Software Engineering Group Project

COMP2002/G52GRP: Final Report

Project information:

Project title: Watson Avatar For New Students

Project sponsor: Ender Ozcan

Academic supervisor: Ender Ozcan

Industry Sponsor: John McNamara

Industry Supervisor(s): John McNamara

Team information:

Team number: 4

(Junhan, LIU, 20321796, scyjl15)

(Zainab, Shakeel, 20337691, psyzs3)

(Zeyu, WEI, 20319939, scyzw11)

(Mengtong, XIE, 20319948, scymx1)

(Inderpal, Thaliwal, 20388814, psyit1)

(James, Thompson, 20366153, psyjt10)

(Huai-Sheng, Cheng, 20210331, efyhc5)

Documentation (links):

Trello board: Trello Board

Code repository: GitLab Repo

Document repository: Google Drive

IBM Watson Repository: Watson Assistant

Table of Contents

Main Report

1. Project Background & Understanding

1.1 Evidence of the own interpretations and specification

Transitioning to university can be an exhilarating experience for freshers as it brings forth a plethora of opportunities and activities to engage with. However, it can also be an arduous task for new students to effectively acclimate themselves to the university's milieu. As a result, many students seek the aid of existing peers or faculty, while others resort to scouring the internet. Nevertheless, such endeavours may prove to be unproductive, especially for introverted or timid students. The university administration has made attempts to address this issue by organising seminars for new students that provide them with strategies to navigate university life. Nonetheless, these seminars entail considerable effort and resources. To explore this matter further, we conducted a survey to gauge the difficulties encountered by freshmen in their initial year, as well as their views on a personalised AI system tailored to assist them. Of the respondents, 69% attested to experiencing difficulties during their first year. Thus, the development of an application that furnishes new students with a personal chatbot is not only an efficient and expedient solution but also serves as a mechanism to alleviate the burden on university staff.

The project entails the development of a web-based application featuring a Watson chatbot, which will be accessible to students via their mobile devices. QR codes will be printed out and placed strategically around the university , targeting new students who may face difficulties navigating and adapting to university life. The chatbot will provide useful information that a university student wished they knew when they arrived at university. The content of the chatbot will be specifically designed to cater to the needs of first-year students at the University of Nottingham, with the aim of facilitating a smooth and rapid transition into the university environment.

1.2 Depth and context of the problem presented

As a component of IBM's DeepQA initiative, a group of researchers led by Principal Investigator David Ferrucci developed the IBM Watson system. This advanced computer program, known as IBM Watson, is designed to respond to inquiries posed in natural language. IBM devised Watson as a question-answering (QA) computer system, intending to harness state-of-the-art developments in automated reasoning, knowledge representation, information retrieval, natural language processing, and machine learning for open-domain question-answering applications. In comparison to other prevalent chatbot development platforms, IBM Watson offers numerous benefits. IBM asserts that Watson employs over 100 distinct techniques to analyse natural language, identify sources, formulate hypotheses, gather and evaluate evidence, and synthesise and rank ideas. Consequently, our Watson-based product demonstrates enhanced proficiency in comprehending user queries, as well as exceptional customization and scalability.

In contrast to seeking guidance from a mentor directly, a chatbot can furnish new students with more rapid and convenient responses. Additionally, it can alleviate the workload for university staff,

as it operates entirely autonomously and necessitates minimal intervention from staff beyond routine maintenance and updates.

2. Requirements & Critical Analysis

2.1 Prioritisation of requirements.

To ensure a mutual understanding of what is required of the product, our team created a software requirement specification (SRS) document. This document outlines the features and functionality of the product and is intended to be used by both our sponsor, IBM, and our group. In the SRS document, each requirement is assigned a priority ranking, ranging from high to low, with high being the most critical. By ranking the requirements based on importance, the SRS document provides a clear indication to both the developers and stakeholders of the priority of each requirement. This helps to delegate and complete tasks efficiently while ensuring that the project meets the needs and expectations of all stakeholders.

1.2 Document Conventions

The conventions used in this document for defining the requirements are described in this section.

Each requirement has a unique identification number that consists of a string of letters followed by a number, for example, REQ1.1. These numbers are assigned when the requirement is created. The requirement numbers are intended to be permanent and must not change when sections or requirements are added to or deleted from this document.

Each requirement is also assigned a short title that allows it to be identified in the table of contents. Following the title is a clear one or two-sentence description of the intention of the requirement.

The requirements will be ordered in terms of priority: High meaning critical, Medium meaning necessary but can wait and Low meaning can wait until resources permit.

Figure 1: Team ’s document conventions

In the SRS Document, our group implemented the MoSCoW method to effectively prioritise tasks based on their importance level. We categorised our requirements into four different groups, namely ‘Must have’, ‘Should have’, ‘Could have’, and ‘Would have’. Focusing on the 'Must haves' ensures that the main components of the project are finished first, and the finer details can be added later. Using the MoSCoW method also allows us to distinguish and cut out unnecessary requirements, keeping our document simple and agile. This approach helps our group minimise wasted time, effort, and resources, ensuring that the project is delivered efficiently and effectively. By prioritising requirements in this manner, we can maintain a clear focus on what is most important for the project's success and make better-informed decisions about resource allocation and project planning.

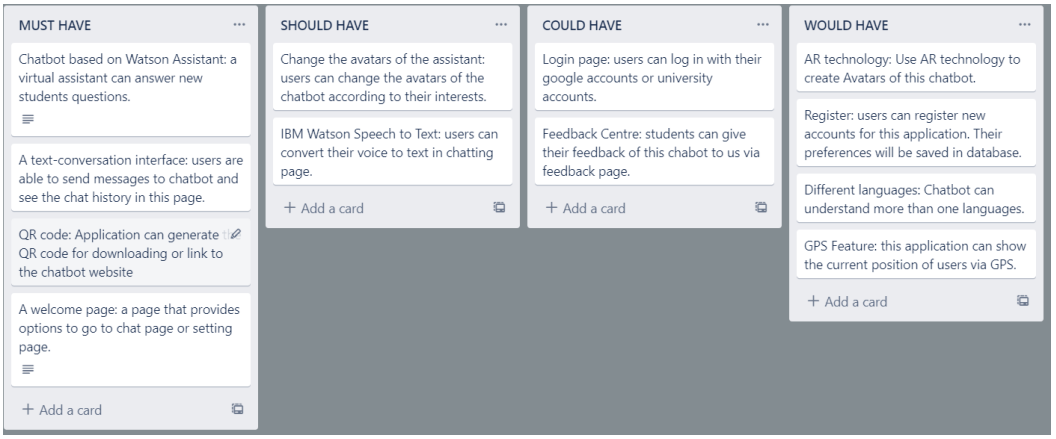


Figure 2: Prove of MoSCoW method

Our team developed user stories and a use case diagram to enhance the user experience and demonstrate how diverse users will interact with the product in various ways, emphasising their distinct priorities. By utilising user stories, we provided valuable context about potential users, aiding developers in prioritising the most important requirements for different stakeholders.

2.5 User Stories

1. As a new student from London, I want the locations and open hour of all the restaurants and canteens in the Park Campus, so that I can know where to have lunch at school
2. As a "2+2" computer science student transferred from Ningbo campus, I need more academic sources in the campus library, so that I can get a high mark on the Operating System module.
3. As an exchange student from the USA, living in Beeston, I want to know the timetable of the Hopper bus and other means of transportation to the Jubilee Campus, so that I will not be late for class every day.
4. As a Chinese student who lives near the Jubilee Campus, I want some good Chinese food restaurants near the campus, so that I can get a taste of my hometown with my Chinese friends.

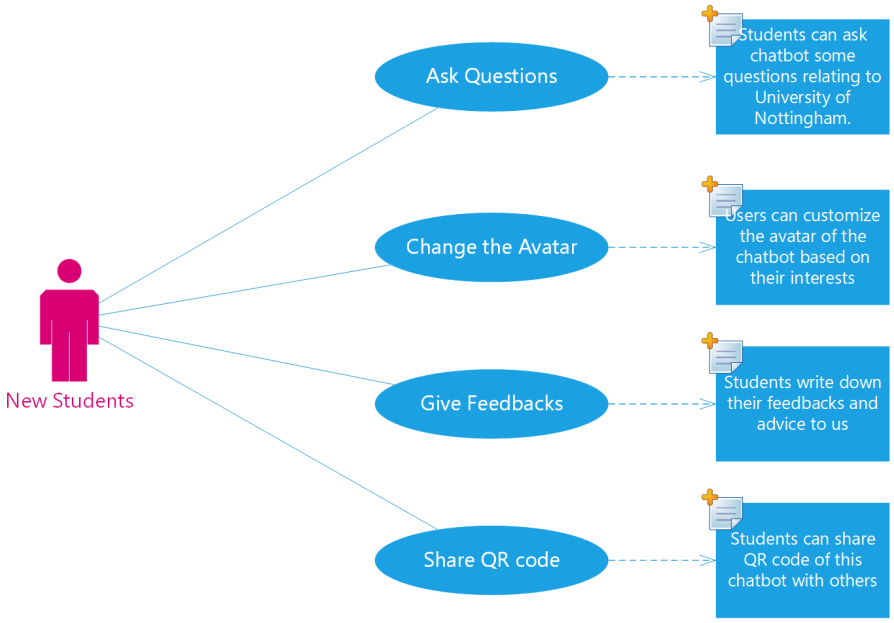


Figure 3: User stories

2.2 Analysis of tools for the project

IBM Cloud platform offers a suite of tools and interfaces that facilitate the management and interaction with Watson services. Among these tools, IBM Watson Assistant stands out as an AI-powered virtual assistant that delivers consistent and accurate responses to clients across various platforms. Using natural-language understanding, natural language processing, and machine learning, this tool enables us to create a chatbot that can better comprehend questions, automate the search for suitable answers, and execute the user's intended action. By leveraging Watson Assistant, our chatbot can engage in more natural and seamless conversations with users by emulating and identifying the numerous permutations of intent in real-world interactions.

Moreover, IBM Cloud's Watson Speech to Text technology provides swift and precise speech transcription in multiple languages for a wide range of use cases. We opted for this tool as it can be seamlessly integrated into our Watson chatbot.

Android Studio, the official integrated development environment (IDE) for Google's Android operating system, will be employed by our team to develop and incorporate an Android-based application for our chatbot. Leveraging Java as our primary programming language, we capitalise on our extensive experience gained from previous university projects.

In addition to the core application, we will create an avatar for the chatbot, utilising one of two possible methods. The first approach involves the use of Blender, an open-source 3D computer graphics software suite, renowned for generating animated films, visual effects, artistic renderings, 3D-printed models, and motion graphics.

2.3 Changed of Requirements

Throughout the project, the principal requisites for the Watson chatbot initiative at the University of Nottingham remained consistent with the original stipulations established during the project's inception. The team had undertaken an exhaustive assessment of the University's needs, formulating a precise set of objectives to accomplish. This meticulous approach guaranteed that the project was clearly delineated and that the team possessed a coherent roadmap to follow.

Nonetheless, several minor adjustments to the requirements emerged during the project's developmental phase. These changes primarily pertained to slight modifications in the chatbot's functionality and user interface, informed by feedback garnered throughout the testing phase. For instance:

1. Enhancing natural language understanding: During testing, it became evident that the chatbot encountered difficulties in comprehending specific user inputs or phrases. To address this, we updated IBM Watson Assistant's training data and refined intent recognition.

2. Integration of external APIs: Our supervisor requested features that relied on data from external sources, necessitating the incorporation of supplementary APIs into the Java backend.

3. Addition or modification of intents: As the project advanced, newly identified use cases or user requirements demanded the addition or alteration of intents within IBM Watson Assistant.

4. Refining context handling: Testing revealed that the chatbot struggled to maintain context during conversations, prompting us to improve the management of context variables in the dialogue nodes.

5. Augmenting the chatbot's appearance: Users proposed alterations to enhance the chatbot's visual appeal and accessibility, leading us to introduce a night mode and modify the size and colour of the fonts within our chatbot applications.

6. Addressing the needs of vulnerable groups, such as colour-blind or hearing-impaired individuals, when crafting the UI design.

Our team demonstrated adaptability by quickly adjusting to these changing requirements, reprioritizing tasks, and reallocating resources as needed.

2.4 Response to Changing Requirements

Our team was committed to ensuring that the project could respond swiftly and effectively to evolving requirements. To achieve this, we collaborated closely in group meetings to identify the most appropriate solutions, and implemented necessary adjustments promptly upon receiving updates. We maintained open and transparent communication among team members and with stakeholders through regular meetings, enabling us to keep everyone informed of any changes and their implications for the project.

For instance, when we finished developing the chatbot's chatting page based on the prototype designed in the requirements gathering process and showed the progress to the supervisor, he proposed two new minor non-functional requirements: enhancing the user interface design of the chat page to offer an elevated visual experience for users and considering the needs of vulnerable groups, such as colour-blind or hearing-impaired individuals, to ensure accessibility and inclusivity. After receiving the new requirements, we took several steps to respond to them.

1. Review and analyse: We reviewed and analysed the changes thoroughly, considering their impacts on users, the importance and priority of these two requirements, and how to improve the UI of the chatting page. After conducting research on various social media platforms' colour designs for chat pages, including WhatsApp, Snapchat, and QQ, our team ultimately decided to adopt the UI design of QQ and make necessary modifications to fit with the main UI theme of our application.

2. Update documentation: We updated the documentation (meeting records and Kanban board) to reflect the changes.
3. Rearrange project plan: We rearranged the project plan and incorporated the new requirements into the development process. All agreements and plans were communicated and updated for all members.

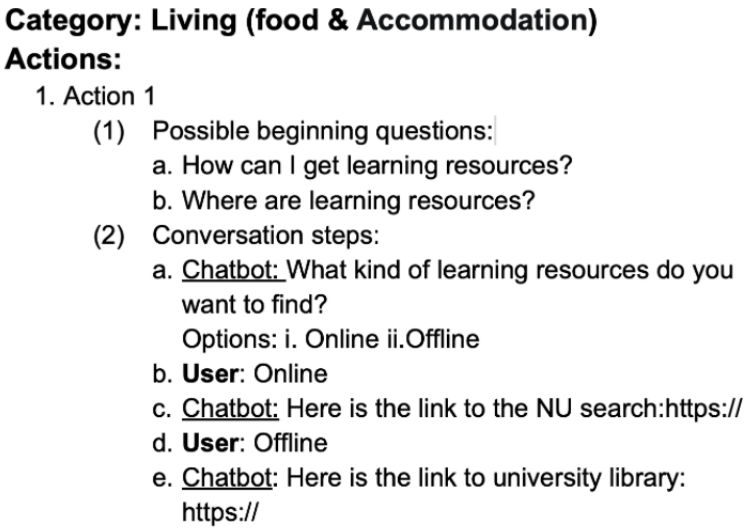
4. Implement changes and seek approval: After implementing the changes, we presented the updated page to all team members and the stakeholder to ask for their approval.

These four steps have become a simple system for us to react to changing requirements and will be valuable lessons that can be applied to future projects. These experiences include the importance of stakeholder engagement, flexible project planning, iterative development, risk management, and documentation. They will help us better anticipate and respond to changes in requirements, ensuring the continued success of our projects.

3. Software Implementation & Testing

3.1 Processes of Creating Chatbot Conversations

The initial step in our process involves transcribing conversations into written form and organising them into distinct documents based on their content classification. We follow a predefined format and strive to generate as many conversations as possible. Through group discussions, we decide whether or not to include them in the system. After we decided, we created an action version of conversation documents and prepared to add it into the AI chatbot system.



