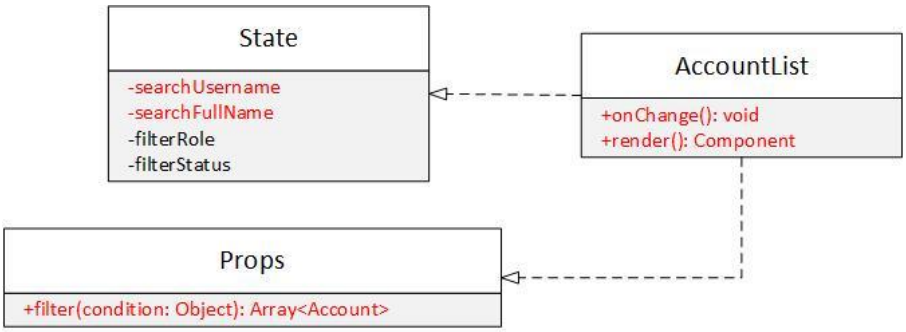


Figure 4-81: Staff searches account class diagram

Class Specification

State

State			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
1	searchUsername	String	
2	searchFullName	String	
Operation			
N/A	N/A		

AccountList

AccountList			
Physical address	src/components/AccountList.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
onChange			
Return Type	void		
Parameters	Name	Type	Description
	N/A	N/A	N/A
render			
Return Type	Component		
Parameters	Name	Type	Description
	N/A	N/A	N/A

Props

Props			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
filter			
Return Type	Array<Account>		

Parameters	Name	Type	Description
	condition	object	

Sequence Diagram

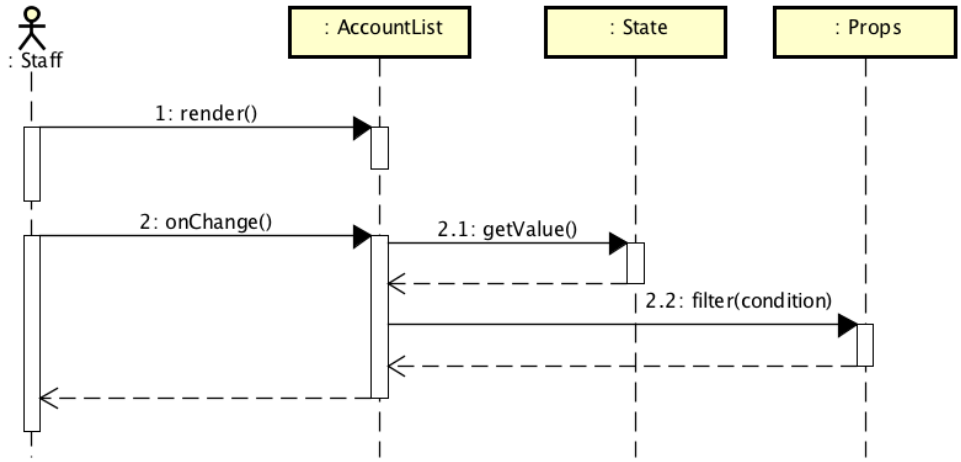


Figure 4-82: Staff searches account sequence diagram

4.3.4.13 Staff filters account

Screen Design

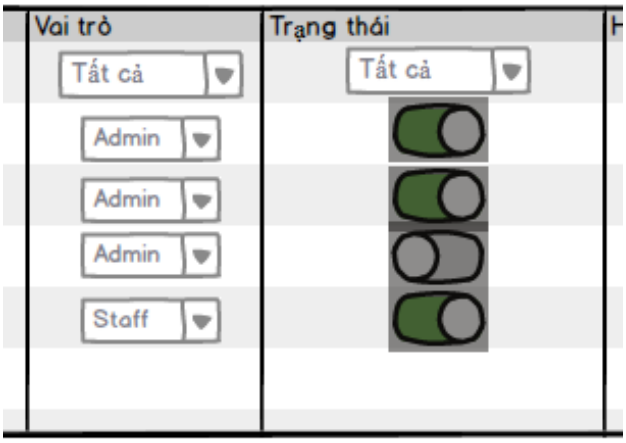


Figure 4-83: Staff filters account screen design

Class Diagram

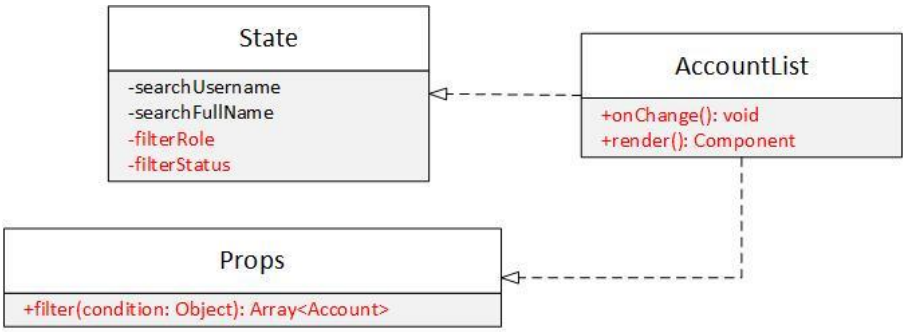


Figure 4-84: Staff filters account class diagram

Class Specification

State

State			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
1	filterRole	String	
2	filterStatus	String	
Operation			
N/A	N/A		

AccountList

AccountList			
Physical address	src/component/AccountList.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
onChange			
Return Type	void		
Parameters	Name	Type	Description
	N/A	N/A	N/A
render			
Return Type	Component		
Parameters	Name	Type	Description
	N/A	N/A	N/A

Props

Props			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
filter			

Return Type	Array<Account>		
Parameters	Name	Type	Description
	condition	object	

Sequence Diagram

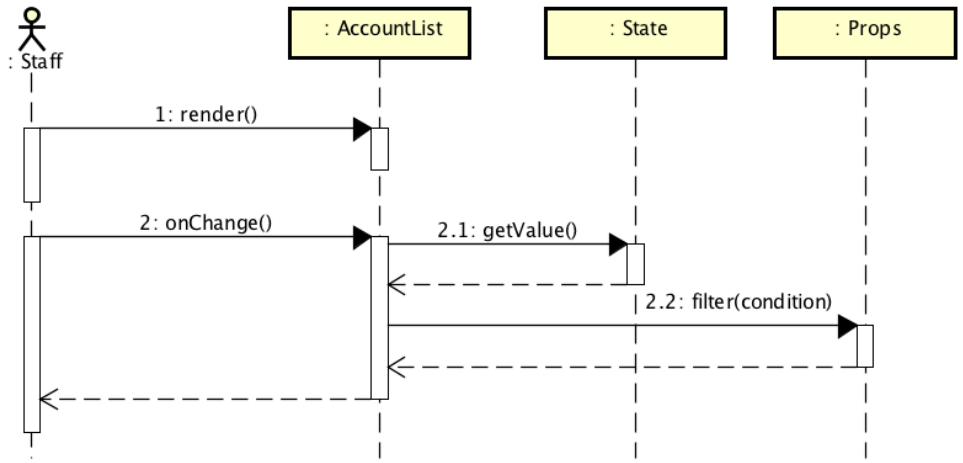


Figure 4-85: Staff filters account sequence diagram

4.3.4.14 Staff views response & answer list

Screen Design

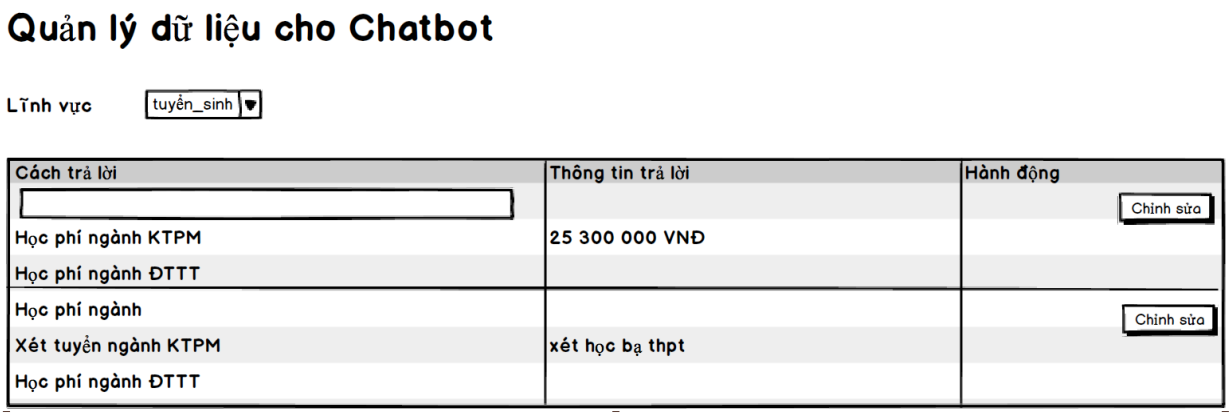


Figure 4-65: Staff views response & answer list screen design

Class Diagram

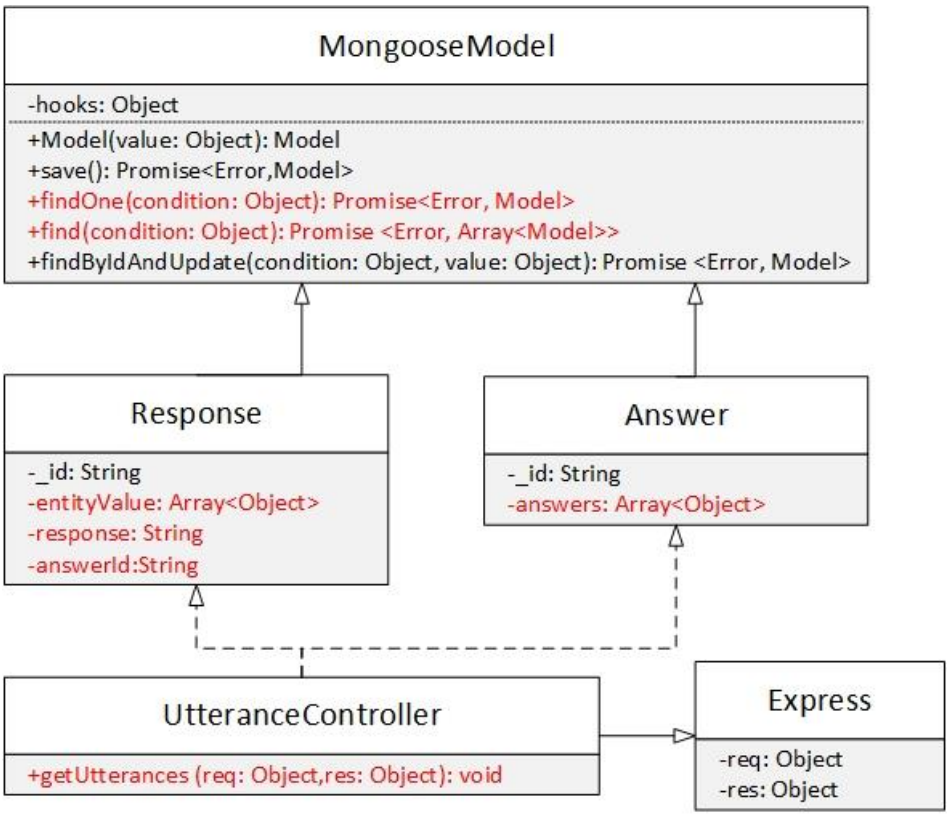


Figure 4-66: Staff views response & answer list class diagram

Class Specification

Response

Response			
Physical address	Backend /models/Response.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	entityValue	Array<object>	
2	response	String	
3	answerId	String	
Operation			

Answer

Answer			
Physical address	Backend /models/Answer.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	answers	Array<object>	
Operation			

UtteranceController

UtteranceController	
Physical address	Backend /controllers/UtteranceController.js
Base class	Express
Attributes	

No	Name	Type	Description
Operation			
getUtterances			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

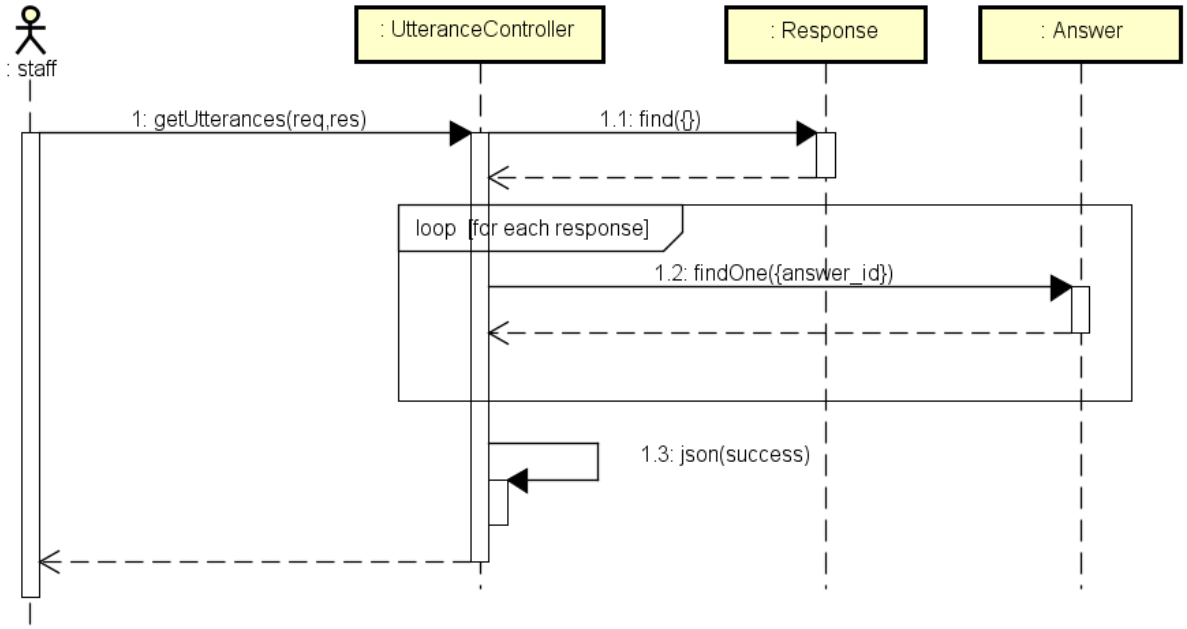


Figure 4-67: Staff views response & answer list sequence diagram

4.3.4.15 Staff filters response & answer

Screen Design

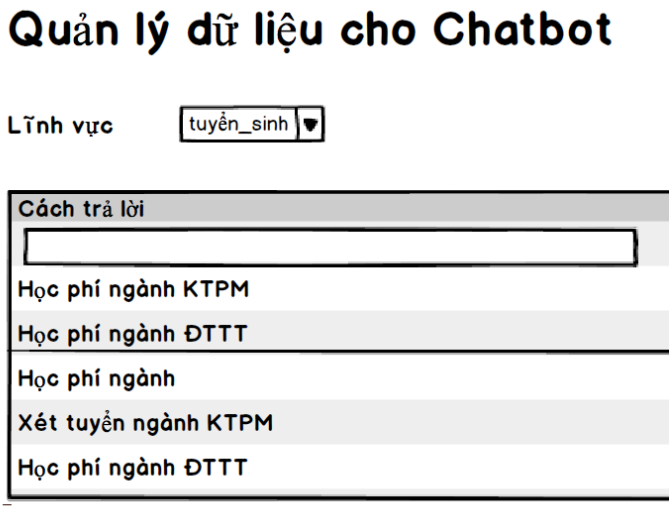


Figure 4-68: Staff filters response & answer screen design

Class Diagram

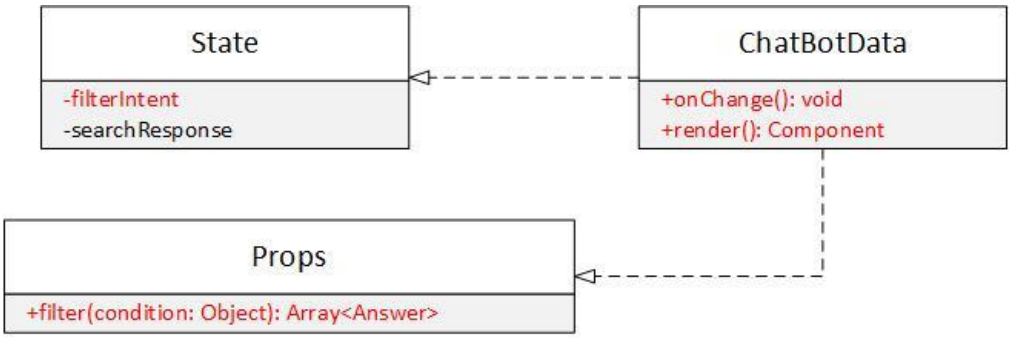


Figure 4-69: Staff filters response & answer class diagram

Class Specification

State

State			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
1	filterIntent	String	
Operation			
N/A	N/A		

ChaBotData

ChatBotData			
Physical address	src/component/ChatbotData.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
onChange			
Return Type	void		
Parameters	Name	Type	Description
	N/A	N/A	N/A
render			
Return Type	Component		
Parameters	Name	Type	Description
	N/A	N/A	N/A

Props

Props			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
filter			
Return Type	Array<Answer>		

Parameters	Name	Type	Description
	condition	object	

Sequence Diagram

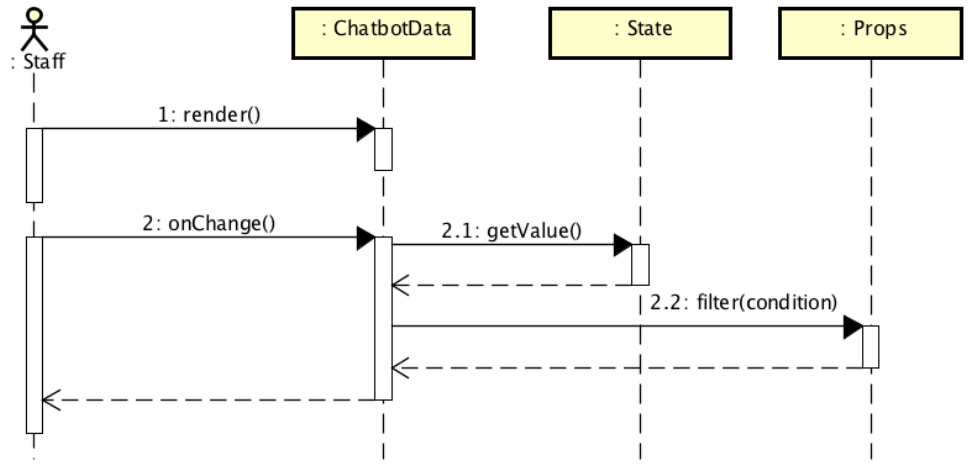


Figure 4-70: Staff filters response & answer sequence diagram

4.3.4.16 Staff searches response

Screen Design

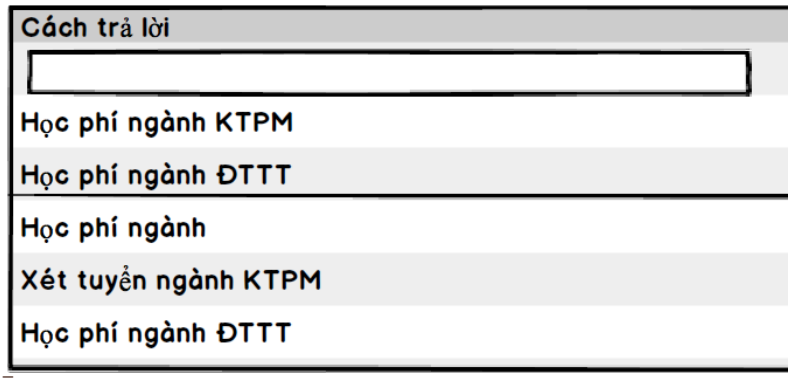


Figure 4-71: Staff searches response screen design

Class Diagram

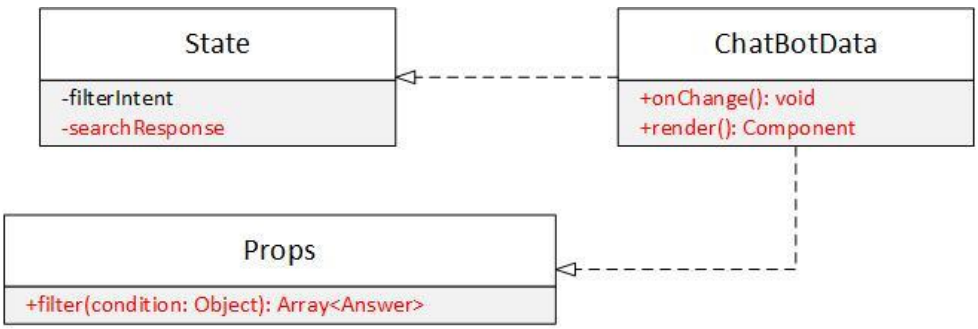


Figure 4-72: Staff searches response screen class diagram

Class Specification

State

State			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
1	filterResponse	String	
Operation			
N/A	N/A		

ChaBotData

ChatBotData			
Physical address	src/component/ChatBotData.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
onChange			
Return Type	void		
Parameters	Name	Type	Description
	N/A	N/A	N/A
render			
Return Type	Component		
Parameters	Name	Type	Description
	N/A	N/A	N/A

Props

Props			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
filter			
Return Type	Array<Answer>		
Parameters	Name	Type	Description
	condition	object	

Sequence Diagram

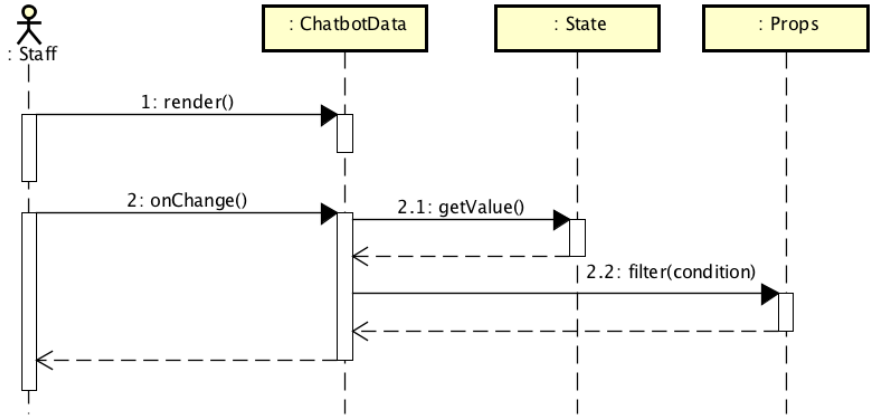


Figure 4-73: Staff searches response sequence diagram

4.3.4.17 Staff requests to edit answer

Screen Design

Kiểm duyệt dữ liệu

STT	Cách trả lời	Thông tin trả lời cũ	Thông tin trả lời mới	Ngày tạo	Người tạo	Trạng thái
1	Học phí ngành KTPM	24 200 000 VND	25 300 000	20/05/2018	anhdv	Đang chờ duyệt
2	Học phí ngành Ngôn ngữ Nhật	20 200 000 VND	20 500 000 VND	21/05/2018	anhdv	Được chấp nhận
3	Học phí ngành Ngôn ngữ Anh	20 200 000 VND	20 800 000	21/05/2018	anhdv	Bị từ chối

Figure 4-86: Staff requests to edit answer screen design

Class Diagram

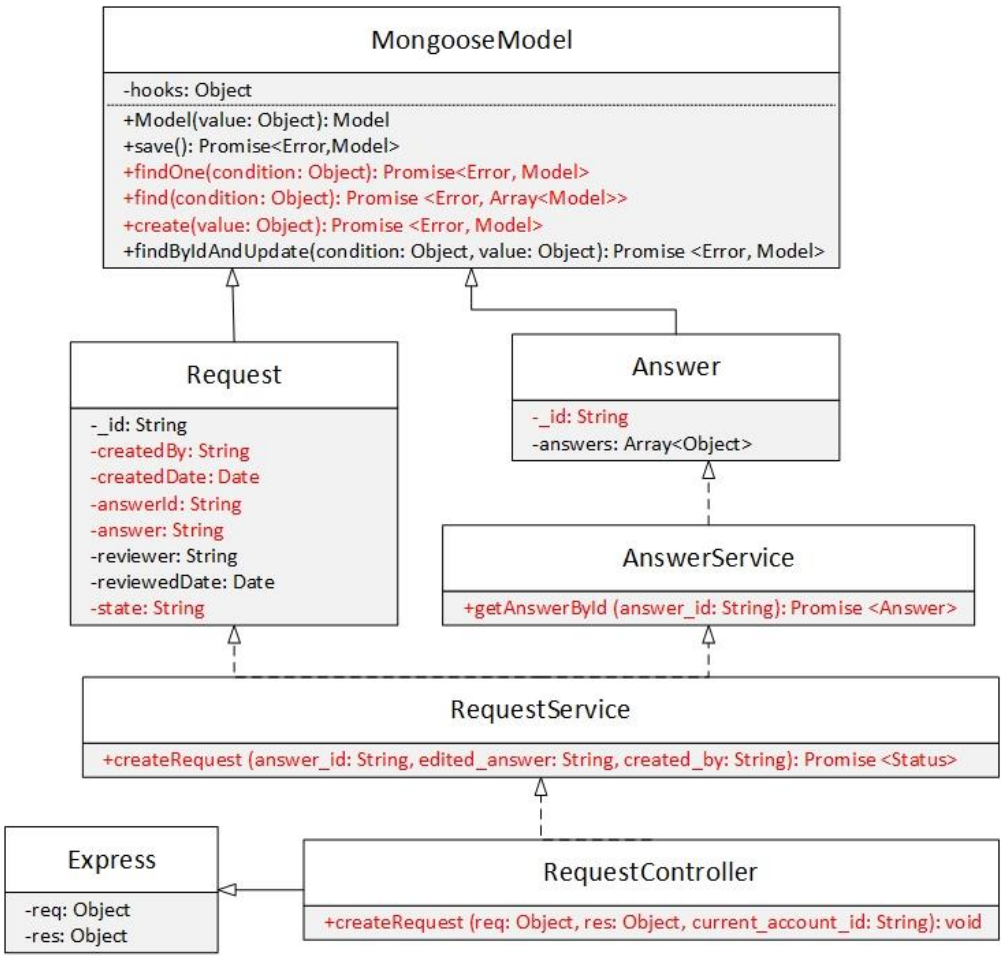


Figure 4-87: Staff requests to edit answer class diagram

Class Specification

Request

Request			
Physical address	Backend /models/Request.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	createdBy	String	
2	createdDate	String	
3	answerId	String	
4	answer	String	
5	state	String	
Operation			

Answer

Answer			
Physical address	Backend /models/Answer.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	_id	String	

Operation	
N/A	N/A

AnswerService

AnswerService			
Physical address	Backend /services/AnswerService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getAnswerById			
Return Type	Promise<Answer>		
Parameters	Name	Type	Description
	answer_id	String	

RequestService

RequestService			
Physical address	Backend /services/RequestService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
createRequest			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	answer_id	String	
	edited_answer	String	
	created_by	String	

RequestController

RequestController			
Physical address	Backend /controllers/RequestController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
createRequest			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.
	current_account_id	String	

Sequence Diagram

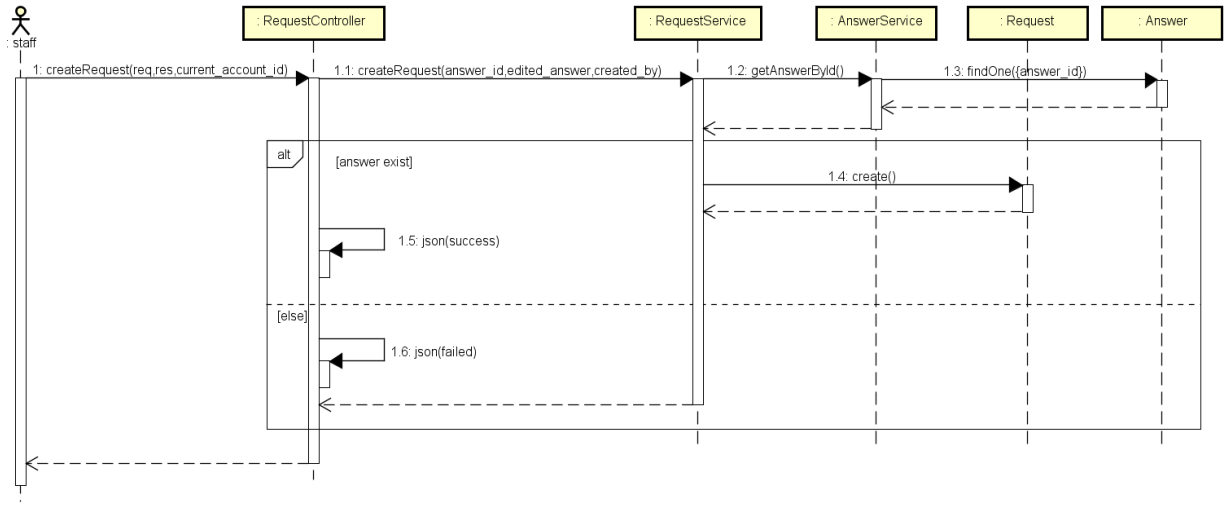


Figure 4-88: Staff request to edit answer sequence diagram

4.3.4.18 Staff receives notification

Screen Design

Figure 4-89: Staff receives notification screen design

Class Diagram

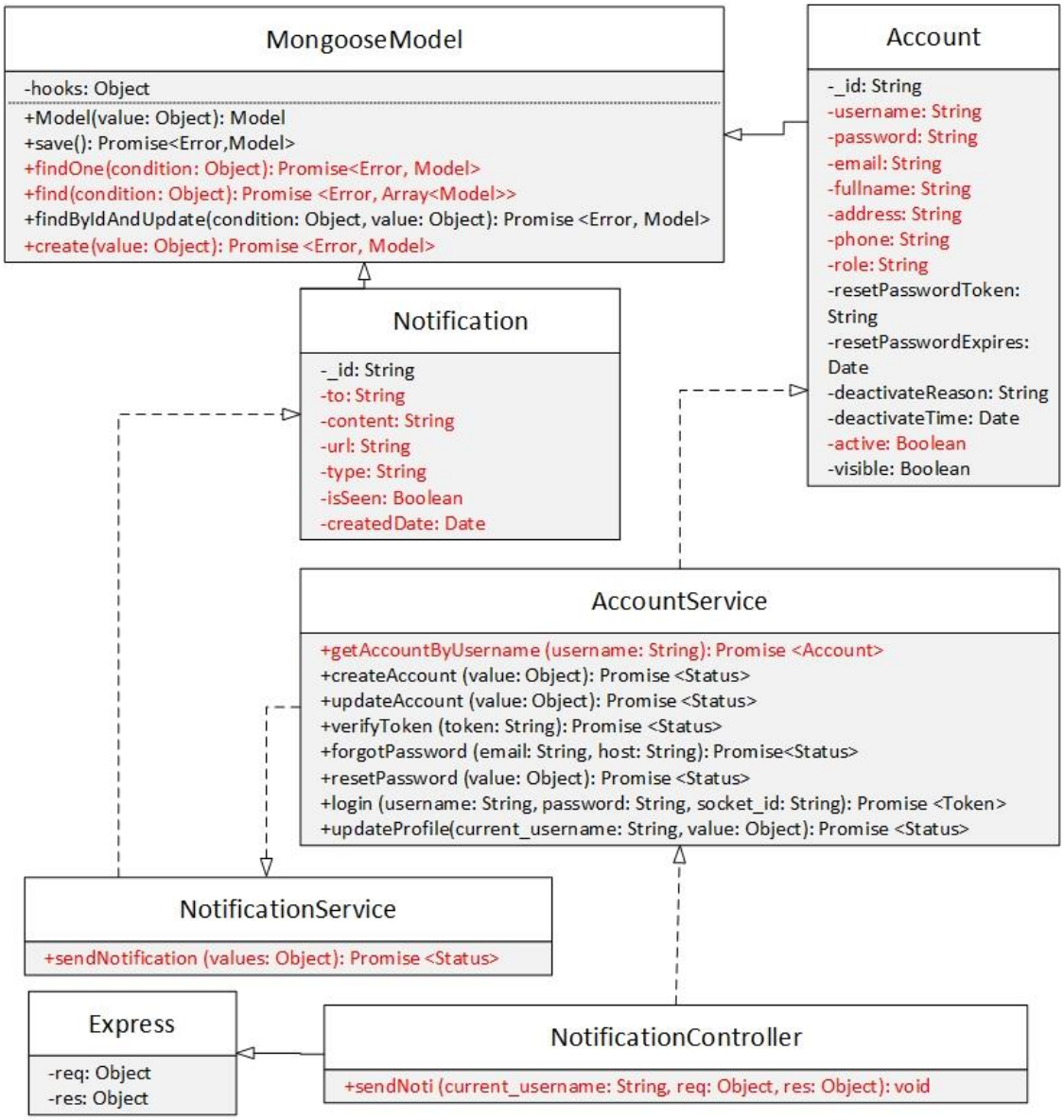


Figure 4-90: Staff receives notification class diagram

Class Specification

Notification

Notification			
Physical address	Backend /models/Notification.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	to	String	
2	content	String	
3	url	String	
4	type	String	
5	isSeen	Boolean	
6	createdDate	Date	
Operation			

