COMP3900

Hal_3900 Report

Ellen Oates	z5098896	Developer
Hayden Le	z5098972	Developer
Yi Wang	z5124282	Developer
Zain Afzal	z5059449	Scrum Master

01/05/2019

e.l.oates@unsw.edu.au
hayden.le@unsw.edu.au
z.afzal@unsw.edu.au
yi.wang7@student.unsw.edu.au

Contents

1 Introduction

1.1 The Problem

Online learning is changing the way students access and engage with higher education. Courses with online delivery increase the flexibility and accessibility of education by providing students with a platform to learn course content in their own time, at their own pace. Increasingly courses which are taught face to face include some online content delivery, including course materials, quizzes, online lecture recordings, and forums to ask questions and discuss the course content outside of class. Because online learning is so prevalent in higher education, it is crucial for universities to ensure students are satisfied with their learning experience.

There is a delay between when students ask a question and when they get a response, and this can vary from hours to days. Answering individual student questions via email or on forums requires a significant amount of time for tutors, course administrators and lecturers. Often the same questions will be asked many times by different students, making it inefficient to have course staff respond to each one individually.

Some key factors that contribute to student satisfaction in the online learning space are:

- Students' preferences for actively participating in learning, rather than through passive learning styles.
- Students' expectations on instructors to facilitate their learning by organising the course resources[1]
- The amount of interaction students have with each other, and the availability of their instructors.[2]
- The availability of strong administrative support when using online learning tools or when confused about assessments and learning expectations.
- Course staff who are concerned with the quality of their course delivery, and want to know what their students need the most help with
- The availability of individual support and extended materials

Making these factors available to students becomes more challenging as classes grow in size. Course staff are thus in need of a more effective way to support with their students.

1.2 Existing Solutions & Problems

Currently at UNSW, learning support is provided to students through email, forums and help sessions. This is very man-hour intensive, requiring many tutors to be on

[1] `https://www.researchgate.net/publication/282699144\_Student\_Satisfaction\_with\_Online\_Learning\_Is\_it\_a\_Psychological\_Contract`

[2] 'Key Factors for Determining Student Satisfaction in Online Courses': `https://www.learntechlib.org/primary/p/2226/article\_2226.pdf`

hand to answer questions which, of themselves, are quite repetitive. In addition, as these courses become larger with increasing enrollment sizes, it becomes more difficult to be able to give students individual attention.

Another side effect of growing cohort sizes is the fact that many tutors and lecturers are forced to spend most of their time answering admin related questions, which is time that could be spent improving the course. The current solution has been to hire more staff and offload the majority of questions to forums, however these are full of repeated questions and require many tutors to moderate them.

This growth is becoming unsustainable, and with the rise of online education platforms, many students are eager to interact with course material in a more meaningful way. Waiting for a tutor to respond often creates a disconnect between the initial question and the answer, which limits the effectiveness of the response. This is provided that the tutor finds time to respond at all.

Chat bots have been deployed in some areas of secondary education, which interact with students in meaningful ways outside of class hours[3], and some have even been created to answer university-level questions[4]. However these tools can not be easily adopted by all university courses, or their administration and assessments.

1.3 Our Solution

1.3.1 Our Contributions

Our bot provides a platform for students to ask questions in real time. This addresses the issue of course staff being too busy to respond to questions in a timely fashion. This also grants lecturers and tutors more time to dedicate towards delivering and improving the course itself.

The bot goes beyond this by providing quiz functionality as well. Students can request to receive quizzes, and see questions and answers provided by the bot. This gives students meaningful interactions and a helpful study tool that doesn't draw from the time of course staff.

1.3.2 Aim, Purpose & Scope

Our goal was to create a course companion chat bot for students to enhance their learning experience. The chat bot should provide students with support by responding to their questions about course administration and the content they are learning in real time. In addition, the bot will monitor students' understanding of the course content with follow-up questions. This is expected to increase both the amount and the quality of student interaction within the course. The bot will also provide students with more frequent and timely interactions. This helps by diverting more complex questions to

[3] `https://botsify.com/education-chatbot`

[4] `https://www.canberra.edu.au/about-uc/media/newsroom/2018/february/students-make-new-friend-in-lucy-the-chatbot`

tutors and lecturers, who in turn will have more time to respond to such questions in depth.

The chat bot will provide administrative support by keeping students informed of their grades and upcoming due dates. This is expected to increase positively affect learning outcomes by boosting students' motivation to study. In addition, it will also enhance course delivery by keeping staff informed of their students' learning needs and frequently asked questions. It will reduce the load on course staff by answering many of the questions that students have and allowing them to focus on the overall delivery of the course.

1.3.3 Differences to Existing Systems

The main difference between our bot and the existing solution of allocating the work to tutors is that the bot does not require an active human at all times. This addresses a number of problems addressed above, primarily saving time for course staff. The bot can be active throughout the day and respond to questions immediately, which is useful for both students and tutors, who previously had to email or post on forums at their convenience.