Category: Living (food & Accommodation)
Actions:

1. Action 1
   (1) Possible beginning questions:
      a. How can I get learning resources?
      b. Where are learning resources?
   (2) Conversation steps:
      a. Chatbot: What kind of learning resources do you want to find?
         Options: i. Online ii.Offline
      b. **User**: Online
      c. Chatbot: Here is the link to the NU search:https://
      d. **User**: Offline
      e. Chatbot: Here is the link to university library: https://

Figure4: Document format of conversation actions

Subsequently, intents are created within the IBM Watson system, which are crucial for enabling the AI to recognize and understand user conversations. In the intents section, pertinent conversation examples or keywords should be entered into the "user example" field. To optimise the system's performance, it is crucial to consider the user's perspective and anticipate the keywords and questions they are most likely to use. This approach helps ensure that the AI can effectively interpret and respond to a wide range of user inputs.

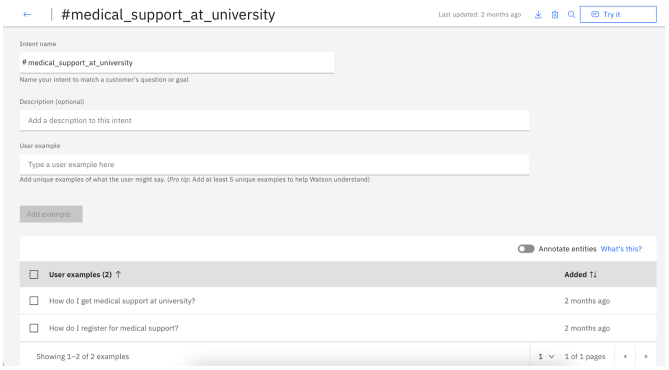


Figure5: Creating intents

In the final phase of implementing the IBM Watson system, answers are generated using nodes within the Dialog section. To organise content, node folders are created based on content classification, with links and responses added under corresponding nodes and subnodes. When adding a new node, it is important to first locate the appropriate node folder.

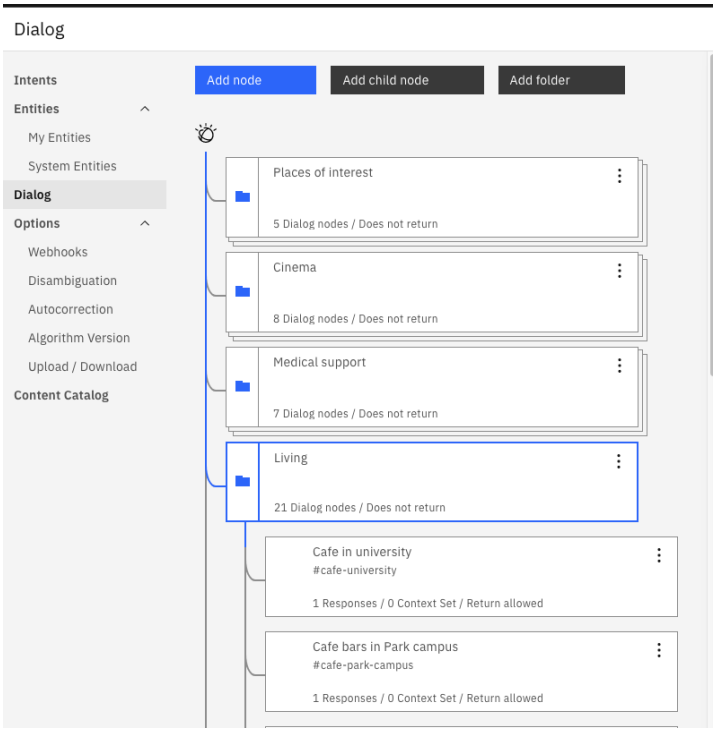


Figure6: Node structure

As illustrated in Figure 4, the node creation interface requires inputting relevant intents at the top. Each node consists of one or more intents. Response types include several response types, which cater to different user needs. Text type and option type are most used. Text type is used for direct textual answers or links to external web pages. While option type enables users to select a more specific question, and these choices are then linked to other nodes containing relevant questions.

Additionally, the system features a welcome node for preliminary interactions and an "Anything else" node for conversations that are not successfully identified.

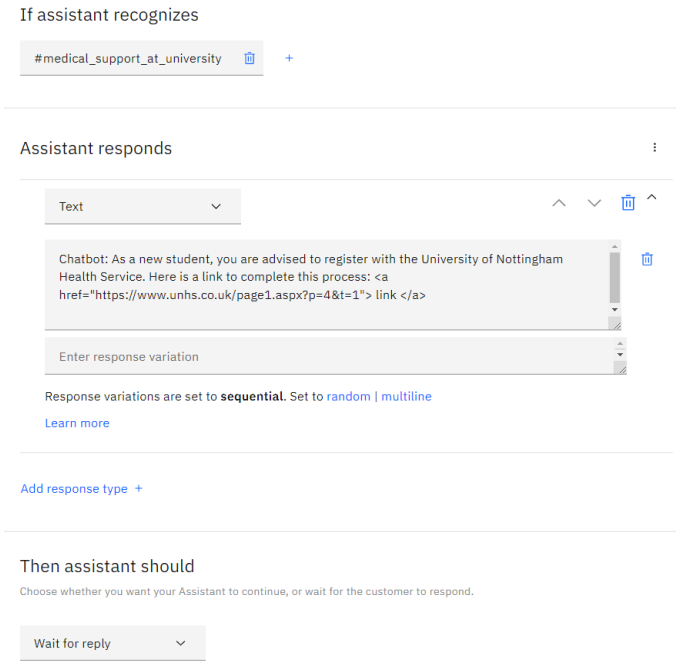


Figure7: Creating new node

In light of the constraints inherent to the IBM Watson platform, our team has developed a comprehensive chatbot solution comprising 100 distinct intents and nodes. This system effectively addresses the multifaceted challenges faced by incoming students in areas such as academics, residential life, and professional development.

3.2 Changing Avatars

Implementation:

The implementation of the code is divided into three parts: attributes, methods, and listeners. Attributes are defined for buttons, image switchers, and gesture detectors. The methods are defined for handling the logic of buttons and the swipe gesture. Finally, listeners are implemented to handle the events when buttons are clicked or swiped.

Testing:

The testing of the code is divided into two parts: functional testing and non-functional testing. Functional testing is performed to ensure that the code meets the requirements and works correctly. Non-functional testing is performed to ensure that the code is efficient, reliable, and user-friendly.

Functional Testing:

The functional testing is performed to ensure that the code meets the following requirements:

1. The code should display the images of avatars.
2. The code should allow the user to switch between the images of avatars using the right and left arrow buttons.
3. The code should allow the user to switch between the images of avatars using swipe gesture.
4. The code should allow the user to save the selected avatar.

To test the functional requirements, the following test cases are performed:

1. The images of avatars are displayed correctly.
2. The right arrow button switches between the images of avatars from left to right.
3. The left arrow button switches between the images of avatars from right to left.
4. The swipe gesture switches between the images of avatars from left to right.
5. The selected avatar is saved correctly.

Non-Functional Testing: The non-functional testing is performed to ensure that the code is efficient, reliable, and user-friendly. The following test cases are performed:

1. The code is efficient in terms of memory usage and processing time.
2. The code is reliable in terms of handling exceptions and errors.
3. The code is user-friendly in terms of displaying the images of avatars, buttons, and error messages.

3.3 UI Designs for the Application

3.3.1 Static UI design:

Design Choices:

1. Colour Scheme:
   The colour scheme was chosen to mirror the University of Nottingham's branding and to provide students with a sense of familiarity. To ensure readability and visual appeal, a combination of the university's official colours and complementary hues was chosen.
2. Font Selection:
   To improve legibility and prevent eye strain, a clean, easy-to-read typeface was chosen for the application. The font used strikes a good balance between professionalism and warmth, making the chatbot more approachable to students.
3. Design of Avatars:
   Four separate avatars were created to provide consumers a sense of individuality and control over their chatbot experience. The avatars cater to various preferences, ensuring that every student can select an avatar that they identify with.
4. Layout and Navigation:
   The layout was created to be simple and intuitive, with clearly defined sections and buttons for easy navigation. This design ensures that even non-technical students can utilise the application comfortably.

Realisation using Photoshop:

Adobe Photoshop was used to generate all static images, including backgrounds, buttons, and avatars. Advanced tools and capabilities in the software allowed for fine control over the design aspects, maintaining aesthetic consistency and high quality.

Multiple variations of the UI design were created in Photoshop and tested for usability, readability, and general appeal. The results of these tests were used to tweak the design until an optimal version was obtained.

3.3.2 Dynamic background

Design Choices:

The dynamic background for our application menu was built on top of our static background design. The objective of this dynamic background was to add to the user experience, ensuring that the chatbot felt modern, interactive and engaging to use. While implementing this feature, it was important to us that our already usable and intuitive interface design was not hindered by any unnecessary moving parts. Therefore, we decided to make a simple scrolling effect with the existing static background design. By doing this, the UI design was given a new layer of depth and quality that a solely static background could not produce, without affecting usability.

Realisation:

To achieve this scrolling effect, we used the static background design and separated it into two separate images. These images were made to have continuity with each other to ensure that scrolling between them was consistent and smooth. By layering these images on top of eachother, we were able to programmatically offset and shift them to create an infinite scrolling effect.

This was done using the ValueAnimator class in Android Studio. The images are controlled by an update listener, which retrieves the current animator value and uses this to calculate the correct coordinates to display and offset each image. This class ensures that our animations occur on the UI thread, limiting their effect and cost on other application features and meaning that they always load promptly and run smoothly.

The speed of the scrolling animation was tested and tweaked multiple times during the development process to verify that it was not distracting, but rather felt natural and truly benefited the application's UI design and reached our design goals.

3.4 IBM Cloudant Database

The IBM Cloudant framework serves as the back-end database for collecting users' feedback in this application. It is a fully-managed Not Only SQL(NoSQL) distributed database service that is based on Apache CouchDB. Apache CouchDB is a NoSQL document-oriented database system that is open-source and emphasises ease of use, dependability, and scalability. The database stores data in JSON document format, providing a flexible and adaptable schema design. It is specifically engineered to be highly scalable, tolerant of faults, and distributed.

The primary objective of IBM Cloudant is to provide a seamless and effortless process for storing, indexing, and querying JSON data while also offering full-text search capabilities. IBM Cloudant is capable of handling vast amounts of unstructured or semi-structured data and is particularly well-suited for mobile, web, and IoT applications. Thus, it is an ideal choice for the feedback centre function of this project, which may involve the transfer of significant amounts of data from users if this application gains widespread adoption.

There are many advantages of this kind of JSON based NoSQL databases in comparison to traditional SQL databases.On the one hand, NoSQL databases usually have a flexible or dynamic schema, which allows for the storage of diverse data types and structures without the need to modify the database schema. This makes NoSQL databases suitable for handling data with varying attributes or when the data structure is expected to change over time. On the other hand, NoSQL databases are designed for horizontal scaling, which means that they can handle increased data volume and throughput by

distributing data across multiple nodes or clusters. This makes them a suitable choice for large-scale applications and rapidly growing datasets, where SQL databases might struggle to scale efficiently. Above are the primary reasons for choosing Cloudant as the back-end database for our project.

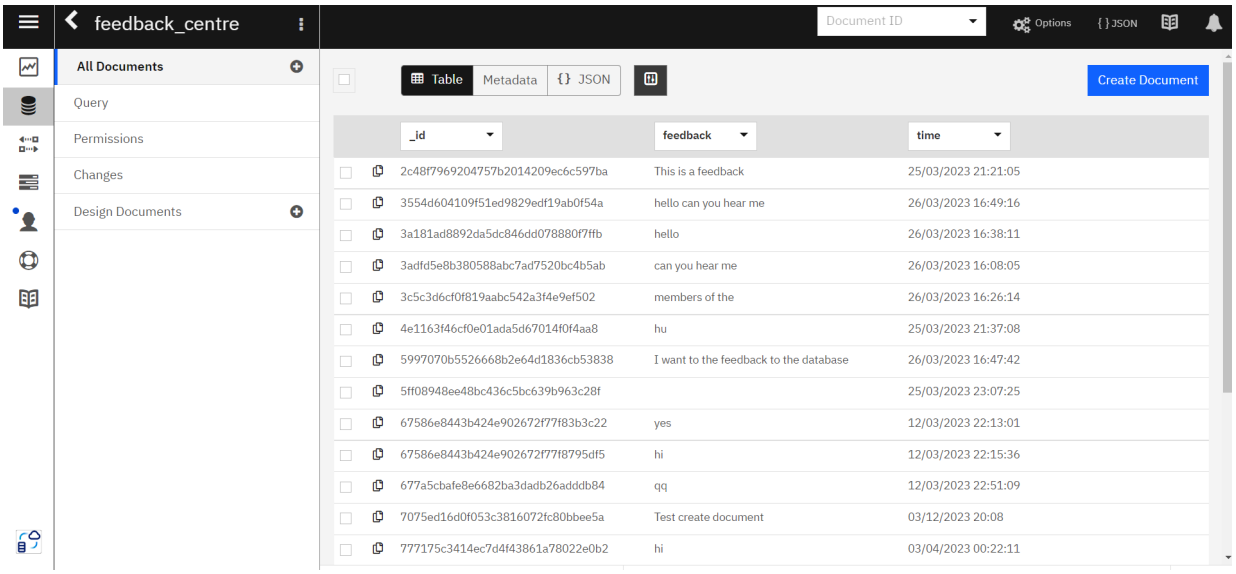


Figure8: The Feedback Centre Database

The database labelled as "feedback_centre" in the figure titled "The Feedback Centre Database" can be considered a flexible database rather than a traditional table. It currently comprises three attributes. What sets it different is that the schema is not predefined, and the number of columns is not fixed. This means that if a new data unit is added to "feedback_centre", it could consist of entirely different attributes. Thus, if later developers want to alter the database composition of "feedback_centre", they can send new data to the existing database without conflicts, and they do not need to modify the original database. Project maintainers can analyse the feedback received from users to make modifications and enhancements to the project accordingly.

Figure9: The JSON document

The figure titled "The JSON document" depicts how each unit of data stored in the database is represented by a JSON document. These documents are designed with a flexible and dynamic schema, allowing them to contain a varying number of fields and values that can be updated without a predefined schema. The database system automatically generates two fields, namely '_id' and '_rev'. The '_id' field serves as a unique identifier for a document within a database, and each document must have a distinct '_id' value, which functions as a primary key for indexing and referencing documents. The '_rev' field represents the revision ID of a document, which is used for conflict detection and synchronisation. All other fields in the document are user-customizable and can be changed flexibly.

3.5 Methods On Collecting Training Data

In order to enhance the performance of the chatbot, it is crucial for our team to gather a substantial amount of questions and answers to train the AI. To accomplish this goal, our team conducted two comprehensive surveys with different types of questions as shown in Figure “Survey Questions”. The surveys were designed to understand the needs and preferences of new students and to define the specific objectives that the chatbot should achieve.