N/A	N/A

Account

Account			
Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	username	String	
2	email	String	
3	fullname	String	
4	password	String	
5	address	String	
6	phone	String	
7	role	String	
8	active	Boolean	
Operation			
N/A	N/A		

NotificationService

NotificationService			
Physical address	Backend /services/NotificationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
sendNotification			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	value	Object	

AccountService

AccountService			
Physical address	Backend /controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getAccountByUsername			
Return Type	Promise<Account>		
Parameters	Name	Type	Description
	username	String	

NotificationController

NotificationController			
Physical address	Backend /controllers/NotificationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			

sendNoti			
Return Type	void		
Parameters	Name	Type	Description
	current_username	String	
	req	object	
	res	object	

Sequence Diagram

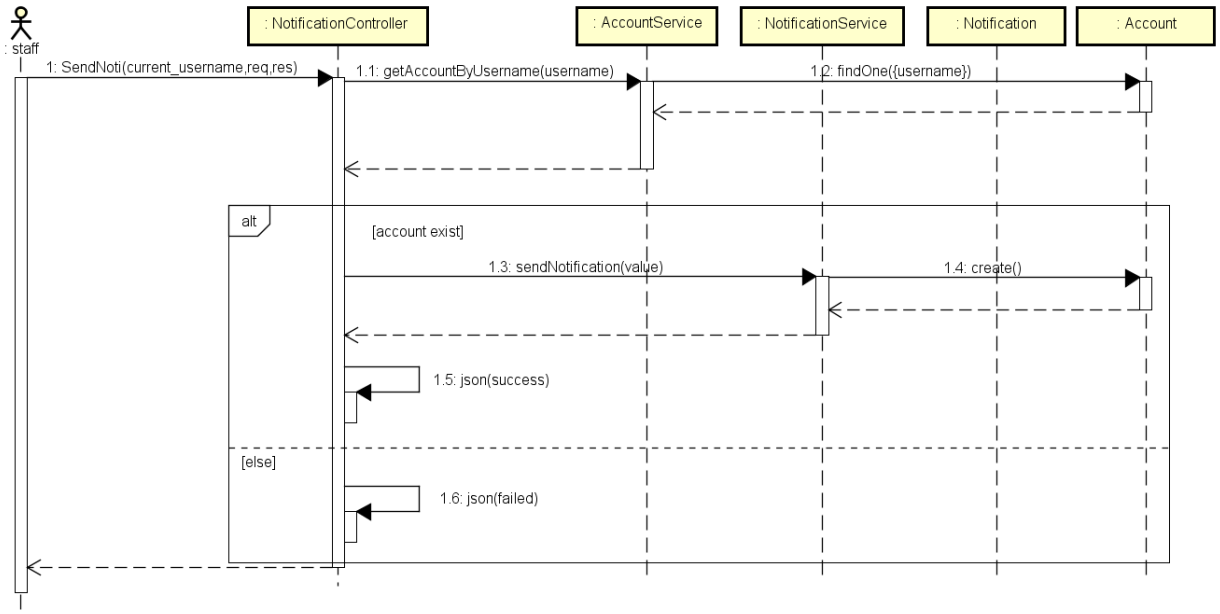


Figure 4-91: Staff receives notification sequence diagram

4.3.4.19 Staff views notification list

Screen Design

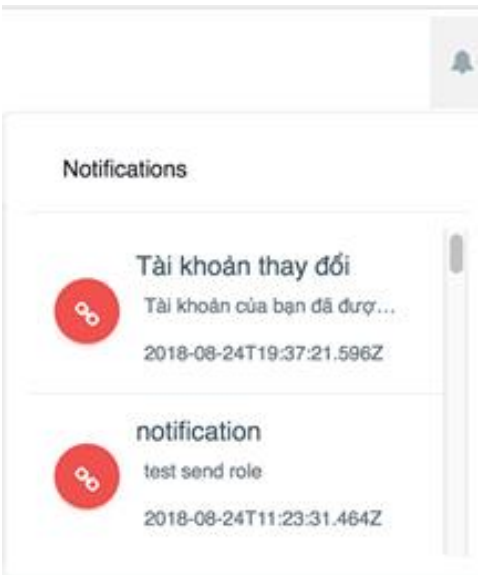


Figure 4-92: Staff views notification list screen design

Class Diagram

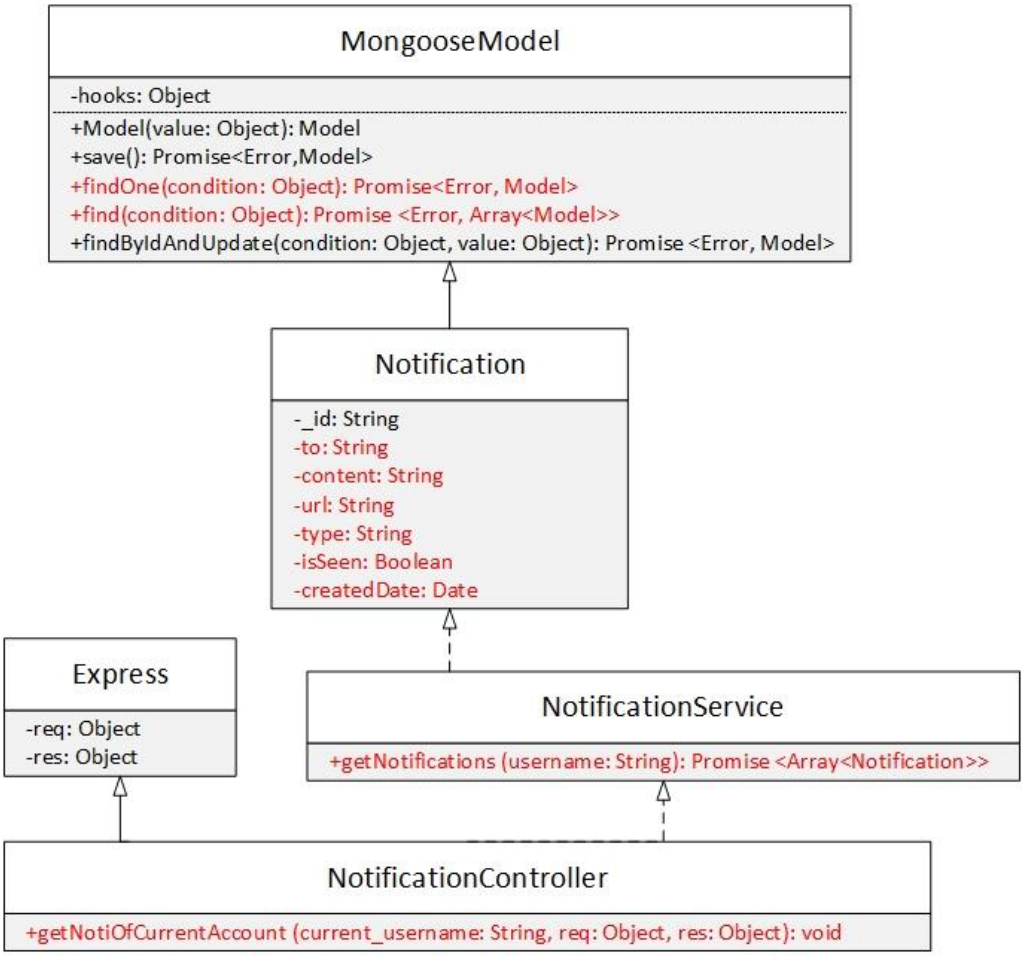


Figure 4-93: Staff views notification list class diagram

Class Specification

Notification

Notification			
Physical address	Backend /models/Notification.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	to	String	
2	content	String	
3	url	String	
4	type	String	
5	isSeen	Boolean	
6	createdDate	Date	
Operation			
N/A	N/A		

NotificationService

NotificationService	
Physical address	Backend /services/NotificationService.js
Base class	N/A

Attributes			
No	Name	Type	Description
Operation			
getNotifications			
Return Type	Promise<Array<Notification>>		
Parameters	Name	Type	Description
	username	String	

NotificationController

NotificationController			
Physical address	Backend /controllers/NotificationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getNotiOfCurrentAccount			
Return Type	void		
Parameters	Name	Type	Description
	req	Object	
	res	Object	
	current_username	String	

Sequence Diagram

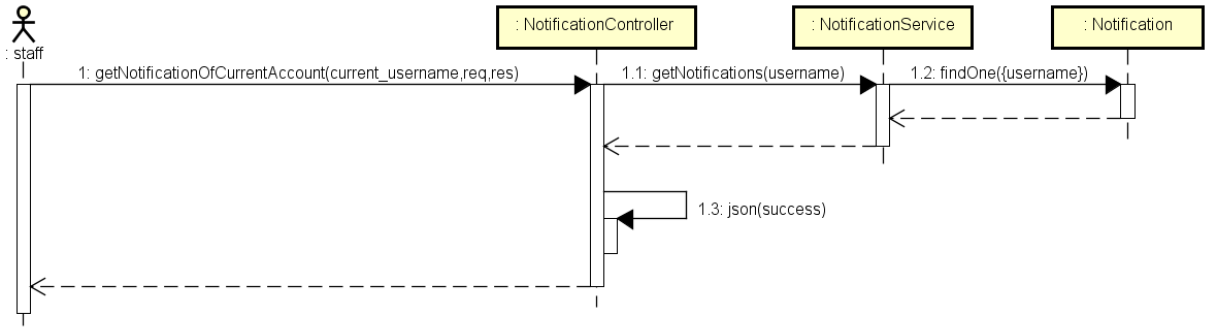


Figure 4-94: Staff views notification list sequence diagram

4.3.4.20 Staff views notification detail

Screen Design

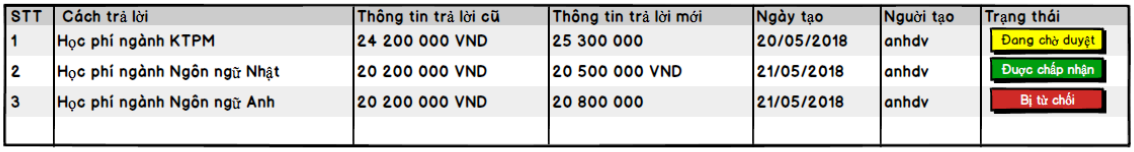


Figure 4- 86: Staff views notification detail screen design

Class Diagram

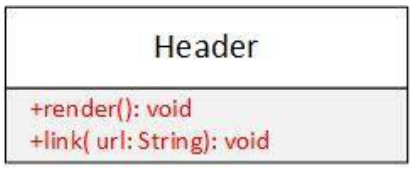


Figure 4-87: Staff views notification detail class diagram

Class Specification

Header

Header			
Physical address	src/component/Header.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
render			
Return Type	Component		
Parameters	Name	Type	Description
	N/A	N/A	N/A
link			
Return Type	Component		
Parameters	Name	Type	Description
	url	String	

Sequence Diagram

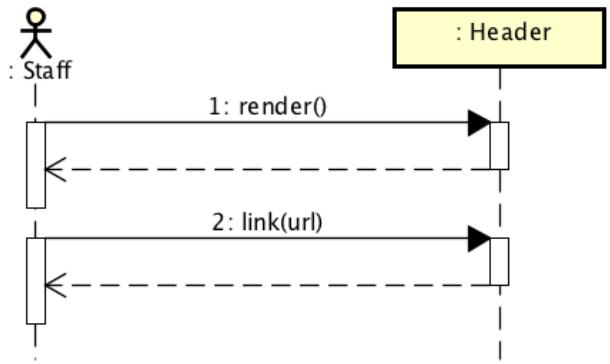


Figure 4-88: Staff views notification detail sequence diagram

4.3.4.21 Staff views Fb Count Page

Screen Design

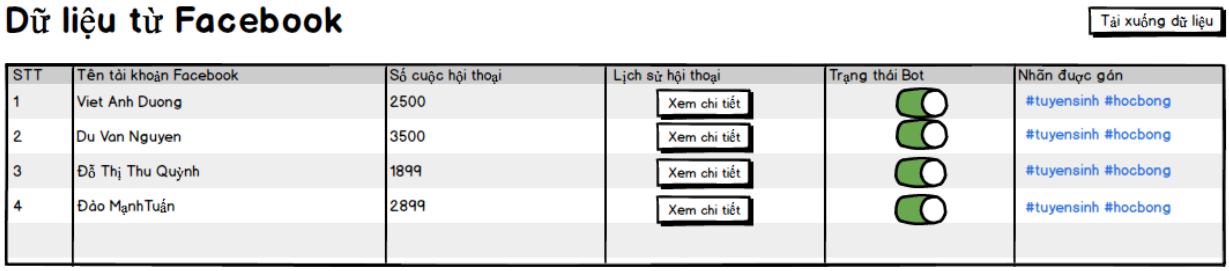


Figure 4-95: Staff views Fb Count Page screen design

Class Diagram

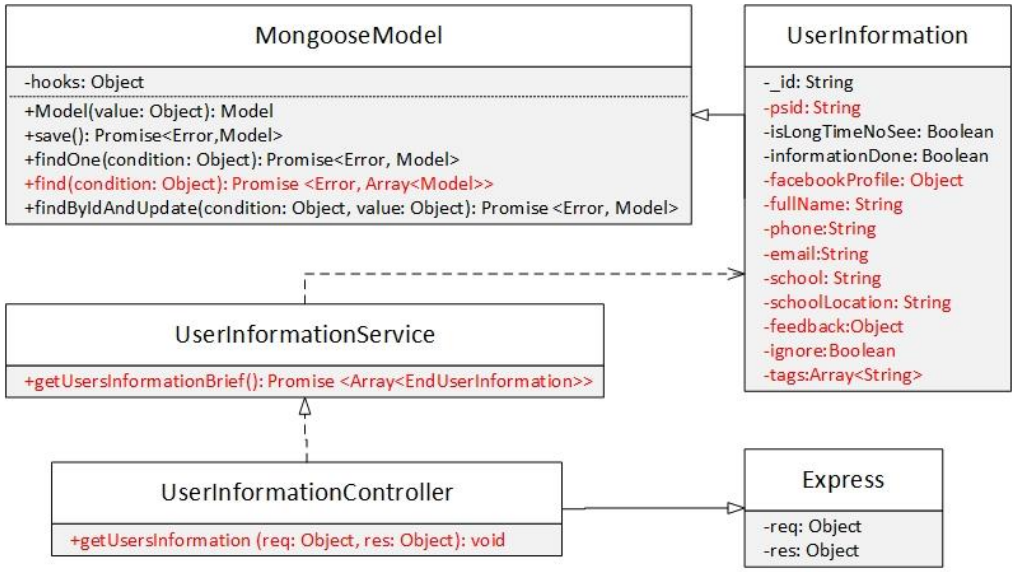


Figure 4-96: Staff views Fb Count Page class diagram

Class Specification

UserInformation

UserInformation			
Physical address	Backend /models/UserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
2	facebookProfile	Object	
3	fullName	String	
4	phone	String	
5	email	String	
6	school	String	
7	schoolLocation	String	
8	feedback	Object	
9	ignore	Boolean	
10	tags	Array<String>	
Operation			

UserInformationService

UserInformationService			
Physical address	Backend /services/UserInformationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getUserInformationBrief			
Return Type	Promise<Array<UserInformation>>		
Parameters	Name	Type	Description
	N/A	N/A	N/A

UserInformationController

UserInformationController			
Physical address	Backend /controllers/UserInformationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getUsersInformation			
Return Type	void		
Parameters	Name	Type	Description
	req	Object	
	res	Object	

Sequence Diagram

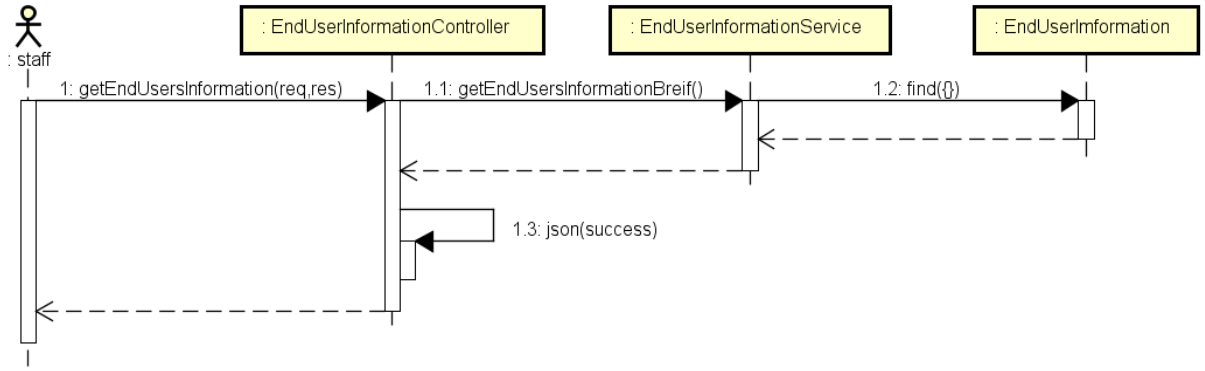


Figure 4-97: Staff views Fb Count Page sequence diagram

4.3.4.22 Staff views chat logs

Screen Design

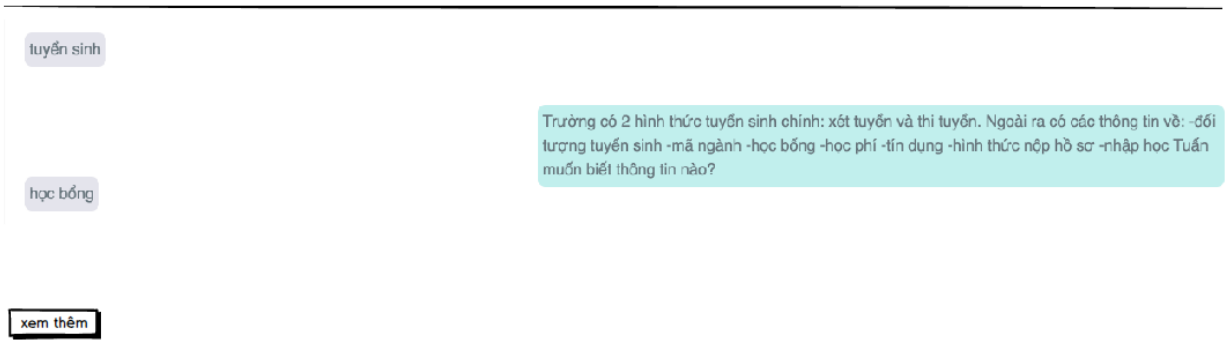


Figure 4-98: Staff views chat logs screen design

Class Diagram

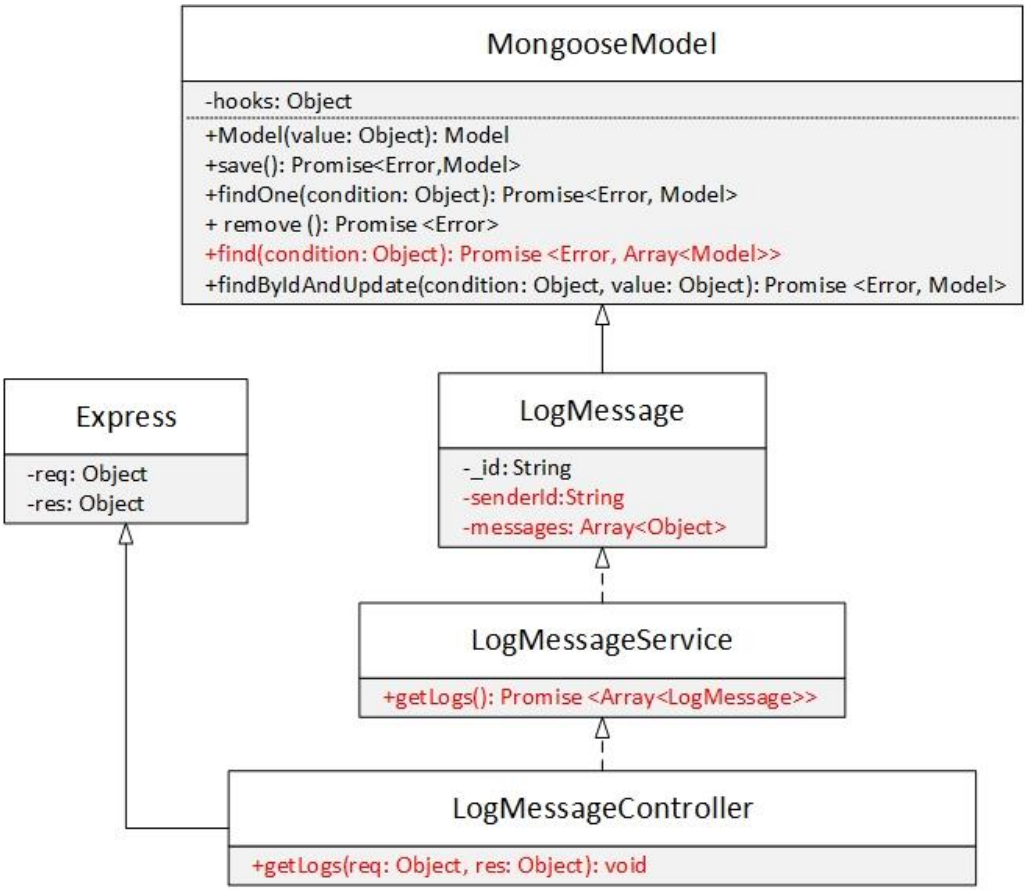


Figure 4-99: Staff views chat logs class diagram

Class Specification

LogMessage

LogMessage			
Physical address	Backend /models/LogMessage.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	senderId	String	

2	messages	Array<object>	
Operation			
N/A	N/A		

LogMessageService

LogMessageService			
Physical address	Backend /controllers/LogMessageService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getLogs			
Return Type	Promise<Array<LogMessage>>		
Parameters	Name	Type	Description
	N/A	N/A	N/A

LogMessageController

LogMessageController			
Physical address	Backend /controllers/LogMessageController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getLogs			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

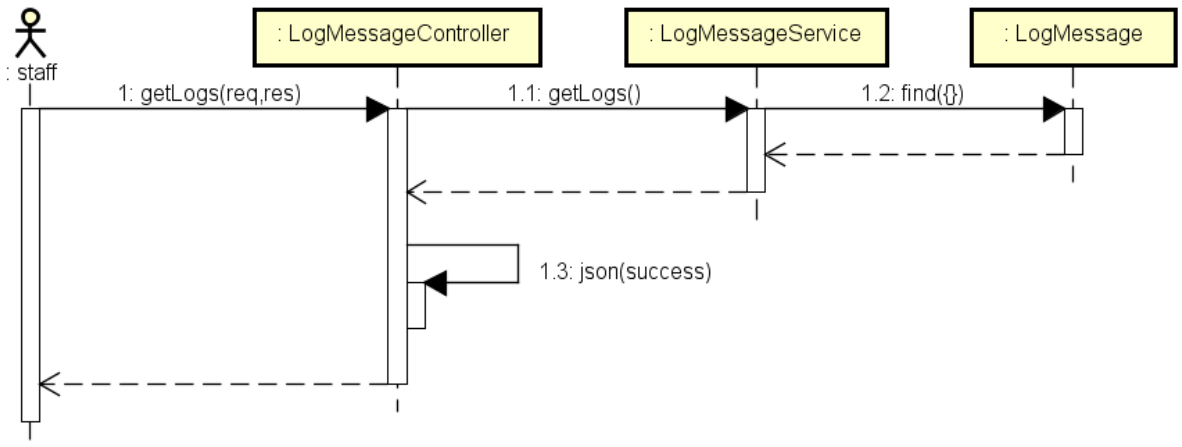


Figure 4-100: Staff views chat logs sequence diagram

4.3.4.23 Staff views Fb account infor

Screen Design

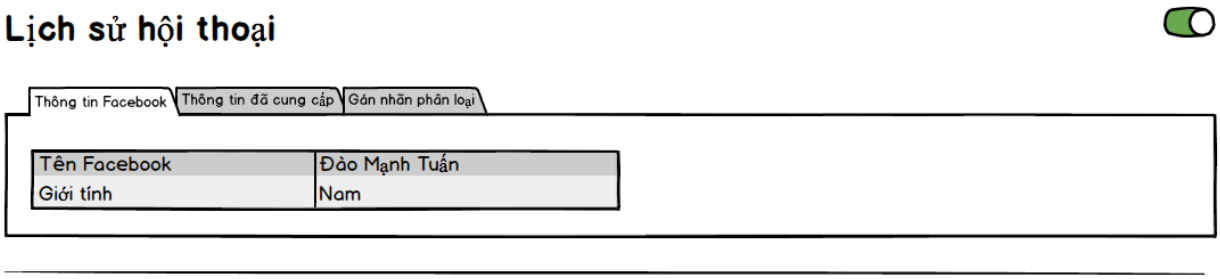


Figure 4-101: Staff views Fb account infor screen desgin

Class Diagram

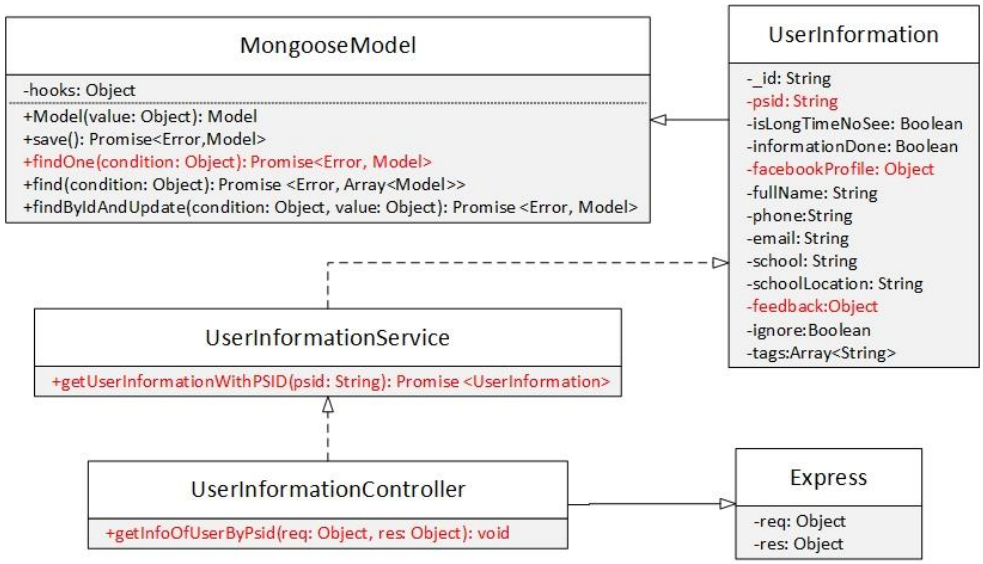


Figure 4-102: Staff views Fb account infor class diagram

Class Specification

UserInformation

UserInformation			
Physical address	Backend /models/UserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
2	facebookProfile	Object	
3	feedback	Object	
Operation			

UserInformationService

UserInformationService			
Physical address	Backend /services/UserInformationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description

Operation			
getUserInformationWithPSID			
Return Type	Promise<UserInformation>		
Parameters	Name	Type	Description
	psid	String	

UserInformationController

UserInformationController			
Physical address	Backend /controllers/UserInformationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getInforOfUserByPsid			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