Our solution to this problem is innovative for a number of reasons. Most notably, we decided to create a bot that could complete what was previously considered human work. While machines replacing humans is no new idea, this is typically only the case in fields of physical labour. We have taken this a step further to create a bot that could supplant not the human muscles, but the human mind.

In addition, our bot also sets itself apart from other chat bots. Most common chatbots use an existing chat frontend such as Facebook Messenger or Slack. However, we have chosen to develop our own web interface from scratch. This gives us significantly more fine-grained control over the features of our bot, allowing us to tune the bot for our specific use case. This is expanded on in section 2.2.4.

2 Background

2.1 Usage Scenarios

2.1.1 Student

User Story: I have just started the course and I want to find more information about my lectures and assignments.

- I log in to the student portal with my name or student number
- I ask the bot questions:
  - Who is the lecturer? *"John Shepherd lecturer"*
  - When is my assignment due? *"Assignment 1 submission week 6 worth 9, assignment 2 submission week 10 worth 11"*
  - How do I do the labs? *"Each week there will be one or more exercises to work on. These exercises will be released in the week preceding the lab class. Labs will be done in pairs and you and your lab partner should discuss the exercises before going to the lab to maximise the usefulness of the class..."*

User Story: I have some questions for COMP1521 that I want the answer to right away. It's a few days until my next tutorial.

- I log in again
- I ask the bot questions:
  - What is qtspim? *"Qtspim... provides a gui front-end useful for debugging"*
  - How do I make a stack frame in MIPS? *"Create a stack frame for itself change $fp and $sp. Save the return address $ra in the stack frame. Save and $s registers that it plans to change"*
  - What are the MIPS floating point registers? *"Mips has 32 32-bit general purpose registers and 16 64-bit floating point registers as well as two special registers hi and lo for manipulating 64-bit integer quantities..."*
  - What is the clock sweep algorithm? *"Uses a reference bit for each frame updated when page is used. Maintains a circular list of allocated frames. Uses a clock hand which iterated over page frame list skipping and resetting reference bit in all reference pages..."*
  - What does execve do? *"Execve... Convert one process into another"*
  - What is envp? *"Envp contains strings of the form key-value"*
  - What does fork do? *"Fork... Create a new child process copy of current process"*
  - What does sigpipe mean? *"Sigpipe... broken pipe no processes reading from pipe"*
  - What does sighup mean? *"Sighup... hangup detected on controlling terminal/process"*
- I get these answers right away and don't have to ask my tutor.

User Story: I want to revise for the midterm test.

- I log in to the chat bot and ask "quiz me"
- I read the quiz questions and try to answer them
- I click "show answer" and check if my answer was right or not
- I am having difficulty with the revision questions for process management in C, so I ask "quiz me on C process management"
- I get some quiz questions specifically about that topic
  - *What happens to a child process if the parent process exits?*
- I think about the answer and once I decide, I check if I was correct by clicking "show answer", which displays the admin inputted data

2.1.2 Course Administrator

User Story: I want to set up my course to work with the chat bot, so that my students can use it.

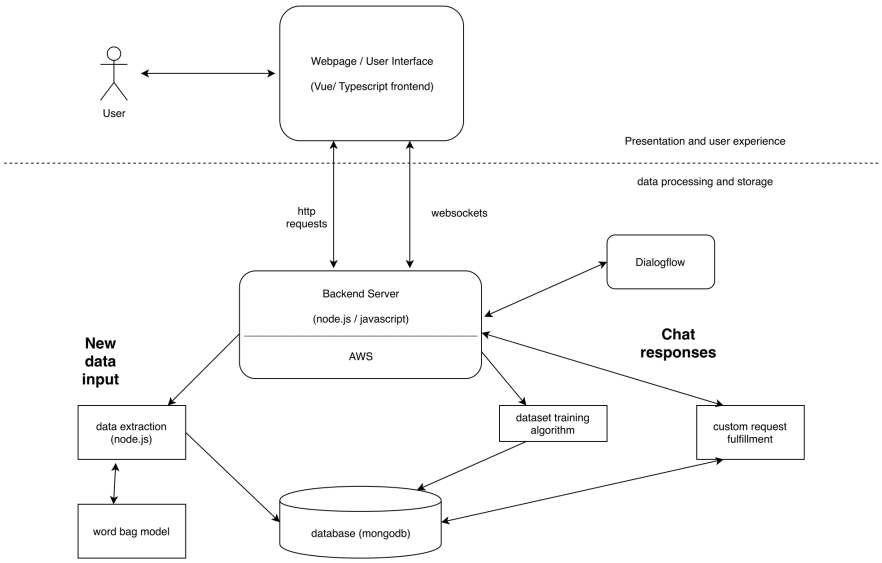
- I register and log in to the admin portal
- I input my course information in the new course setup page
- I supply links to the html pages I want to be included in the data: course outline, assignment specifications, and course content pages that are available online
- I submit the form, and then when the course setup is ready, I'll be able to see it in the admin portal when I refresh the page
- The setup might take a few minutes as the data is processed, but I can do something else while I wait

User Story: I want to add quiz questions for my course to help my students revise.