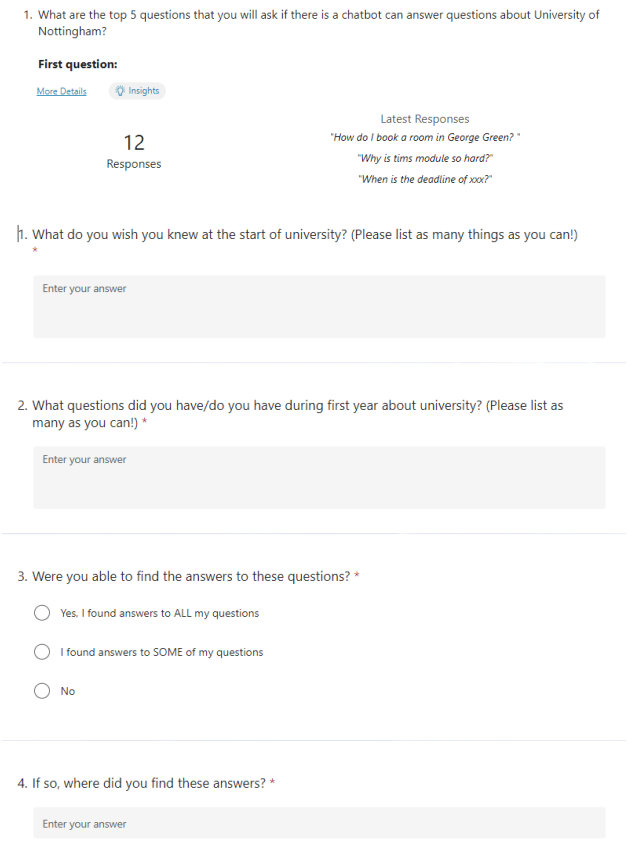


Figure10: Survey Questions

Regrettably, our team faced some challenges while promoting and disseminating the survey to new students. Initially, the team leader shared the survey link on the Year 1 Discord group, but received a meagre response. However, with the assistance of our supervisor, we managed to promote the questionnaire by sending an email to all Year 2 computer science students, encouraging them to participate in the survey. Despite the initial obstacles, we were able to collect a sufficient number of survey responses to inform the design of the chatbot's conversational rules and flow.

The results of our survey indicate that the chatbot should be designed to provide general information regarding the university, address frequently asked questions, and assist with tasks such as locating campus resources. Additionally, the chatbot could be utilised for academic and administrative purposes, such as checking grades or scheduling appointments, as well as providing social and wellness support.

Furthermore, we collected and filtered the responses from the aforementioned survey, and utilised the IBM Watson design system to organise them into conversational rules and flow to define the chatbot's functionalities.

In addition to collecting surveys from students, our team also sought help from university staff in relevant departments to obtain questions and answers regarding the university. We arranged a meeting with the Careers and Employability Office at our university to introduce our project and discuss potential collaboration opportunities. The office's staff expressed interest in our project and agreed to assist us as a stakeholder. They provided us with a list of frequently asked questions from students and official responses. This list proved invaluable in developing chatbot conversations related to careers services at the university.

1. What does work experience cover
Work experience can include part-time work, volunteering, internships during holidays and year-long placements. You can also get involved with employer projects. You can find out more details at Gaining experience (nottingham.ac.uk).
2. How can I get work experience?
Opportunities exclusive to Nottingham students can be found at What's on offer at Nottingham? - The University of Nottingham. You can also look across the UK for opportunities Where to look for vacancies in the UK - The University of Nottingham.

Figure11: Questions and Answers List regarding Careers

We utilised two methods to collect training questions and answers for the chatbot by conducting surveys to understand the needs of new students and gaining a list of frequently asked questions from university staff in relevant departments. Both methods proved to be helpful in developing our chatbot's conversational rules and flow.

3.6 Version Control System

Version control systems are software tools designed to facilitate the tracking of code modifications over time. Our team employs GitLab as our chosen version control system, as depicted in the "Version Control" figure.

We maintain two primary, permanent branches: master and develop. To safeguard the master branch, direct code pushes from team members are prohibited. After the 1.0 tag in our git repository, each point in the master branch represents a subsequent version of the project. The final version will also be released within the master branch.

All developers on the team commit their code to the develop branch. When a member is tasked with creating a new feature, they must switch to a new branch, named "feature-featureName," which originates from the develop branch, and push their code to this newly-created feature branch. Upon completion, the member must merge this temporary feature branch into develop. Once a series of functionalities have been implemented and a new version release is planned, we transition to a

release branch for testing purposes. Bug fixes must be executed within the release branch. Upon successful completion of tests, this branch is merged into the master branch and tagged with a new version name.

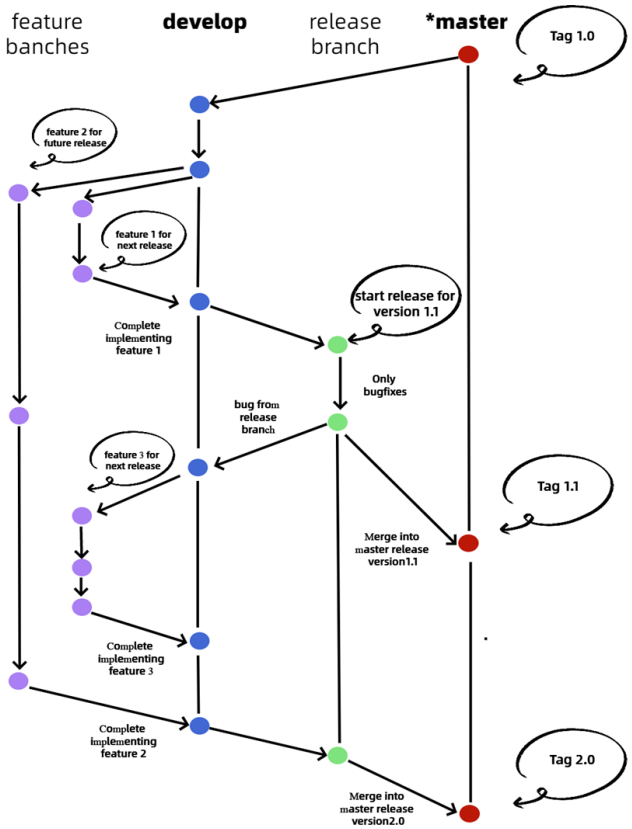


Figure 12 : Version Control

3.7 Evidence of testing

Testing was performed using both manual and automated testing techniques to ensure the correct functionality and performance of the application.

3.7.1 Functional Testing

Test Cases:

WelcomeFragment
a. Test navigation to ChangeAvatarFragment.
b. Test navigation to FeedbackFragment.
c. Test navigation to QRCodeFragment.
d. Test navigation to SettingFragment.
e. Test navigation to WebChatFragment.

ChangeAvatarFragment
a. Test avatar selection.

b. Test avatar change functionality.
c. Test navigation back to SettingFragment.

FeedbackFragment
a. Test submission of feedback.
b. Test input validation (e.g., checking for empty fields).
c. Test navigation back to SettingFragment.

QRCodeFragment
a. Test QR code generation.
b. Test saving QR code to the device's gallery.
c. Test navigation back to SettingFragment.

SettingFragment
a. Test loading previously saved avatar.
b. Test navigation to ChangeAvatarFragment.
c. Test navigation to FeedbackFragment.
d. Test navigation to QRCodeFragment.
e. Test navigation to WebChatFragment.

WebChatFragment
a. Test loading of the web chat interface.
b. Test interaction with the chatbot.
c. Test navigation back to WelcomeFragment.

Message
a. Test message creation.
b. Test message content and type.

MessageAdapter
a. Test message display.
b. Test correct handling of received and sent messages.

Result:
All test cases passed successfully, with no critical issues or defects found. The application's navigation and functionality are working as expected. Based on the successful completion of all test cases, the NottingBot Android application is considered stable and ready for release. Any minor issues or improvements can be addressed in future updates.

3.7.2 Dialog Testing

We randomly selected ten questions related to the University of Nottingham, encompassing topics such as academics, dining options, student services, and accommodations. We compared the duration required to obtain accurate answers for these queries using a chatbot with that of using the "Bing" search engine on the internet. The time elapsed from the beginning of the search to accessing the desired web page was measured. The average time required for the chatbot to provide responses to these ten questions was 16.27 seconds, whereas the search engine required 30.25 seconds. If students ask questions that are pre-defined in the chatbot system, approximately 46% of their time

can be saved, implying that using the chatbot to obtain information about the university is more efficient than searching for answers on the internet. This outcome may be due to the excessive number of possible options displayed on the internet search results for a specific question, many of which may lead students to incorrect answers.

3.8 Inline document

The Javadoc tool processes declarations and documentation comments in a set of Java source files, generating a collection of HTML pages that describe public and protected classes, nested classes (excluding anonymous inner classes), interfaces, constructors, methods, and fields (by default). This tool can be employed to create API (Application Programming Interface) documentation or implementation documentation for a group of source files.

Each team member was mandated to compose Javadoc comments for Java classes, fields, and methods in a standardised format, which enhances code readability and facilitates collaboration. Utilising the Javadoc tool, a series of HTML pages containing the declarations and documentation comments from a collection of Java source files is generated. This output can be used to produce API (Application Programming Interface) or inline documentation for a group of source code.

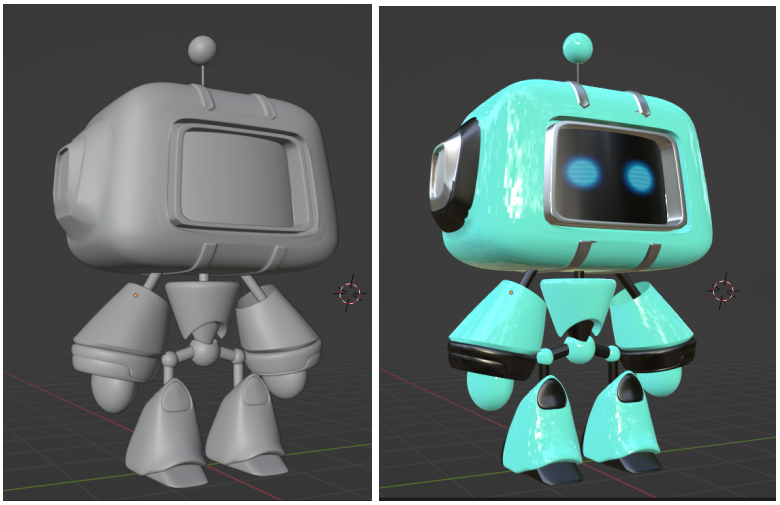
```java
/**
 * This class is the controller class of change_avatar.xml page and
 * it defines the logic of the buttons in this page
 *
 * @version 1.0
 * @author Junhan LIU
 */
public class ChangeAvatarFragment extends Fragment {
    /**
     * binding variable of change_avatar.xml which is
     * designed in MVVM model
     */
    private ChangeAvatarBinding binding;

    /**
     * Called to have the fragment instantiate its user interface view.
     *
     * @param inflater           The LayoutInflater object that can be used to inflate any views in the fragment
     * @param container          If non-null, this is the parent view that the fragment's UI should be attached to.
     *                           The fragment should not add the view itself,
     *                           but this can be used to generate the LayoutParams of the view.
     * @param savedInstanceState If non-null, this fragment is being re-constructed from a previous saved state as given here.
     * @return
     */
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                             @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        binding = ChangeAvatarBinding.inflate(inflater, container,  attachToParent: false);
        return binding.getRoot();
    }
```

Figure 13: Javadoc

3.9 AR Avatar

3.9.1 3D Model design



The 3D models of the AI chatbot were created using Blender, after first creating 2D illustrations to use as reference. Using these as a guide, these 3D models were sculpted and shaped using various primitive objects. Once the basic model was complete, textures, lighting and materials were used to bring the model to life.

3.9.2 AR Core

In this application, the 3D avatar representation is facilitated by the integration of ARCore technology and the Sceneform framework. ARCore, a software development kit (SDK) devised by Google, empowers the development of augmented reality (AR) applications for Android devices. By harnessing the camera, motion sensors, and other device hardware, ARCore determines the real-time position and orientation of the device, thus enabling virtual objects to be situated within the real world. Consequently, this generates an interactive and immersive AR experience for users. ARCore incorporates features such as motion tracking, environmental understanding, and light estimation, and is compatible with an extensive array of Android devices. The application employs ARCore version 1.36.0, the most recent version available.

Sceneform 1.17.1 is used to manage the positioning of AR avatars, as it is a 3D framework for Android that relies on ARCore to render 3D objects in real-world settings. Sceneform streamlines the creation of AR applications by offering a high-level API for 3D graphics rendering and user input handling. The incorporation of AR features in NottingBot enhances its appeal and engagement, particularly for new university students.

4. Project Management & Progress

4.1 Distribution of Work

4.1.1 Main principle

The allocation of tasks among different group members was based on a fair and rational principle that took into account the preferences and abilities of each group member. Whenever new tasks or issues emerged, the team leader would consult with the group members about their interests and availability and assign the tasks accordingly. For some specialised tasks, such as UI design, the team

leader would directly delegate them to a group member who had demonstrated proficiency and expertise in that area.

Another principle that guided our task allocation was splitting into subgroups to finish a large task. For some complex and challenging tasks that required more than one person's effort, such as writing the final report, our group would split into smaller subgroups and each subgroup would take responsibility for a specific section or aspect of the task. This method enhanced the engagement and participation of every group member and increased the efficiency and quality of accomplishing large-scale tasks.

These were the main methods that our team employed to divide the work among ourselves.

4.1.2 Use of Trello

We employed Trello boards as a tool to manage and monitor the assignment tasks. Our team held a regular sprint meeting at the end of each month, which was mandatory for all group members to attend. In this meeting, each group member provided a summary of their performance and progress for that month by answering three questions: 1. What did I do well or poorly for this project this month? 2. How can I improve my performance and contribution next month? 3. What suggestions do I have to enhance our team's processes and outcomes? Their actual achievements and challenges for that month were documented in the meeting minutes. Then, we engaged in a discussion about some critical decisions and strategies on how to improve our work quality and efficiency next month. After reaching a consensus, we formulated an action plan and listed all the tasks for the next month in a Kanban board in Trello, as illustrated in the figure.

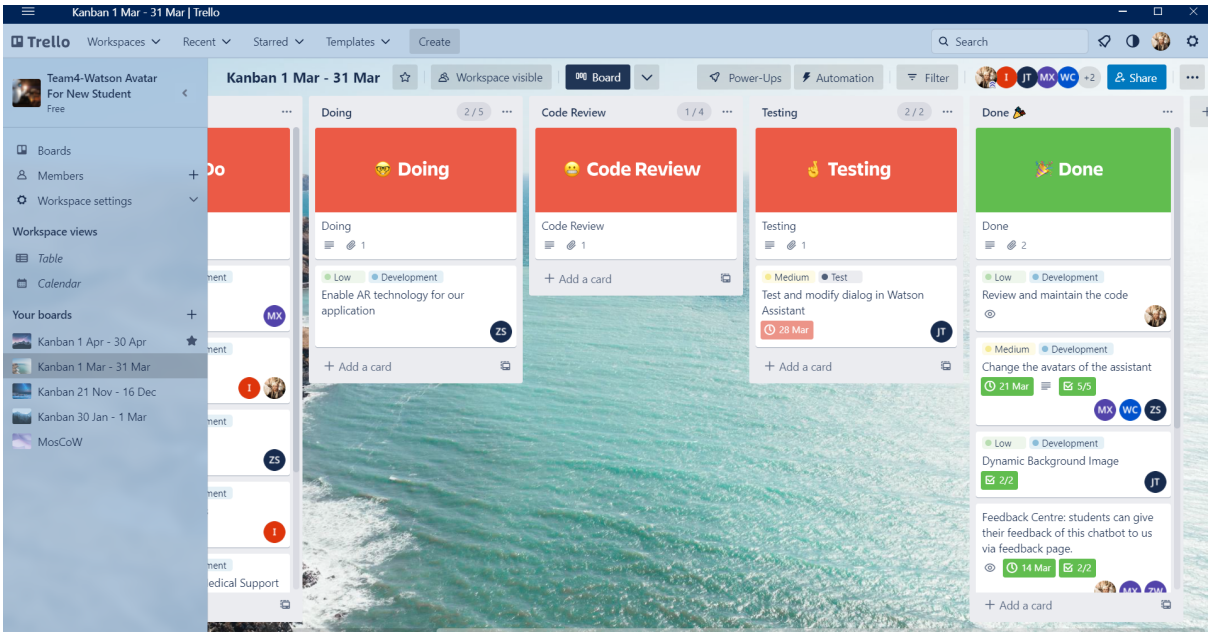


Figure 14: All tasks in a Month

Regarding a specific task, the figure "Task Card" shows the detailed information that each task contains. It specifies the group members who are assigned to complete the task and the labels that indicate its priority and category. Moreover, it displays the deadline for the task and the subtasks that outline the steps to accomplish it. In this way, every group member can have a clear and

comprehensive understanding of their roles and responsibilities and the procedures and expectations for the task.