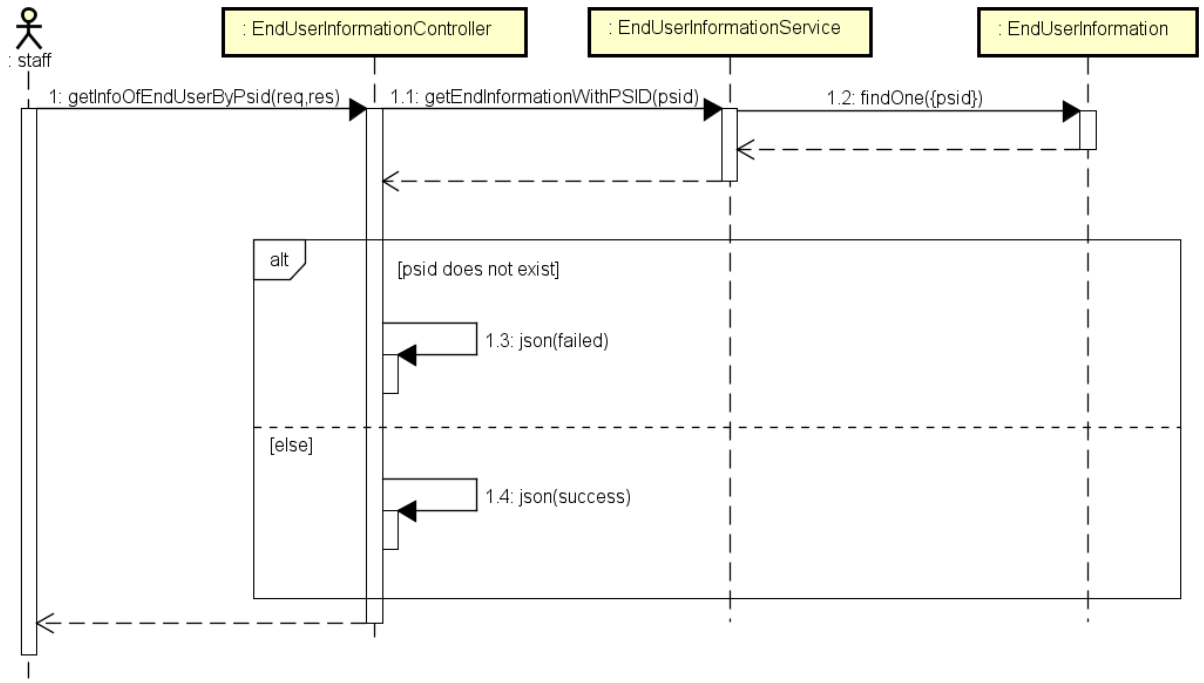


Figure 4-103: Staff views Fb account infor sequence diagram

4.3.4.24 Staff views providing infor

Screen Design

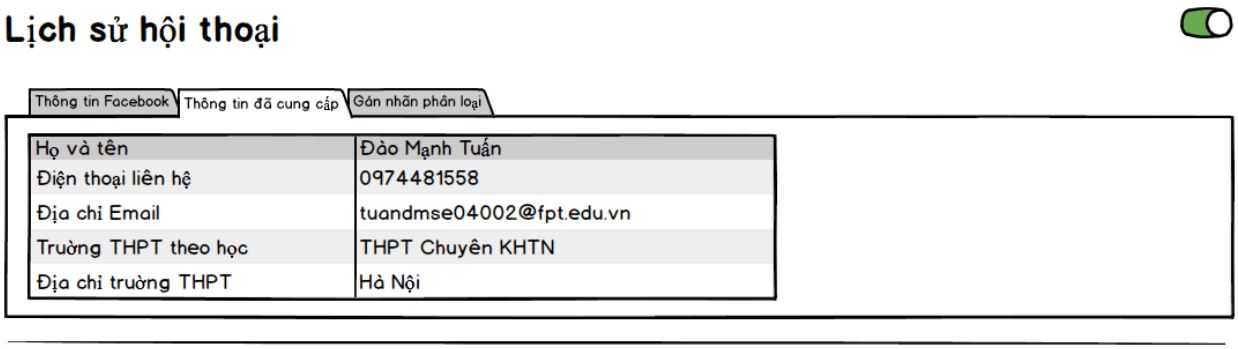


Figure 4-104: Staff views providing infor screen design

Class Diagram

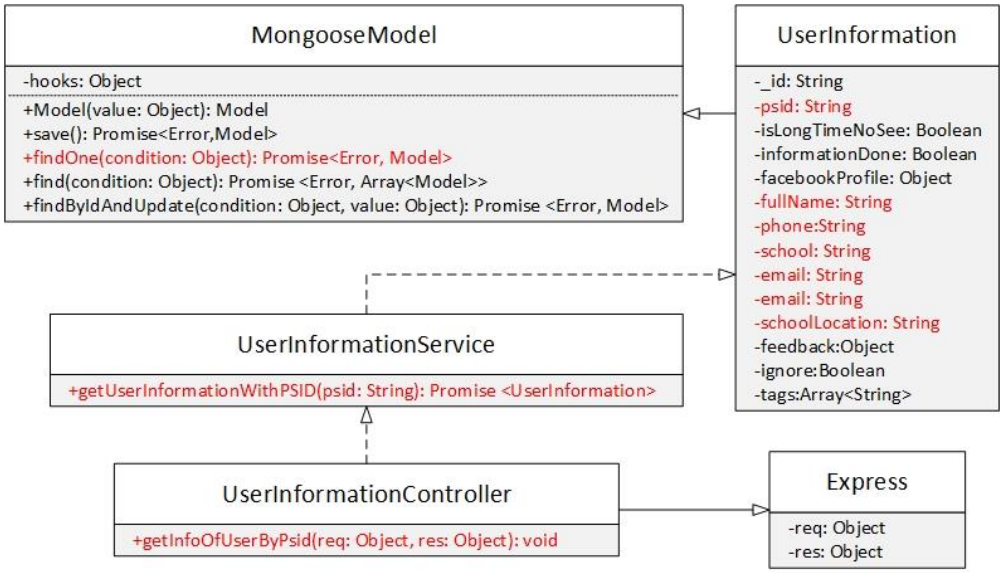


Figure 4-105: Staff views providing infor class diagram

Class Specification

EndUserInformation

EndUserformation			
Physical address	Backend /models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
2	fullname	String	
3	phone	String	
4	school	String	
5	schoolLocation	String	
6	email	String	
Operation			
N/A	N/A		

EndUserInformationService

EndUserInformationService			
Physical address	Backend /services/EndUserInformationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
getUserInformationWithPSID			
Return Type	Promise<EndUserInformation>		
Parameters	Name	Type	Description
	psid	String	

EndUserInformationController

EndUserInformationController			
Physical address	Backend /controllers/EndUserInformationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
getInforOfUserByPsid			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

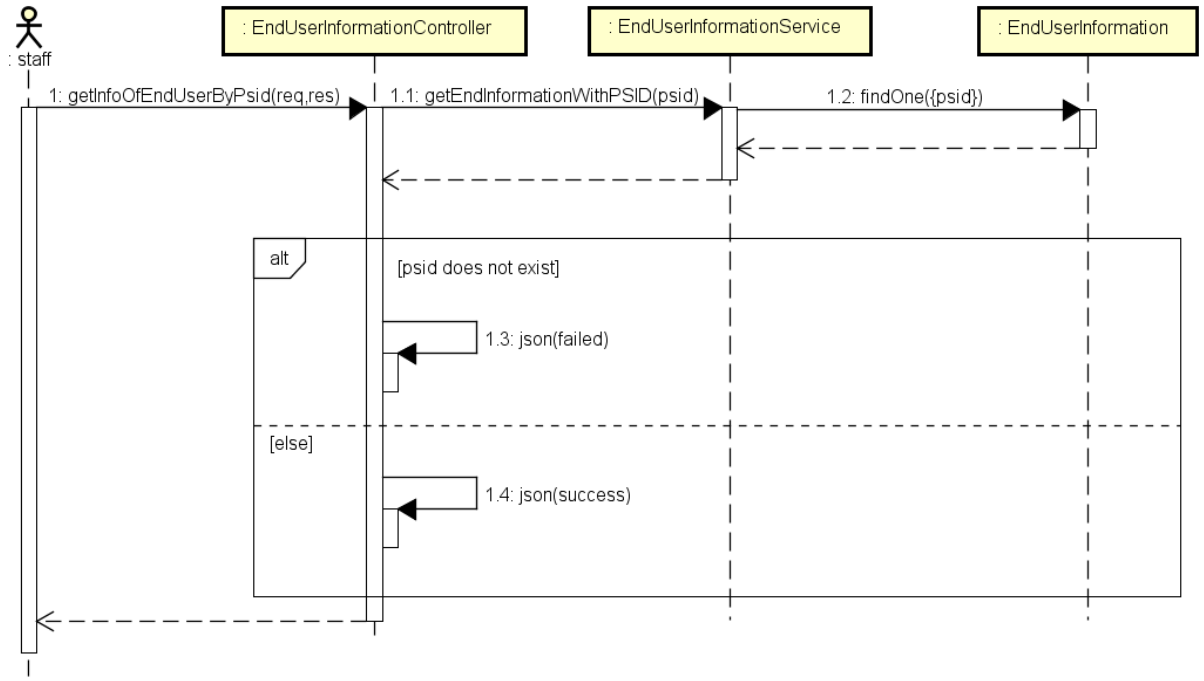


Figure 4-106: Staff views providing infor sequence diagram

4.3.4.25 Staff adds tag

Screen Design

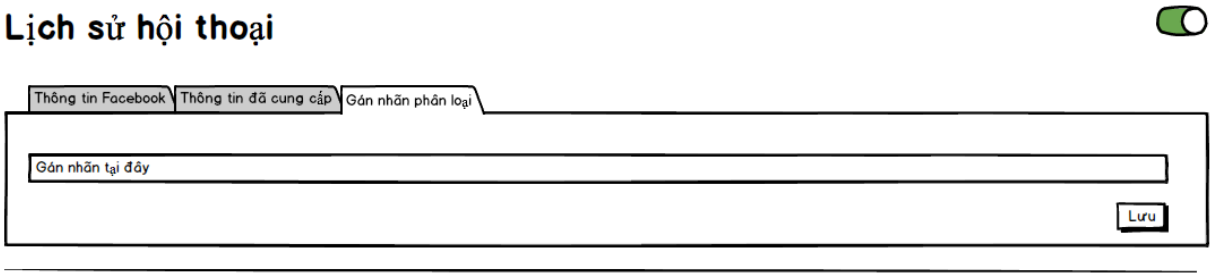


Figure 4-107: Staff adds tag screen design

Class Diagram

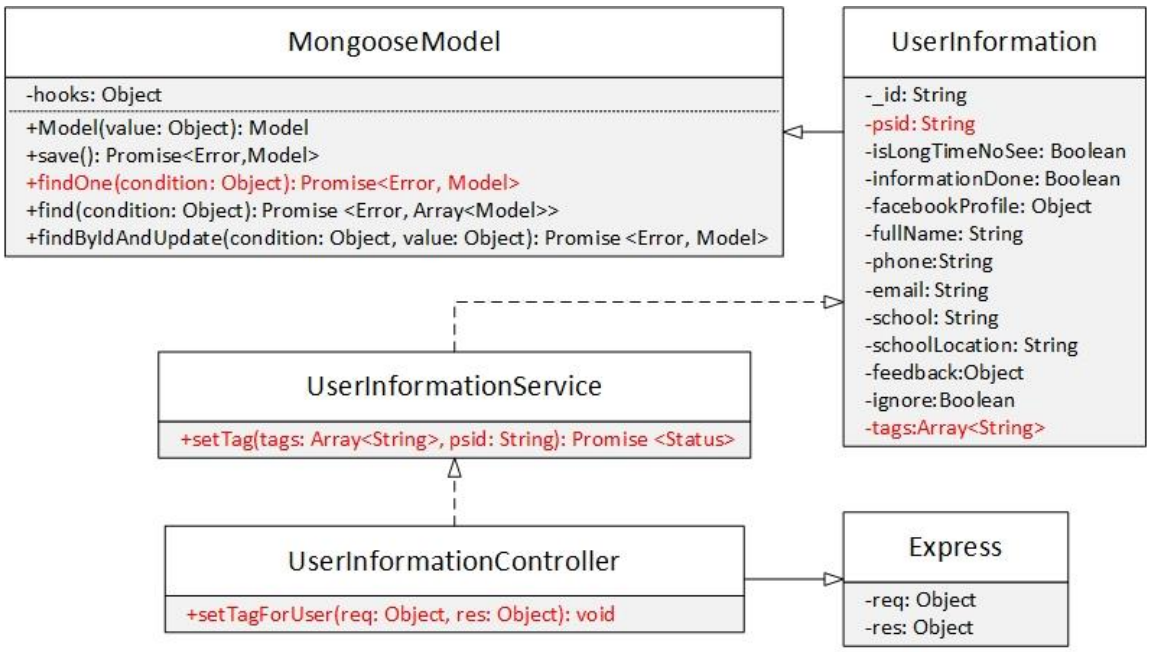


Figure 4-108: Staff adds tag class diagram

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend /models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
2	tags	Array<String>	
Operation			
N/A	N/A		

EndUserInformationService

EndUserInformationService	
Physical address	Backend /controllers/EndUserInformationService.js

Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
setTag			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	tags	Array<String>	
	psid	String	

EndUserInformationController

EndUserInformationController			
Physical address	Backend /controllers/EndUserInformationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
setTagForUser			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

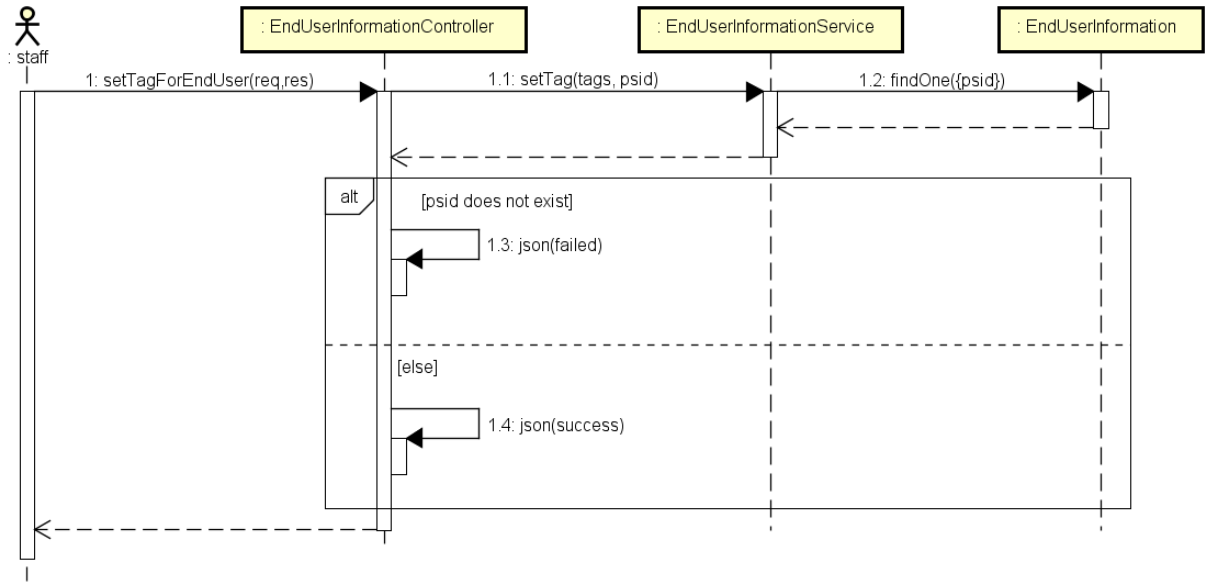


Figure 4-109: Staff adds tag sequence diagram

4.3.4.26 Staff removes tag

Screen Design

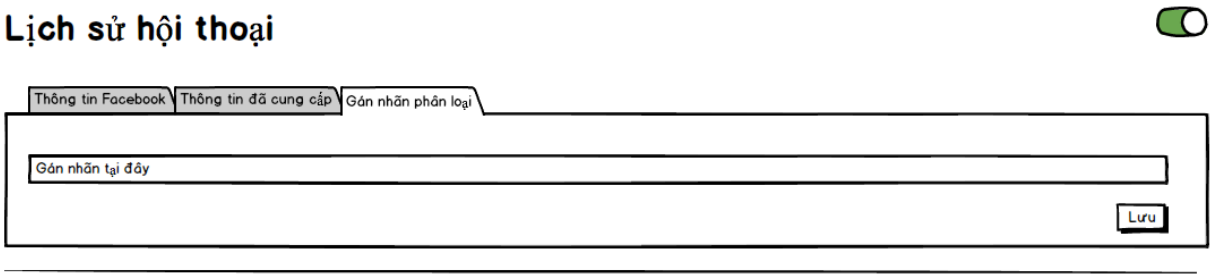


Figure 4-110: Staff removes tag screen design

Class Diagram

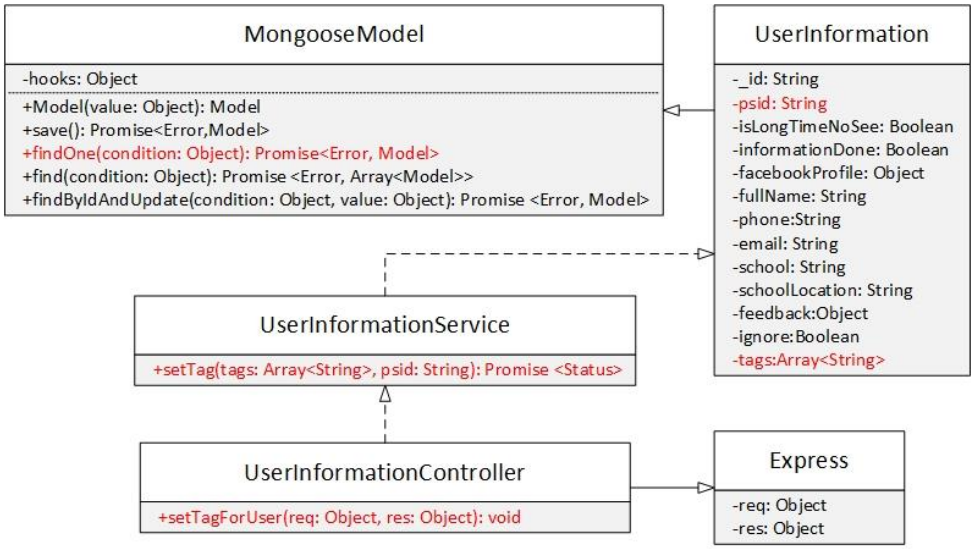


Figure 4-111: Staff removes tag class diagram

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend /models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
2	tags	Array<String>	
Operation			
N/A	N/A		

EndUserInformationService

EndUserInformationService			
Physical address	Backend /controllers/EndUserInformationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
setTag			

Return Type	Promise<Status>		
Parameters	Name	Type	Description
	tags	Array<String>	
	psid	String	

EndUserInformationController

EndUserInformationController			
Physical address	Backend /controllers/EndUserInformationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
setTagForUser			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

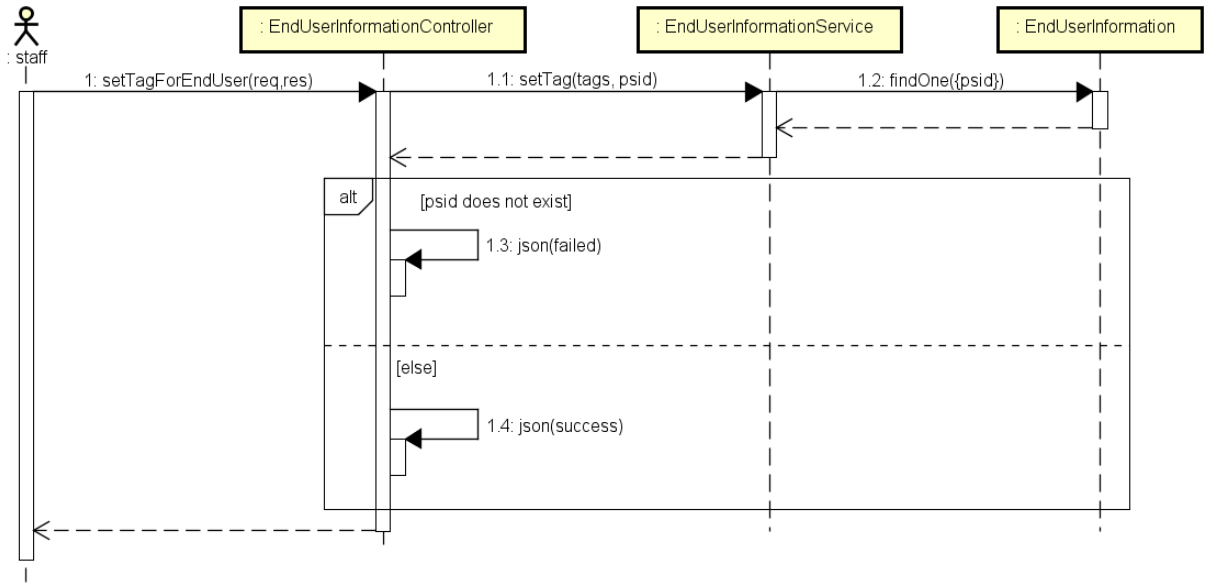


Figure 4-112: Staff removes tag sequence diagram

4.3.4.27 Staff exports end user list

Screen Design

Figure 4-113: Staff exports end user list screen design

Class Diagram

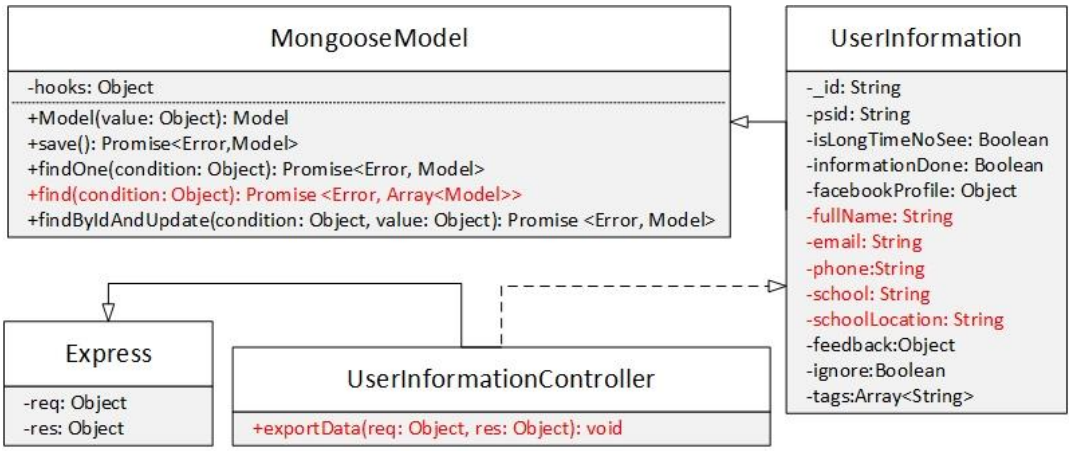


Figure 4-114: Staff exports end user list class diagram

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend /models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	fullName	String	
2	email	String	
3	phone	String	
4	school	String	
5	schoolLocation	String	
Operation			
N/A	N/A		

EndUserInformationController

EndUserInformationController			
Physical address	Backend /controllers/EndUserInformationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
exportData			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

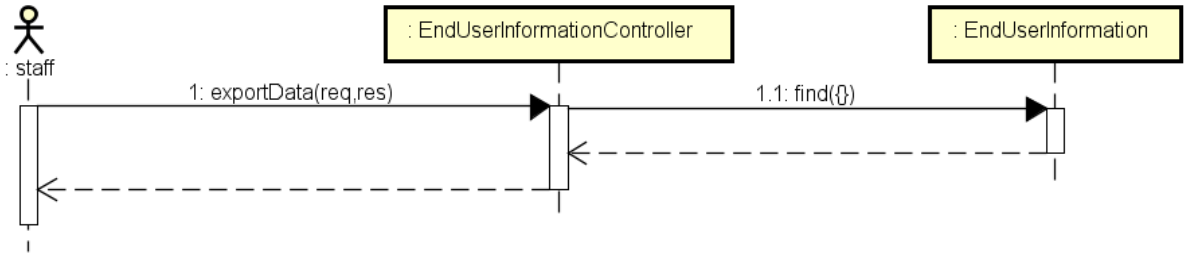


Figure 4-115: Staff exports end user list sequence diagram

4.3.4.28 Staff changes bot status

Screen Design

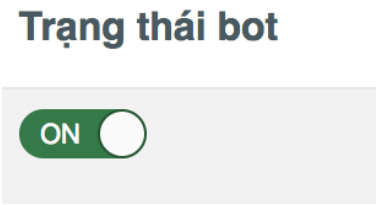


Figure 4-116: Staff changes bot status screen design

Class Diagram

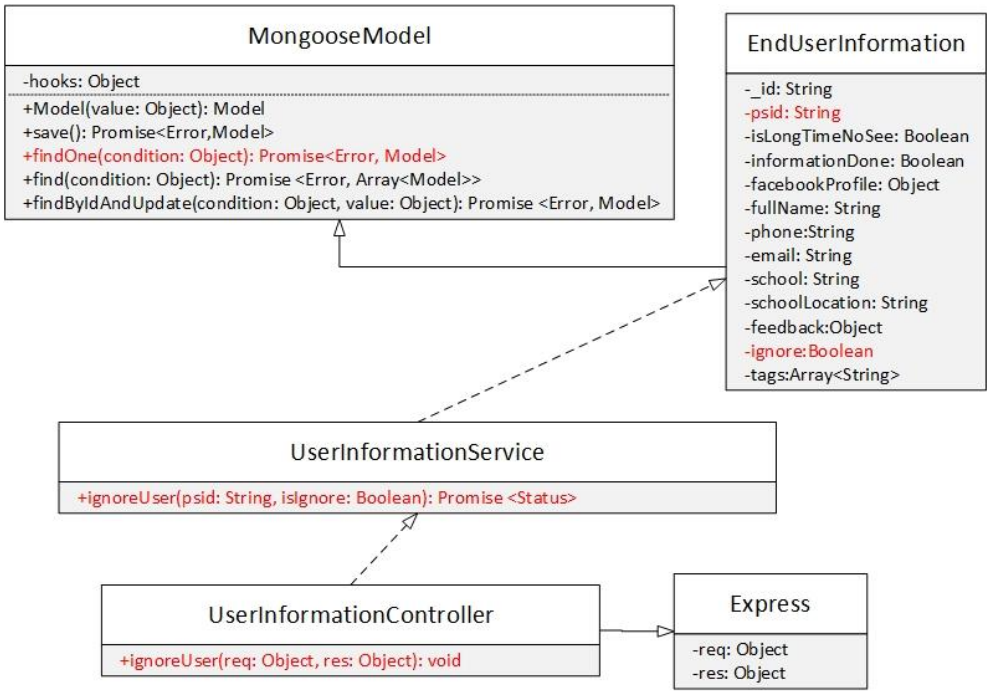


Figure 4-117: Staff changes bot status class diagram

Class Specification

EndUserInformation

EndUserInformation	
Physical address	Backend /models/EndUserInformation.js
Base class	MongooseModel

Attributes			
No	Name	Type	Description
1	psid	String	
2	ignore	Boolean	
Operation			
N/A	N/A		

EndUserInformationService

EndUserInformationService			
Physical address	Backend /controllers/EndUserInformationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
ignoreUser			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	psid	String	
	isIgnore	Boolean	

EndUserInformationController

EndUserInformationController			
Physical address	Backend /controllers/EndUserInformationController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
ignoreUser			
Return Type	void		
Parameters	Name	Type	Description
	req	object	
	res	object	

Sequence Diagram

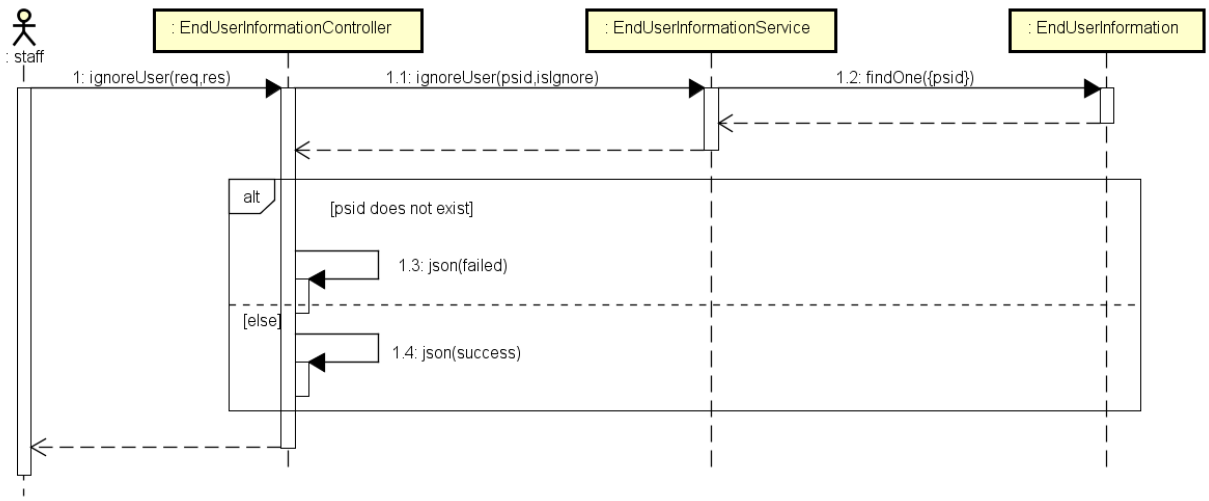


Figure 4-118: Staff changes bot status sequence diagram

4.3.4.29 Staff logouts

Screen Design

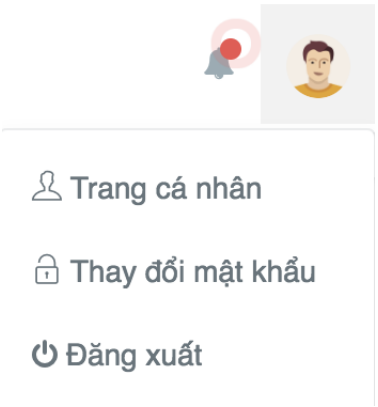


Figure 4-119: Staff logouts screen design

Class diagram

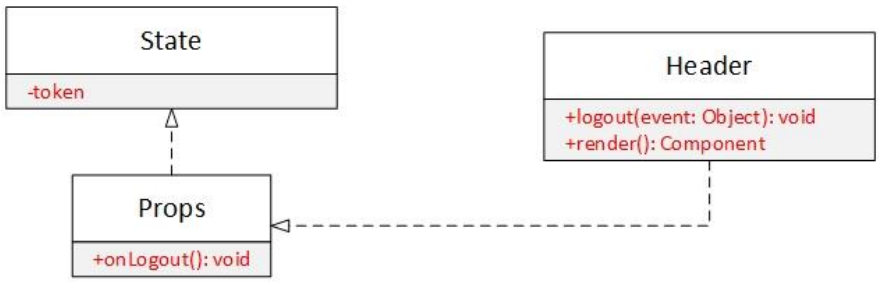


Figure 4-120: Staff logouts class diagram

Class Specification

State

State			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
1	token	String	
Operation			
N/A	N/A		

Props

Props			
Physical address	N/A		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
onLogout			
Return Type	void		
Parameters	Name	Type	Description
	N/A	N/A	N/A

Header

Header			
Physical address	Src/components/Header.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
Operation			
logout			
Return Type	void		
Parameters	Name	Type	Description
	event	object	
render			
Return Type	Component		
Parameters	Name	Type	Description
	N/A	N/A	N/A