- I log in to the admin portal and select my course from the menu
- I view the quiz questions I have already added
- I delete some old questions I no longer want my students to see
- I select "add questions" and input more questions
  - What is the size of the general registers in MIPS? Why can't we store a C long long int, in the $t0 register? *mips is a 32 bit architecture and as such the registers and relevant logic circuits only support 32 bit numbers.*
  - What happens to a child process if the parent process exits? *The child process runs independently and does not exit.*
- These will be available right away to students

2.2 System Architecture

2.2.1 Architecture Diagram



2.2.2 Data Scraping & Processing

The bot needs to be initialised when a new course is created. Without any initial data, it will take a significant amount of time before the bot has been trained to a sufficient degree such that it can give meaningful responses to students.

To solve this issue, we developed our own system to scrape and process the data on webcms3. There is a simple interface where the course administrator(s) can input the course code, name, forum and links to pages of content. Our data extraction module recursively follows these links to process and add the data to a database, to create an initial working data set.

It was necessary for us to build this component so that users could be onboarded with minimum hassle. This creates a very low barrier of entry, as the user only needs to provide a small amount of information, and the server will take care of the heavy lifting. Furthermore, webcms3 content is difficult to parse with prebuilt, generic web scraping tools due to its lack of ids and labels, hence the need to build our own tool from scratch.

2.2.3 Extensible Backend

Our bot was built entirely from scratch, with limited dependency on external APIs. The backend is responsible for a number of tasks, including but not limited to processing user queries, generating intelligent responses, managing user sessions and providing access to the relevant databases for user accounts and quiz questions. These tasks can be classfied into one of two loose categories: chat bot and internal API.

The chat bot is effectively the brain and bulk of our system, made to handle user queries. They are processed for keywords and intents to figure out what the user is asking about. Responses are generated accordingly, and are sent back to the user via the web interface, completing one iteration the interaction. This allows students to ask questions about the course content and logistics and receive correct information in a timely manner. The chat bot also provides the ability for students to ask for quizzes, which aid their study and help them revise with official, instructor-endorsed material.

The internal API is a subset of the backend that manages features that complement the functionality of the chat bot. Requests are made to the internal API on the backend via the frontend when the user interacts with the web interface. This enables features such as registering user accounts and adding quiz questions to the database. User state is managed to create personal instances of the bot for each student, which keeps track of frequent questions and problem areas. Adding quiz questions is part of the admin API, which gives course administrators the power to manage the bot for their course.

2.2.4 Interactive User Interface

As briefly mentioned in section 1.3.3, our frontend was built from the ground up. It was designed to be easy to use, aesthetically pleasing and extendable. To this end, we could develop additional features for our system that would not be possible if we were to use an existing chat platform, such as Facebook Messenger or Slack.

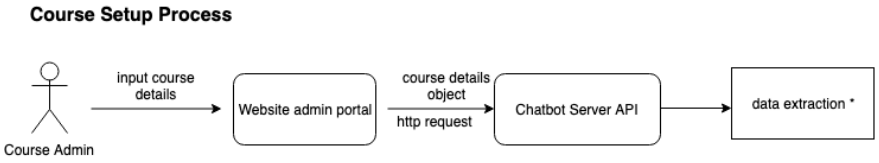
We have implemented a wide array of additional features that are only possible because of our custom frontend. For instance, the chat bot can ask for feedback on the quality of its response, and directly interact with the backend to adjust the weighting of the data in the database. It can also render tables and show images, and maintain a user state as mentioned in section 2.2.3. This is useful for one of our other custom features: the ability to display usage statistics to the course administrators.

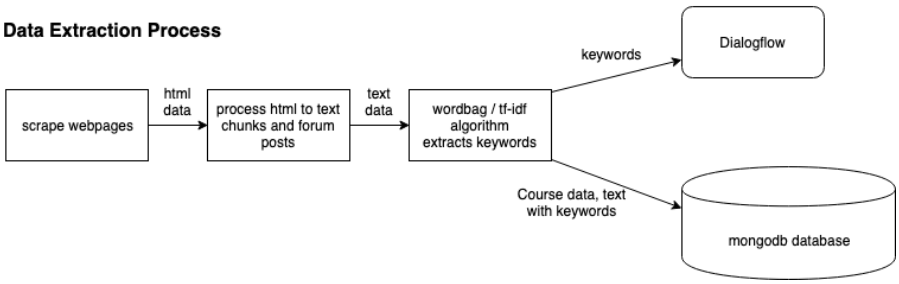
Another section of our frontend is the administrator dashboard. This is the interface that allows course administrators to interact with the system and complete actions provided by the internal API. This includes inputting data (both to expand the bot's dataset, and in the form of quiz questions for students), as well as view information and statistics that the bot has collected.