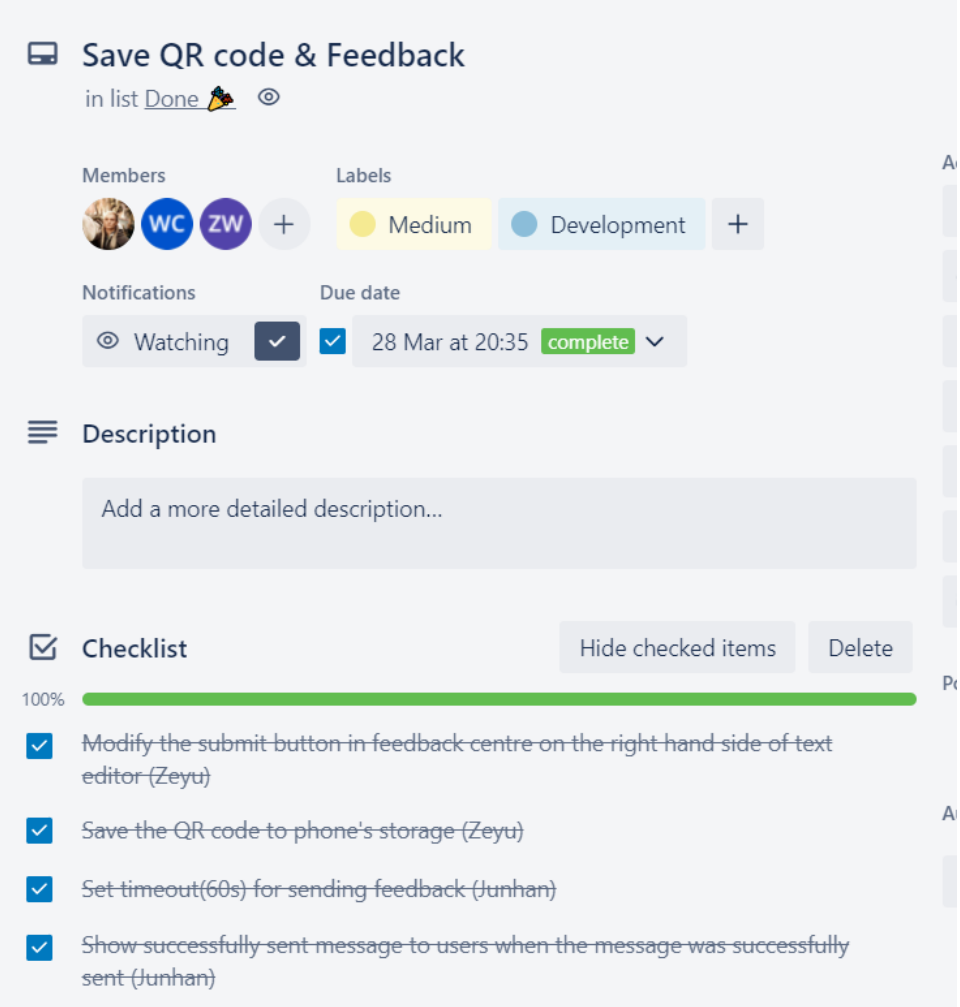


Figure 15: Task Card

Every task card is required to have two labels, one indicating the priority and the other indicating the category of the task. The figure "Labels" displays all the labels that our team has created and used. We have established four levels of urgency for the tasks, ranging from low to critical. Tasks with higher priority are given precedence over tasks with lower priority, which enables us to manage the order and timing of the tasks in a rational and efficient manner.

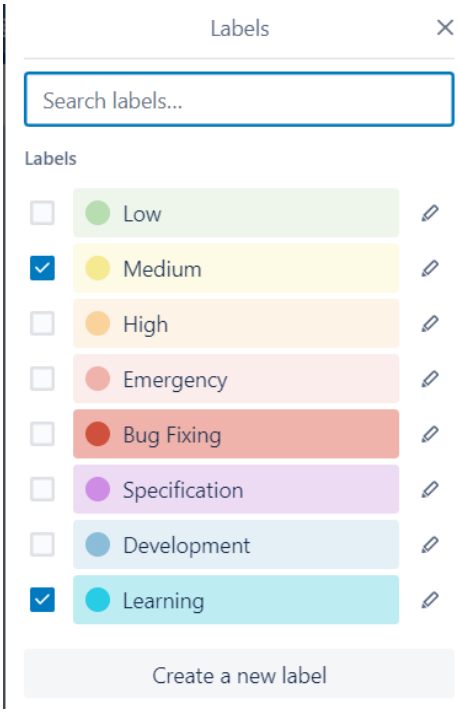


Figure 16 : Labels

4.1.3 Details of record distributions and assign a new task

At the start of each regular weekly meeting, every group member would report on their activities and achievements in the previous week, and these contributions would be documented in the meeting minutes. At the conclusion of the weekly meeting, each group member would outline their plans and goals for the following week to the rest of the group, which would enhance the awareness and understanding of our own and others' tasks. This method would enable us to identify and resolve any ambiguities or uncertainties in the plan before implementing it. Consequently, many potential misunderstandings and errors in execution would be avoided or minimised.

4.2 Agile Software Development Process using Kanban

4.2.1 Introduction of Kanban Framework

We adopted the Kanban framework as a method to facilitate agile software development. All tasks' progress was made visible and accessible, enabling group members to monitor and track the status of every piece of work at any given time. The Kanban method is an approach for us to adapt and improve our processes gradually and incrementally. It enables us to visualise work processes, limit work-in-progress, focus our team's attention and foster a collaborative work environment. All of these benefits can optimise our work quality and efficiency and help us attain our goals and outcomes more quickly. Therefore we chose Kanban as our main project management method.

4.2.2 Fast delivery and flexibility

One of the significant benefits of employing the Kanban agile project management process is its ability to facilitate faster delivery compared to traditional management methodologies such as the "waterfall" process. In Kanban, once a task is completed, it can immediately move to the next stage or be released, minimising the time spent on handoffs or waiting for other tasks to finish. As a result,

the duration of each iteration of our product in Kanban is shortened, enabling us to continuously showcase the latest updates to our supervisors and stakeholders on a regular basis. Timely feedback from these stakeholders helps to prevent the development of features that may deviate from their expectations, thereby enhancing our work efficiency. In contrast, in waterfall development, tasks are completed in a sequential manner, and the next phase cannot commence until the previous phase is completed, leading to longer lead times and delayed delivery. This may result in spending time on tasks that are not aligned with the overall project direction.

Kanban allows for flexibility and adaptability in our project management, as it allows our team to respond to changing requirements and priorities in real-time. With Kanban, our team can quickly adjust the workflow, prioritise tasks, and allocate resources as needed, based on the current status of the project. For example, when we attempted to implement more emergence requirements for this project, we only created the essential task cards related to those requirements. This helped our team focus on completing the most important tasks first and avoid spending time on irrelevant work. In contrast, waterfall development follows a rigid, sequential process, making it less adaptable to changes in requirements or priorities, and can result in delays and rework.

4.3 Contingency measures helped to manage unexpected circumstances

4.3.1 Unexpected circumstances that we encountered

During the beginning of the second semester, our team encountered an unexpected event when all members' IBM cloud accounts remained inactive for an extended period during the winter vacation, resulting in automatic cancellation of our accounts. As a consequence, all chatbot conversations, except for those that were previously backed up appropriately, were lost. In response to this issue, we contacted our industrial supervisor, who provided us with two potential approaches for resolution. The first approach involved creating a support request on IBM Cloud using a newly registered account, while the second approach involved sending an email to the corresponding service team. Our team leader provided a detailed description of the issue and a list of cancelled accounts to these IBM support teams. However, both support teams responded that our accounts could not be reactivated at this time and recommended that we create a new account instead. Despite our efforts to retrieve the lost data saved in the IBM Cloud accounts, we were unable to recover it, which posed significant challenges to our project. Ultimately, we made the decision to restart on a new account during a group meeting. Fortunately, due to the dialog backup measures that were in place to mitigate data loss, we were able to quickly rebuild a substantial number of conversations within a two-week timeframe, surpassing the progress made in the previous semester.

Another challenge we faced was reaching the maximum number of "intents" in the Watson Assistant due to the limitations of our low-level version account with restricted functionalities. "Intents" refer to the purposes or goals expressed in a customer's input, such as asking a question. The Watson Assistant service relies on accurately recognizing the intent expressed in a customer's input to select the appropriate dialog flow for generating a response. However, due to this issue, we were unable to add any new conversations to our chatbot, significantly reducing its usability.To address this issue, we maintained communication with IBM staff and eventually, they offered us a higher version account that allows us to create more dialogs and overcome the limitation of intents in our chatbot.

The detailed methods and strategies we employed to handle unexpected issues will be further elaborated in the following paragraphs.

4.3.2 Dialog backup and recovery system

Our project aims to create conversational scenarios that provide some basic information about the university to help new students adjust to their new environment quickly. We used IBM Cloud's Watson Assistant to generate and store all the conversations that we used to train the AI chatbot. However, this approach has some risks because our IBM cloud account might be cancelled for some reasons such as inactivity, which would result in the loss of our conversational data. To prevent this potential problem, we developed two backup systems for our dialogues.

The first backup system involved saving the content of our conversations in word documents in our group Google Drive repository. We required every member to write their dialogues in word documents first and then add them to Watson Assistant accordingly. This way, we can ensure that our conversational data is stored in two locations, and if any unforeseen events such as data loss in the Watson server occur, we can quickly restore the data from one backup to another. This approach offers several advantages, such as facilitating the organisation and management of a substantial volume of dialogue information in a structured manner based on its respective topics. By segregating the data into distinct documents and folders for categorization, it becomes possible to expedite the process of locating dialogue segments that require editing and identifying any gaps in our topics. However, this system also presents a limitation. Due to the disparity between the dialogue formats in Word and Watson Assistant, it takes more time than adding new conversations directly in IBM Cloud. Consequently, some team members tended to bypass this step, resulting in the loss of not adequately backed up data that was created during the previous semester. We used this method to address the first problem that was discussed in the previous section.

Another backup methodology was discovered during the second semester. The Watson Assistant platform offers a feature that enables the export of all dialog data in a chatbot to a JSON file. This JSON file can subsequently be uploaded to the Watson Assistant server, thereby replacing all existing conversations with the uploaded data. The team leader adopted a practice of regularly downloading and archiving JSON files containing the chatbot's dialog data on a weekly basis, particularly when updates or modifications were made. This method proved to be significantly faster than the first system, making it particularly suitable for transferring all data from one account to another. This approach was successfully utilised during the resolution of Crisis 2, as mentioned in the previous context, when our team reached the maximum limit of conversation intents and IBM staff provided us with a higher version of Watson Assistant with enhanced functionalities. Our team was able to swiftly transfer all previous data to the new account using this method. However, it is worth noting that this approach has limited flexibility, as it does not allow for the manipulation or modification of individual conversations in the backup data. Instead, the entire dialog defined in the account is replaced by the newly uploaded file, which may result in potential data loss.

4.3.3 Emergency response meetings

To proactively address emergencies that could potentially have negative impacts, our team implemented a collaborative approach. We held emergency meetings where team members would come together to discuss and devise strategies for responding to these unexpected events. These meetings typically covered several key topics, including updates on the situation, assessment of risks and impacts, identification of immediate actions to be taken, allocation of resources, and communication strategies. During these meetings, agreements and decisions were made collectively to address the issue at hand.

This proactive approach was implemented for both unexpected events mentioned earlier, and it proved to be effective in facilitating timely decision-making and action plan development. For instance, when our accounts were unexpectedly cancelled, convening these meetings and engaging in comprehensive discussions allowed our team to promptly assess the reasons behind the cancellations, identify potential solutions for account reactivation, and develop strategies to seek help from relevant parties. By collectively evaluating the available options, we formulated a comprehensive action plan to manage the emergency and mitigate its impact on our project.

4.3.4 Crisis communication plans

A communication list was created in case potential emergencies occur, which included various individuals who could potentially provide assistance. This list encompassed our academic supervisor, industrial sponsors, and even other project groups who were also working on IBM supervised projects. In the case of the first crisis mentioned in the previous context, we reached out to all the individuals on the communication list, and the group leader even contacted IBM client service's number in the US for assistance. Information and advice obtained from these sources were carefully collected, and immediate actions were taken based on their recommendations.

4.3.5 Dialog testing methods

In our team, some members were assigned the responsibility of reviewing the current flow of conversations in the chatbot. They assumed the role of potential users, actively engaging with the chatbot to identify any logical defects or uncomfortable conversations and evaluated the chatbot's responses to ensure that they were accurate, relevant, and aligned with the intended goals of the conversations. These issues were then reported to the responsible team members for further action. The corresponding team members would then modify the dialogues based on the feedback provided by the reviewers.

5. Reflection

5.1 Achievements of the Project

The University of Nottingham embarked on a groundbreaking Watson chatbot project to aid new students in adapting and navigating their academic lives with ease. The project's success was measured based on various factors, with the primary objective being to assist new students. One of the most notable accomplishments of the project was the implementation of IBM Watson chatbot technology to create an Android application that enables students to conveniently access the chatbot on their mobile devices.

The project requirement was to create a web application incorporating a Watson chatbot that students can access through their mobile devices by scanning strategically placed QR codes throughout the university. The chatbot provides essential information that students wish they had known upon arriving at the university. The content of the chatbot is designed to cater specifically to new students in a particular course or every first-year student at the University of Nottingham, which is our primary goal.

Overall, all the must-have and should-have requirements were achieved using IBM Watson chatbot technology and Java Android Development. Our project created a chatbot that employs natural language processing and machine learning algorithms to provide accurate and relevant responses to student inquiries with practical features like QR code sharing, avatar setting, and speech-to-text

conversion integrated. IBM Watson technology empowered the chatbot to learn and improve its performance as it interacted with more students, providing the best possible support and information.

One of the most significant accomplishments of our project is the ability of our chatbots to offer accurate and dependable responses to a broad range of student inquiries. These inquiries can vary from questions about student life, academics, health and wellness, career advice, and transportation. Additionally, our chatbot has the capability to collect and analyse user data, allowing us to identify areas where students may need further support or guidance. This information will be invaluable in making future improvements to the chatbot's functionality and content, ensuring that it remains relevant and helpful to students throughout their university experience.

The Watson chatbot project for the University of Nottingham is poised to have a transformative impact on the student experience by offering precise and convenient assistance to new students while demonstrating the University's dedication to leveraging cutting-edge technology to augment the student experience. Additionally, the project will ease the burden on university staff by addressing common queries and providing instant answers to frequently asked questions, aiding students who face challenges in accessing conventional support channels. This includes individuals who are reticent or prefer seeking help via online platforms.