Sequence Diagram

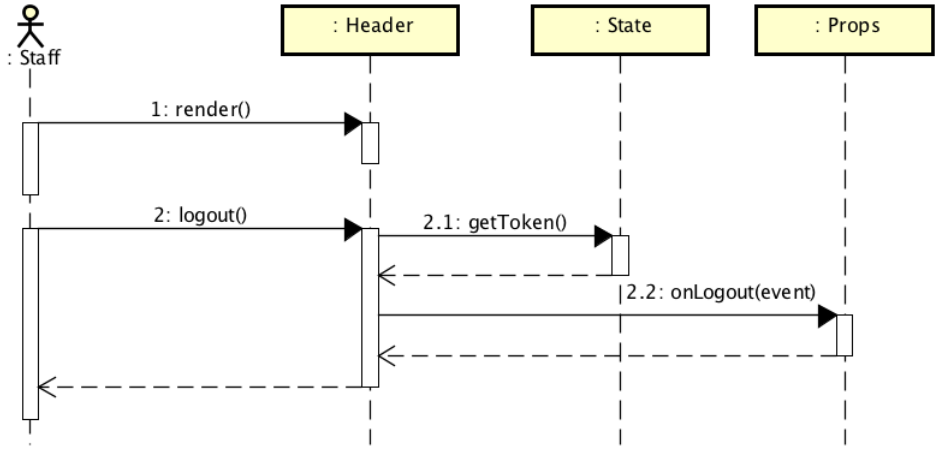


Figure 4-115: Staff logouts sequence diagram

4.3.4.30 Admin views answer editing requests

Screen Design

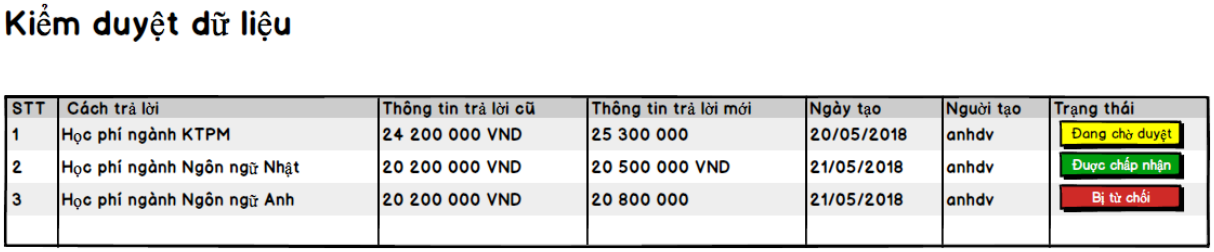


Figure 4-121: Admin views answer editing requests screen design

Class Diagram

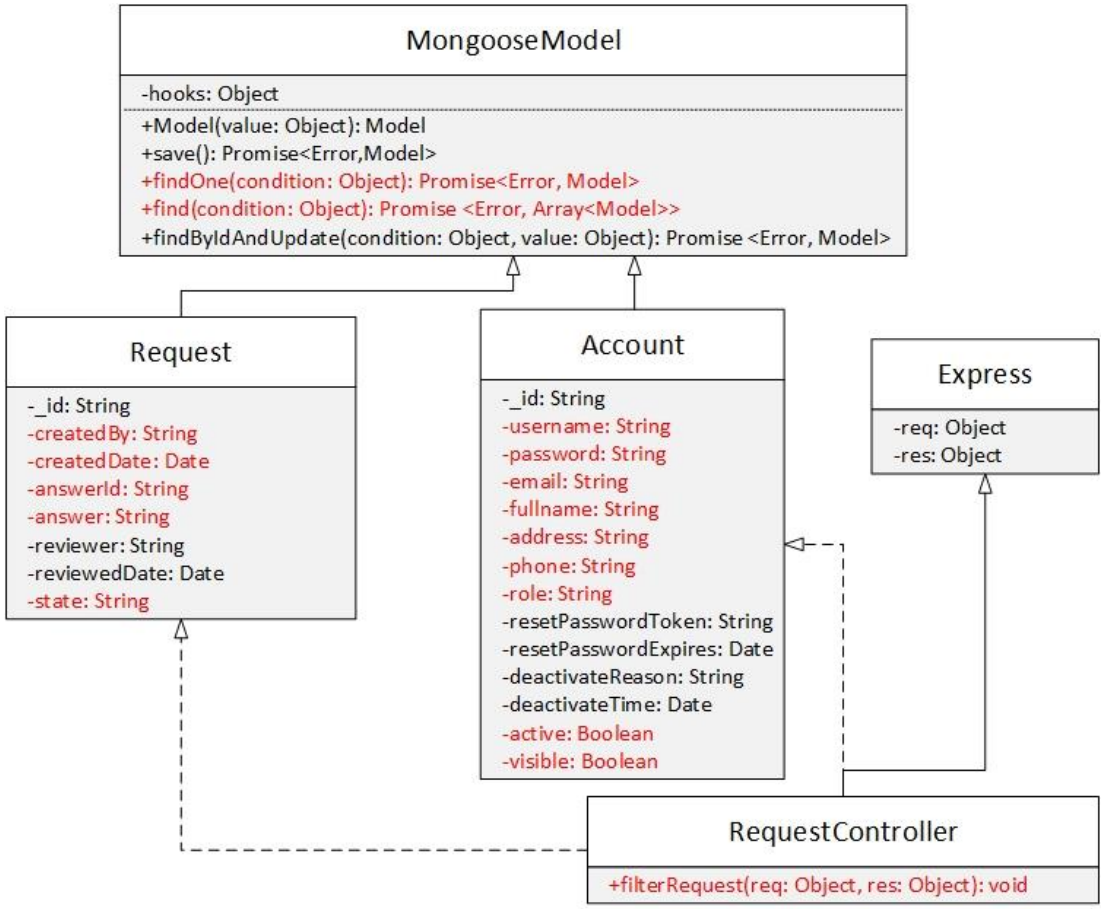


Figure 4-122: Admin views answer editing requests class diagram

Class Specification

Request

Request			
Physical address	Backend /models/Request.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	createdBy	String	
2	createdDate	Date	
3	answerId	String	
4	answer	String	
5	state	String	
Operation			

Account

Account			
Physical address	Backend /models/Account.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	username	String	
Operation			

RequestController

RequestController			
Physical address	Backend /controllers/RequestController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
filterRequest			
Return Type	void		
Parameters	Name	Type	Description
	req	Object	
	res	Object	

Sequence Diagram

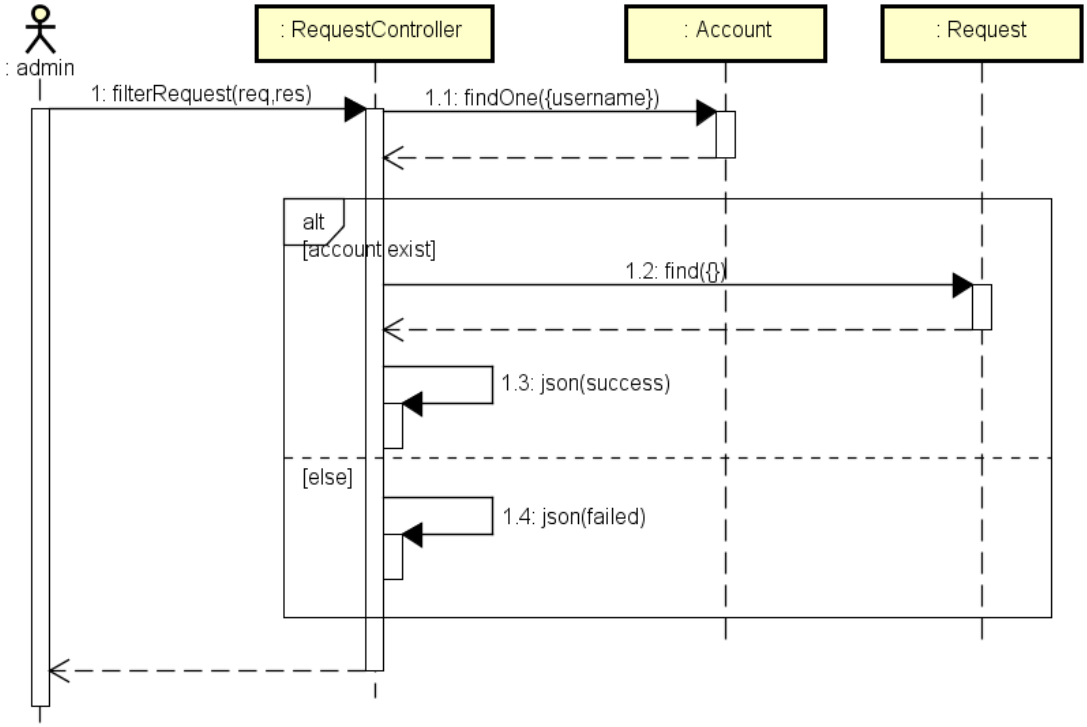


Figure 4-123: Admin views answer editing requests sequence diagram

4.3.4.31 Admin accepts an answer editing request

Screen Design

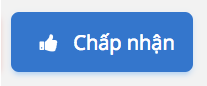


Figure 4-124: Admin accepts an answer editing request screen design

Class Diagram

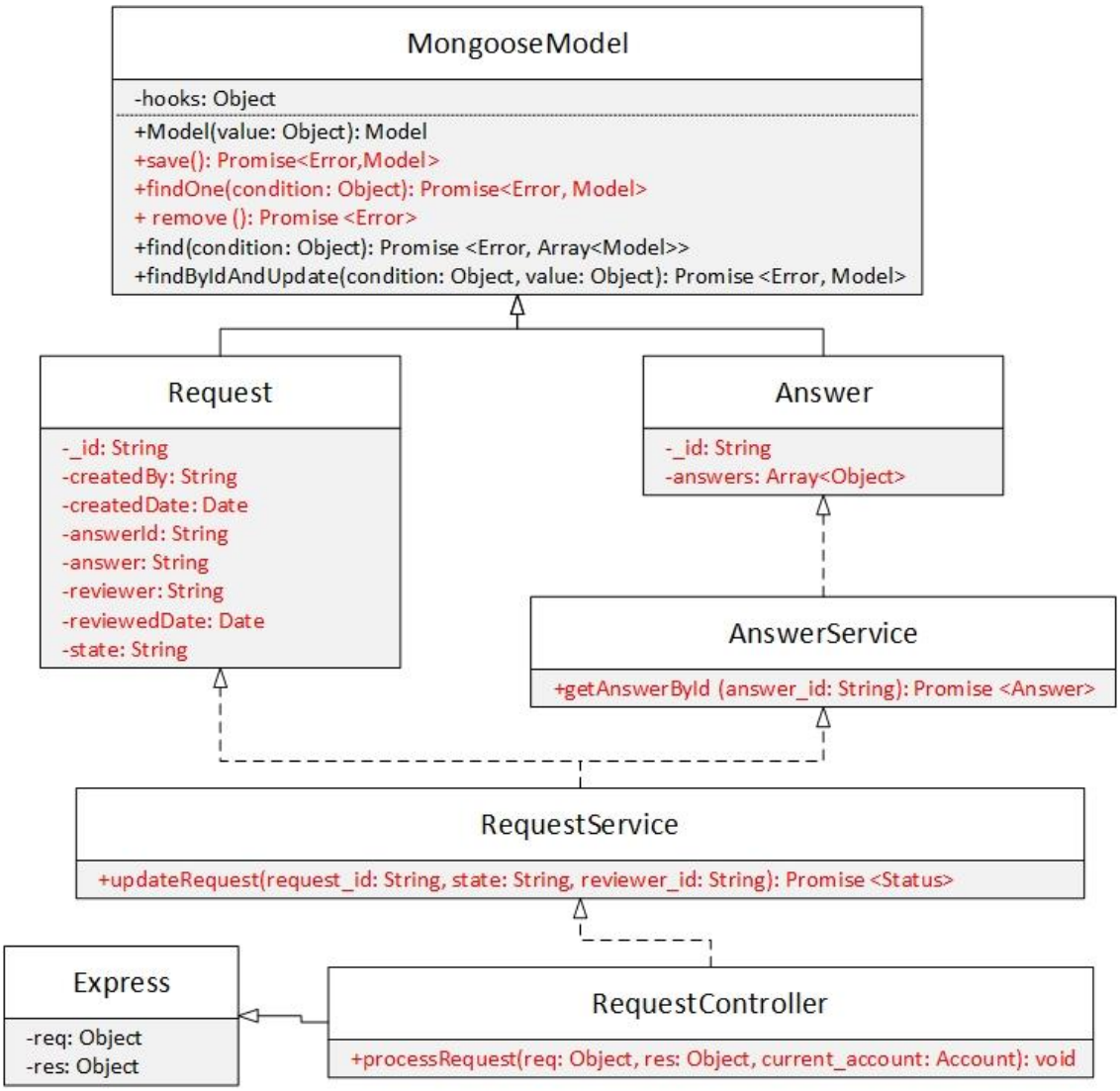


Figure 4-125: Admin accepts an answer editing request class diagram

Class Specification

Request

Request			
Physical address	Backend /models/Request.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	_id	String	
2	createdBy	String	
3	createdDate	String	
4	answerId	String	
5	answer	String	
6	reviewer	String	
7	reviewedDate	Date	
8	state	String	

Operation	
N/A	N/A

Answer

Answer			
Physical address	Backend /models/Answer.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	_id	String	
2	answers	Array<Object>	
Operation			
N/A	N/A		

AnswerService

AnswerService			
Physical address	Backend /services/AnswerService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getAnswerById			
Return Type	Promise<Answer>		
Parameters	Name	Type	Description
	answer_id	String	

RequestService

RequestService			
Physical address	Backend /services/RequestService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
updateRequest			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	request_id	String	
	reviewer_id	String	

RequestController

RequestController			
Physical address	Backend /controllers/RequestController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
processRequest			

Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.
	current_accoun	Account	

Sequence Diagram

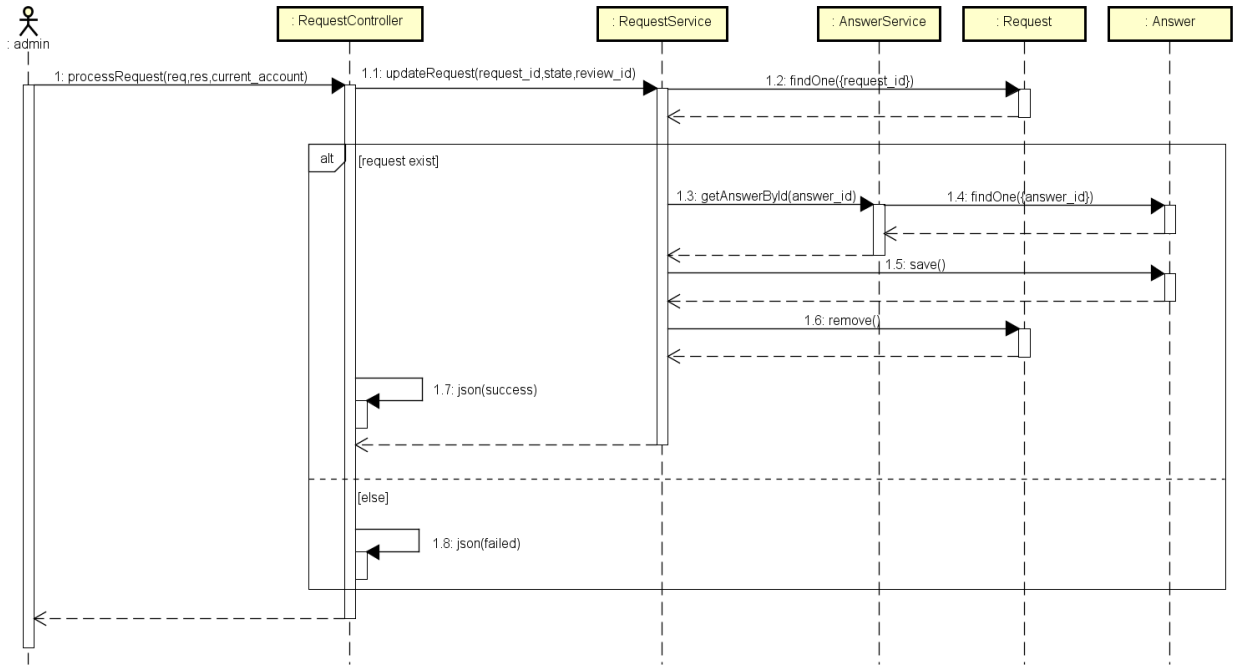


Figure 4-126: Admin accepts answer editing request sequence diagram

4.3.4.32 Admin rejects an answer editing request

Screen Design

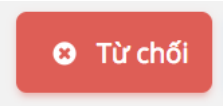


Figure 4-127: Admin rejects an answer editing request screen design

Class Diagram

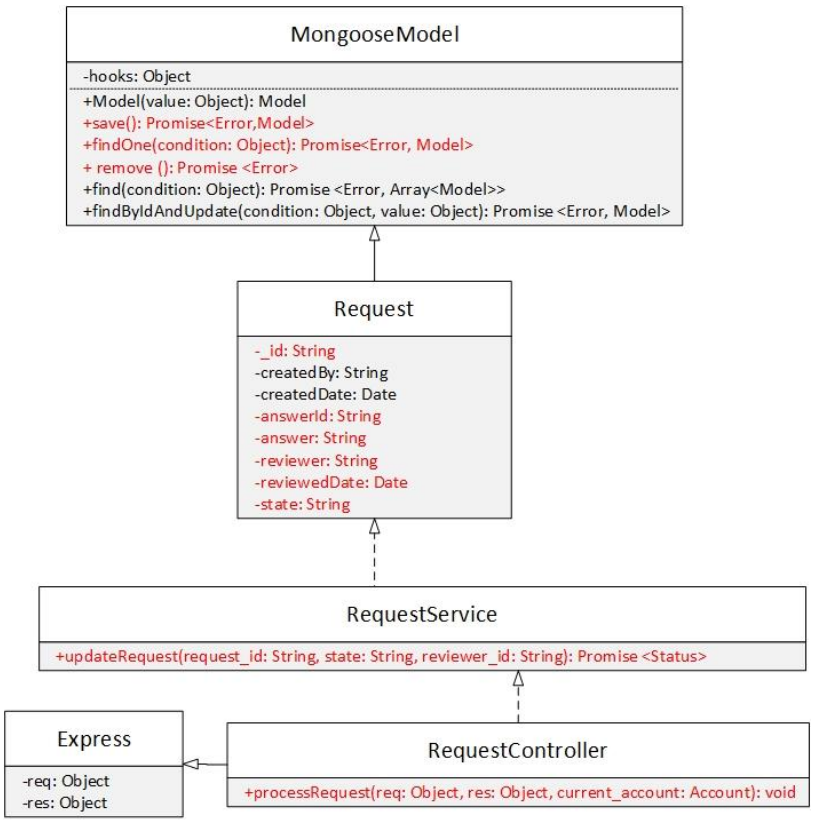


Figure 4-128: Admin rejects an answer editing request class diagram

Class Specification

Request

Request			
Physical address	Backend /models/Request.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	_id	String	
2	reviewer	String	
3	answerId	String	
4	answer	String	
5	state	String	
6	reviewedDate	Date	
Operation			
N/A	N/A		

RequestService

RequestService			
Physical address	Backend /services/RequestService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			

updateRequest			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	request_id	String	
	reviewer_id	String	

RequestController

RequestController			
Physical address	Backend /controllers/RequestController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
processRequest			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.
	current_accoun	Account	

Sequence Diagram

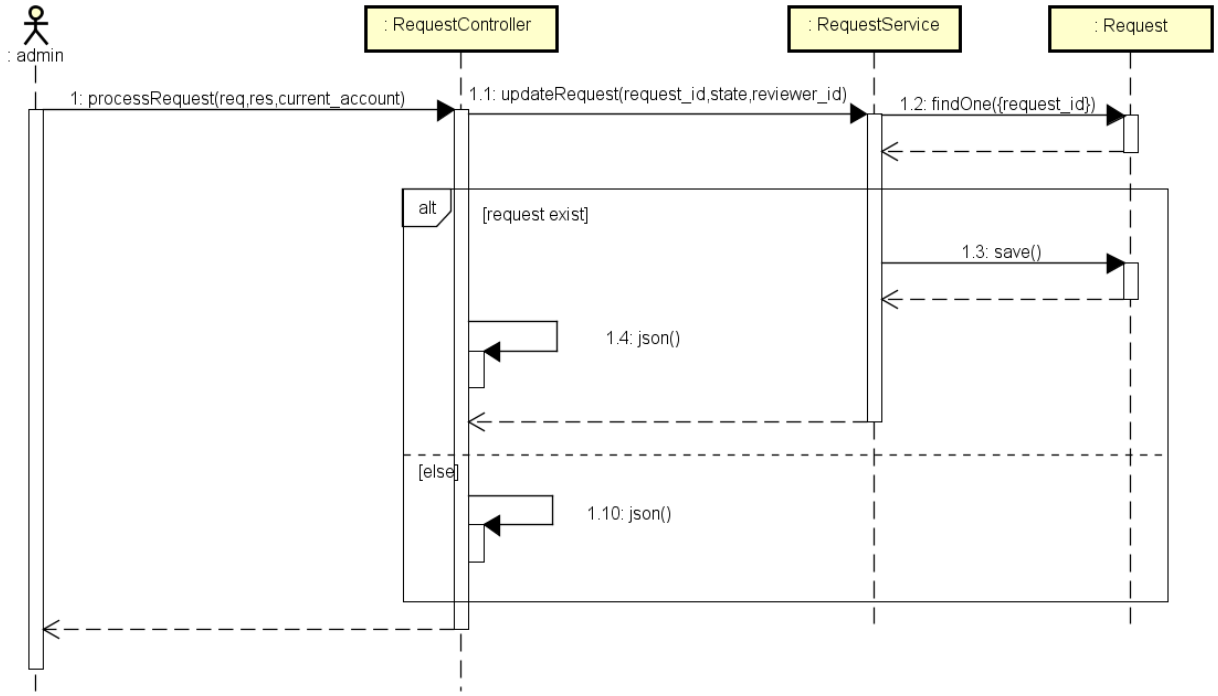


Figure 4-129: Admin rejects an answer editing request sequence diagram

4.3.4.33 Admin creates account

Screen Design

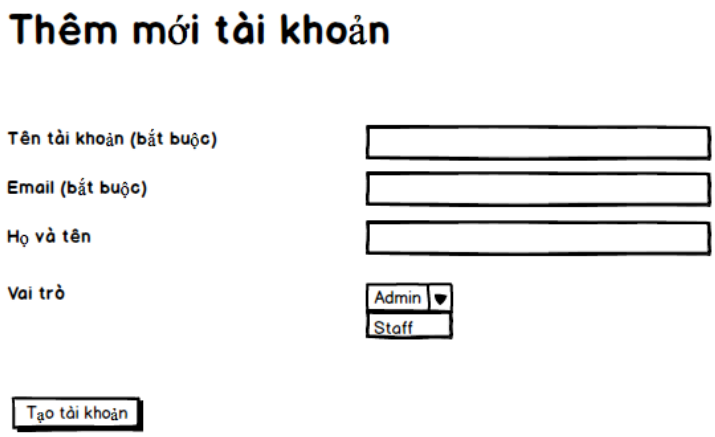


Figure 4-130: Admin creates account screen design

Class Diagram

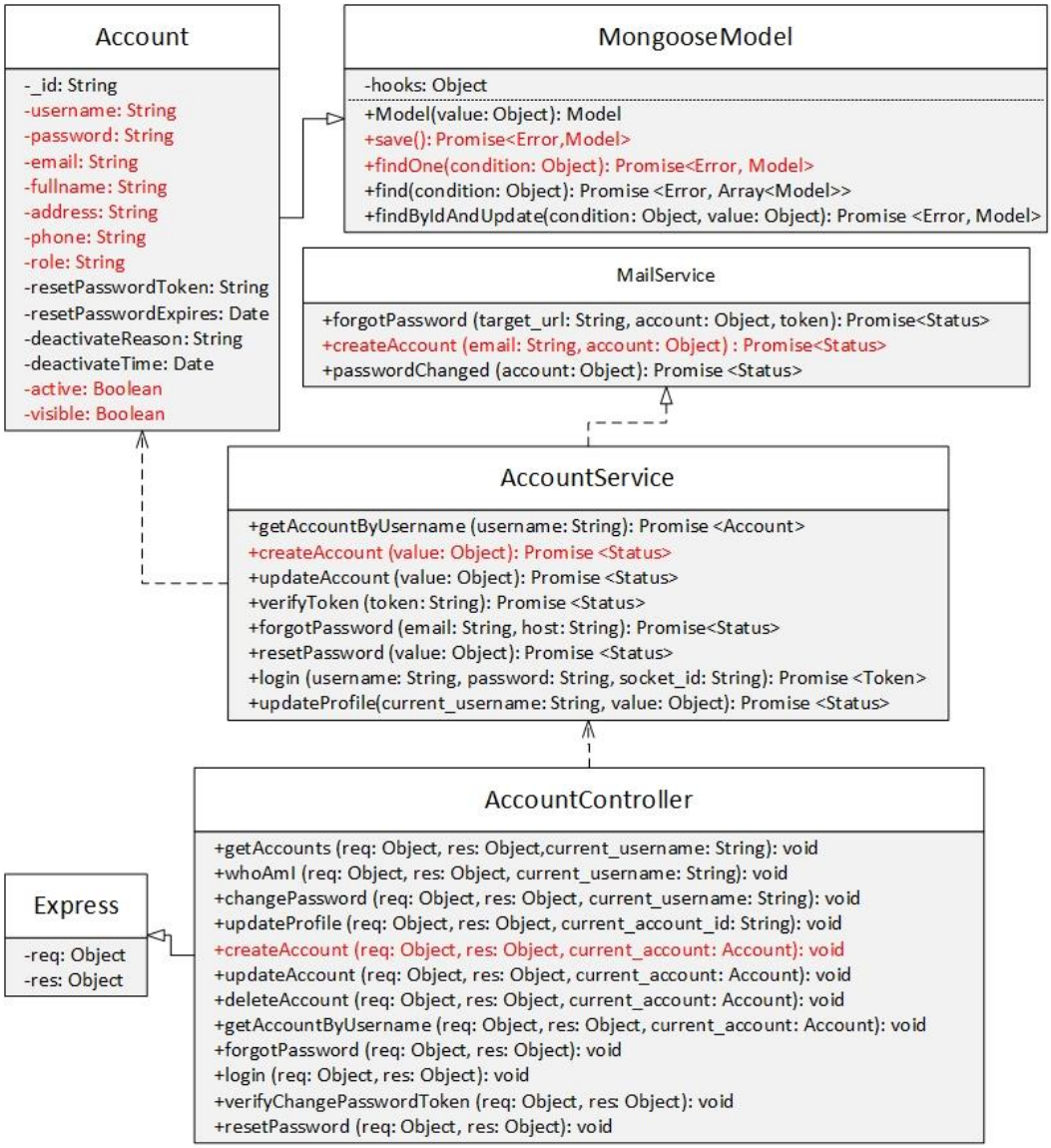


Figure 4-131: Admin creates account class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	username	String	
2	password	String	
3	fullname	String	
4	email	String	
5	address	String	
6	role	String	
7	active	Boolean	
8	visible	Boolean	
Operation			
N/A	N/A		

AccountService

AccountService			
Physical address	Backend /controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
createAccount			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	value	object	The object contains account information

MailService

MailService			
Physical address	Backend /controllers/MailService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
createAccount			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	email	String	
	account	object	

AccountController

AccountController	
Physical address	Backend /controllers/AccountController.js

Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
createAccount			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.
	current_account	Account	

Sequence Diagram

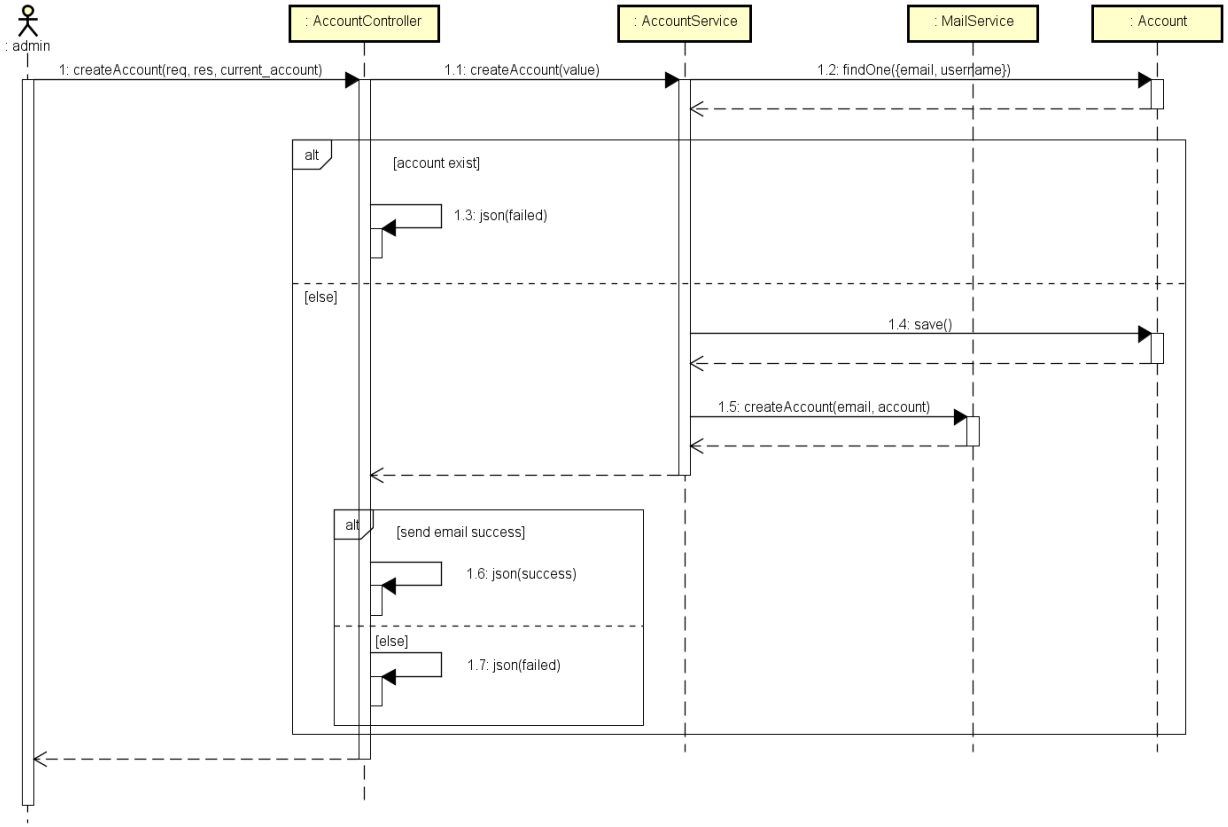


Figure 4-132: Admin creates account sequence diagram

4.3.4.34 Admin deletes account

Screen Design

Figure 4-133: Admin deletes account screen design

Class Diagram

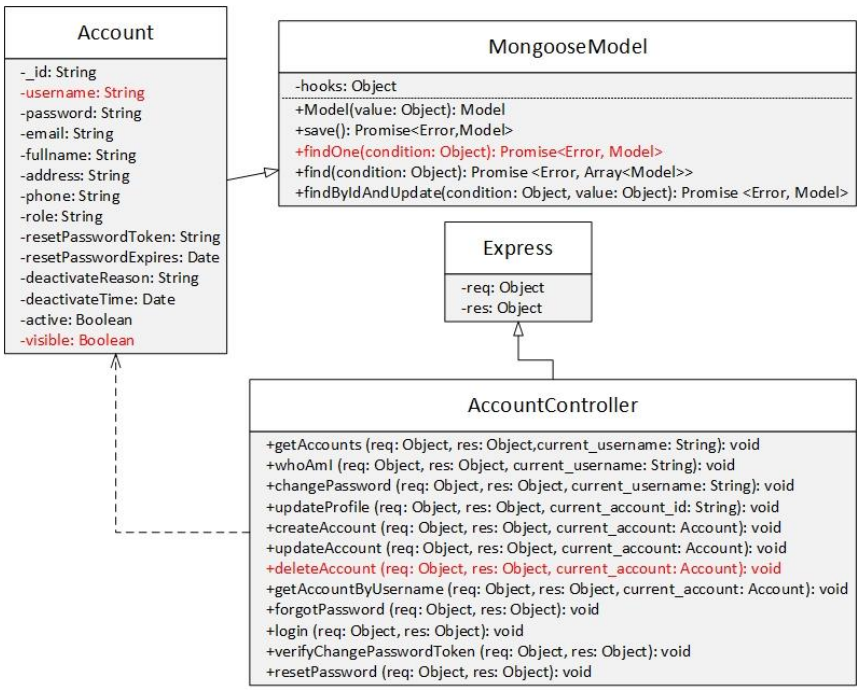


Figure 4-134: Admin deletes account class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	username	String	
2	visible	String	
Operation			
N/A	N/A		