3 Data Scraping & Processing

3.1 Process

1. User provides input to the simple interface to set up a course.
2. Input is sent to the server and starts the scraper module.
3. Module scrapes the `HTML` from the provided webpages, and also follows forum links recursively to collect data of individual posts.
4. Our custom data extraction algorithm filters the different structures on the page and extracts the relevant text.
5. The data is analysed for keywords using the tf-idf algorithm.
6. The data and its corresponding keywords are inserted into the database.
7. An interactive model of the dataset appears on the webpage once the course is ready.
8. Entities are updated automatically in Dialogflow to include new keywords, so our bot can recognise new requests from students

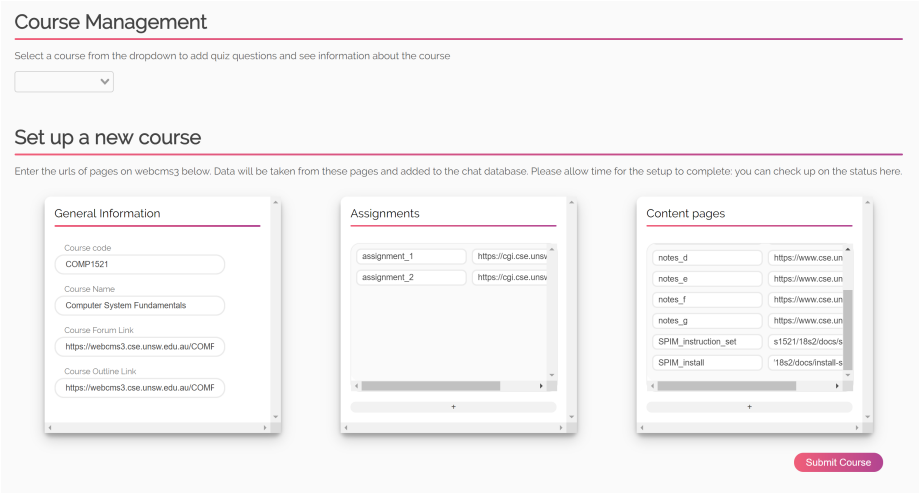




3.2 Example Usage

This is an example use case based on the first user story described in section 2.1.2.

The course administrator registers an account, and then logs in via the front end. They are greeted with the setup dashboard, where they provide links to web pages that will be included in the bot's data set.



Once the form is submitted, the data scraper module will process the pages and extract data. This data will be stored in the database and is ready for use.

```
{ courseCode: 'COMP1521',
  courseName: 'Computer System Fundamentals',
  forum: 'https://webcms3.cse.unsw.edu.au/COMP1521/18s2/forums/',
  outline:
   [ { name: 'course_outline',
       address: 'https://webcms3.cse.unsw.edu.au/COMP1521/18s2/outline' } ],
  assignment:
   [ { name: 'assignment_1',
       address:
        'https://cgi.cse.unsw.edu.au/~cs1521/18s2/assignments/assign1/index.php' },
     { name: 'assignment_2',
       address:
        'https://cgi.cse.unsw.edu.au/~cs1521/18s2/assignments/assign2/index.php' } ],
  content:
   [ { name: 'notes_a',
       address:
        'https://www.cse.unsw.edu.au/~cs1521/18s2/notes/A/notes.html' },
     { name: 'notes_b',
       address:
        'https://www.cse.unsw.edu.au/~cs1521/18s2/notes/B/notes.html' },
     { name: 'notes_c',
       address:
        'https://www.cse.unsw.edu.au/~cs1521/18s2/notes/C/notes.html' },
     { name: 'notes_d',
       address:
        'https://www.cse.unsw.edu.au/~cs1521/18s2/notes/D/notes.html' },
     { name: 'notes_e',
       address:
        'https://www.cse.unsw.edu.au/~cs1521/18s2/notes/E/notes.html' },
     { name: 'notes_f',
       address:
        'https://www.cse.unsw.edu.au/~cs1521/18s2/notes/F/notes.html' },
     { name: 'notes_g',
       address:
        'https://www.cse.unsw.edu.au/~cs1521/18s2/notes/G/notes.html' },
     { name: 'SPIM_instruction_set',
       address: 'https://cgi.cse.unsw.edu.au/~cs1521/18s2/docs/spim.php' },
     { name: 'SPIM_install',
       address:
        'https://cgi.cse.unsw.edu.au/~cs1521/18s2/docs/install-spim.php' } ] }
[2019-04-29T14:58:54.236] [INFO] Database - Connected to database successfully
Scraping Content Pages...
Scraping forum posts...
```

3.3 Technical Details

The frontend is developed in `Vue.js` using `TypeScript`. This helps enforce data types within `JavaScript`, to ensure that the input sent to the backend is valid.