5.2 Motivation Behind the Technical Approaches

The technical approaches used in this project were chosen based on the team's experience and the requirements of the project. For instance, the team chose to use Android Studio with Java programming language because of the team's familiarity with Java from previous university projects, making it easier to develop and implement the project. The Android Studio's 'Design' view also made it easier to create layouts and templates that are used in the code to create content dynamically, and implement the UI prototype created beforehand.

Another technical approach used was implementing the IBM Watson chatbot development system for the AI chatbot, which was built on deep learning, machine learning, and natural language processing. The chatbot's purpose was to answer frequently asked questions, provide directions, and recommend points of interest in different areas. This allowed the program to provide a natural and conversational interface, which could improve engagement and enhance the user experience.

In addition to the chatbot, the team also considered accessibility features, such as text-to-speech and speech-to-text functionality. This feature was made available through Watson and was essential for users who may have difficulty reading text on a screen.

The team also used the MoSCoW method to prioritise requirements and specifications, which helped them to focus on critical features and deliverables. Several Trello boards were used to track and manage the project's progress, making it easier to identify and resolve any issues that arose during development.
Finally, the team used OpenAR and Blender to create and implement 3D models for the AR section. These tools provided the team with the necessary resources to create a visually appealing and interactive interface that would capture the users' attention and provide an enjoyable experience.

In conclusion, the team's motivation for using these technical approaches was to ensure the project's success, accessibility, and ease of use. By using familiar tools, prioritising requirements, and considering accessibility features, the team was able to deliver a high-quality project that met the users' needs and expectations.

6. Future Direction

6.1 Possible Future Direction

Here are some possible future directions for extending the content of our project.

1. Campus Navigation Enhancement: The current app offers campus maps and other location images, which could be improved by incorporating an interactive map feature. This would assist students in navigating the campus, locating study halls, student services, and other essential facilities more effectively. Interactive maps would provide clearer visuals for students seeking specific locations.

2. Peer-to-Peer Communication Platform: Establishing a platform for students to connect with their peers, exchange experiences, and offer support would foster a sense of community among them.

3. Feedback Analysis Implementation: While we have created a feedback centre for users to share their experiences and suggestions, it is essential to conduct regular user feedback analysis. This would help us identify trends, common issues, and areas for improvement, enabling data-driven decisions to enhance the app's functionality.

4. Event Calendar Integration: To keep students informed and engaged, integrating a calendar featuring upcoming university events, test information, club meetings, and social gatherings would be a valuable addition.

5. iOS System Integration: This application can be extended to an iOS version thus enabling access to a broader audience, including individuals who prefer iOS devices. The process of developing an iOS version requires the adaptation of the existing Android codebase to conform to iOS platform and design guidelines, while taking into account the exclusive features and functionalities of iOS devices.

User Manual

1. Introduction

This is the user manual of the chatbot application for Nottingham new students developed by GRP Team-4. Expected that this manual could help with our application use.

This is a web-based application chatbot, which will be accessible to students via their mobile devices. The chatbot will provide useful information to cater to the needs of first-year students.

This user manual is prepared for non-technical students at the University of Nottingham who intend to use the chatbot to get information about university.

2. Getting started

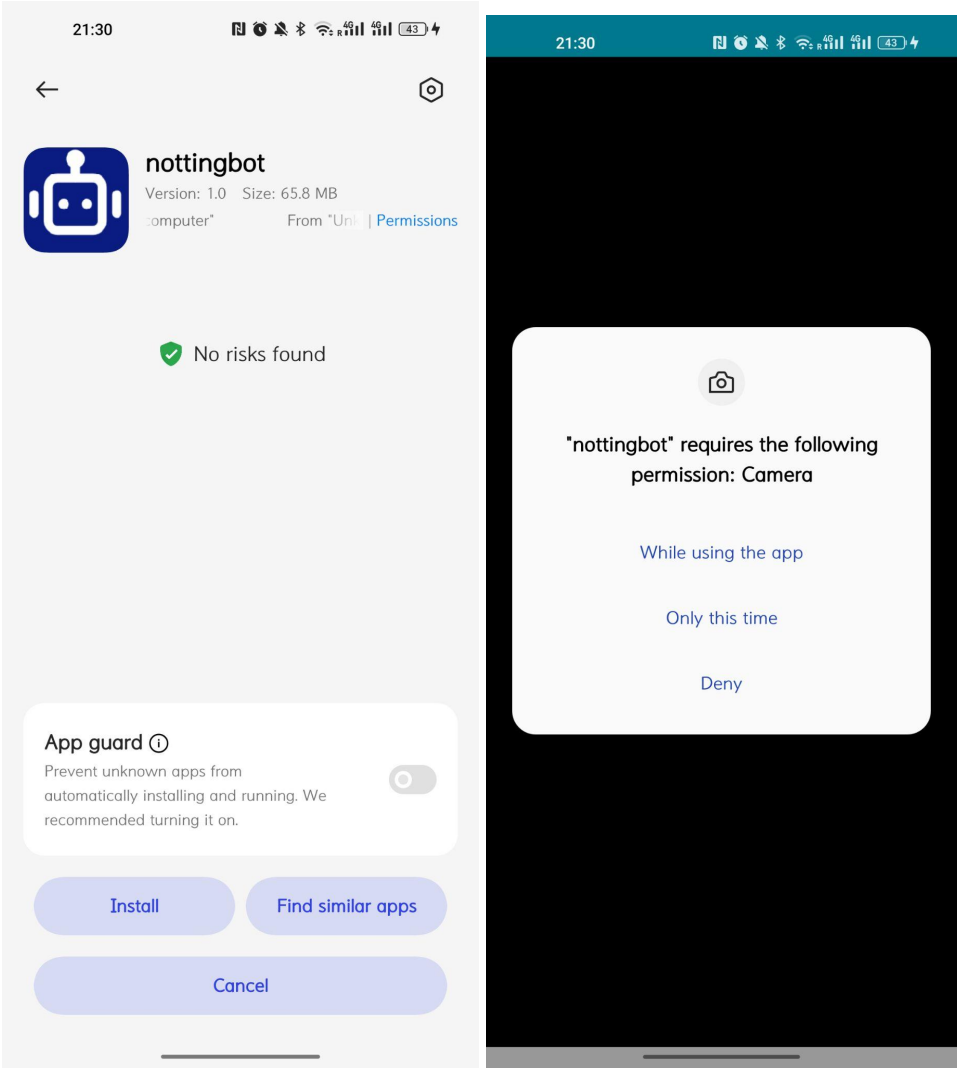


Figure 1. Install the Application

a. Download and install the “nottingbot” on the local device.
b. Give the app the necessary permissions.

c. Begin using the application.

3. Features and functionalities

This section describes the features and general functionality of the software, how to use chatbots and customise your settings.

3.1 Overview

The web application offers a user-friendly platform for new students, integrating a Watson chatbot for quick access to essential information. You may quickly start the chat with chatbot by clicking the 'Get Started' button on the welcome page. There are 4 main functionalities: chat with chatbot,change avatars, leave feedback and share QR code. Below is a detailed description of how the functions mentioned above are used.

3.2 Start Page

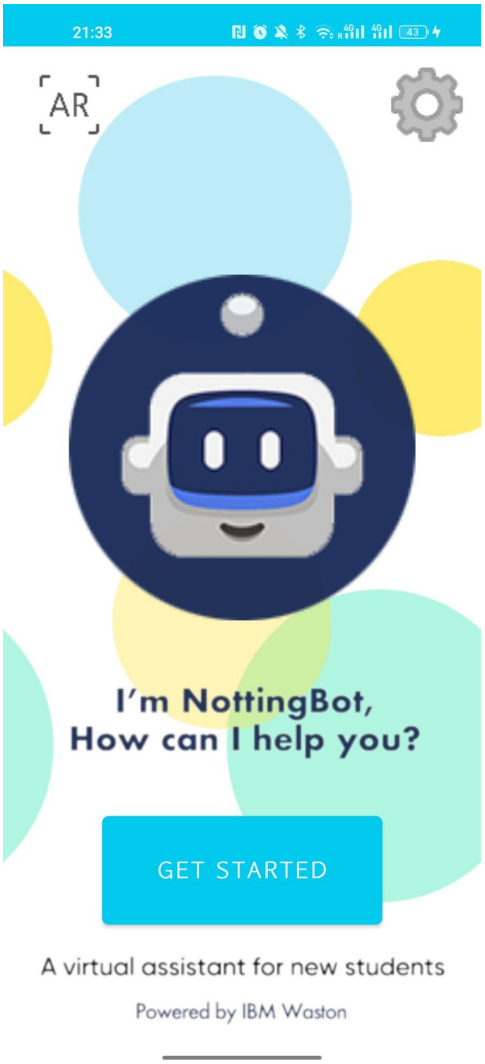


Figure 2. Start Page

As shown in Figure 1, upon opening the app, the start page is displayed.
a. Click the 'Get Started' button to access the chat page and start using the chatbot.
b. Click the 'Settings' icon button at the top right corner to access the settings page and customise your app experience.

3.3 Questions-Answering

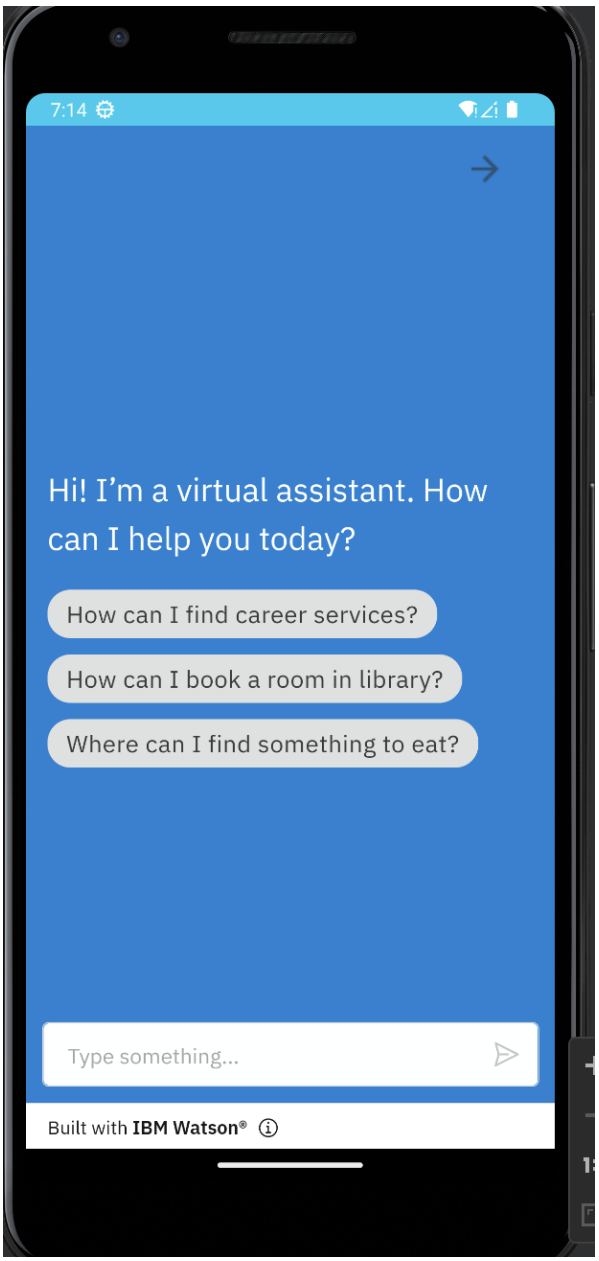


Figure 3. Presetted Questions Page

a. Click the 'Get Started' button on the start page to initiate a chat session with the chatbot.
b. Type your questions or concerns into the chat window and press the trigonal arrow.
c. The chatbot will respond with relevant information based on your query.
d. Also, you can click the presetted questions on the screen and the chatbot will automatically provide the answer.

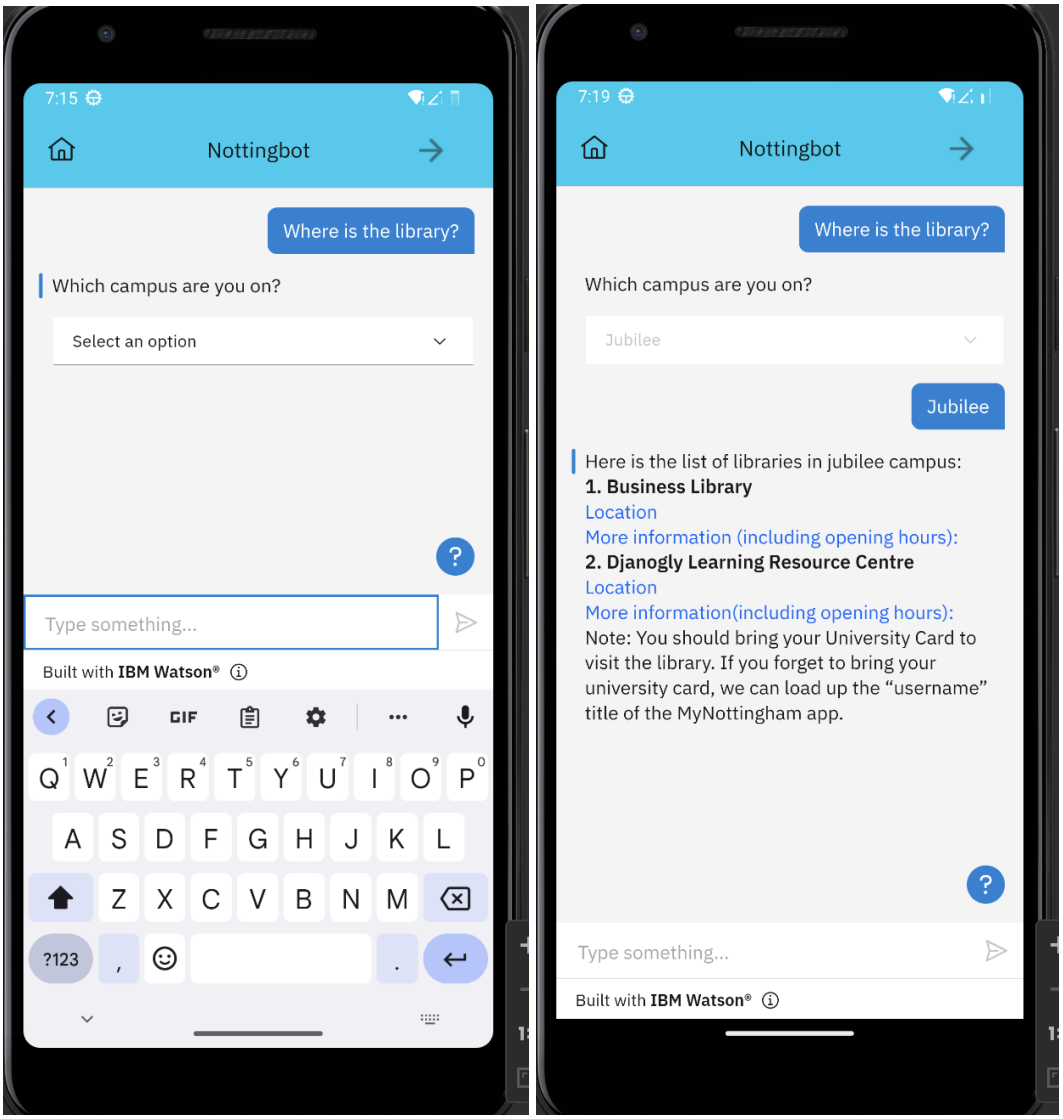


Figure 4. Questions Answering

e. Click the 'house shape' icon at top left corner go to the start page
f. Click the arrow at top right corner go to the Figure 2.Presetted Questions Page

3.4 Setting Menu

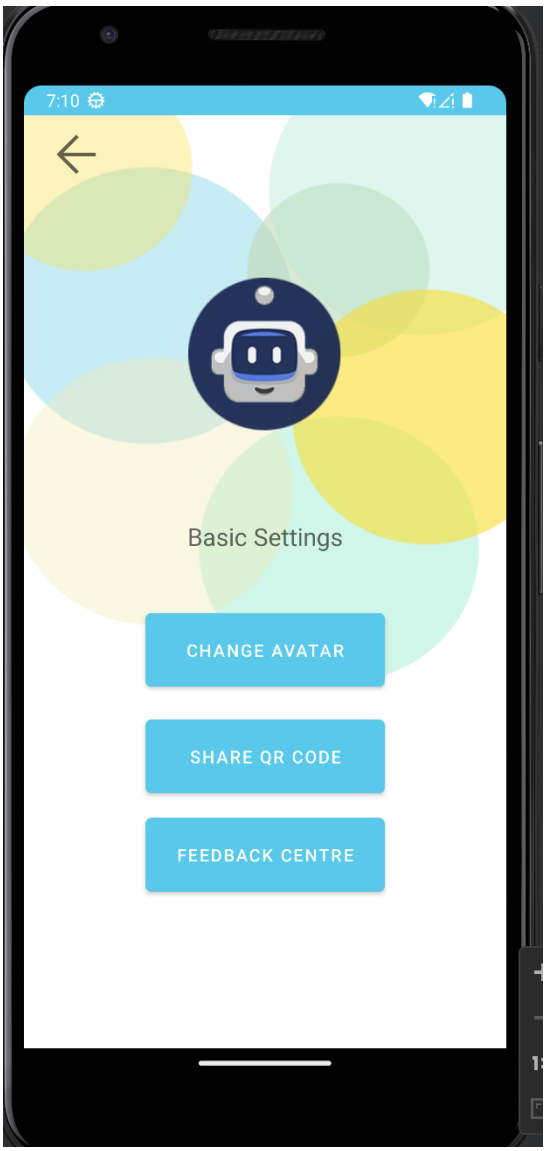


Figure 5. Setting Page

a. On the settings page, you can access three functionalities: 'Change avatar', 'Share QR code', and 'Feedback centre'.