AccountController

AccountController			
Physical address	Backend /controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
deleteAccount			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.

	res	object	The object contains response information.
	current_account	Account	

Sequence Diagram

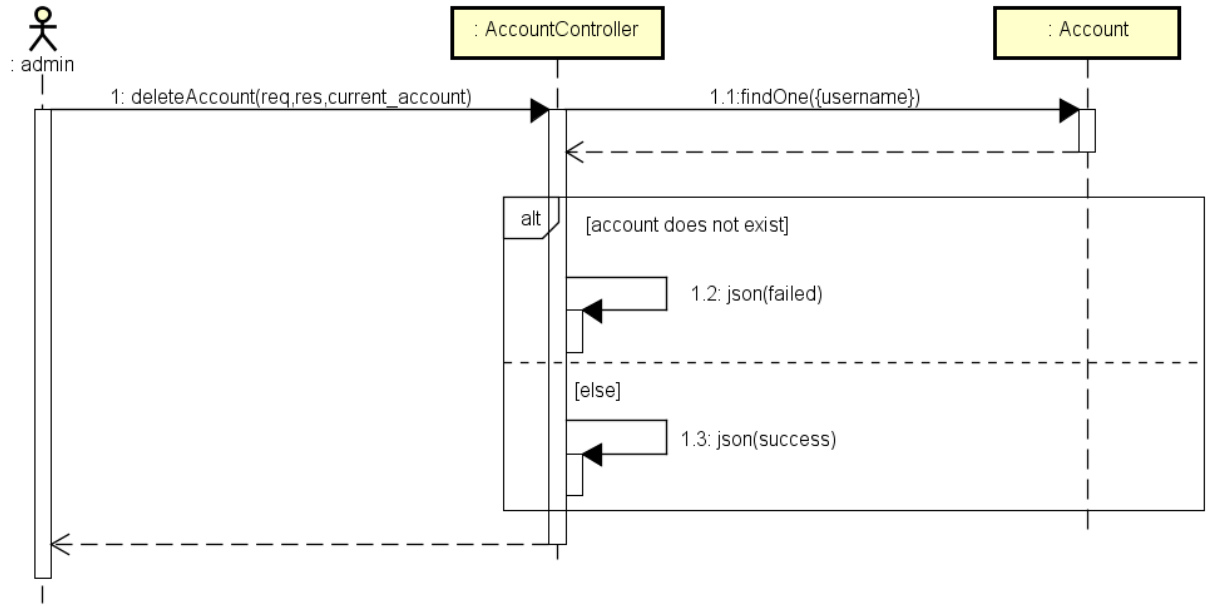


Figure 4-135: Admin deletes account sequence diagram

4.3.4.35 Admin changes account role

Screen Design

Figure 4-136: Admin changes account role screen design

Class Diagram

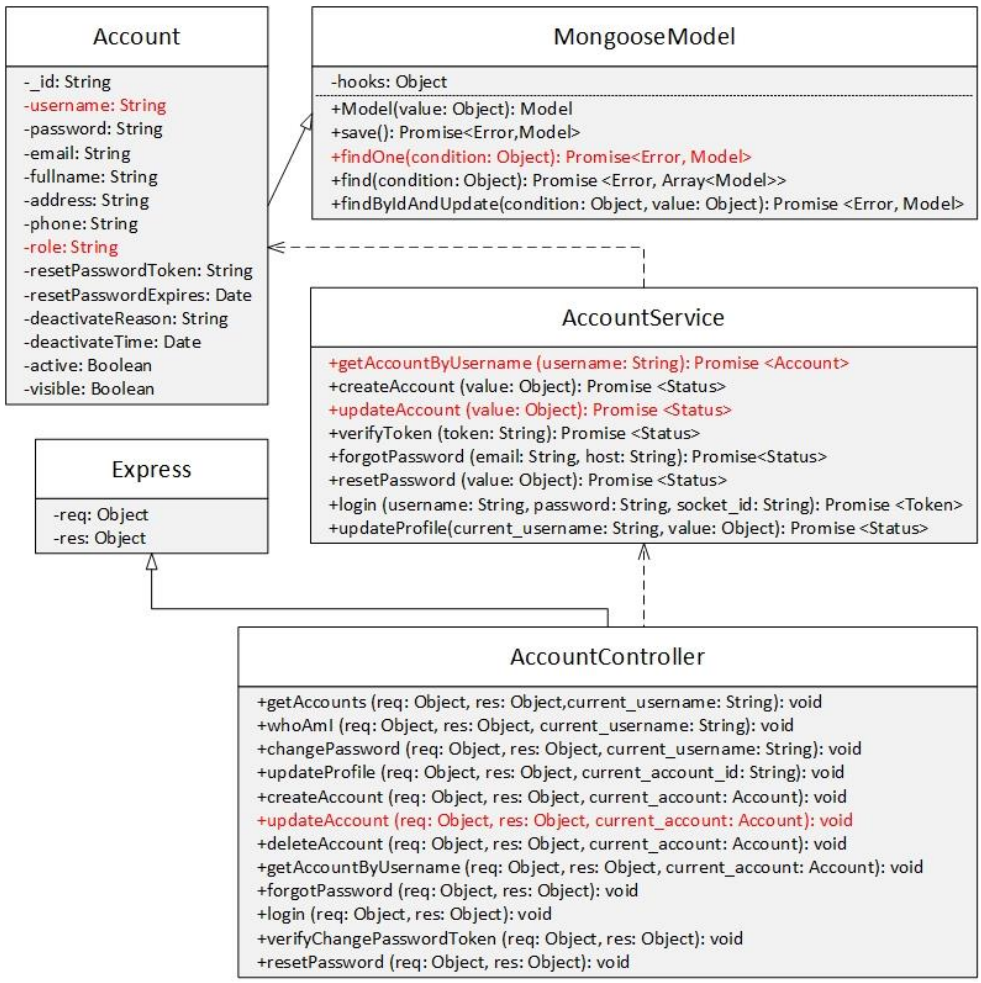


Figure 4-137: Admin changes account role class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	username	String	
2	role	String	
Operation			

AccountService

AccountService			
Physical address	Backend /controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getAccountByUsername			

Return Type	Promise<Account>		
Parameters	Name	Type	Description
	username	String	
updateAccount			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	value	object	The object contains account information.

AccountController

AccountController			
Physical address	Backend /controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
Operation			
updateAccount			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.
	current_account	Account	

Sequence Diagram

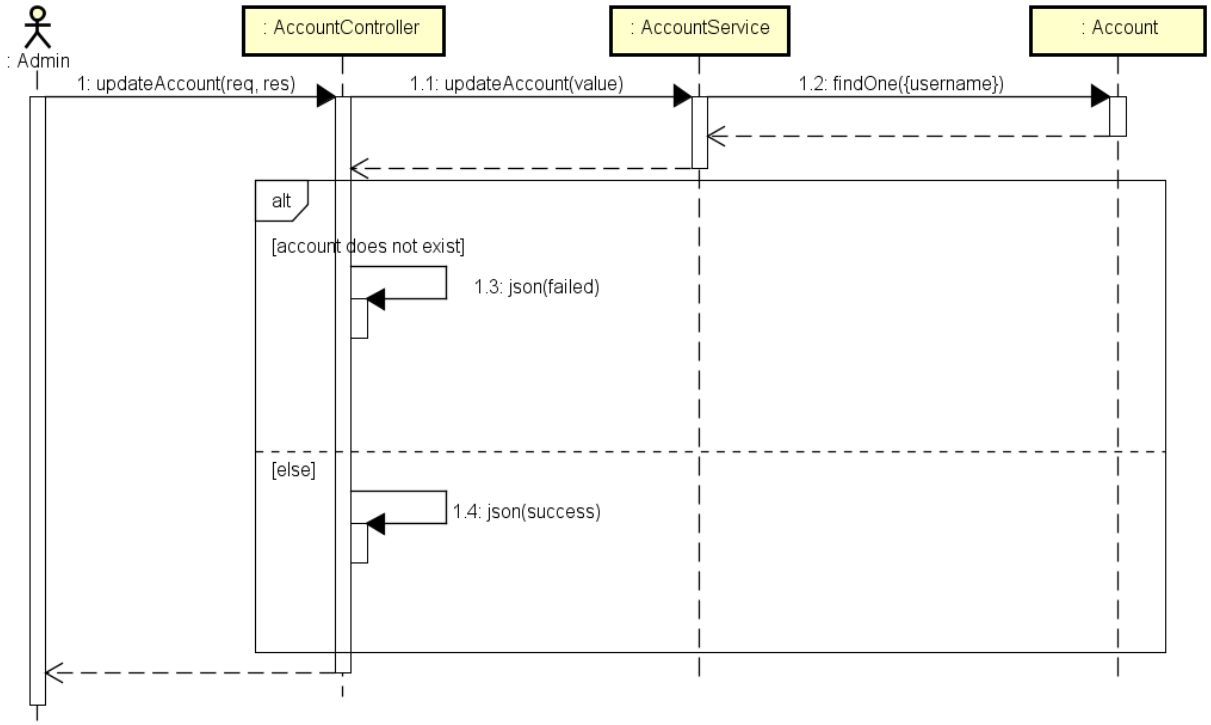


Figure 4-138: Admin changes account role sequence diagram

4.3.4.36 Admin changes account status

Screen Design

Figure 4-139: Admin changes account status screen design

Class Diagram

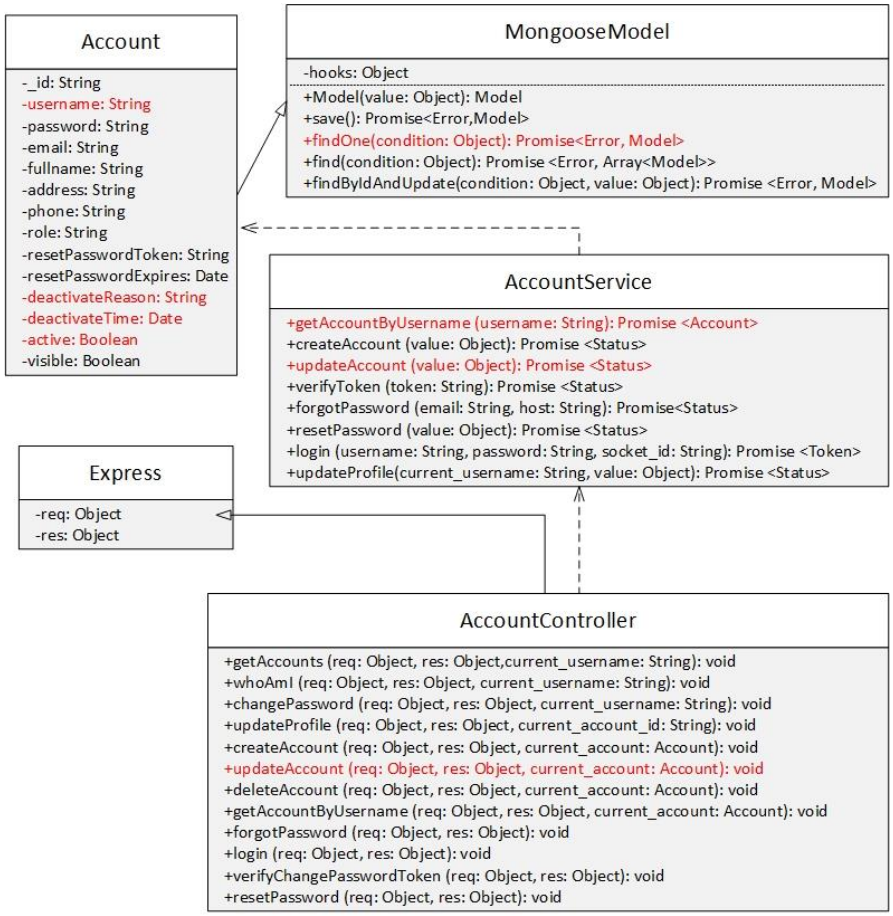


Figure 4-140: Admin changes account status class diagram

Class Specification

Account

Account			
Physical address	Backend /models/Account.js		
Base class	Mongoose Model		
Attributes			
No	Name	Type	Description
1	username	String	
2	active	String	
3	deactivateReason	String	
4	deactiveTime	Date	
Operation			

AccountService

AccountService

Physical address	Backend /controllers/AccountService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getAccountByUsername			
Return Type	Promise<Account>		
Parameters	Name	Type	Description
	username	String	
updateAccount			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	value	object	The object contains account information.

AccountController

AccountController			
Physical address	Backend /controllers/AccountController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
updateAccount			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.
	current_account	Account	

4.3.4.37 Admin broadcasts

Screen Design

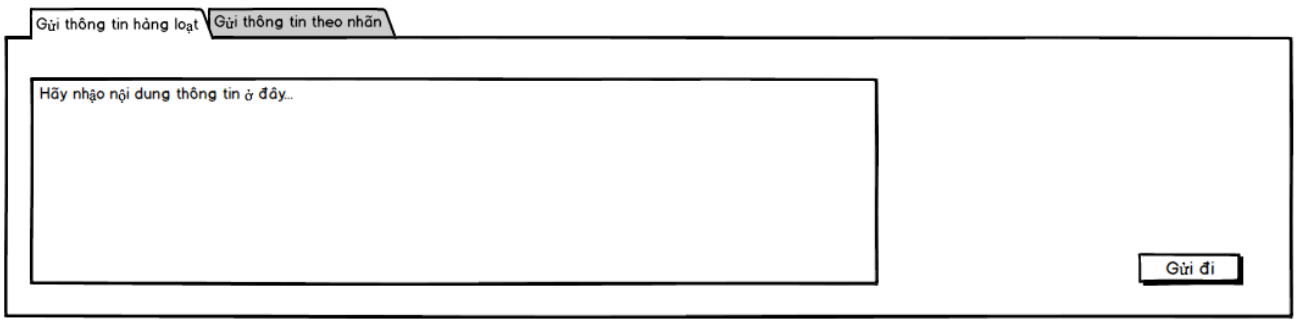


Figure 4-141: Broadcast screen design

Class Diagram

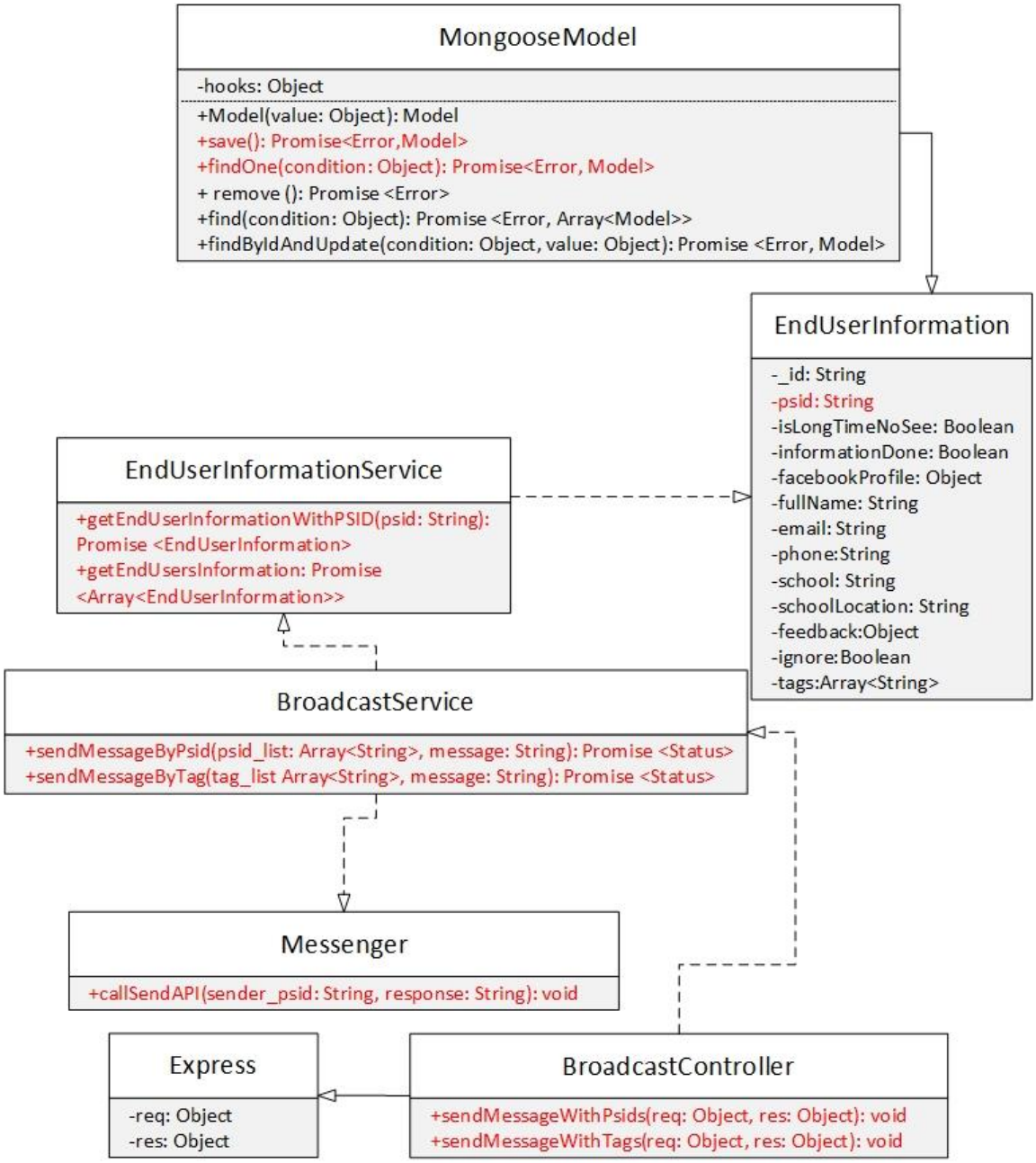


Figure 4-142: Admin broadcasts

Class Specification

EndUserInformation

EndUserInformation			
Physical address	Backend /models/EndUserInformation.js		
Base class	MongooseModel		
Attributes			
No	Name	Type	Description
1	psid	String	
Operation			

EndUserInformationtService

EndUserInformationService			
Physical address	Backend /controllers/EndUserInformationService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
getEndUserInformationWithPSID			
Return Type	Promise<EndUserInformation>		
Parameters	Name	Type	Description
	psid	String	
getEndUserInformation			
Return Type	Promise<Array<EndUserInformation>>		
Parameters	Name	Type	Description
	N/A	N/A	N/A

BroadcastService

BroadcastService			
Physical address	Backend /controllers/BroadcastService.js		
Base class	N/A		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
sendMessageByPsid			
Return Type	Promise<Status>		
Parameters	Name	Type	Description
	psid_list	Array<String>	
sendMessageByTag			
Return Type	Promise<Array<Status>>		
Parameters	Name	Type	Description
	tag_list	Array<String>	
	message	String	

BroadcastController

BroadcastController			
Physical address	Backend /controllers/BroadcastController.js		
Base class	Express		
Attributes			
No	Name	Type	Description
N/A	N/A	N/A	N/A
Operation			
sendMessageWithPsids			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.

sendMessageWithTags			
Return Type	void		
Parameters	Name	Type	Description
	req	object	The object contains request information.
	res	object	The object contains response information.

Sequence diagram

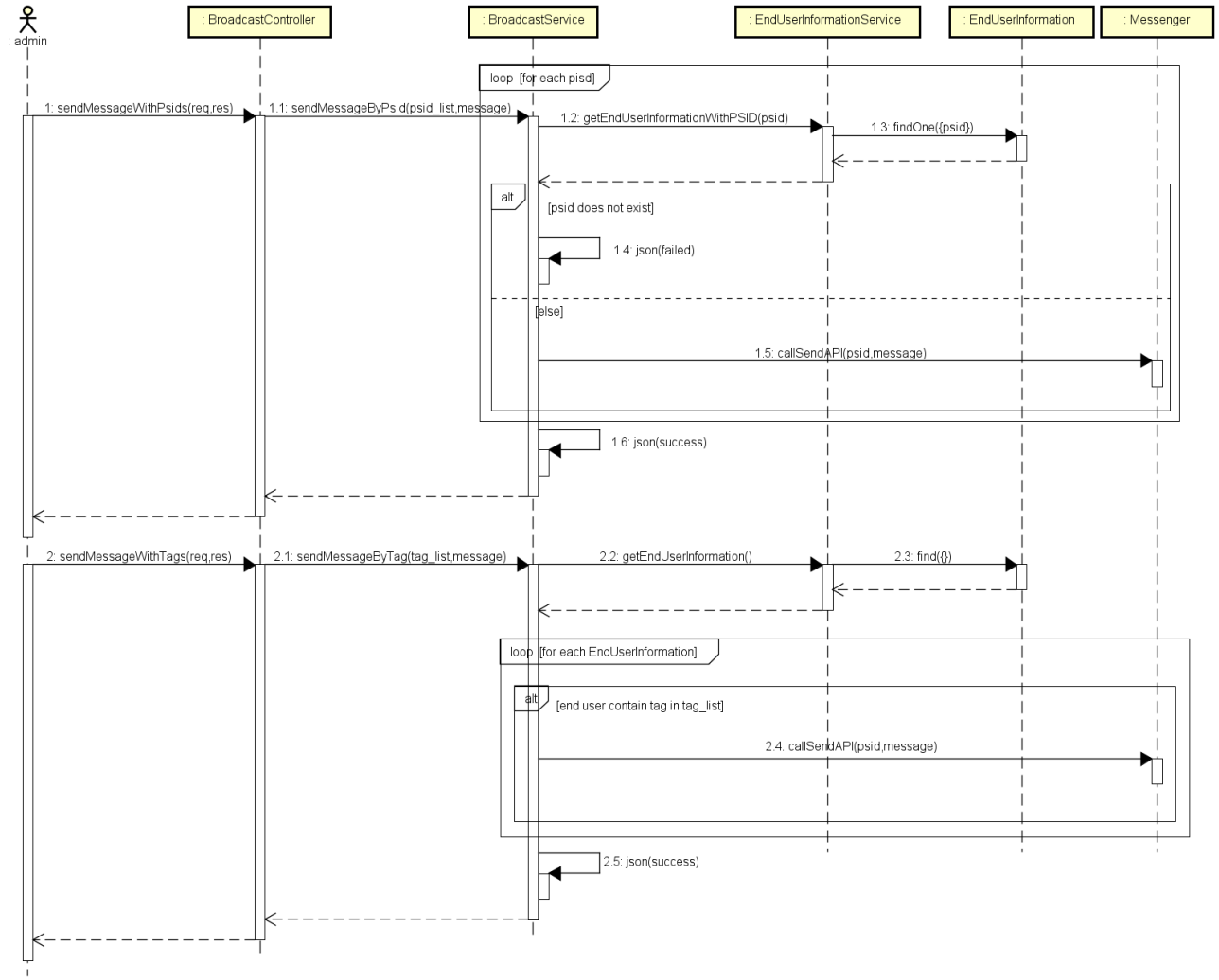


Figure 4-143: Broadcast sequence diagram

Chapter 5 : Software Testing Documentation

5.1 Purpose

The primary purpose of this chapter is to detect and prevent defects which may be created by developers while developing the software and this may lead to software failures. On the other hand, another objective of this chapter is to provide information about the level of quality and to make sure that the end result meets the business and user requirements. It contains the following sections:

- Scope of Testing.
- Testing Tool and Environment.
- Resources and responsibilities.
- Test strategy: Test approach, test stages.
- Test schedule.
- Feature to be tested.
- Feature not to be tested.
- Defect Log.
- Test report.

5.1.1 Scope of testing

- Testing phases

There are 4 phases in Testing Process: Unit testing, Integration testing, System testing and Acceptance testing.

ID	Testing Phase	Description
1	**Unit testing**	The main aim of this endeavor is to determine whether the application functions as designed. In this phase, a unit can refer to a function, individual program or even a procedure, and a White-box Testing method is usually used to get the job done.
2	**Integration testing**	Integration testing allows individuals the opportunity to combine all of the units within a program and test them as a group. This testing level is designed to find interface defects between the modules/functions. This is particularly beneficial because it determines how efficiently the units are running together.
3	**System testing**	System testing is the first level in which the complete application is tested as a whole. The goal at this level is to evaluate whether the system has complied with all of the outlined requirements and to see that it meets Quality Standards.
4	**Acceptance testing**	Acceptance testing (or User Acceptance Testing), is conducted to determine whether the system is ready for release. During this phase, the tester and some alpha test user will test the system to find out whether the application meets their business' needs.

- **Testing types**
    - GUI testing
    - Function testing
    - Performance testing
    - Regression testing
    - Acceptance testing including alpha testing
    - Unit testing
    - Load testing
- **Testing range**

  Team performs all functions defined in the SRS based on the approved version.

5.2 Test plan

5.2.1 Testing tools and environment

5.2.1.1 Testing tools

5.2.1.1.1 ASC Front-end and Project testing

- **Chrome DevTools:** To view logs, inspect elements.

Figure 5-1: Chrome Developer Tools

- **Redux DevTools:** To view Redux action and state tree.
- **Backlog:** To manage bugs: Show all bugs and life cycle of process to resolve bug.

Figure 5-2: Backlog

- **Microsoft Excel:** To manage test cases

Figure 5-3: Microsoft Excel

- **Postman:** to manage the list of all APIs and manually test API

Figure 5-4: Postman

5.2.1.1.2 ASC API testing

- **Mocha [1]:** a feature-rich JavaScript test framework running on Node.js and in the browser, using to make asynchronous testing simple.

Figure 5-5: MochaJS

- **Chaijs [2]:** Chai is a BDD / TDD assertion library for node, to help assertion result in a test.

Figure 5-6: ChaiJS

- **Chai as Promised [3]:** Chai as Promised extends Chai with a fluent language for asserting facts about promises, to assert result with Promises.
- **SuperTest [4]:** A Super-agent[5] driven library for testing Node.js HTTP servers using a fluent API, use to make a test with from high-level abstraction to the low-level API.
- **Nyc[6]**: command-line-client for Istanbul works well with most JavaScript testing frameworks: tap, mocha, AVA, …

5.2.1.2 Testing environment

---

[1] Mocha main page: https://mochajs.org
[2] Chai assertion library main page: http://chaijs.com
[3] Chai as promised project: https://github.com/domenic/chai-as-promised
[4] SuperTest project: https://github.com/visionmedia/supertest
[5] SuperAgent main page: http://visionmedia.github.io/superagent
[6] Nyc project: https://github.com/istanbuljs/nyc

Type of testing	Software	Hardware
System test and Acceptance test	- Microsoft Office Excel 2016. - Microsoft Office Word 2016 - Chrome Version 68.0.3071.115	Personal computer for developing with the minimum configuration: - Windows 10 Professional 64-bit - Intel® Core™ i3 - Installed memory (RAM): 8.00GB - or - Ubuntu 16.04 64-bit - Intel® Core™ i3 - Installed memory (RAM): 8.00GB
Unit testing and API testing	- Chaijs v4.1.2 - Chai as promised v4.1.2 - Mocha v5.2.0 - SuperTest v3.1.0 - Nyc v12.0.2	Personal computer for developing with the minimum configuration: - Windows 10 Professional 64-bit - Intel® Core™ i3 - Installed memory (RAM): 8.00GB - or - Ubuntu 16.04 64-bit - Intel® Core™ i3 - Installed memory (RAM): 8.00GB

5.2.2 Resources and responsibilities

ID	Resources	Responsibilities
1	Project Manager	• Responsible for Project Schedules and overall success of the project. • Review Test-case and report.
2	Tester	• Preforming the actual system testing. • Manage test resource and assign test tasks. • Create Test Plan. • Create Test Cases. • Create Test Report. • Execute Test. • Test Log report.
3	Developer	• Create unit test and integration test scripts. • Fix bugs.

5.2.3 Test strategy

5.2.3.1 Testing model

Overall, ASC project uses the Iterative and Incremental Software Process Model, ASC is divided into 2 sub-systems: ASC API and ASC Frontend. In each phase of process, we use specific process for each sub-system team to fit the requirement, the characteristic and the human resource of each team.

Since APIs lack GUI and need to change source code rapidly as ASC Frontend requires, so that ASC API applies Test-driven development (TDD) and Behavior Driven Development (BDD) process, which covers source code by Unit testing and API testing at the message layer. In the development time, whenever we add a new feature or change the old features, we will add/modify the tests first, then write code to make the test pass then refactor the code and refactor the test at the last.

ASC API has 2 levels of test:

- Unit testing: Automation tests that cover logic of XXXService.js files.
- API testing: Automation tests that involve testing APIs directly (in isolation) to determine whether APIs return the correct response (in the expected format) for a broad range of feasible requests, react properly to edge cases such as failures and unexpected/extreme inputs.

ASC Frontend works mostly with GUI instead of logic and it depends on ASC API, so that ASC Frontend applies System testing which covers the whole ASC system.

5.2.3.2 Testing types

Testing ASC Project will be carried out in each release package as defined in project plan and will depend on internal delivery by development side. Critical tests will be defined as any new or modified tests for the ASC system. Project Manager will decide which closed defects the test team cannot determine.

Each type of ASC Project Testing will be designated to cover a level of project, from each model level, API level, ASC API sub-system level and ASC Project level with Unit testing combined with Database testing, API testing, System testing and Acceptance testing. On the other hand, ASC will use several of testing type to prevent defects from code modifications and latent bugs.

The different types of testing that will be carried out this project are:

- Unit testing:
  - Testing all individual implemented methods, functions of XXXService.js files.
  - Unit test also includes database testing to verify constraint, transaction, default value, data types, data format which are mentioned in database design and software requirement.
  - Test case will have to cover all logic branch that function or method could execute with difference data input. Another alternative logic branch should be covered if not, that logic branch should be detected at API testing level.
  - Implemented function’s error message and database error message will be included in this test.
- API testing:
  - Involves testing APIs directly to determine if they meet expectations for functionality, reliability, performance, and security. API testing will test all of individual implemented API of ASC API.

- Testing will mock all possible datasets and call to corresponding API with separate testing database by using Chaijs, SuperTest and Mocha.
- Test case will verify constraint of data which be mention in Business rule.
- Basically, almost all API test cases are executed as automation test. After that all API with standard sample datasets will be saved and confirmation tests will be executed by using Postman with developer's local database.

- UI testing
  - User Interface testing verifies a user's interaction with the software. The goal of GUI testing is to ensure that the GUI provides the user with an appropriate access and navigation through the functions of the target-of-test. In addition, GUI testing ensures that the objects within the GUI function as expected and conform to requirement.
  - This testing type targets to cover the verification of the overall look and feel of the ASC system including initial position, font, text size, color, focus, initial button, tab order, label, screen sizes and sentences width.
  - Check all the GUI elements for size, position, width, length and acceptance of characters or numbers. For instance, it must be able to provide inputs to the input fields.

1. Check if Error Messages are displayed correctly.
2. Check if Font used in application is readable.
3. Check if the alignment of the text is proper.
4. Check if the Color of the font and warning messages is aesthetically pleasing.
5. Check if images have good clarity.
6. Check if images are properly aligned.
7. Check the positioning of GUI elements for different screen resolution.