The server is created with `Node.js`. The data extraction makes use of asynchronous `Node.js` to reduce the amount of time spent in blocking processes such as api calls and filesystem/`http` calls. This is particularly important because it would otherwise cause the server to block and fail to process requests. Instead of having to spawn a new process, `Node.js` is able to manage all running processes and allow the data extraction module to run in the background.

We also use the `cheerio js` library, as it provides tools to parse `HTML` into usable data.

`MongoDB` is used for the database. It is the most appropriate software to handle the unstructured data that we deal with.

4 Extensible Backend

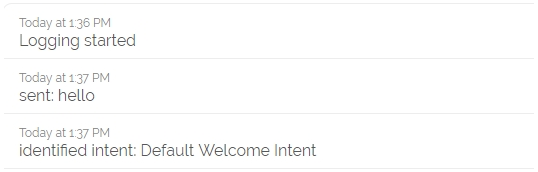
4.1 Chat Bot

4.1.1 Process

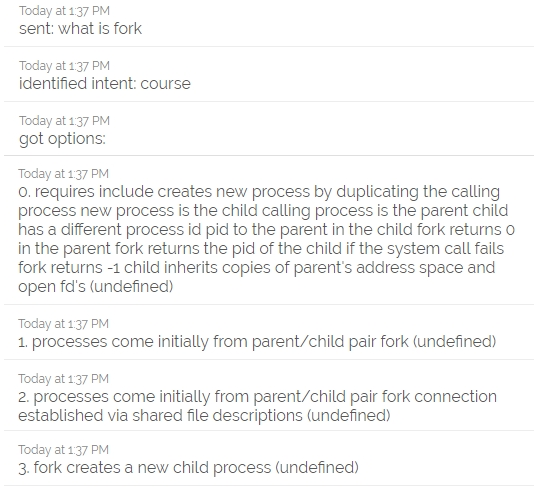
1. The user sends a query to the chat bot via the web interface.
2. The chat bot sends the query to Dialogflow to extract the keywords and intents.
3. It will then search the database to find candidate responses.
4. A score is calculated for each candidate based on its theta value and salience.
5. The candidate with the highest score is considered the best answer.
6. When deciding on a response, the chat bot also considers whether or not the user's query is a question.
7. The final chosen answer is returned to the user via the web interface.

4.1.2 Example Usage

The user accesses the chat interface and sends a greeting message. This is parsed and detected as a "Default Welcome Intent", and the bot responds accordingly.



The user then asks "What is fork?" This query is also processed and the intent is identified as corresponding to the course content. The chat bot queries the database and chooses from the answers with the highest score. The answer with the highest score is displayed at the top in the debug log shown below.



4.1.3 Technical Details

The chat bot is retrieval-based and functions on two layers. The initial layer is the Dialogflow component. It handles the natural language processing of the query and extracts keywords, as well as dealing with responses for non-relevant questions. Dialogflow was chosen as it is a known, reliable module and has core NLP features such as small talk, which help our bot appear more human. It also has an API for communication with our backend, which suited our needs perfectly.

The second layer of our chat bot is the bag of words model. This was designed to process only relevant queries, once the keywords have been extracted by Dialogflow in the first layer. As we don't rely on a set of training data, our system cannot be pre-trained. Thus, any deep learning module such as LSTM would not work in our case. The bag of words model was deemed the most appropriate for our use case, as it is able to retrieve data points without any prior training.

This was chosen over the Google Cloud NLP's keyword extraction module because the latter occasionally missed important tags, and also because it calculated the salience such that the sum of one data point will be one. This is not ideal for our system, as long sentences will tend to have a lower salience and be less likely to be presented to users. Instead, the tf-idf algorithm that we implemented suits our use case better, as it does not miss any tags for our data points, and also assigns a lower salience to common words, which helps us calculate accurate scores for each data point.