3.5 QR Code Sharing

Figure 6. Share QR code

a. Click the 'Share the QR code' button to save the code to your device.
b. Share the code with other students to enable them to download and use the app.

3.6 Sending Feedback

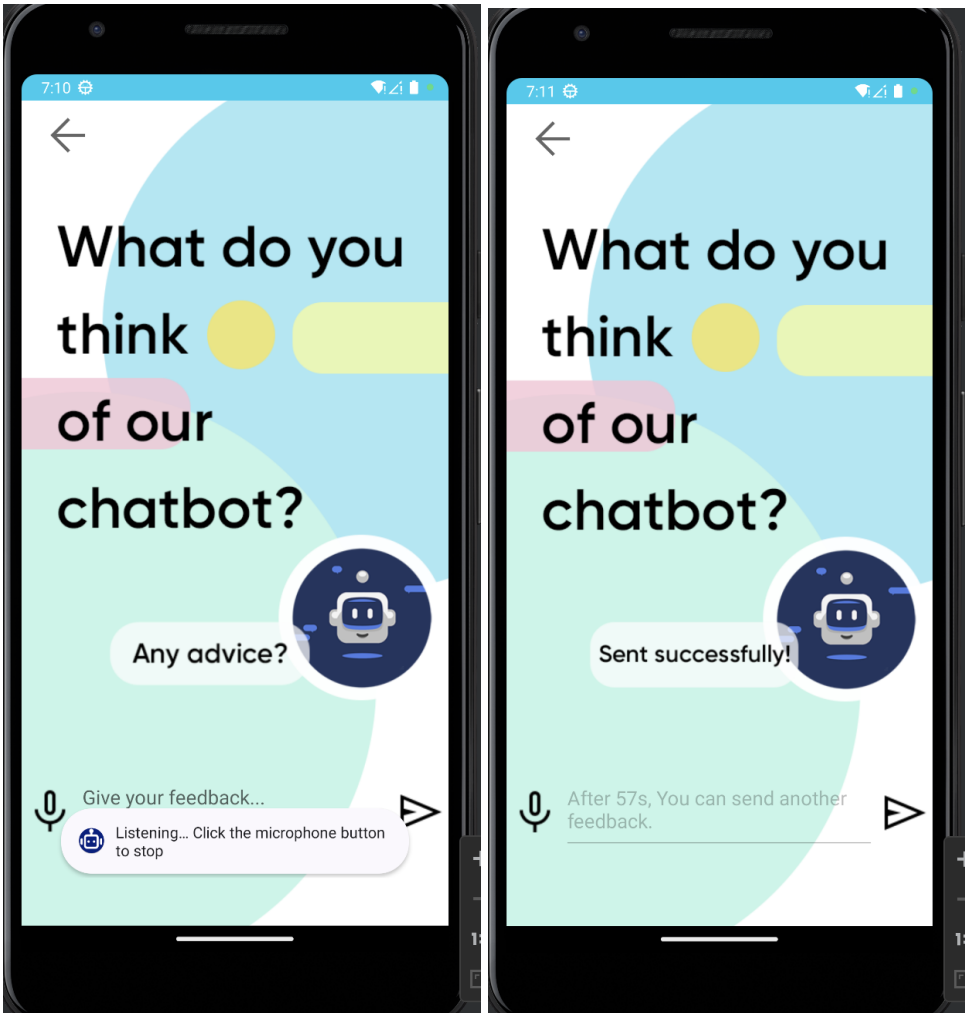


Figure 7. Send feedback

a. Click the 'Give feedback' button to access the feedback form.
b. Fill in and submit the form to provide your insights and suggestions for improving the app.
c. You can also use the speak to text functionalities by clicking the microphone icon

3.7 Change Avatar

Figure 8. Change avatar

a. Select the 'Change your avatar' option at the setting page.
b. Choose from the available avatars by clicking the left and right arrows
c. Click on the ‘Save’ button to apply your preferred avatar.

3.8 AR Avatar

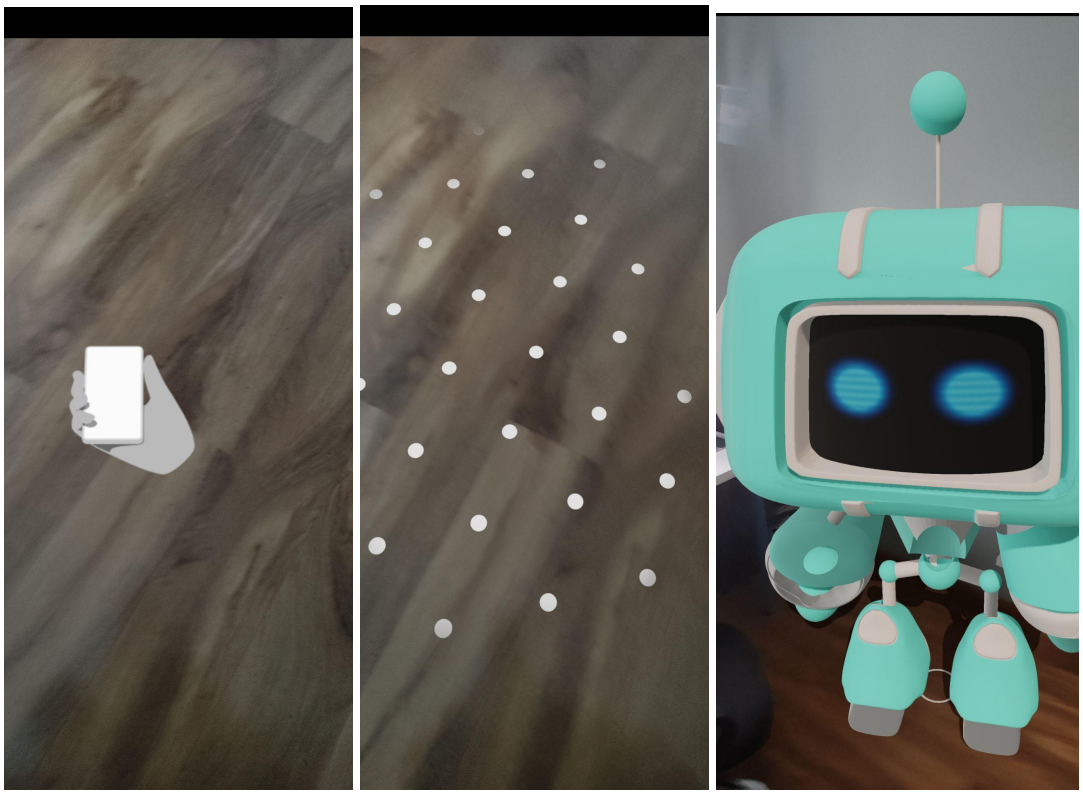


Figure 9. AR Avatar

a. Wave the phone to find the flat surface.
b. The white scatter represents the plane where the avatar can be placed。
c. Click the white scatter to place the 3D avatar

4. Frequently asked questions

1. . How do I access the chatbot?

Use the camera on your mobile device or a QR code scanning program to identify a QR code distributed around the campus and scan it to access the chatbot. As soon as the web application has loaded, you can start chatting with the chatbot.

2. What kind of information can the chatbot provide?

The chatbot may provide crucial details about the University of Nottingham, such as specifics about programs, campus amenities, activities, and support services for students. Its content can be modified to meet the particular requirements of certain courses or made accessible to all first-year students.

3. How can I provide feedback on the chatbot?

Go to the settings page and select the "Give feedback" button to submit feedback. To share your feedback with the development team, fill out the feedback form with any criticisms or ideas you have for enhancing the program.

4. Can I share the chatbot with other students?

Yes, you may let your friends know about the chatbot by showing them the QR code. To accomplish this, navigate to the settings page, select "Share the QR code," save the displayed QR code to your smartphone, and then distribute it to your other students.

5. Will the chatbot improve over time?

Machine learning techniques are used by the Watson chatbot to progressively enhance its comprehension and response capabilities. Its effectiveness will increase over time as it engages with more students and develops a better grasp of their needs and issues.

6. Who can I contact for additional support?

For further assistance, you may contact the University of Nottingham IT Helpdesk or visit the university's website for more information.

Software Manual

1. Introduction

1.1 Problem Description

The project aims to develop a web-based application featuring a Watson chatbot to assist first-year students at the University of Nottingham in their transition to university life. This chatbot will be accessible through mobile devices by scanning QR codes strategically placed around the campus. The chatbot will provide useful information and guidance, specifically tailored to cater to the needs of new students, facilitating a smoother and quicker adaptation to the university environment.

1.2 Goal of the Project

The primary goal of this project is to create an efficient, expedient, and personalised AI solution to help alleviate the challenges faced by freshmen during their initial year at the university. By utilising IBM Watson's advanced natural language processing and machine learning capabilities, the chatbot aims to enhance the student experience while reducing the burden on university staff. Ultimately, the project seeks to foster a more positive and supportive environment for incoming students, promoting their academic success and overall well-being.

2. Definitions of project specific terminology

2.1 Watson Assistant

Watson Assistant is an artificial intelligence platform that enables the development and deployment of conversational interfaces into diverse applications, websites, and messaging platforms. The system uses natural language processing and machine learning techniques to comprehend the user's intention and provide precise responses. It offers customisation and training to accomplish a broad range of tasks, including the resolution of commonly asked questions and the automation of intricate business processes. Furthermore, Watson Assistant is capable of integrating with other IBM Watson services such as language translation and speech-to-text, to deliver a more seamless user experience. For this project, all the conversations that respond to new students' potential queries are constructed utilising Watson Assistant.

2.2 Intents

Intents are defined as the goals or purposes that are conveyed in a customer's input, such as a request for information. They are used to comprehend and classify user inputs so that the chatbot can offer suitable responses. Through recognizing the intent conveyed in a customer's input, the Watson Assistant service can select the appropriate dialog flow to generate a response. To create an intent, developers must assemble a group of similar questions with the same meaning, allowing the chatbot to perform expected actions in response to such types of questions.

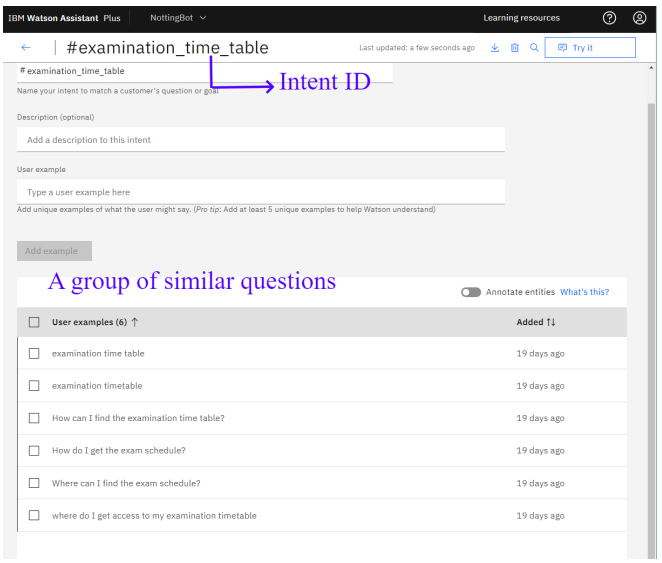


Figure1: Intents

2.3 Dialog Tree

The dialog tree is a hierarchical structure that models the conversation flow in a chatbot. It consists of multiple dialog nodes organised into branches, which represent different paths the conversation can take based on user inputs and context.

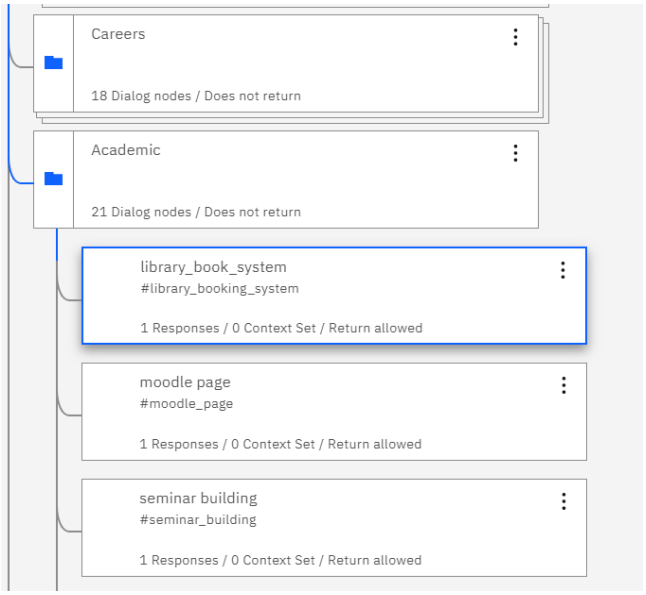


Figure2: Dialog Tree

2.4 Dialog Node

A dialog node is an individual element within the dialog tree that defines a specific response or action. Each node has a trigger condition, which is usually based on the intent or entity recognized in the user input. When the trigger condition is met, the node is executed, and the chatbot provides the response or action associated with that node. Dialog nodes can have the following components:

- Trigger condition: Determines when a dialog node should be executed based on user input. It can be intents, entities, or context variables. This condition acts as a decision point in the conversation flow, allowing the chatbot to respond

appropriately to the user's input. For instance, a node could be triggered when the intent #greeting is recognised, or when a specific entity value is detected.