- Performance testing
  - The testing is to confirm to cover for response time for ASC System with different network conditions with the throttling tool of Google Chrome
- Regression testing
  - The testing is to confirm that the bug was removed including the extent of the impact, when developers fix bug, developers and testers will confirm with each other what is the impacts of fix bug modification, after that all impact unit or function will be retested by developers then testers after that.
- Acceptance testing
  - This test type will be executed by tester with designed test cases, acceptance test is a test type conducted to determine if the requirements of a specification or contract are met.
  - It also includes alpha testing alpha testing takes place at close relation user's site and are free test to detect bug and strange behavior. By that, development team will improve UX and UI of system
- Load testing

- The testing is executed by developers to estimate a web server performance in order to find if the server can serve sufficiently high workload by using Artillery[1]

5.2.3.3 Testing phases

Table below are the stages in which common tests are executed:

Testing type	Testing phase			
	Unit testing	**Integration testing**	**System testing**	**Acceptance testing**
Unit Testing	x	x	x	
API Testing	x	x		
UI Testing				x
Regression Testing	x	x	x	x
Performance Testing			x	x
Acceptance Testing			x	
Load Testing			x	x

Table 5-1: Testing phase

5.2.3.4 Test schedule

Table below is the Test Schedule for ASC Project:

Test Schedule	Start Date	End Date
Phase 1: Authentication and Bot's related features	6/19/2018	7/9/2018
Unit Testing and API Testing	6/19/2018	7/8/2018
User Interface Testing	7/1/2018	7/2/2018
System Testing	6/30/2018	7/9/2018
Performance Tests	7/8/2018	7/8/2018
Regression Tests	7/9/2018	7/9/2018
Phase 2: Dashboard related features and optimization	7/13/2018	8/9/2018
Unit Testing and API Testing	7/13/2018	8/8/2018
User Interface Testing	1/8/2018	8/2/2018
System Testing	7/30/2018	8/9/2018
Performance Tests	8/7/2018	8/7/2018
Regression Tests	8/8/2018	8/8/2018
Acceptance testing	8/9/2018	8/9/2018

Table 5-2: Test schedule

5.2.3.5 Deliverables

Table below is the Deliverables for ASC Project:

Deliverables	Responsibilities	Complete date
Phase 1		
Test Plan	Tester	6/21/2018
Test Cases	Tester	6/29/2018
Test case review	Tester + PM	6/30/2018
Defect report	All members	7/8/2018

[1] https://artillery.io

Phase 2		
Test Plan	Tester	7/16/2018
Test Cases	Tester	7/26/2018
Test case review	Tester + PM	8/27/2018
Defect report	All members	8/7/2018
Final test Summary report	PM	8/10/2018

Table 5-3: Deliverables schedule

5.2.4 Features to be tested

Actor	Name
End User	Chat
	Leave feedback
	Leave personal information
Staff	Reset Password
	Login
	View statistics
	View profile
	Update profile
	Change password
	View account list
	View account information detail
	Search account
	Filter account
	View response & answer list
	Search response
	Filter response & answer
	Request to edit answer
	Receive notification
	View notification list
	View notification detail
	View Fb count page
	View chat logs
	View Fb account infor
	View providing infor
	Add tag
	Remove tag
	Export end user list
	Change bot status
	Logout
	View answer editing requests
	Accept answer editing request
	Reject answer editing request
Admin	Create account
	Delete account
	Change account role
	Change account status
	Broadcast

Table 5-4: Features to be tested

5.3 Test Case

5.3.1 Automation testing with API testing and Unit testing

Unit testing and API testing will be done by the developers and approved by team leader.

The ASC development team embrace this feature to gain the following advantages:

- Reduce the level of bugs in production code.
- Save development time.
- Automation tests can be run as frequently as required.
- Make it easier to change and refactor code by improving the design of code especially with Test-Driven Development.
- Can easily form a document from the tests.
- Easier to maintain than GUI tests which are difficult to maintain with the short release cycles and frequent changes and with a complex system
- Reduce cost of resource to corresponding GUI testing

5.3.1.1 Automation testing framework

- For API testing and Unit testing, we use Mocha testing framework combined with Chai assertion library and Istanbul for code coverage observation.
- All testing frameworks and libraries will be installed to ASC APIs project automatically by using npm package manager:

```
$ npm install
```

- All automation test scripts will be created manually and saved to **tests** directory of ASC API. Unit tests are stored in **api** and **service** folder, API tests will be storage in **api** folder.

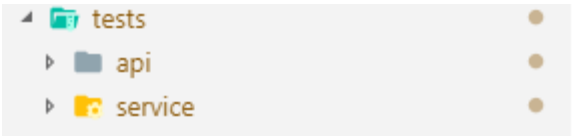


Figure 5-7: Test directory structure

- Unit tests focus on individual functions in class and will be created simply as below

```
describe("AccountService", () => {

    before(done => {
        mongoose.connect("mongodb://futo:abcd%402018@13.76.0.40:27017/futo-data'
    });

    describe("getAccountByUsername", function () {
        function callService(username, callback) {
            AccountService.getAccountByUsername(username).then(account => {
                callback(account);
            }).catch(err => callback(err));
        }
        describe("username exist", () => {
            it("it should be return a account", done => {
                const username = "anhdv";
                callService(username, function (account) {
                    expect(account.username).to.equal(username);
                    done();
                });
            });
        });
        describe("username does not exist", () => {
            it("it should be return undefined", done => {
                callService("hello", function (account) {
                    expect(account).to.equal(null);
                    done();
                });
            });
        });
    });
```

Figure 5-8: Unit test case sample
This getAccountByUsername function is used to retrieve an account info.
Test case will check if the getAccountByUsername handles the case that user does not exist.

- API tests focus on individual API and use SuperTest to test these API based on the business rule.

```
describe("Account API", () => {

    describe("GET /api/account", () => {···
    });

    describe("GET /api/account/me", () => {
        describe("get information of current account", () => {
            it("it should be return a account", done => {
                request(app)
                    .get("/api/account/me")
                    .set("Authorization", `Bearer ${TOKEN}`)
                    .set("content-type", "application/json")
                    .expect(200).end(function (err, res) {
                        expect(res.body.account).is.exist;
                        expect(res.body.account.username).to.equal("anhdv")
                        done();
                    });
            });
        });
```

Figure 5-9: API test case sample

Test behavior of HTTP method which is used to get account information with correct token

- Test case are executed automatically by using Nyc (in command line).

```
$ npm run test
```

```
"scripts": {
  "test": "nyc mocha --exit --timeout 20000 tests/**/*.test.js",
  "coverage": "nyc --reporter=lcov --reporter=text-lcov mocha --exit --timeout 20000 tests/**/*.test.js "
},
```

*Figure 5-10: Configure script in **package.json** file*

```
  setTag
    end user does not exist
      √ it should be return a error message (78ms)
    set tags for end user success
      √ it should be return a success message (78ms)


  448 passing (1m)
```

Figure 5-11: Run test by npm command

By using Nyc combining with Istanbul, test report can be exported in 2 formats: static html and console table.

```
UserInformationService
  getUserInformationWithPSID
    psid exist
      √ it should be return a end user (79ms)
    psid does not exist
      √ it should be return a error message (76ms)
  getTags
    test get all tag
      √ it should be return a list tag (161ms)
  getUsersInformation
    test get all end user
      √ it should be return a list end user (174ms)
  getUsersInformationBrief
    test get all end user short version
      √ it should be return a list end user (874ms)
  countMessage
    test get amount message of end user
      √ it should be return amount message of end user (686ms)
    end user does not exist
      √ it should be return error message (82ms)
    end user exist but no log
      √ it should be return error message (416ms)
  ignoreUser
    end user does not exist
      √ it should be return a error message (94ms)
    ignore end user success
      √ it should be return a success message (85ms)
  setTag
    end user does not exist
      √ it should be return a error message (77ms)
    set tags for end user success
      √ it should be return a success message (96ms)
```

Figure 5-12:Sample Unit test report

```
Dashboard API
  GET /api/dashboard/card
    get statistic for card
      √ it should be return statistics (1008ms)
  GET /api/dashboard/chart
    get statistic for chart
      √ it should be return statistics (190ms)
  GET /api/dashboard/interest
    get all interest
      √ it should be return a success message (170ms)
  GET /api/dashboard/interest/:year/:keyword
    bad request
      √ it should be return a error message (85ms)
    filter success
      √ it should be return a list of result (180ms)
```

Figure 5-13: Sample API test report

```
File                          |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
------------------------------|----------|----------|----------|----------|-------------------|
All files                     |    92.17 |    83.33 |    85.71 |    93.97 |                   |
 config                       |      100 |      100 |      100 |      100 |                   |
  commonConfig.js             |      100 |      100 |      100 |      100 |                   |
  databaseConfig.js           |      100 |      100 |      100 |      100 |                   |
  elasticConfig.js            |      100 |      100 |      100 |      100 |                   |
  witConfig.js                |      100 |      100 |      100 |      100 |                   |
 constants                    |      100 |      100 |      100 |      100 |                   |
  Action.js                   |      100 |      100 |      100 |      100 |                   |
  Constant.js                 |      100 |      100 |      100 |      100 |                   |
  ContextType.js              |      100 |      100 |      100 |      100 |                   |
  Menu.js                     |      100 |      100 |      100 |      100 |                   |
  Message.js                  |      100 |      100 |      100 |      100 |                   |
  Response.js                 |      100 |      100 |      100 |      100 |                   |
  State.js                    |      100 |      100 |      100 |      100 |                   |
 controllers                  |    91.95 |    88.78 |    84.48 |     92.9 |                   |
  accountController.js        |    93.13 |    89.13 |    89.47 |    93.65 |... 59,170,183,202 |
  broadcastController.js      |    95.65 |      100 |       75 |    95.65 |                42 |
  dashboardController.js      |    86.21 |      100 |    66.67 |    86.21 |        9,17,37,55 |
  logMessageController.js     |       90 |      100 |    83.33 |       90 |                 7 |
  notificationController.js   |    84.62 |      100 |    64.71 |    84.21 | 11,35,38,59,68,76 |
  requestController.js        |    94.55 |       75 |      100 |      100 |    13,19,24,31,76 |
  userInfomationController.js |    92.65 |    91.67 |    89.47 |    93.85 |       12,59,60,68 |
  utteranceController.js      |    93.33 |      100 |      100 |    92.31 |             42,43 |
 models                       |      100 |      100 |      100 |      100 |                   |
  account.js                  |      100 |      100 |      100 |      100 |                   |
  answer.js                   |      100 |      100 |      100 |      100 |                   |
  interest.js                 |      100 |      100 |      100 |      100 |                   |
  logMessage.js               |      100 |      100 |      100 |      100 |                   |
  notification.js             |      100 |      100 |      100 |      100 |                   |
  request.js                  |      100 |      100 |      100 |      100 |                   |
  response.js                 |      100 |      100 |      100 |      100 |                   |
  scenario.js                 |      100 |      100 |      100 |      100 |                   |
  stateMachine.js             |      100 |      100 |      100 |      100 |                   |
  userInformation.js          |      100 |      100 |      100 |      100 |                   |
 routes/api                   |    98.25 |      100 |    94.87 |    98.23 |                   |
  account.js                  |      100 |      100 |      100 |      100 |                   |
  broadcast.js                |      100 |      100 |      100 |      100 |                   |
  dashboard.js                |      100 |      100 |      100 |      100 |                   |
  extract.js                  |       80 |      100 |        0 |       80 |              8,48 |
  forgot.js                   |      100 |      100 |      100 |      100 |                   |
  log.js                      |      100 |      100 |      100 |      100 |                   |
  login.js                    |      100 |      100 |      100 |      100 |                   |
  notification.js             |      100 |      100 |      100 |      100 |                   |
  policy.js                   |    85.71 |      100 |        0 |    85.71 |                 7 |
  request.js                  |      100 |      100 |      100 |      100 |                   |
  reset.js                    |    91.67 |      100 |    66.67 |    91.67 |                 8 |
  userInfomation.js           |      100 |      100 |      100 |      100 |                   |
  utterance.js                |      100 |      100 |      100 |      100 |                   |
 services                     |    88.61 |    79.63 |    82.35 |    91.93 |                   |
  accountService.js           |    87.92 |    83.33 |     81.4 |    91.43 |... 22,256,274,284 |
  answerService.js            |      100 |      100 |      100 |      100 |                   |
  broadcastService.js         |    95.45 |      100 |    83.33 |       95 |                32 |
  dashboardService.js         |    89.32 |       80 |       75 |    89.11 |... 63,164,183,193 |
  logMessageService.js        |    86.67 |      100 |       75 |    85.71 |             13,23 |
  mailService.js              |    90.32 |       50 |      100 |      100 |          37,56,75 |
  notificationService.js      |    89.29 |       75 |    79.31 |    91.46 |... ,83,95,128,161 |
```

Figure 5-14: Istanbul console test coverage report

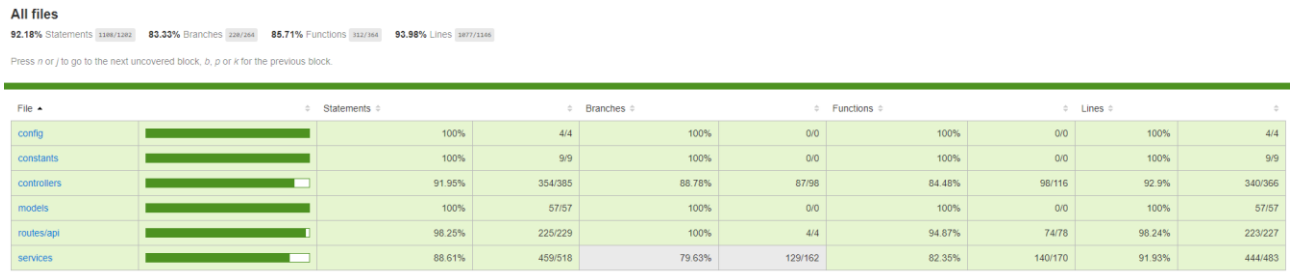


Figure 5-15: Istanbul static html test coverage report

5.3.2 System testing

Detailed Test cases will be described in ***ASC_TestCase.xlsx*** file.

As a standard definition, ASC Project defines that a test case is:

A set of test data and test programs (test scripts) and their expected results. A test case validates one or more system requirements and generates a pass or fail.

A good test case should follow two basic aspects, the Contents and the Style. Test cases for functional testing are derived from the target of test's use cases Test cases should be developed for each use case scenario. The use case scenarios are identified by describing the paths through the use case that traverse the basic flow and alternate flows start to finish through the use case.

By using good automation test, ASC Project System testing will not focus on common logic of system like length of text but focus on behavior of website and aims to validate that all software module dependencies are functionally correct and that data integrity is maintained between separate modules for the entire solution.

5.3.3 Acceptance testing

Acceptance Testing is a level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery. But our project will use the Checklists as a substitute for Acceptance testing.

The content of the Checklist is shown in the table below

ID	Checklists	Yes	No
	General		
CL-001	Text on all pages for spelling and grammatical errors.	√	
CL-002	Functionality of buttons available on all pages.	√	
CL-003	All mandatory fields are validated.	√	
CL-004	Validation error messages are displayed properly below the field.	√	
CL-005	All error messages are displayed in red color.	√	
CL-006	Delete functionality for any record on page are asked for confirmation.	√	
CL-007	All numeric values are formatted properly.	√	
CL-008	Unavailable pages are redirected to not found page.	√	
	GUI and Usability		
CL-009	Screens are designed follow project standards.	√	
CL-010	The screen is well organized.	√	
CL-011	All fields on page (e.g. text box, radio options, dropdown lists) should be aligned properly.	√	
CL-012	Information is arranged symmetrically with adequate spacing between components.	√	
CL-013	The most important fields are located where they are easy to see.	√	
CL-014	Information is presented in the order that the user needs it.	√	
CL-015	Do not use slang, acronyms, and abbreviations.	√	
CL-016	Icons and images are designed impression and copyright.	√	
CL-017	Font size, style and color for headline, description test, labels, infield data, and grid information standard as specified in SRS.	√	
CL-018	The static text is clear, concise, and meaningful.	√	
CL-019	A list view is used to allow a collection of items that are on a single hierarchical level.	√	
CL-020	A tree view is used to allow a collection of items to be displayed and manipulated within varying hierarchical levels.	√	
CL-021	Pop-up menus are provided for the user to access information about an object's properties or perform specific tasks on the object.	√	
CL-022	Command buttons are used to trigger application processes.	√	

CL-023	Toggle buttons are used to show independent on/off choices.	√	
CL-024	System displays notification message when trouble or error occurs.	√	
CL-025	Quick reply is used to show choices.	√	
CL-026	Bubble splitting in a long message.	√	
CL-027	Typing indicator is used to make real.	√	
Database			
CL-028	Correct data is saved in database upon successful page submit.	√	
CL-029	Columns does not accept null values.	√	
CL-030	Data should be stored in single or multiple collections based on design.	√	
CL-031	Input data is not truncated.	√	
CL-032	Input numeric fields with minimum, maximum, and float values.	√	
CL-033	Drop down list are saved correctly in database.	√	
CL-034	Database fields are designed with correct data type and data length.	√	
Performance			
CL-035	Real time connection	√	
CL-036	Page load time is within acceptable range.	√	
CL-037	Page load on slow connections.	√	
CL-038	Response time for any action under light, normal, moderate and heavy load conditions.	√	
CL-039	Database query execution time.	√	
CL-040	Load testing of application.	√	
Security			
CL-041	Test password security and password policy enforcement.	√	

Table 5-3: Check list table

5.3.4 Defect Log

ASC project used to manager tasks and defects.

PM created ASC project in Backlog and add other members' email to allow the team take part in activities: control bugs, fix bugs, re-test bugs and close bug. Bug will be logged by tester or developer in develop progress.

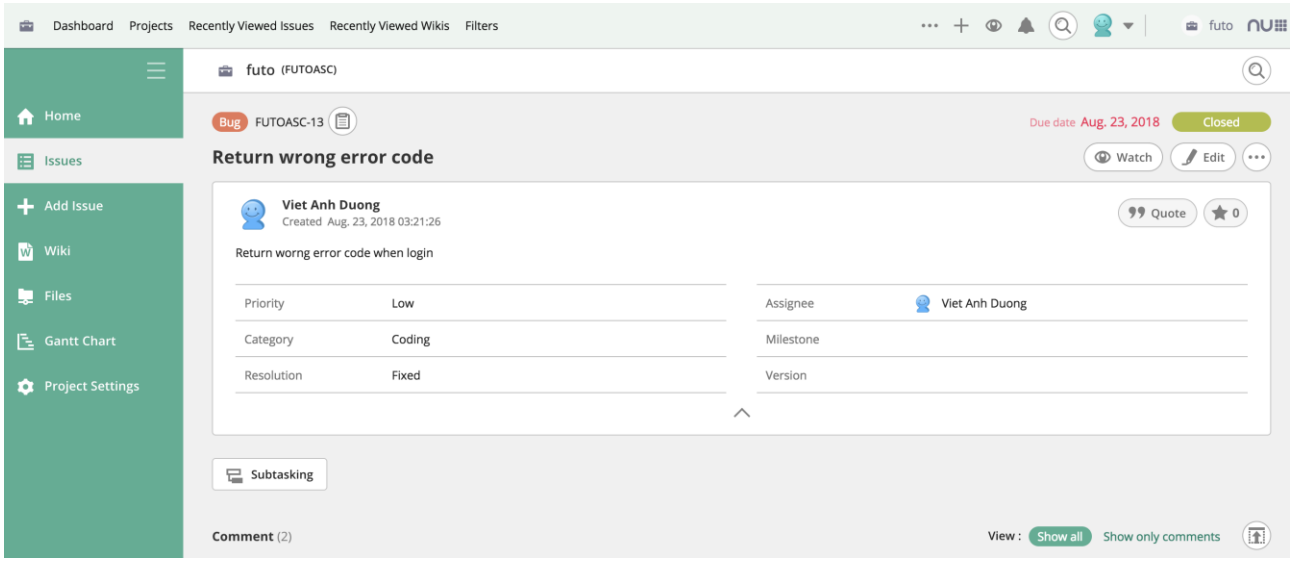


Figure 5-16: Control task and bug with Backlog

5.4 Test Report

5.4.1 Automation test case report

The contents of the Automation Test Case Report are shown in the table below:

Type of test case	**Test Case**	**Pass**	**Fail**	**Not available**	**Number of Test Case**
API Test - Account	Get all account	2	0	0	2
	Get information of current account	3	0	0	3
	Change password	4	0	0	4
	Update profile	3	0	0	3
	Create account	4	0	0	4
	Update account	5	0	0	5
	Delete account	5	0	0	5
	Total	26	0	0	26
API Test - Broadcast	Send message with psids	3	0	0	3
	Send message with tags	3	0	0	3
	Total	6	0	0	6
API Test - Dashboard	Get statistic for card	1	0	0	1
	Get statistic for chart	1	0	0	1
	Get all interest	1	0	0	1
	Filter interest	2	0	0	2
	Total	5	0	0	5
API Test – Forgot password	Forgot password	3	0	0	3
	Total	3	0	0	3
API Test – Log message	Get all log message	1	0	0	1
	Get log message by id	2	0	0	2
	Total	3	0	0	3
API Test - Login	Login	5	0	0	5
	Total	5	0	0	5
API Test – Notification	Get all notification of current account	1	0	0	1
	Seen all notification	1	0	0	1
	Seen a notification	1	0	0	1
	Send a notification	4	0	0	4
	Send a notification with role	2	0	0	2

	Total	9	0	0	9
API Test – Request	Filter request	2	0	0	2
	Get all request answer	1	0	0	1
	Create request answer	3	0	0	3
	Process request answer	4	0	0	4
	Cancel request answer	2	0	0	2
	Total	12	0	0	12
API Test – Reset password	Verify token	3	0	0	3
	Reset password	2	0	0	2
	Total	5	0	0	5
API Test – User information	Get all end user information	1	0	0	1
	Export all end user information	1	0	0	1
	Get information of end user by id	2	0	0	2
	Get all tags	1	0	0	1
	Amount message of end user	2	0	0	2
	Ignore end user	3	0	0	3
	Add tag for end user	3	0	0	3
	Total	13	0	0	13
API Test – Utterance	Get all utterance	1	0	0	1
	Get answer by id	2	0	0	2
	Total	3	0	0	3
API Test - Bot	Chat	9	0	0	9
Unit Test - Account	Get account by username	2	0	0	2
	Create account	3	0	0	3
	Update account	5	0	0	5
	Update profile	4	0	0	4
	Login	4	0	0	4
	Forgot password	3	0	0	3
	Verify token	2	0	0	2
	Reset password	3	0	0	3

	Total	23	0	0	23
Unit Test – Answer	Get answer by id	3	0	0	3
	Total	3	0	0	3
Unit Test – Broadcast	Send message with psids	1	0	0	1
	Send message with tags	2	0	0	2
	Total	3	0	0	3
Unit Test – Dashboard	Get statistic for card	1	0	0	1
	Get statistic for chart	1	0	0	1
	Process entities	2	0	0	2
	Get all interest	1	0	0	1
	Filter interest	2	0	0	2
	Total	7	0	0	7
Unit Test – Log message	Get log message by psid	2	0	0	2
	Get logs	1	0	0	1
	Total	3	0	0	3
Unit Test – Notification	Send a notification	2	0	0	2
	Seen a notification	1	0	0	1
	Seen all notification	1	0	0	1
	Get notifications	1	0	0	1
	Send notification with Role	1	0	0	1
	Send notification for all account	1	0	0	1
	Total	7	0	0	7
Unit Test – Request	Create request	2	0	0	2
	Update request	5	0	0	5
	Total	7	0	0	7
Unit Test – User information	Get end user information by Psid	5	0	0	5
	Get tags	1	0	0	1
	Get end users information	1	0	0	1
	Get end users information brief	1	0	0	1
	Count message	3	0	0	3
	Ignore end user	2	0	0	2

	Set tag	2	0	0	2
	Total	15	0	0	15
Unit Test - Bot	Chat	265	13	0	278
Total of Test Case		428	13	0	441

Table 5-4: Automation test case report

5.4.2 Automation test report

Automation test is an integral part of development process. So that, there are 2 phases as 2 phases of Iterative and Incremental Software Process Model.

Test case	Phase 1		Phase 2		Final
	Pass	Fail	Pass	Fail	
API Test – Account	11	0	15	0	26
API Test – Broadcast	2	0	4	0	6
API Test – Dashboard	0	0	4	0	5
API Test – Forgot Password	1	0	2	0	3
API Test – Log message	1	0	2	0	3
API Test – Login	2	0	3	0	5
API Test – Notification	3	0	6	0	9
API Test – Request	5	0	7	0	12
API Test – Reset password	2	0	3	0	5
API Test – User information	6	0	7	0	13
API Test – Utterance	1	0	2	0	3
API Test - Bot	1	0	8	0	9
Unit Test – Account	10	0	13	0	23
Unit Test – Answer	1	0	2	0	3
Unit Test – Broadcast	1	0	2	0	3
Unit Test – Dashboard	2	0	5	0	7
Unit Test – Log message	1	0	2	0	3
Unit Test – Notification	2	0	5	0	7
Unit Test – Request	1	0	6	0	7
Unit Test – User Information	3	0	12	0	15
Unit Test - Bot	73	13	192	0	278
Total all test case	131	13	297	0	441

Table 5-5: Automation test report

Phase	Coverage			
	Statements	Branches	Functions	Lines
Phase 1	89.01%	80.23%	86.31%	92.07%
Phase 2	92.18%	83.33%	85.71%	93.98%

Table 5-6: Automation test coverage report

5.4.3 System test case report

The contents of the System Test Case Report are shown in the table below:

Module	Test Case	Pass	Fail	Not available	Number of Test Case
Authorization	View Login form	1	1	0	2
	Login by ASC account	3	1	0	4
	Logout	1	0	0	1
	Reset password	4	1	0	5
	Deactivated user logins	1	0	0	1
	Total	**10**	**3**	**0**	**13**
Broadcast	Send message	3	1	0	4
	Send message with tags	3	1	0	4
	Total	**6**	**2**	**0**	**8**
Staff	View statistics	6	3	0	9
	View profile	1	0	0	1
	Update profile	3	0	0	3
	View account list	1	0	0	1
	Change password	3	1	0	4
	Edit answer	3	1	0	4
	View editing answer request	1	0	0	1
	View account detail	1	0	0	1
	Total	**19**	**5**	**0**	**24**
Log message	View Facebook count	1	0	0	1
	View chat logs	3	0	0	3
	View Fb account infor	1	0	0	1
	View providing infor	1	0	0	1
	Add tags	2	1	0	3
	Remove tags	1	1	0	2
	Export end user list	3	1	0	4
	Load more message	1	0	0	1
	Total	**13**	**3**	**0**	**16**
Notification	Receive notification	5	2	0	7
	Open event on notification	1	0	0	1

	Load more notification	1	0	0	1
	View notification in details	6	2	0	8
	Total	**13**	**4**	**0**	**17**
Request	Filter request	1	0	0	1
	View request	1	0	0	1
	Create request	3	1	0	4
	Reject request	1	1	0	2
	Accept request	1	1	0	2
	Total	**7**	**3**	**0**	**10**
Search	Search account by full name	2	0	0	2
	Search account by username	2	0	0	2
	Search response	2	1	0	3
	Filter account by role	1	0	0	1
	Filter account by status	1	0	0	1
	Filter response	1	1	0	3
	Total	**9**	**3**	**0**	**12**
Admin	Create account	3	1	0	4
	Delete account	1	1	0	2
	Staff view	2	0	0	2
	Change account status	1	0	0	1
	Logout	1	0	0	1
	Total	**8**	**2**	**0**	**10**
Bot	**Training**	**702**	**126**	**0**	**828**
Total of Test Case		**800**	**141**	**0**	**941**

Table 5-7: System test case report

Test case list:

ID	Test Case Description	Test Case Procedure	Expected Output	Status
View Login & Forget Password Form				

[Staff-1]	View Login	1. Access https://futo-dashboard.luzotech.com 2. On Screen HomePage, click "Đăng nhập" button	1. Homepage is displayed 2. System displays with the following information: - Title: "Hệ thống quản lý ứng dụng Trợ lý ảo" - Account name field: "TÊN ĐĂNG NHẬP" - Password field: "MẬT KHẨU" - Checkbox: "Nhớ tên đăng nhập lần sau" - Forget Password: "Quên mật khẩu?" - Login button: "Đăng nhập"	Pass
[Staff-2]	View Forget Password	1. On Screen HomePage, click to access "Quên mật khẩu?" 2. On Screen ForgetPassPage, click "Xin cấp lại mật khẩu" button	1. Forget Password Page is displayed. 2. System displays with the following information: - Title: "Quên mật khẩu?" - Description field. - Email field - Button: "Xin cấp lại mật khẩu"	Pass
Login & Forget Pasword by ASC account				
[Staff-3]	Login by invalid account	1. Enter invalid account 2. Enter valid pasword 3. Click "Đăng nhập" button	1. Login Page is displayed. 2. System display an error alert message: "Đăng nhập thất bại".	Pass
[Staff-4]	Login by blank account	1. Do not fill account 2. Enter valid pasword 3. Click "Đăng nhập" button	1. Login Page is displayed. 2. System display an error alert message: "Đăng nhập thất bại".	Pass
[Staff-5]	Login by invalid password	1. Enter valid account 2. Enter invalid pasword 3. Click "Đăng nhập" button	1. Login Page is displayed. 2. System display an error alert message: "Đăng nhập thất bại".	Pass
[Staff-6]	Login by blank password	1. Enter valid account 2. Do not fill pasword 3. Click "Đăng nhập" button	1. Login Page is displayed. 2. System display an error alert message: "Đăng nhập thất bại".	Pass
[Staff-7]	Login by valid email and password	1. Enter valid account 2. Enter valid pasword 3. Click "Đăng nhập" button	1. Login Page is displayed. 2. No error alert message is displayed. 3. Login successfully	Pass

[Staff-8]	Forget password by invalid email	1. Enter invalid email 2. Click "Xin cấp lại mật khẩu" button	1. Forget Password Page is displayed. 2. System display an error alert message: "Please input an '@' in the email address. Invalid email is missing an '@'.	Pass
[Staff-9]	Forget password by valid email	1. Enter valid email 2. Click "Xin cấp lại mật khẩu" button	1. Forget Password Page is displayed. 2. Successfully	Pass
View Dashboard after login				
[Staff-10]	View Statistics	1. Login Successfully 2. On Screen "Bảng tin" is statistics.	1. Statistics Pgae is displayed. 2. System displays with the following information: - 4 blocks information: "Người dùng, Người mới hỏi trong hôm nay, Người dùng cũ hỏi trong hôm nay, Người đã cung cấp thông tin cá nhân". - 2 charts information: "Số lượng người đánh giá Bot (bar chart), Các ngành đang được quan tâm (pie chart)".	Pass
[Staff-11]	View Profile	1. On the right navigation bar, click to avatar. 2. Click to "Trang cá nhân"	1. Personal Page is displayed. 2. System displays with the following information: - Thông tin tài khoản card: "Tên tài khoản, Vai trò". - Thông tin liên hệ card: "Họ và tên, Email, Số điện thoại, Địa chỉ". 3. "Lưu thông tin cá nhân" button	Pass
[Staff-12]	Update Profile by valid info	1. On the right navigation bar, click to avatar. 2. Click to "Trang cá nhân" 3. Edit/ change valid information for "Thông tin liên hệ" card.	1. Personal Page is displayed. 2. System displays with the following information: - Can be change information in "Thông tin liên hệ" card. 43. System display an alert message: "Cập nhật hồ sơ thành công".	Pass