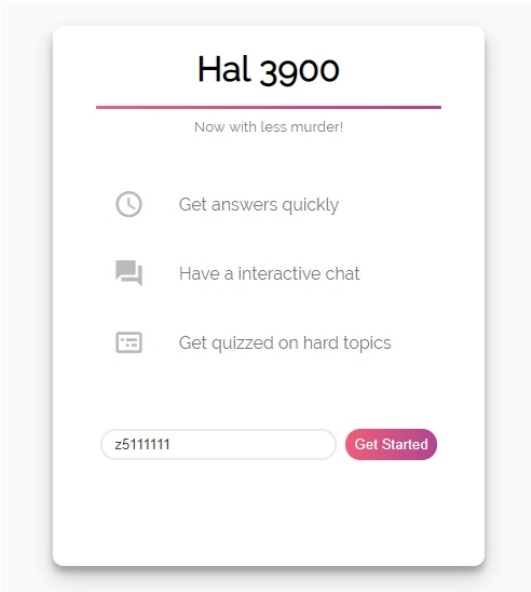
4.2 Internal APIs

4.2.1 Process

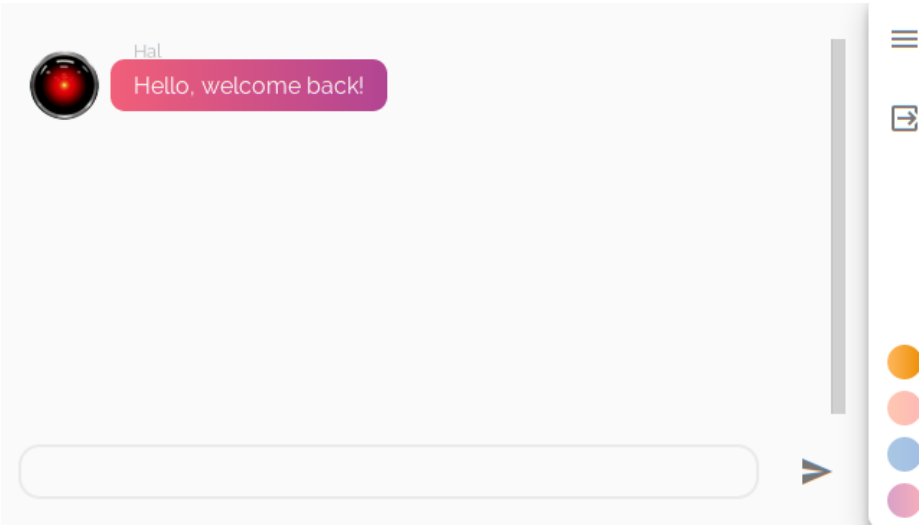
1. The backend API runs in its own instance, separate from the frontend.
2. The user makes a request via the interface on the frontend.
3. The frontend sends an `HTTP` request to the API with the details of the user's request.
4. The API receives the request and routes it appropriately.
5. The corresponding function is called and handles the user's request.
6. An `HTTP` response is sent back to the frontend based on the behaviour of the function.
7. The frontend displays an appropriate response to the user based on the response it receives.

4.2.2 Usage

The internal API is accessed when a user wants to register. When accessing the homepage, the user will see a registration form. Using this, they can choose a name and confirm it by pressing the button.



The `Get Started` button sends an `HTTP POST` request to the internal API at a specific URL. The backend router associates this with the function to register or log in a user. The function runs, and checks if the user already exists. If they do, their previous session will be loaded. Otherwise, a new session will be created for the user. The API then sends an `HTTP` response back to the frontend and the user can proceed to use the system.



4.2.3 Technical Details

The backend is primarily constructed using `Node.js` in `TypeScript`. It was chosen due to its high amount of usage in the industry and abundance of packages and documentation that could be taken advantage of. One of the main advantages of using `Node.js` over something like `Django` for example, is the availability of an easy-to-use web socket implementation. This allowed us to easily create a real-time chat system, which is part of our core functionality.

`Express.js` and `express-session` were used to handle routing and user sessions, respectively. Considering there was already a reliable implementation of these core but basic features, we did not see a need to create our own. Similarly, `express-ws` was used to deal with the web socket functionality that we needed.

`TypeScript` was chosen mainly for development purposes. It helped to ensure our code was robust and error free by enforcing data types to avoid common `JavaScript` errors.

5 Interactive User Interface

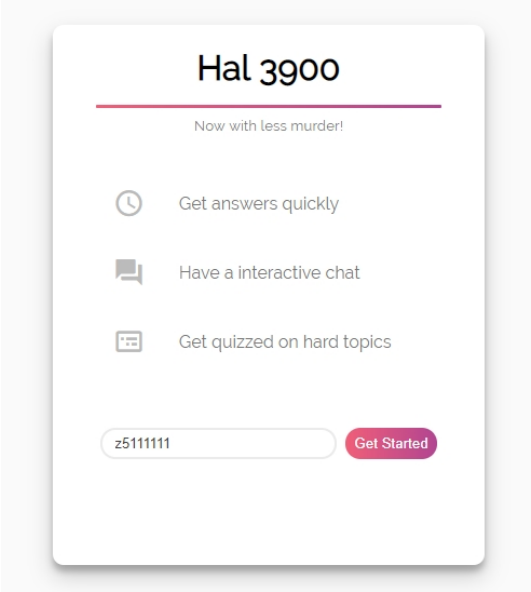
5.1 Process

1. The user makes a request to the frontend by visiting the website in their browser.
2. The frontend router handles the request and displays the page requested.
3. The user sends a message by typing into the text field on the chat page.
4. The frontend detects that the user is awaiting a response and displays a loading animation.
5. The message is stored in an internal `Vuex` state and is also sent to the backend via a lazily created websocket.
6. The backend sends a response back to the frontend through the same websocket.
7. The response is stored in the `Vuex` state.
8. The frontend detects a change and renders the page again to end the loading animation and display the response.