- Response: Defines the message or messages that the chatbot should send to the user when the node is executed. These responses can be plain text or include variables, allowing for dynamic and personalised conversations. Additionally, response variations can be utilised to add variety to the conversation and keep the user engaged.

- Actions: Specifies any actions the chatbot should perform when the node is executed. These actions can range from setting context variables and making API calls to jumping to another node in the dialog tree.

- Child nodes: A dialog node can have one or more child nodes, creating branches within the dialog tree. These child nodes can be executed based on additional conditions or user inputs, allowing for more complex and dynamic conversation flows. By utilising child nodes, chatbot developers can build sophisticated conversation trees that can handle a wide range of user inputs and intents.

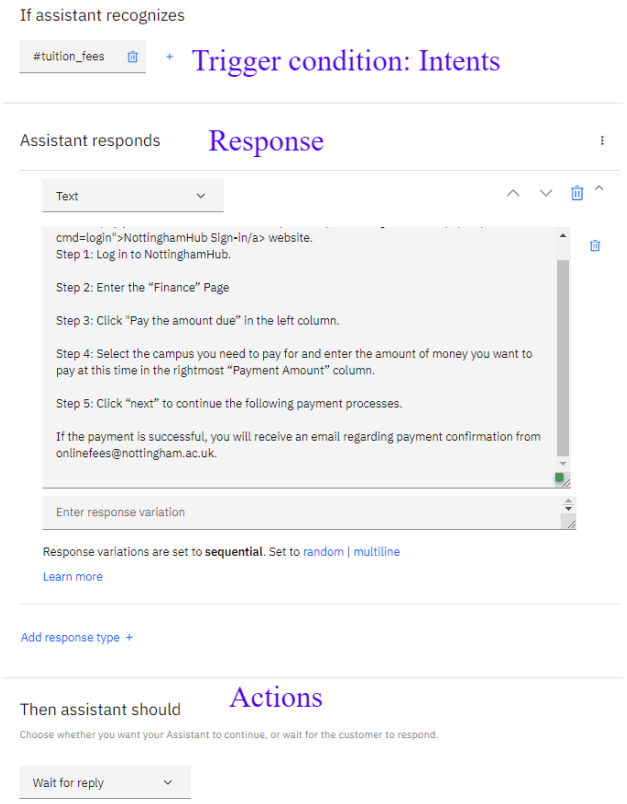


Figure3: Dialog Node

3. Project Diagrams

3.1 Use Case Diagram

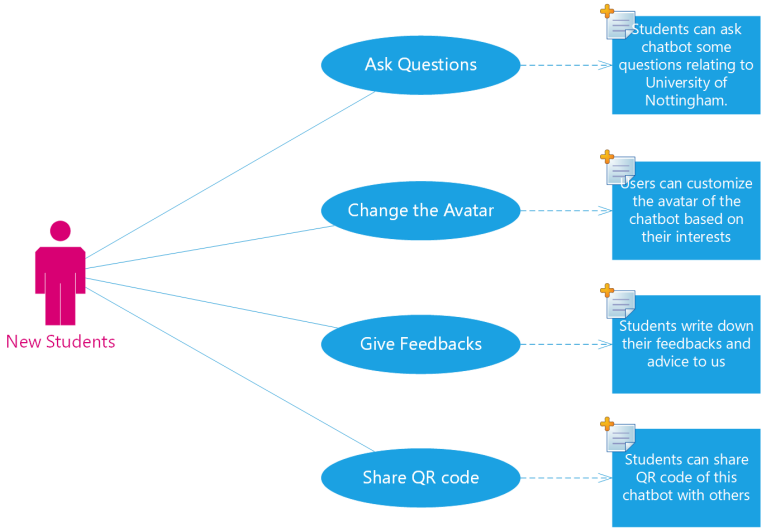


Figure4: Use Case

A use case diagram was developed to emphasise the user experience and delineate how the main group of users interact with the product. Through presenting a succinct and clear summary of the system's behaviour, a use case diagram facilitates efficient communication and collaboration among project participants.

3.2 Sequence Diagrams

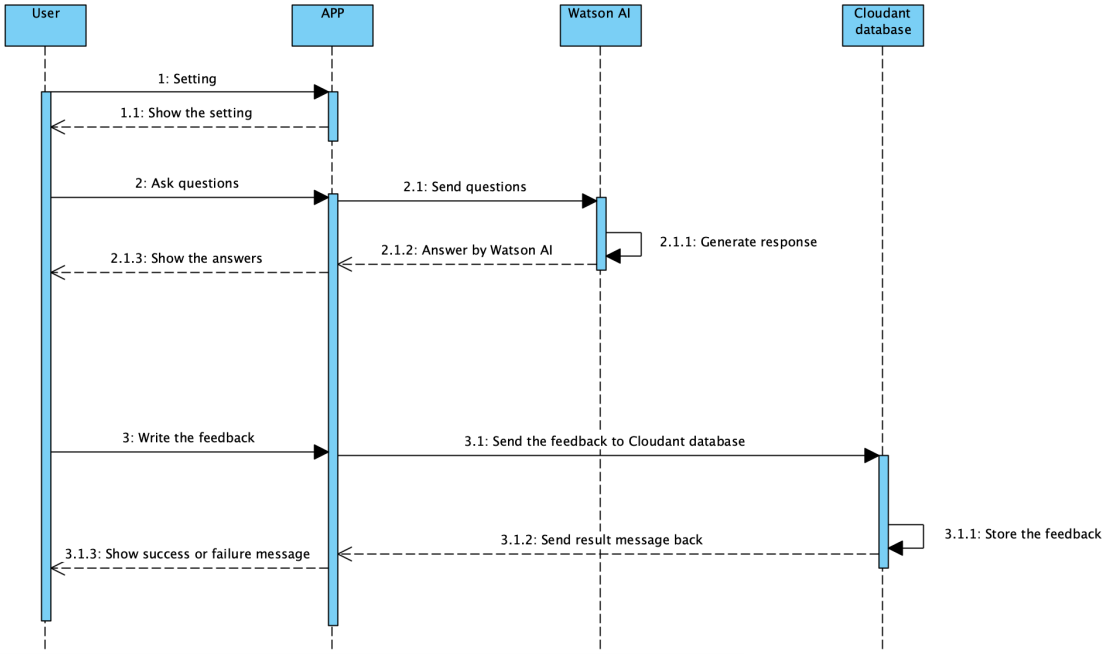


Figure5: Sequence Diagram

This is a sequence diagram illustrating the core functionality of our project. First, users configure their preferences within the app, which then displays the outcome of these settings. Users can pose

questions in the app's dialogue box, which are then transmitted to Watson AI via the internet. Watson AI processes the questions, generates corresponding responses, and sends them back to the app for user display.

Additionally, users can provide feedback through the app, which is submitted to a Cloudant database over the internet. Cloudant stores the feedback and subsequently sends a success or failure confirmation message to users, which is displayed within the app.

3.3 Activity Diagrams

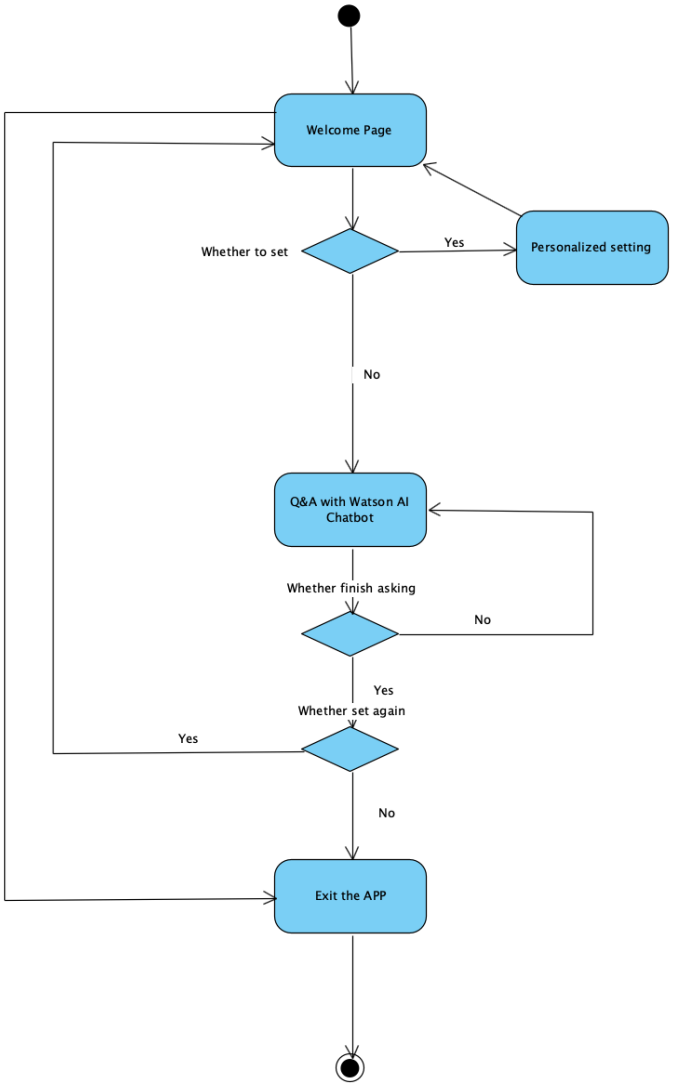


Figure6: Activity Diagram

This is an activity diagram illustrating the primary functionality of our app. Upon launching the app, users are greeted with a welcome page where they can decide whether or not to personalise their settings. The basic settings encompass changing the avatar, saving a QR code, and submitting feedback within the app.

Following the customization, users can engage in a Q&A session with the Watson AI Chatbot integrated into the app. Afterward, they have the option to ask additional questions, return to the welcome page to adjust their settings, or exit the app and conclude the activity.

3.4 Class Diagrams

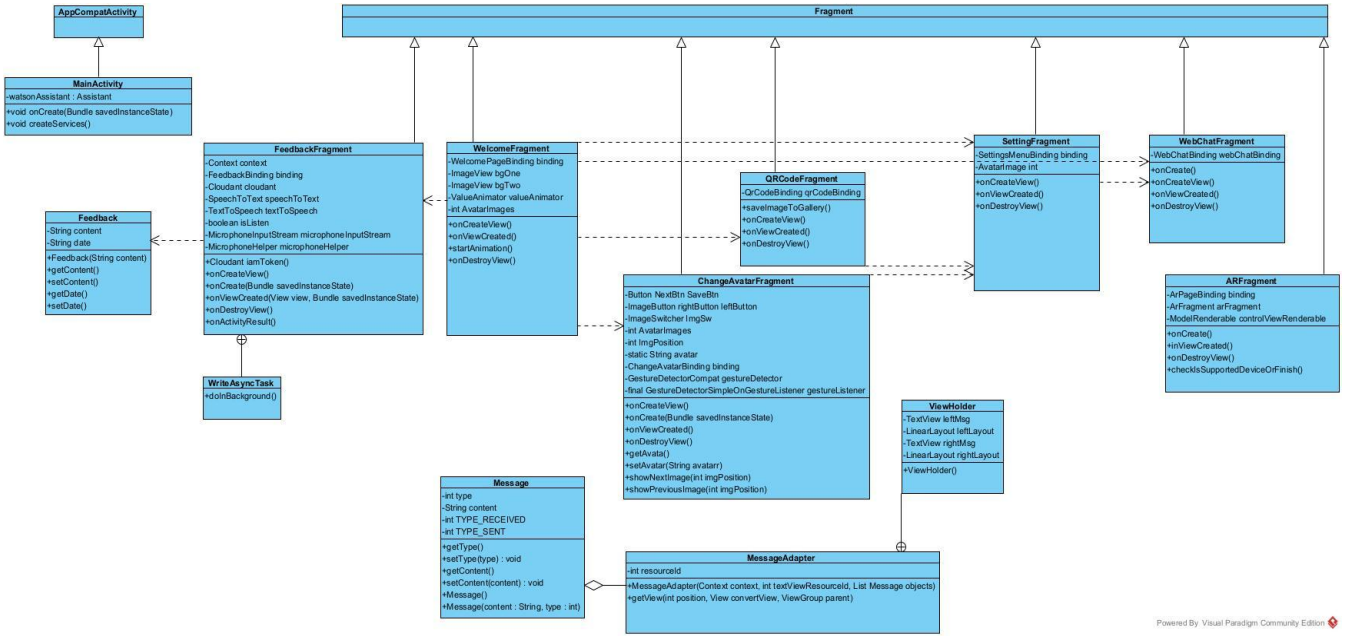


Figure7: Class Diagram

This class diagram illustrates the primary structure and relationships of the classes in our app. The app starts with the MainActivity, which hosts the various fragments responsible for handling user interactions and displaying the UI.

Upon launching the app, users are greeted with a WelcomeFragment where they can either navigate to the WebChatFragment to engage in a Q&A session with the Watson AI Chatbot or visit the SettingFragment to personalise their settings, such as changing their avatar, saving a QR code, or submitting feedback via the FeedbackFragment. The conversation with the AI chatbot is managed by Message and MessageAdapter classes, which handle the display of messages in the conversation view. The QRCodeFragment allows users to save a QR code to their gallery, while the FeedbackFragment handles user feedback using the Feedback class.

Users can navigate through these fragments to access various functionalities of the app, interact with the UI components, and personalise their experience. The relationships between the classes mainly revolve around navigation and data management.

4. Architectural Design

4.1 Android Application's Architecture

4.1.1 Model-View-ViewModel

The software architecture of the Android application under consideration has been implemented using the Model-View-ViewModel (MVVM) design pattern. This design pattern involves three primary components: the model, the view, and the view model, each serving a specific purpose in the application. The model layer is responsible for retrieving and updating data from various sources, such as databases or web services. The view layer, which represents the user interface, is responsible for displaying data and receiving user input. The view model acts as a mediator between the view and the model and exposes data and commands that the view can bind to, while also handling the communication between the two components. The interactions and relationships between the three components are depicted in a diagram presented below.

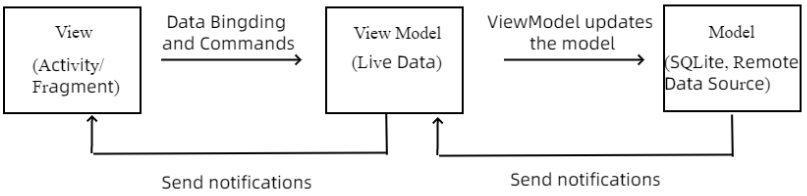


Figure8: MVVM Model

The Model layer of an Android application is responsible for accessing and updating data from various sources, including databases or web services. This layer is decoupled from the user interface and presentation logic of the application. The specifics of the Model layer, including local and remote data repositories, will be elaborated on in the subsequent section titled "Overview of Data Storage Mechanisms."

The ViewModel serves as an intermediary between the View and the Model. It exposes data and commands that the View can bind to, and it manages the communication between the View and the Model. The ViewModel abstracts the View, which simplifies testing and maintenance. Data binding is a key feature of the MVVM pattern.

The View is responsible for presenting the user interface, including the layout and visual elements of the Android application. It displays data and captures user input, but it is not aware of the application's data or business logic. The View is bound to the ViewModel, and any changes in the ViewModel are automatically reflected in the View. This eliminates the need for manual updates and reduces the risk of errors. In this Android application, two example classes of the ViewModel are provided. The 'FeedbackBinding' variable is used to bind XML layouts with the 'FeedbackFragment' object, allowing them to send data to each other.

```
public class FeedbackFragment extends Fragment {
    // data binding member variable
    private FeedbackBinding binding;
}
public class MainActivity extends AppCompatActivity {
      // class body
}
```

4.1.2 Navigation

The Navigation component refers to the process of moving between Activities or Fragments of the application. It is an important aspect of user experience and can greatly impact the usability of the app. Android provides a built-in navigation component that allows developers to create and manage navigation within their applications easily. 'res/navigation/nav_graph.xml' of this project defines all the destinations in the app and the actions that can be taken to move between them. And it provides a visual representation of the app's navigation flow, making it easier for developers to understand and manage the navigation within their app.