[Staff-12]	Update Profile by invalid info	1. On the right navigation bar, click to avatar. 2. Click to "Trang cá nhân" 3. Edit/ change invalid information for "Thông tin liên hệ" card. 4. Click "Lưu thông tin cá nhân button.	1. Personal Page is displayed. 2. System displays with the following information: - Can be change information in "Thông tin liên hệ" card. 3. No error message for field information 3. System display an alert message: "Cập nhật hồ sơ Không thành công".	Fail
[Staff-13]	Change password	1. On the right navigation bar, click to avatar. 2. Click to "Thay đổi mật khẩu"	1. Change Password Page is displayed. 2. System display with the following information: - Title page - Password field: "Mật khẩu cũ". - Password field: "Mật khẩu mới". - Password field: "Nhập lại mật khẩu". 3. "Đổi mật khẩu" button	Pass
[Staff-14]	Change password by invalid old password, valid new password, valid confirm new password	1. On the right navigation bar, click to avatar. 2. Click to "Thay đổi mật khẩu" 3. Enter invalid old password 4. Enter valid new password 5. Enter valid confirm password 6. Click to "Đổi mật khẩu".	1. Change Password Page is displayed. 2. System display an alert message: "Mật khẩu chưa khớp".	Pass
[Staff-16]	Change password by valid old password, invalid new password, invalid confirm new password	1. On the right navigation bar, click to avatar. 2. Click to "Thay đổi mật khẩu" 3. Enter valid old password 4. Enter invalid new password 5. Enter invalid confirm new password 6. Click to "Đổi mật khẩu".	1. Change Password Page is displayed. 2. System display an error alert message: "Mật khẩu chưa khớp".	Pass
[Staff-17]	Change password by valid old password, valid new password, valid confirm new password	1. On the right navigation bar, click to avatar. 2. Click to "Thay đổi mật khẩu" 3. Enter valid old password 4. Enter valid new password 5. Enter valid confirm new password 6. Click to "Đổi mật khẩu".	1. Change Password Page is displayed. 2. System display an error alert message: "Thay đổi mật khẩu Thất bại".	Fail

[Staff-18]	Change password by blank old pasword, blank new password, blank confirm new password	1. On the right navigation bar, click to avatar. 2. Click to "Thay đổi mật khẩu" 3. Do not fill old password 4. Do not fill new password 5. Do not fill confirm new password 6. Click to "Đổi mật khẩu".	1. Change Password Page is displayed. 2. System display an error message on field: "Please fill out this field".	Pass
[Staff-19]	View Account list	On the side bar, click to "Tất cả tài khoản"	1. Show all account list on the system. 2. System display with the following information: - Table account list: "STT, Tài khoản, Họ và Tên, Vai trò, Trạng thái, Hành động". 3. Dropdown list "Vai trò" is displayed. 4. Dropdown list "Trạng thái" is displayed. 5. "Hoạt động" button 6. "Ngừng hoạt động" button 7. "Xem" button 8. "Xoá" button	Pass
[Staff-20]	Search Account	1. On the side bar, click to "Tất cả tài khoản" 2. On the table, click to seach input into table. 3. Enter username, "họ và tên" to search.	1. Show all account list on the system. 2. Table display result for search.	Pass
[View Login-21]	View Account Information Detail	1. On the side bar, click to "Tất cả tài khoản" 2. On the table, click to "Xem" button, redirect to "Thông tin tài khoản" Page.	1. "Thông tin tài khoản" Page is displayed. 2. System display with the following information: - "Tên tài khoản" field. - "Họ và tên" field. - Email field. - "Số điện thoại" field. - "Vai trò" field. - "Trạng thái" field.	Pass
[Staff-22]	Filter Account	1. On the side bar, click to "Tất cả tài khoản" 2. On the table, click to Dropdown list into table. 3. Dropdown list can be filter account: - "Vai trò" dropdown list - "Trạng thái" dropdown list.	1. Show account when filter account by click to "Vai trò" dropdown list. 2. Show account when filter account by click to "Trạng thái" dropdown list.	Pass

Table 5-8: Test case

For more details about all test suites, please look at file "**ASC_TestCase.xlsx"**

5.4.4 System test report

We execute test in 2 phases of process model, to finish project. With 2nd phase, we test with 2 small tasks:

The first one: re-test phase 1 and implemented function of phase 2
The last one: test all test case and re-test to confirm bug

The contents of the Test Report are shown in the table below

Module	Phase 1		Phase 2-1		Phase 2-2		Final
	Pass	Fail	Pass	Fail	Pass	Fail	
Authorization	2	1	2	2	8	0	13
Broadcast	1	0	2	1	4	0	8
Staff	4	2	4	2	12	0	24
Log message	3	1	5	1	6	0	16
Notification	4	1	2	1	9	0	17
Request	1	1	3	2	3	0	10
Search	3	0	2	1	9	0	15
Admin	2	0	2	1	5	0	10
Bot	201	72	250	54	251	0	828
Total all Test Case	**221**	**78**	**272**	**63**	**307**	**0**	**941**

Table 5-9: System test report

5.4.5 Benchmarks

5.4.5.1.1 Requirements

We did some benchmarks to see if we have any issues with performance of the system. Because of limited resource, out target is that the system should be able to serve **0.5 user can do the search request per second with the response time less than 500ms.**

5.4.5.1.2 Benchmark Tool

Figure 5-17: Artillery

We use **Artillery** to do our benchmark. **Artillery** is a modern load-testing framework. It offers rich functionality, has a strong focus on developer happiness, and is open-source. It supports: **HTTPS, WebSockets, Scriptable, HTML reports.**

5.4.5.1.3 Benchmark implementation

With **Artillery** we can write a benchmark script that simulates realistic user behavior with scenarios.

We will configure the **Artillery** as following:

- Run on address: https://futo.luzotech.com/
- We include a valid token to every request.
- There will be 150 virtual users created and lasts for 300 seconds, so the arrival rate is 0.5 user/second

```
config:
  target: 'https://futo.luzotech.com'
  phases:
    - duration: 300
      arrivalRate: 150
  defaults:
    headers:
      Authorization: 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1MzU2MzM0MjksImFjY291bnQiOnsiYWN0aXZlIjp0cnVlL
scenarios:
  - name: search request
    flow:
    - get:
        url: "/api/request/answer/filter?state=pending"
```

Figure 5-18: Artillery configuration for REST API

Then we run the benchmark with commands:

```
nguyendu195@FUTO-BackEnd:~/backend-futo$ artillery run benchmark-REST.yml
```

5.4.5.1.4 Benchmark result

```
All virtual users finished
Summary report @ 14:26:47(+0000) 2018-08-26
  Scenarios launched:  45004
  Scenarios completed: 45004
  Requests completed:  45004
  RPS sent: 18.89
  Request latency:
    min: 30.2
    max: 2773
    median: 280.1
    p95: 455.6
    p99: 973.7
  Scenario counts:
    search request: 45004 (100%)
  Codes:
    200: 45004
```

Figure 5-19: REST API benchmark result

The benchmark shows that our system works as expected as we defined in the first place:

At request latency, the maximum response time is 2773 ms for REST API

The median response time is 280.1ms for REST API.

The p99 latency is 973.7 ms for REST API (p99 means that 99% of the requests should be faster than given latency)

Chapter 6 : User manual

6.1 Deployment guidelines

6.1.1 Environment for development

- Apple Macbook Pro with OS High Sierra 10.13.5
- Asus Laptop with Microsoft Windows 10
- Integrated Development Environment: Visual Studio Code 1.25.0
- MongoDB Management Tool: Robomongo 1.2.1
- Git extension for Visual Code

6.1.1.1 Setup development environment

6.1.1.1.1 Install Visual Studio Code

Go tohttps://code.visualstudio.com/, click on “Download” button to start download the installation file and follow the instruction to install.

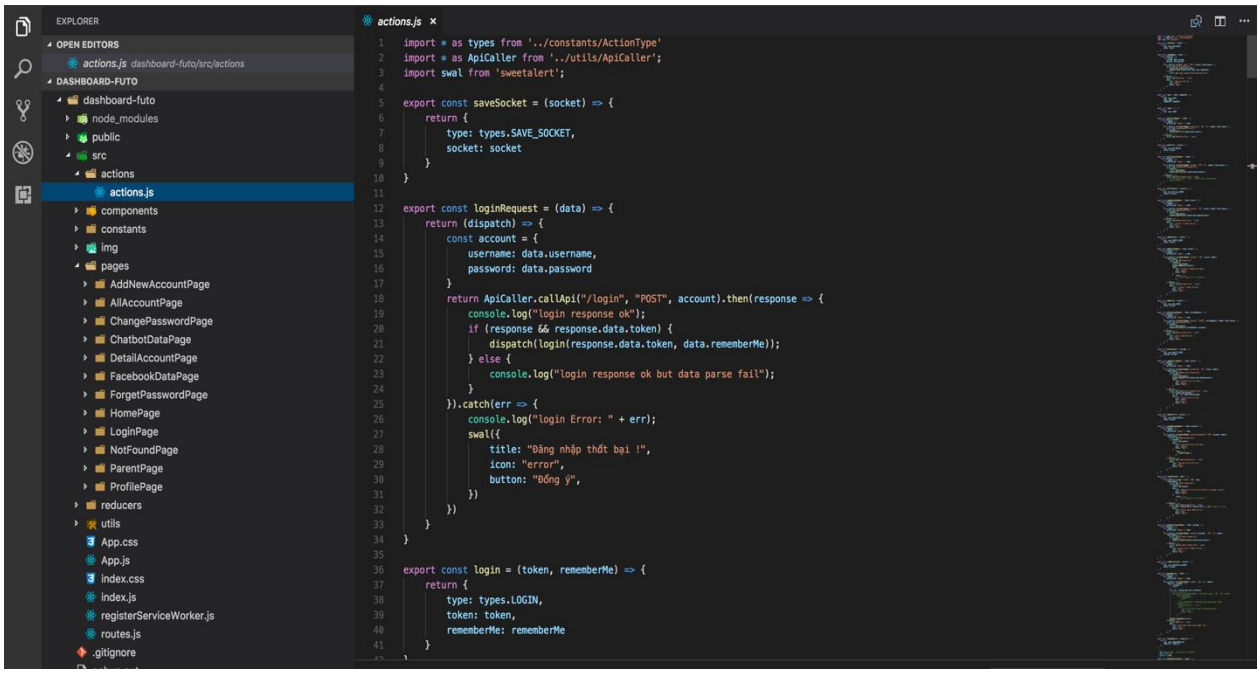


Figure 6-1: Visual Code

6.1.1.1.2 Install Robomongo

- Go to https://robomongo.org/download click on “Download” button to start download the installation file and follow the instruction to install.

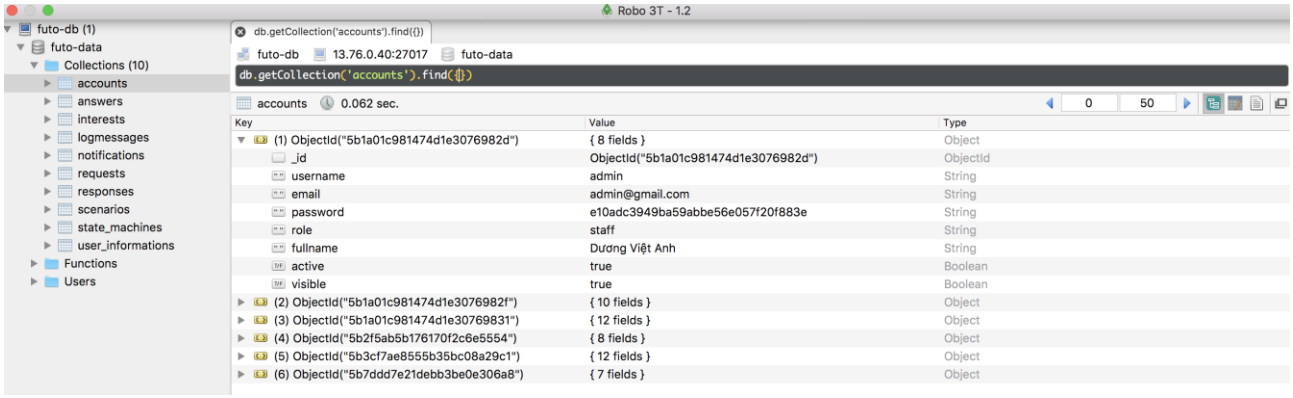


Figure 62: -Robomongo

6.1.1.1.3 Install Git extension

- Integrated on Visual Studio Code

Figure 6-3: Git on Visual Studio Code

6.1.1.2 Run tests in development environment

To run all test suites, run the follow command in the source code directory:

```
xHadesvn-3:dashboard-futo xHadesvn$ npm test
```

6.1.2 Environment for deployment

- Ubuntu 16.04 (provided by Azure Virtual Machine) **(ASC System Architecture)**
  - ASC System Architecture
  - MongoDB
- Elasticsearch **(for full-text search feature)**
- Memcached **(a distributed memory)**
- Nginx **(a web server)**

6.1.2.1 Setup deployment environment for ASC Backend

6.1.2.1.1 Create new Ubuntu 16.04 on Virtual Machine (VM) on Microsoft Azure

- Login an account on Microsft Azure Portal at http://portal.azure.com/
- From the Microsoft Azure Portal, choose **Virtual Machines**
- Select Add on menu bar, search **Ubuntu Server**
- Next, select **Ubuntu Server 16.04 LTS** on right bar, click to **Create**
- Setup name, VM disk type, Username, Password, Location of server.
- Choose a size of server, in this project we will create 3 B2S (2 vCPUs, 4 GiB memory) instances: each for 2-3 **ASC System Architecture.**
- Finally **Review** and **Create**
- We will have 3 Virtual Machine like this:

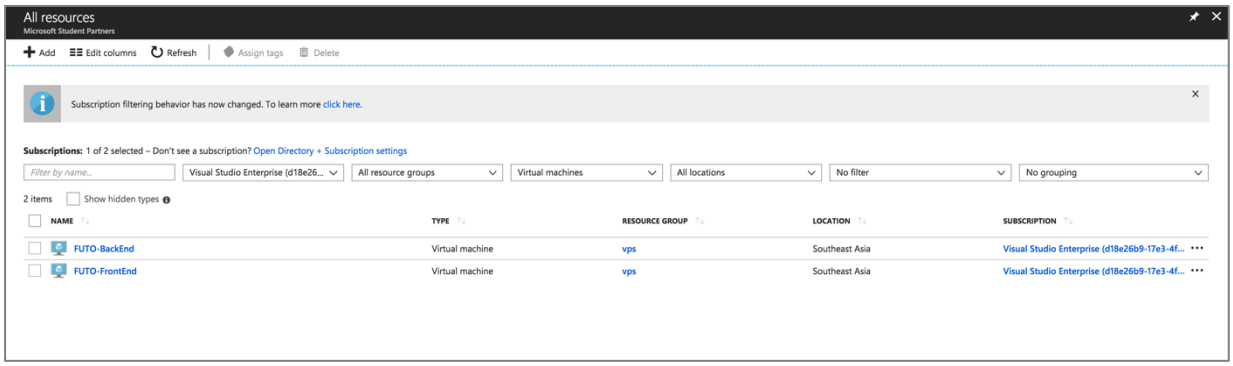


Figure 6-4: Microsoft Azure Virtual Machine Dashboard

6.1.2.1.2 Configure MongoDB

- Go to https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/, follow the instruction to install and configure MongoDB for Ubuntu 16.04

6.1.2.1.3 Configure Elasticsearch

- Go to https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-elasticsearch-on-ubuntu-16-04, follow the instruction to install and configure Elasticsearch for Ubuntu 16.04.

6.1.2.1.4 Configure Memcahed

- Go to https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-memcached-on-ubuntu-16-04, follow the instruction to install and configure Memcached for Ubuntu 16.04

6.1.2.1.5 Configure Nginx

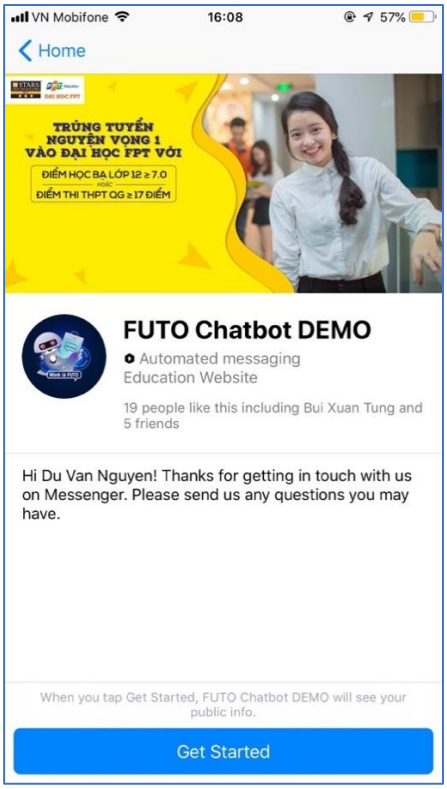
- Go to https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04, follow the instruction to install and configure Nginx for Ubuntu 16.04

6.2 User guidelines

6.2.1 End User

6.2.1.1 Chat

1. Open Facebook Messenger and click “Get started” for English version or “Bắt đầu” for Vietnamese version.



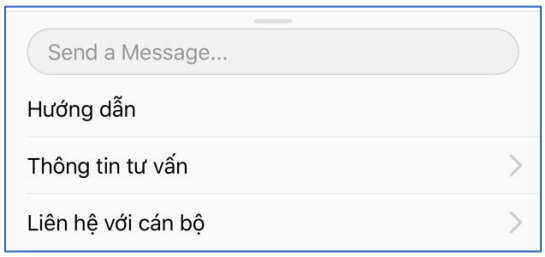
2. Use menu or enter any text directly to send message.

6.2.1.2 Leave Feedback

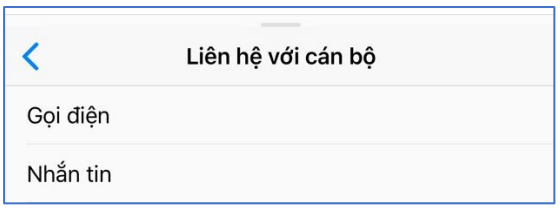
On screen, after 5 minutes if the user does not have any interaction, then Futo Chatbot will give the rating to the user selecting.

6.2.1.3 Leave Information

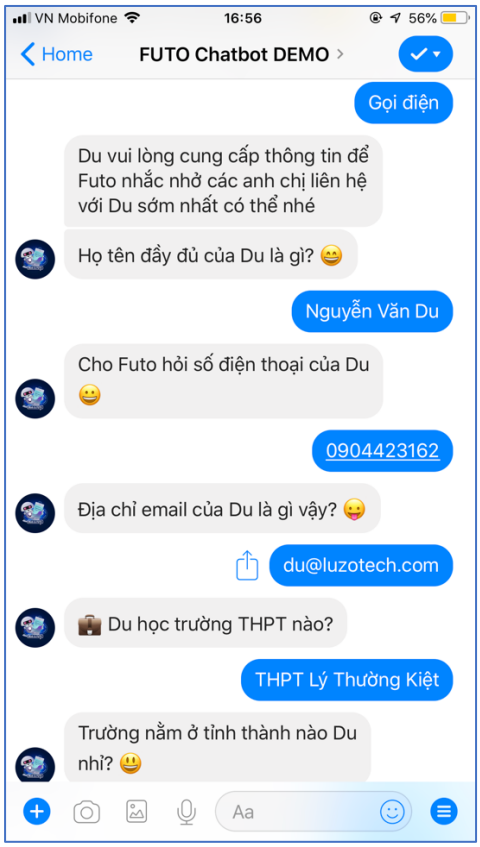
1. On menu, the end user selects “Liên hệ với cán bộ”



2. Next, the end user selects “Gọi điện”



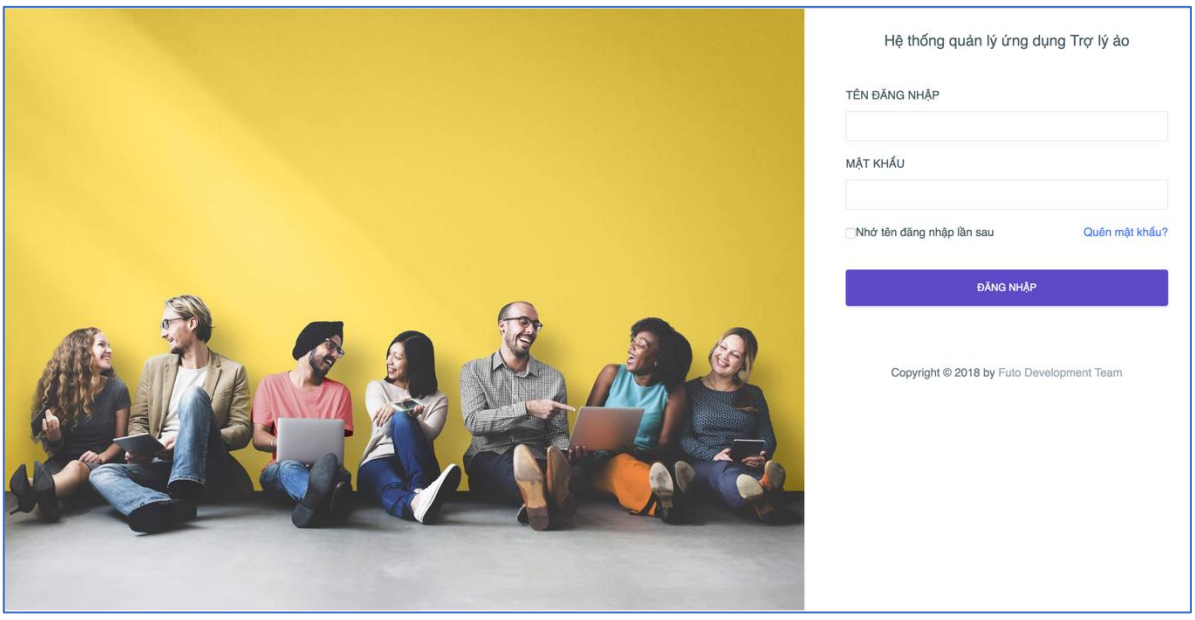
3. FUTO will ask you to provide information.



6.2.2 Staff & Admin

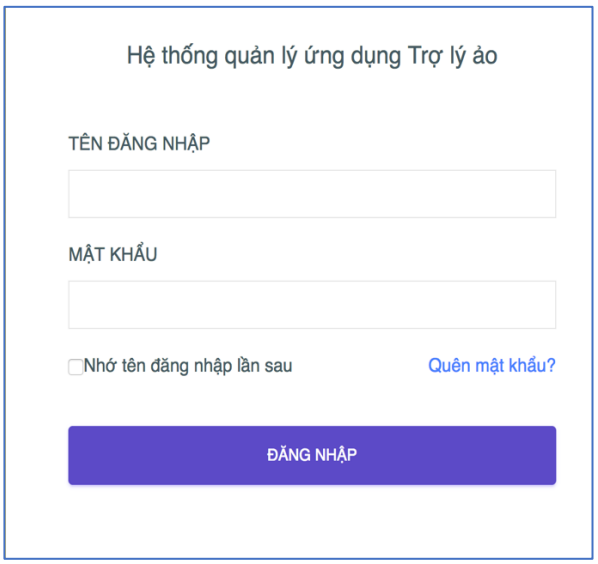
6.2.2.1 Login

Access to https://futo-dashboard.luzotech.com, enter “TÊN ĐĂNG NHẬP”, “MẬT KHẨU” and clicks “ĐĂNG NHẬP”.

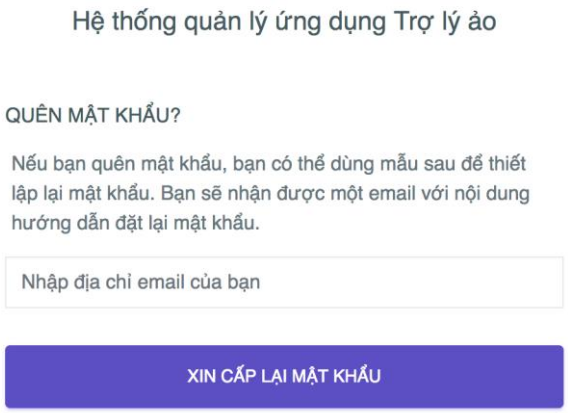


6.2.2.2 Reset Password

1. On screen, user clicks “Quên mật khẩu”

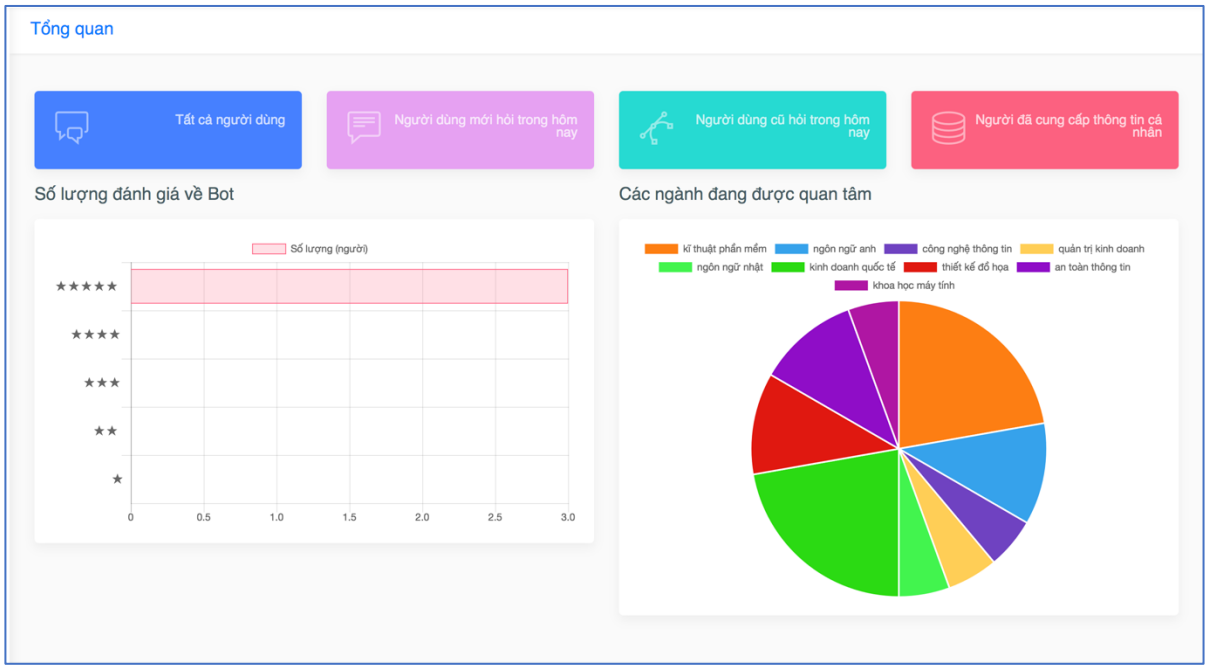


2. On next screen, enter email and clicks “XIN CẤP LẠI MẬT KHẨU”



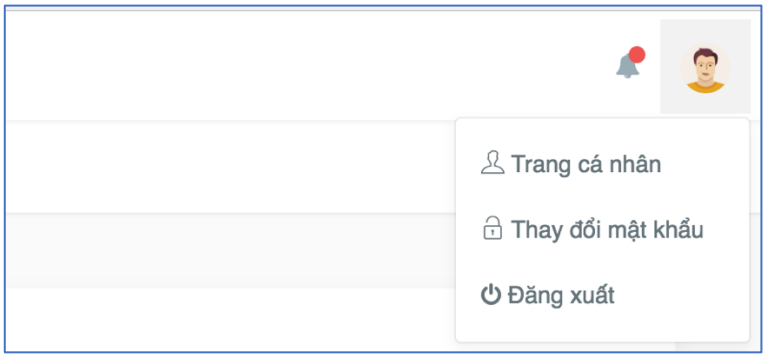
6.2.2.3 View Statistics

After login, Statistics will display on the screen.

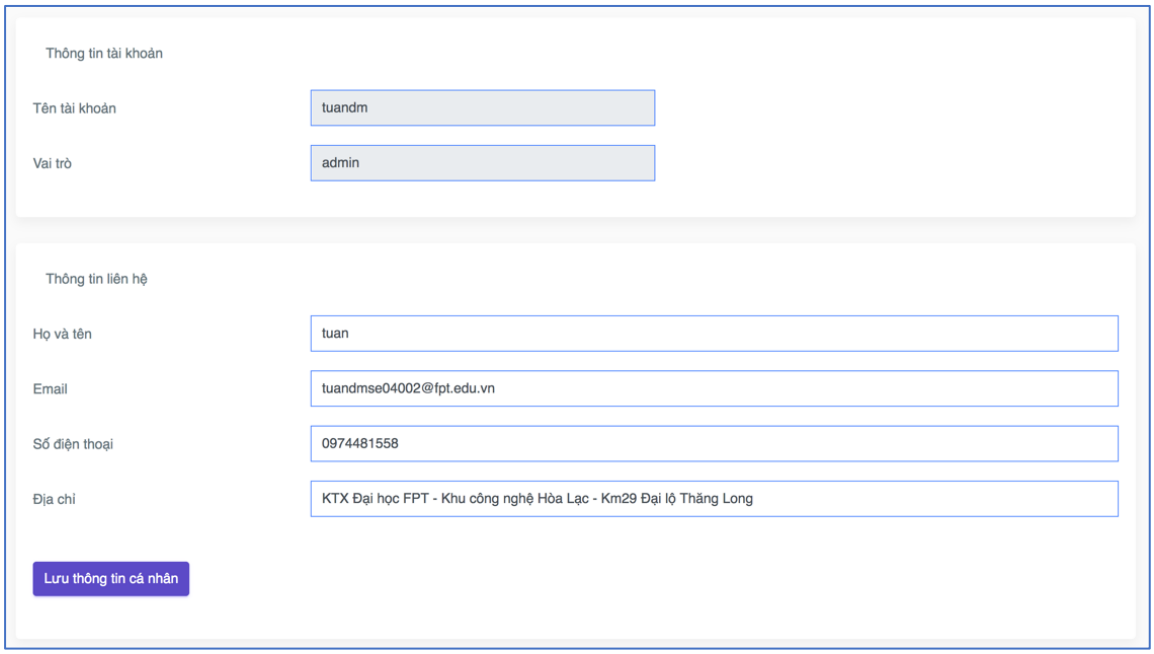


6.2.2.4 View Profile & Update Profile

1. On Dashboard, clicks avatar on the right, select “Trang cá nhân”.

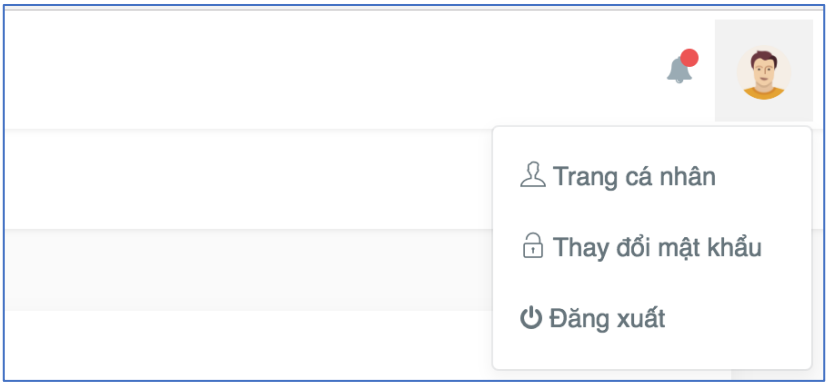


2. View Profile and Update Profile on the screen.



6.2.2.5 Change Password

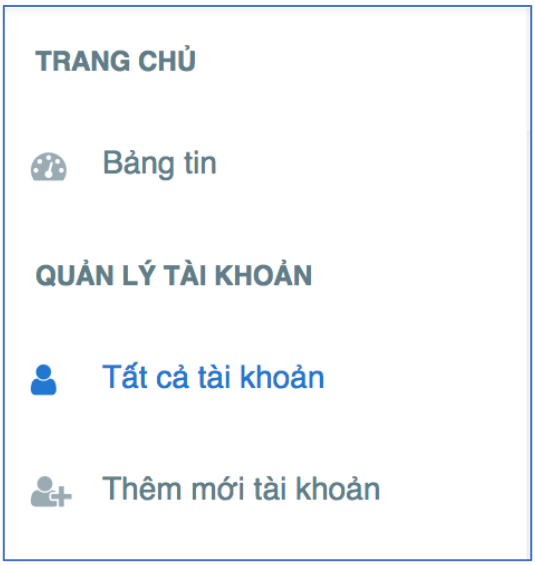
1. On Dashboard, clicks avatar on the right, select “Thay đổi mật khẩu”.