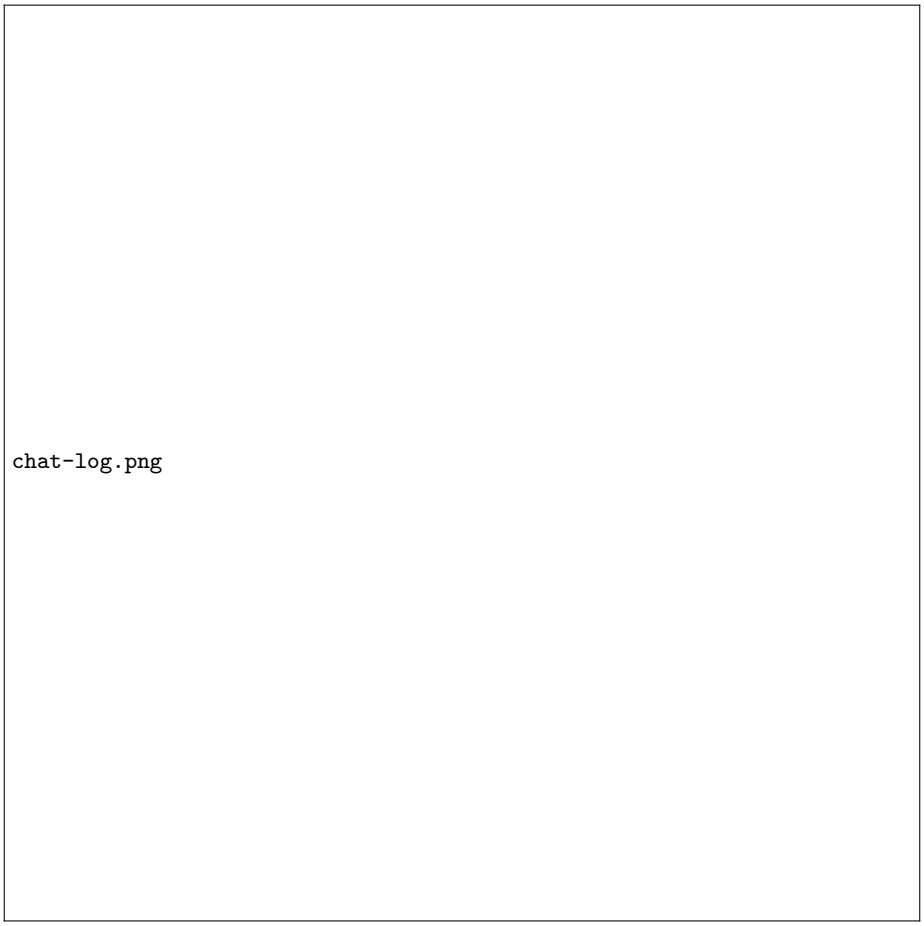
5.2 Example Usage

This is an example use case based on the first user story described in section 2.1.1.

The student visits the homepage for the chat bot and is greeted by the welcome screen. They are then able to register by inputting a username, which can be anything from their real name to their zID. There is no other information required for registration to make it fast and easy to use. Password authentication was deemed not required, as no sensitive data can be accessed.



Once logged in, the user can use the intuitive chat interface to interact with the chat bot. They are free to ask questions about anything relevant, including the course content or for a quiz. In this case, the student would like to learn more about the course administration.



5.3 Technical Details

The web interface is developed using `Vue.js`. This allows us to build a modular and extensible frontend with reusable components. For example, the generic message base component was extended to allow the bot to respond with a number of message formats, such as a regular message, a set of options that can be selected from, a quiz question and answer pair, or a table.

`Vuex` is a state management pattern and library that was used to centralise the data that the frontend needs to access. This helps to limit the number of times the page needs to be rendered after a change is made. It also allows for complex logic, such as expressing the chat bot as a set of components. Implementing this helped us add

features in short sprint cycles as it eliminated the need to constantly move around large amounts of information.

`TypeScript` was also used here for reasons similar to those described in other facets of our system. It helps mitigate the problems that come with `JavaScript` such as undefined or malformed variables, allowing us to avoid or handle errors better.

We also used `Sass` to quickly and cleanly style our website. This was done to make the user interface as pleasant and usable as possible for the end user.

6 Conclusion

6.1 How Existing Problems Were Addressed

Section 1 highlights three main problems with the existing system of hiring tutors to answer student questions. Our solution addresses all of these problems to a high degree.

1. Responses to students are delayed.

When a student wants to ask about something related to the course, they would typically have to email their tutor or a lecturer, or post on the course forum. Alternatively, they would have to wait until a lecture or tutorial to ask the question in person. As course staff are not constantly available, there is often a time lag between when a question comes up and when it is answered, which ranges from hours to days.

Our system solves this problem by virtue of being a chat bot. It is hosted live on an `EC2` instance from Amazon Web Services, and is running around the clock. Students can ask questions at any time of the day, and receive an instantaneous response, effectively eliminating any and all human delay.

2. Tutors have a limited amount of time.

Casual tutors are typically only paid for one hour of associated work per class. Lecturers are also not expected to spend every waking moment attending to their course. As such, there are often times when many student queries go unanswered due to lack of time, or also simple negligence. This is true especially in times of spikes in question frequency, particularly around exam period or assignment due dates.

This problem has also been addressed by our system. Our bot is able to process any reasonable number of student queries, as it opens a separate web socket for each connection. This allows it to respond to all queries it receives without fail.

3. Hiring tutors costs money.

With growing cohort size, it becomes difficult to fund a large number of tutors to handle an exponentially growing number of student queries. Many courses are already having trouble with funding and have had to cut back on the number of tutors they hire, or the amount of work they assign to them.

Similarly, this problem is also solved by our system. The only upkeep cost for our system is a few dollars of AWS credits (several hundred of which are given to academic staff for free, from Amazon). The system is effectively free to run, and does not even contribute to the electricity bill due to it being hosted by Amazon. Despite this, it is able to handle a large proportion of the work that previously required paid tutors.

6.2 Challenges

The first challenge we faced when developing our system came about when we were looking for a solution to process the data from the required source, webcms3. As mentioned in section 2.2.2, the structure of the data on the webpage proved difficult

to extract, and we had to build our own solution to address this issue. The details of this solution can be found in section 3.

We were also using Google NLP originally, but as our data set grew larger, we noticed a significant decrease in performance. This affected the real time experience that we wanted to deliver to users, so we ended up designing our own NLP algorithm. This algorithm is described in section 4.1.

Another roadblock we hit was during deployment. Our frontend and backend run on `hal-3900.com` and `backend.hal-3900.com` respectively, but both reference a different server. To resolve this, we needed to configure `Apache` to direct the traffic of each domain to the corresponding `Docker` container. This posed issues as `Apache` assumed an `HTTP` protocol, but web sockets run on an entirely different protocol. After much struggle, we managed to solve this by rewriting a rule to specifically find the web socket endpoint and change its protocol to `ws`.

6.3 Improvements

One of the main deficiencies of our current system is that it relies on a predetermined list of keywords. If it receives queries that do not contain words in this list, it will be unable to handle the request. In its current state, the system is unable to provide any meaningful information about this beyond counting the number of occurrences. For further development, it would be possible to parse the user input periodically with our wordbag algorithm to potentially create new entities for Dialogflow.

Another area for improvement is the course statistics. The current implementation is quite basic and only keeps track of a cumulative set of data over the duration of the course. It would be more beneficial to course staff to provide options for this, such as tracking them by week. This could further be extended to generate notifications for course staff, for example by requesting more time be spent on topics students are having trouble with (perhaps by highlighting quiz questions that were frequently answered incorrectly).

7 Appendix

7.1 Technologies

The primary technologies used in development of this system can be found below, along with the version used and license type. The justifications for the choice of each technology can be found in the sections describing the components that use them.

Technology	Version	License
Node.js	11.11.0	BSD
Nodemon	1.18.10	MIT
Vue.js	2.6.6	MIT
vue-class-component	6.0.0	MIT
vue-property-decorator	7.0.0	MIT
vue-router	3.0.1	MIT
vue-template-compiler	2.5.21	MIT
Vuex	3.0.1	MIT
TypeScript	3.2.1	Apache
node-sass	4.9.0	MIT
sass-loader	7.1.0	MIT
babel-eslint	10.0.1	MIT
eslint	5.8.0	MIT
eslint-plugin-vue	5.0.0	MIT
Express.js	4.16.4	MIT
express-session	1.15.6	MIT
express-ws	4.0.0	BSD
Dialogflow Node.js Client	0.8.2	Apache
cheerio	1.0.0	MIT
MongoDB	3.1.13	SSPL
Mocha	6.0.2	MIT
uuid	3.3.2	MIT

The license for each of the dependencies is provided with the software to comply with the requirements of their use. The dependency manager `npm` handles this in deployment.

7.2 Installation Manual

7.3 Usage Manual

7.4 Source Code Structure

Our root directory contains a number of directories. Each corresponds to a certain component of our system. They can be thought of as a collection of standalone modules. These are **Backend**, **Frontend**, **Data_Extraction**, **Database**, **Dialogflow** and **IR**.

Backend

`index.js` is the main file for the backend which gets run when the system starts. All other files are its modules. `bot.js` contains the actual chat bot module, described in section 4.1.

`router` contains files that deal with the routes of the backend. `routes/talk.js` is the module that handles the web socket that the chat bot runs on. `routes/router.js` defines all of the routes (paths) for access, including the web socket and all of the API paths.

The API modules are stored in `routes/api`. `routes/api/users.js` handles the API functions for users (fetching users, logging in/registering) and `routes/api/quiz.js` contains the API functions dealing with the quiz feature.

Frontend

Data_Extraction

Database

Dialogflow

IR