2. Change Password on the screen

6.2.2.6 View Account List

1. On the Sidebar of Dashboard, select “Tất cả tài khoản”.

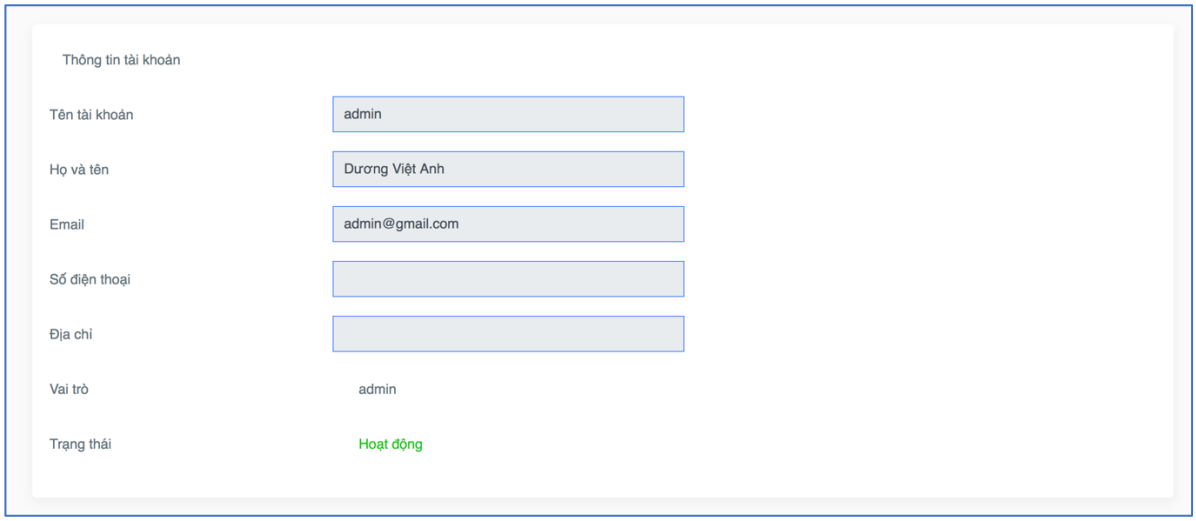


2. All Account List on the screen.

6.2.2.7 View Account Information Detail

1. On the table of All Account List, clicks “Xem” on the any user.

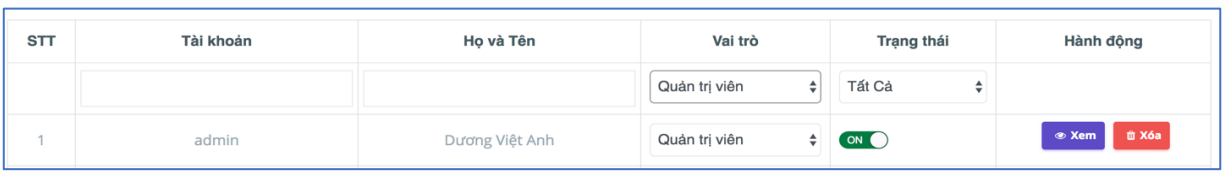
2. View Account Information Detail on the screen

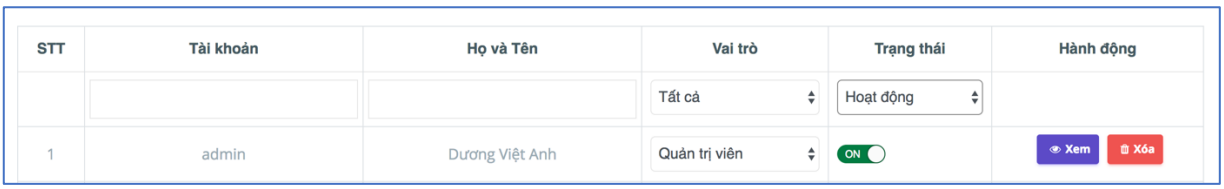


6.2.2.8 Search Account & Filter Account

1. On the table of All Account List, clicks on text input below “Tài khoản” or “Họ và tên” to Search Account.

2. On the table of All Account List, clicks on dropdown list below “Vai trò” or “Trạng thái” to Filter Account.

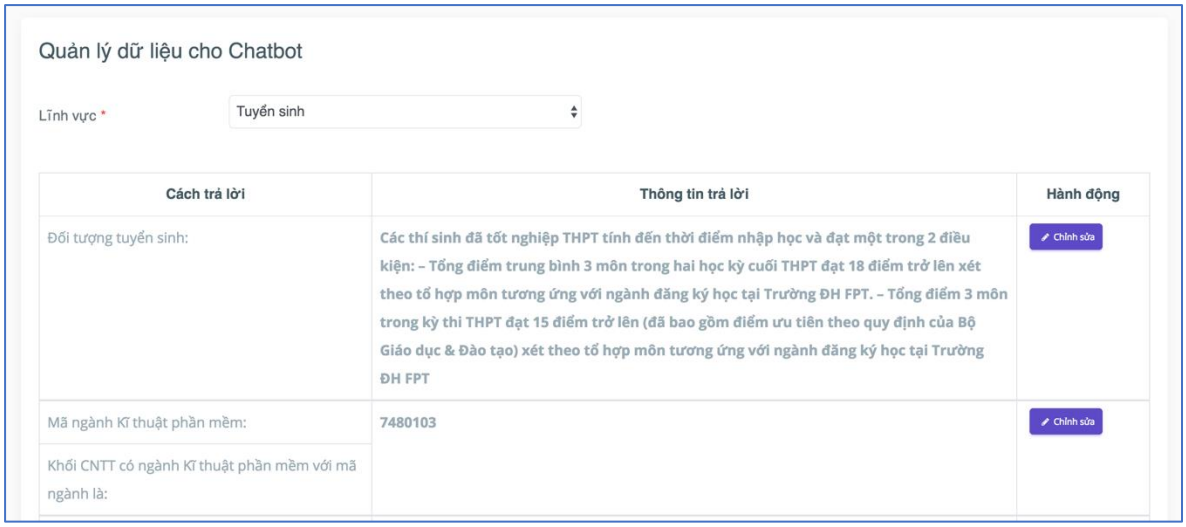




6.2.2.9 View Response & Answer List

1. On the Sidebar of Dashboard, select “Quản lý dữ liệu của Chatbot”.

2. View Response & Answer List on the screen.



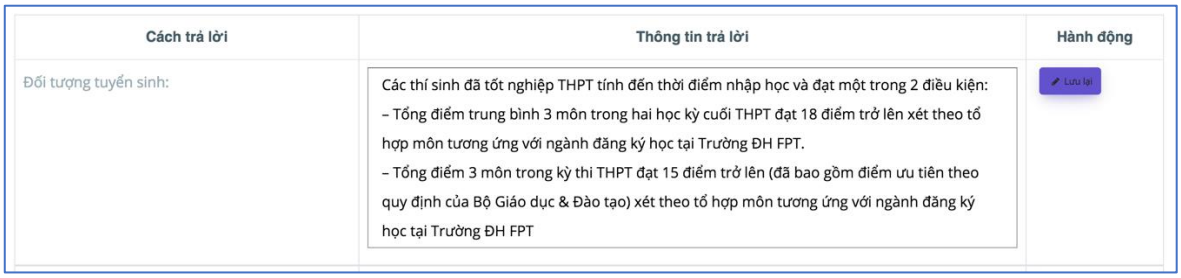
6.2.2.10 Filter Response & Answer List

Clicks on dropdown list from “Lĩnh vực” to Filter Search Response & Answer List.

6.2.2.11 Search Response

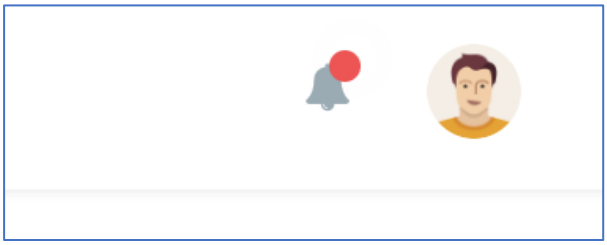
6.2.2.12 Request to Edit Answer

On the table of “Quản lý dữ liệu cho Chatbot” page, clicks “Chỉnh sửa”.



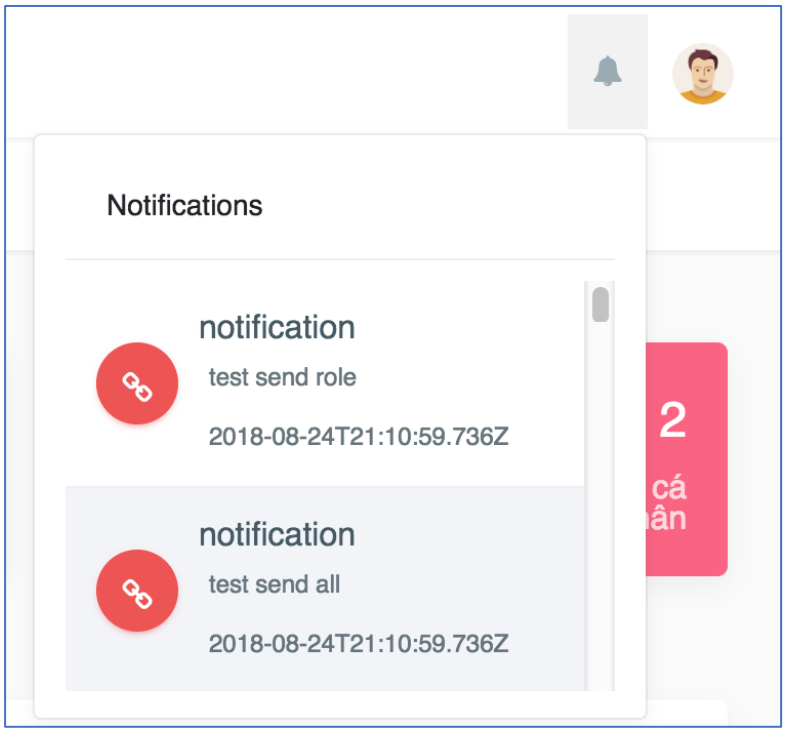
6.2.2.13 Receive Notification

On the Dashboard, notice the bell icon on the right.



6.2.2.14 View Notification List & Notification Detail

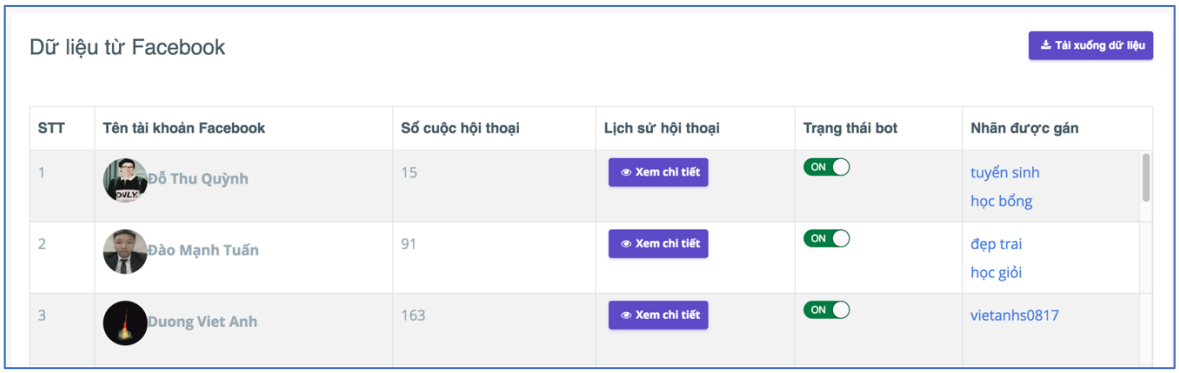
On the Dashboard, clicks the bell icon on the right.



6.2.2.15 View Facebook Account Page

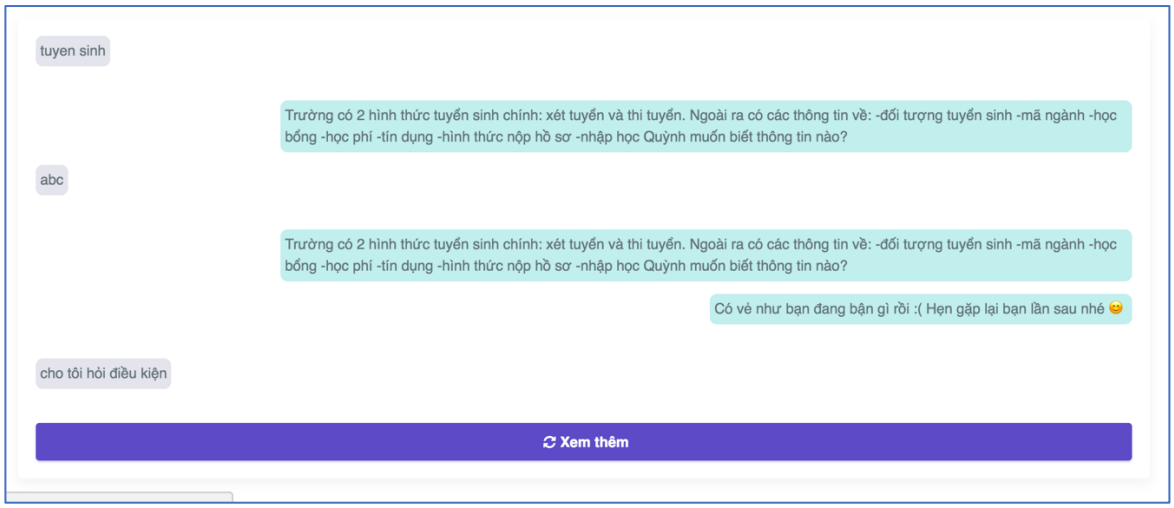
1. On the Sidebar of Dashboard, select “Quản lý dữ liệu từ Facebook”.

2. View Facebook Account on the screen.



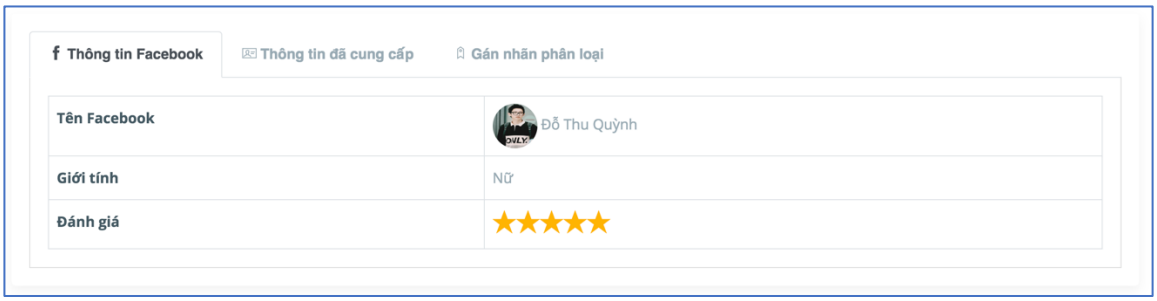
6.2.2.16 View Chat Logs

On the table of View Facebook Account page, clicks "Xem chi tiết" on the any user facebook to see view chat logs.



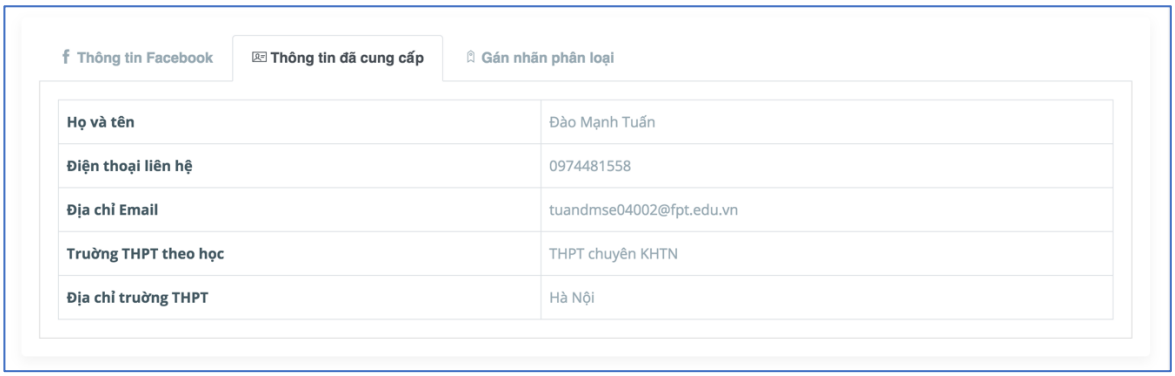
6.2.2.17 View Facebook Account Information

On the table of View Facebook Account Page, clicks "Xem chi tiết" on the any user facebook to see view facebook account information of that user.



6.2.2.18 View Providing Information

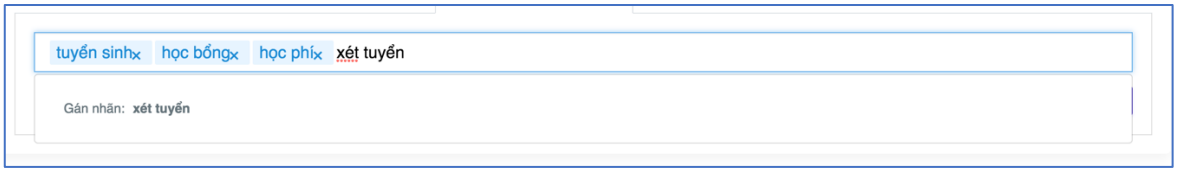
On the table of View Facebook Account page, clicks "Xem chi tiết" on the any user facebook to see view providing information from that user facebook.

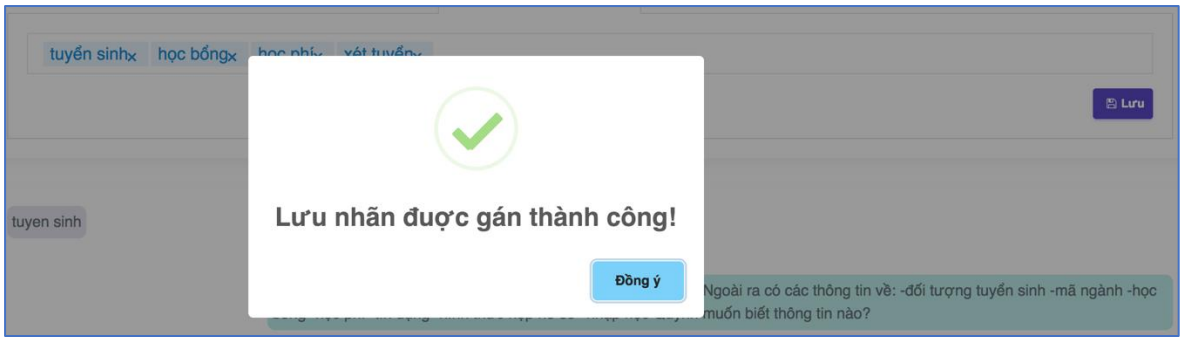


6.2.2.19 Add tag & Remove tag

On the table of View Facebook Account page, clicks “Xem chi tiết” on the any user facebook. Select tab “Gán nhãn phân loại”.

- Clicks text input, enter name tag, press enter and clicks “Lưu” to add tag.
- Clicks “x” beside name tag, and clicks “Lưu” to remove tag.



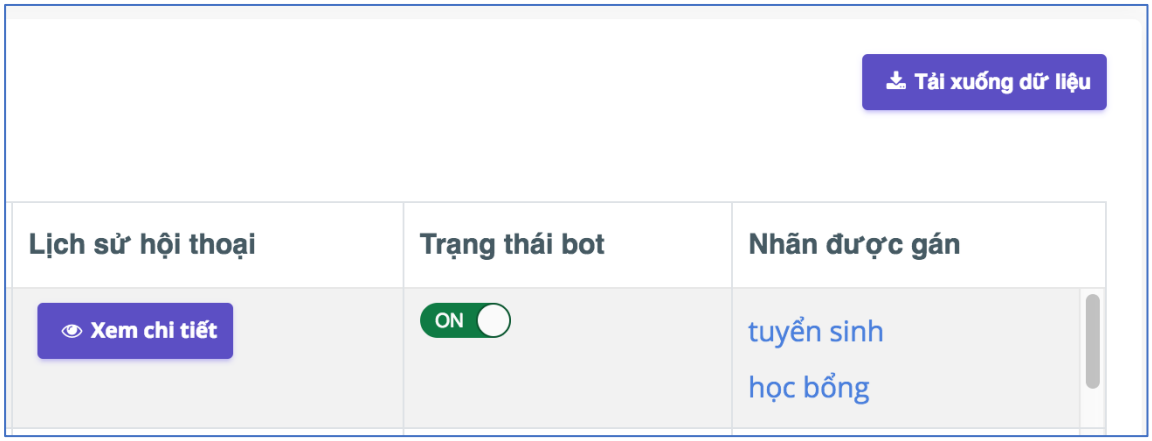


6.2.2.20 Export End User List

On the View Facebook Account page, clicks “Tải xuống dữ liệu” on the right, to export end user list.

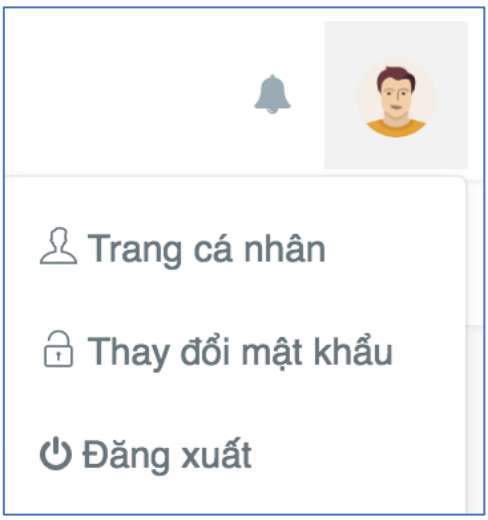
6.2.2.21 Change Bot Status

On the table of View Facebook Account page. At “Trạng thái bot” column, clicks toggle button to On/ Off bot status on the any user. Or clicks “Xem chi tiết” on the any user, clicks toggle button at the right page.



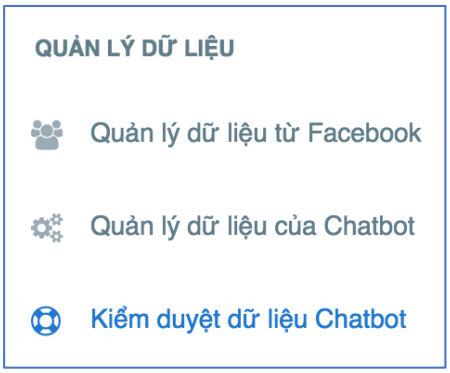
6.2.2.22 Logout

On the Dashboard, clicks avatar on the right. Select “Đăng xuất”.



6.2.2.23 View Answer Editing Requests (Only Admin)

1. On the Sidebar of Dashboard, select “Kiểm duyệt dữ liệu Chatbot”.

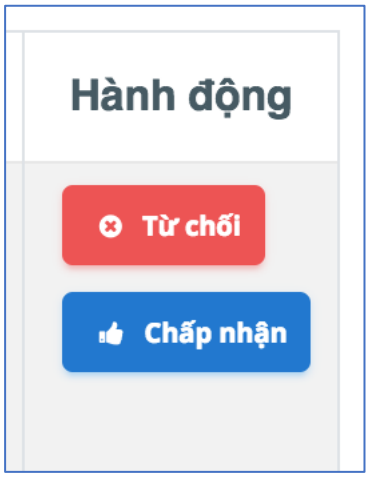


2. View Answer Editing Requests on the screen

6.2.2.24 Accept & Reject Answer Editing Requests (Only Admin)

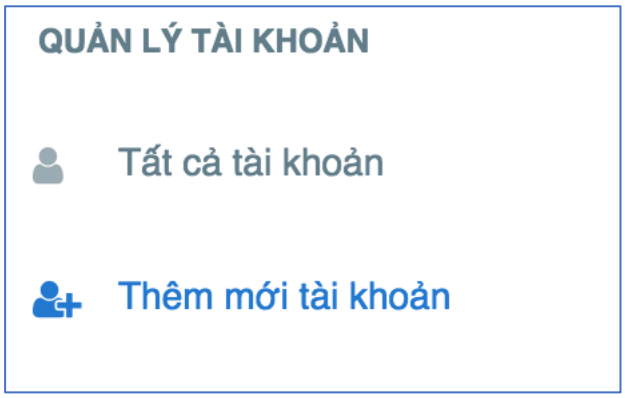
On the table of View Answer Editing Requests page, at "Hành động" column:

- Clicks "Chấp nhận" to accept answer editing requests.
- Clicks "Từ chối" to reject answer editing requests.

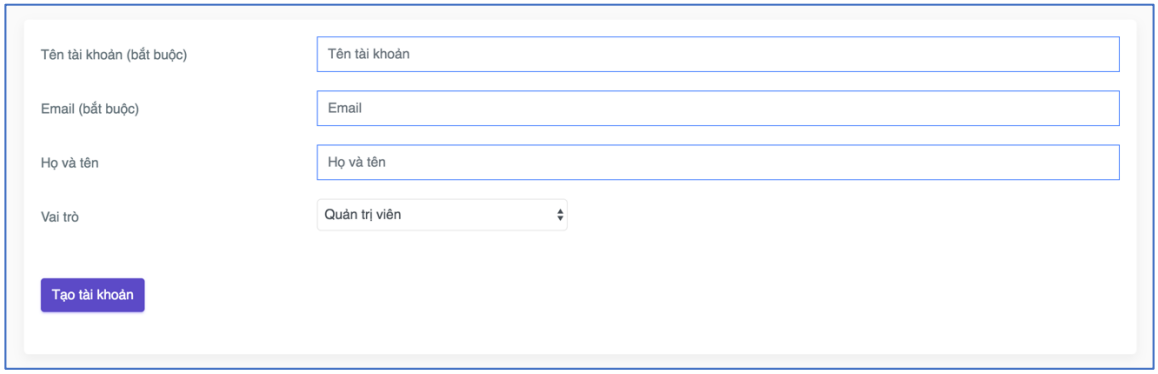


6.2.2.25 Create Account (Only Admin)

1. On the Sidebar of Dashboard, select "Thêm mới tài khoản".

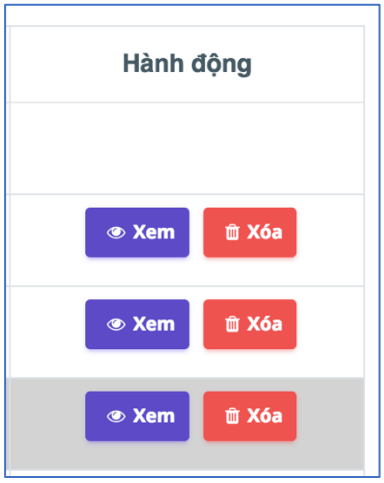


2. Create Account on the screen



6.2.2.26 Delete Account (Only Admin)

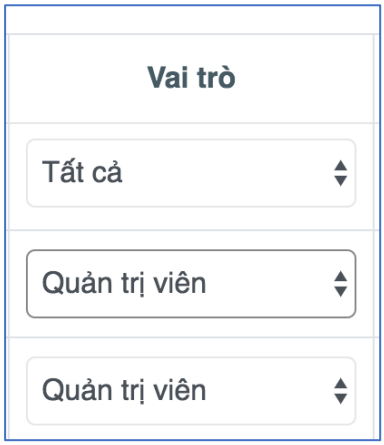
On the table of All Account List, clicks “Xoá” on the any user at “Hành động” column, to delete account.



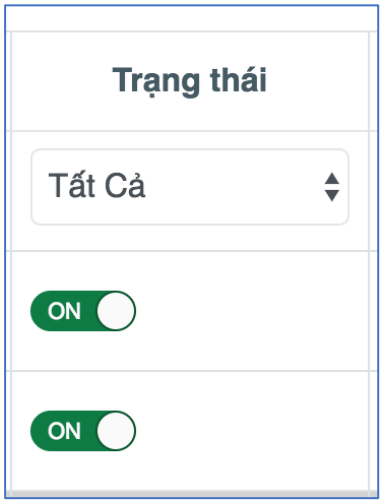
6.2.2.27 Change Account Role & Account Status (Only Admin)

On the table of All Account List:

- Clicks on dropdown list at “Vai trò” column on the any user to change account role.



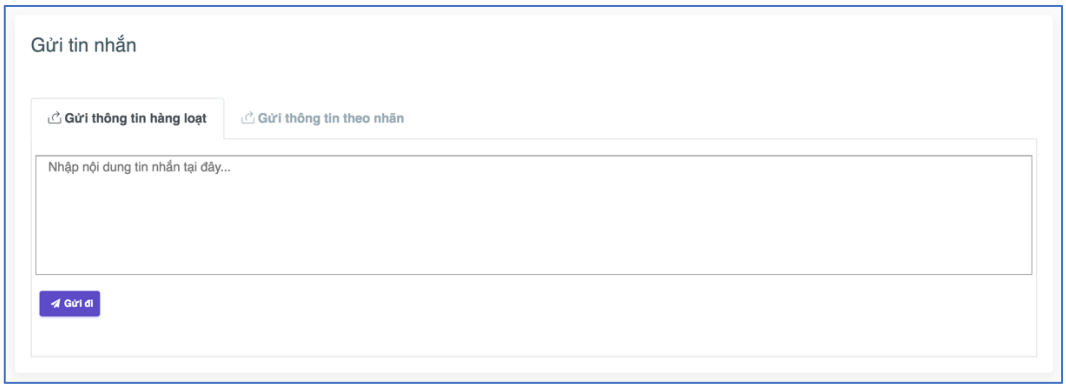
- Clicks on dropdown list at “Trạng thái” column on the any user to change account status.



6.2.2.28 Broadcast (Only Admin)

On the View Facebook Account page:

- Send a general messages to all



- Send messages follow tag