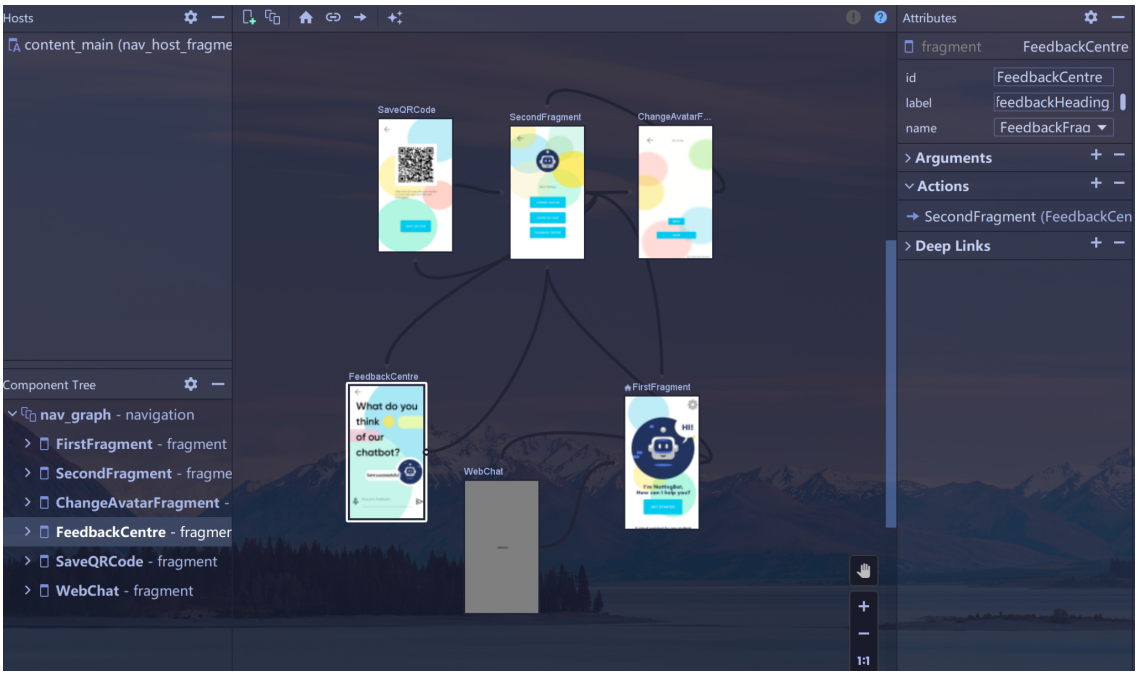


Figure9: Navigation

4.1.3 Background Tasks

In this project, there is a need to perform background tasks that can execute long-running operations without obstructing the user interface. The task of sending requests to IBM Cloudant Database server in class 'FeedbackFragment.java' is performed by the 'WriteAsyncTask' class that extends the built-in 'AsyncTask' class in Android. 'AsyncTask' class provides a convenient way for developers to perform background operations and present the results on the UI thread without having to manipulate threads or handlers explicitly. The 'WriteAsyncTask' class implements the 'doInBackground()' method, which is executed in a separate thread and receives input parameters as

an argument. Once completed, it returns a result encapsulated in the ‘Feedback’ class, which contains the content and date of one piece of feedback.

4.2 Watson Assistant’s Architecture

4.2.1 User Interface

IBM Watson Assistant provides a web-based user interface that allows developers to create, manage and deploy chatbots. The interface consists of several components that are designed to facilitate the development process. The dialog builder is the main interface for creating and managing the dialog flow of the chatbot. It provides a visual representation of the dialog tree and allows developers to create nodes that correspond to specific intents and responses. Each node can be customised with various settings, such as trigger conditions and response messages. The intent editor is used to create and manage the intents that the chatbot can recognize. Developers can create new intents, add example user inputs, and assign them to specific nodes in the dialog tree. The analytics dashboard provides insights into how the chatbot is being used, including metrics such as user sessions, intent usage, and user satisfaction ratings. This data can be used to improve the chatbot's performance and user experience.

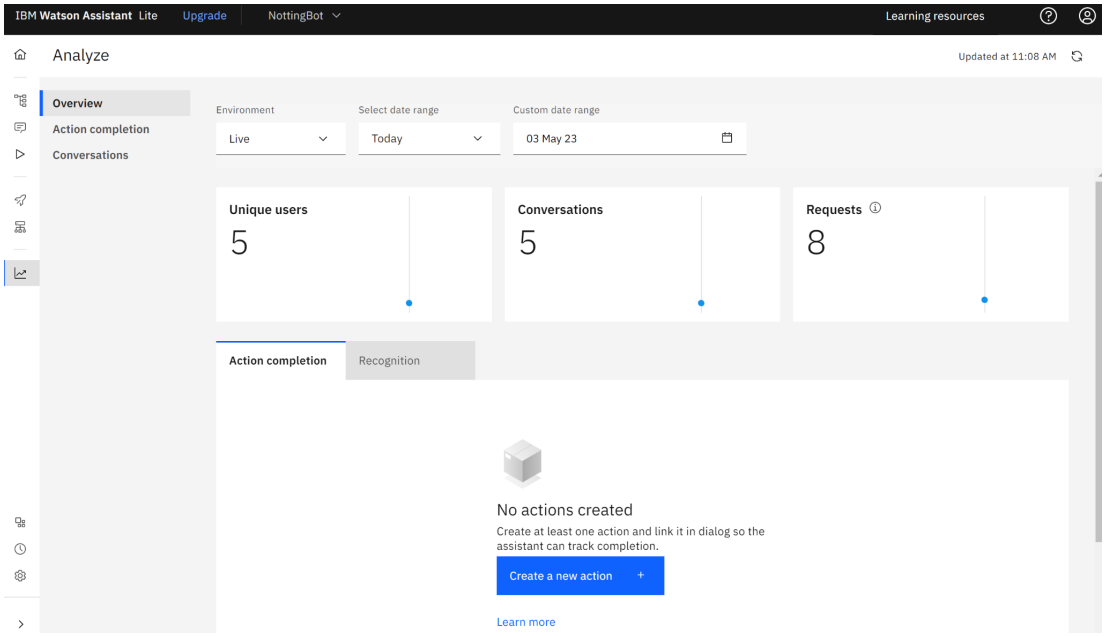


Figure10: Analytics Dashboard

4.2.2 Natural Language Understanding

The Natural Language Understanding (NLU) is an essential component of Watson Assistant, which allows the chatbot to comprehend and analyse user input expressed in natural language. The application of machine learning and deep learning algorithms enables NLU to extract relevant information from user input, including entities, concepts, emotions, and sentiment. Furthermore, the identification of concepts, which are more abstract ideas or topics related to the user input, is also possible with NLU. For instance, if a user requests information about "restaurants in Park Campus," NLU can recognise "restaurants" as the concept and "Park Campus" as the context. In summary, NLU is a powerful tool for enhancing the natural language understanding abilities of chatbots, facilitating better comprehension and response to user input that appears more natural and intuitive.

4.2.3 Dialog Management

Dialog Management in Watson Assistant is the process of controlling the chatbot's and the user's interaction flow. The exchange of messages between the user and the chatbot occurs at each turn in the conversation, which is set up as a sequence of conversational turns. Recognizing the user's purpose, understanding the context of the discussion, choosing the proper answer, and managing the conversation's state are all included in the dialog management technique. In order to determine the proper response or action based on the determined intents, entities, and context variables, it uses the dialog tree, which comprises dialog nodes.

5. Overview of Data Storage Mechanisms

5.1 Local Machine Data Storage

5.1.1 Introduction of SQLiteOpenHelper

The Android Software Development Kits (SDKs) offer developers a technique to persist non-trivial data, such as chat history, to local machines. By employing this method, the user experience can be greatly enhanced through the implementation of data persistence.

Android applications employ SQLite databases to save data locally on mobile devices, for various reasons. For instance, SQLite offers a lightweight, serverless, and self-contained database engine, making it an optimal solution for mobile devices that possess limited storage and processing capabilities. Notably, SQLite does not necessitate a distinct server process, simplifying deployment and reducing the intricacies of application development. Additionally, the database engine does not require any particular setup or administration, enabling easy integration into Android applications. To manage the creation, versioning, and updating of SQLite databases, Android provides the "*SQLiteOpenHelper*" class within the "*android.database.sqlite*" package. This class abstracts the intricacies of database creation and versioning management, allowing developers to concentrate on developing data storage and retrieval logic. Crucially, the principal reason for using SQLiteOpenHelper to save data locally is that we can extend SQLiteOpenHelper to create customised database helper classes to match their application's specific needs.

5.1.2 Processes of Using SQLiteOpenHelper to Manipulate Data

The following are the general steps for utilising the SQLiteOpenHelper class to manipulate a database. More coding details can be found in the "*ChatDao.java*" file of this project.

Step1: Extend the SQLiteOpenHelper class: Create a new customised subclass that extends SQLiteOpenHelper and implement the *onCreate()* and *onUpgrade()* methods.

Step2: Implement the *onCreate()* method: This method, called *onCreate()*, is executed when the database is created for the first time. It is in this function that you define the structure of your database, including tables and indexes, by using SQL statements.

Step3: Implement the *onUpgrade()* method: This method is called when the database needs to be upgraded due to a version change. Handle schema updates, such as adding or removing tables and columns, in this method.

Step4: Create an instance of your custom SQLiteOpenHelper class: Instantiate your custom SQLiteOpenHelper class in your activity or other application components where you need to access the database.

Step5: Use the *getWritableDatabase()* or *getReadableDatabase()* methods: These methods return an instance of SQLiteDatabase that can be used to perform CRUD (Create, Read, Update, and Delete) operations on the database. Some commonly used methods of SQLiteDatabase instance are: *execSQL(String sql)*: Executes a single SQL statement that doesn't return any data, *insert(String table, String nullColumnHack, ContentValues values)*: Inserts a new row into the specified table using the key-value pairs from the ContentValues object. Returns the row ID of the newly inserted row or -1 if an error occurred.

5.1.3 Introduction of SharedPreferences

SharedPreferences is an application programming interface (API) available in the Android platform, which facilitates the storing and retrieval of key-value pairs of primitive data types, including Boolean, Integer, Float, and String. The data is persisted as an XML file in the app's private storage, making it accessible only to the app. One of the significant advantages of using *SharedPreferences* API is its simplicity of implementation, which does not necessitate the creation of a database or helper classes. This approach is particularly suitable for storing small amounts of data, such as user preferences or application settings, that do not require complex querying or sorting operations. In our project, we utilised the *SharedPreferences* API to save the user's preferred Avatar image.

5.1.4 Processes of Using SharedPreferences

This project used IBM Cloudant

The following are general guidelines for using SharedPreferences. For more detailed coding information regarding reading data from SharedPreferences, refer to the "*SettingFragment.java*" file, while the "*ChangeAvatarFragment.java*" file contains information on how to write data to it.

Step1 Access SharedPreferences:

To access *SharedPreferences*, call the *getSharedPreferences()* method in your activity or other application components, passing the name of the preference file and the mode (usually *Context.MODE_PRIVATE*). This method returns a *SharedPreferences* object.

Step2 Write data to SharedPreferences:

When saving data to SharedPreferences, the *SharedPreferences.Editor* interface can be used. To obtain an editor, the *edit()* method should be called on the SharedPreferences instance, after which the desired key-value pairs can be set using the appropriate methods such as *putString()*, *putInt()*, *putFloat()*, *putLong()*, or *putBoolean()*. Finally, to save the changes, the *apply()* or *commit()* method should be called.

Step3 Read data from SharedPreferences:

When attempting to access data from *SharedPreferences*, you can employ the getter methods (such as *getString()*, *getInt()*, *getFloat()*, *getLong()*, or *getBoolean()*) on the *SharedPreferences* instance that you have created, along with the key corresponding to the desired data and a default value that should be returned in case the desired key is not found.

5.2 Online NoSQL Database

Here are some basic instructions on how to enable IBM Cloudant service for this project. You can find more technical details on how to send a message to the online database by referring to the "*FeedbackFragment*" class in the inline document or source code.

Step1: Sign up for IBM Cloud and create a Cloudant service instance
To get started with IBM Cloudant, you need an IBM Cloud account. Once you're logged in, create a new Cloudant service instance by navigating to the "Create resource" option in the dashboard, then select "Databases" and choose "Cloudant".

Step2: Create a new database and document
From the Cloudant dashboard, click the "Create database" button and provide a unique name for your new database. Once created, you can add and manage documents, create secondary indexes, and set up replication tasks. In Cloudant, data is stored as JSON documents. To add a new document, click the "New Doc" button in your database view. You can then provide a JSON object containing your data, or you can use the form-based editor to input your data. Save your document to store it in the database.

Step3: Add dependencies
Add or update the following dependency in the '*build.gradle*' (module level) file of this project.

```
dependencies {
    implementation group: 'com.cloudant', name: 'cloudant-client',
version: '2.6.0'
    implementation 'com.ibm.cloud:cloudant:0.+'
}
```

Step 4: Create Cloudant Instance in the Code
To create a Cloudant instance in the code, you can start by creating an IamAuthenticator instance with the Cloudant API_KEY string. This authenticator can be used to create a Cloudant instance and set the service URL for that instance.

Step 5: Create a New Document and Post it to the Database
To create a new document, you can instantiate a Document object and use the put method to add data to it. Once the document is ready, you can use the cloudant instance's inbuilt method, such as cloudant.postDocument(postDocumentOptions).execute(), to post the document to the database. It is important to perform these posting tasks in an asynchronous thread to ensure expected application performance.

6. Coding conventions

6.1 Android Coding Conventions

1. Naming conventions:
    - Classes: Use PascalCase (e.g.,MessageAdapter)

- Methods: Use camelCase (e.g., getView)
- Constants: Use UPPER_SNAKE_CASE (e.g., VERSION_CODE)
- Variables and member fields: Use camelCase (e.g., viewHolder)
- XML files: Use lowercase and underscores (e.g., message_item.xml)
- IDs in XML: Use camelCase (e.g.,codeToSetting)

2. Code organisation:
    - Organise imports alphabetically
    - Group related code together and separate different sections with a blank line
    - Use packages to organise code by functionality
3. Formatting:
    - Use the Android Studio default code style settings
    - Use spaces instead of tabs for indentation
    - Use a line length of 100 characters max
    - Use braces even for single-line statements
4. Comments and documentation:
    - Use Javadoc-style comments for classes, methods, and member fields
    - Add inline comments to explain complex code sections
    - Keep comments up-to-date when code is changed

6.2 Git Conventions

1. Commit messages:
    - Write short, descriptive commit messages (50-72 characters) in the imperative mood
    - Provide a more detailed description, if necessary, after a blank line
    - Use bullet points or lists for multiple changes
2. Branching strategy:
    - Use a consistent naming scheme for branches (e.g., feature/feature-transition or bugfix/bug-transition)
    - Create feature branches for new features, develop branches for features, bugfix branches for bug fixes
    - Merge branches back into the main branch after code review and testing
3. Rebasing and merging:
    - Prefer rebasing to merging when updating a feature branch with the latest changes from the main branch
    - Use git merge --no-ff when merging feature branches back into the main branch to maintain a clean history
4. Tagging:
    - Use semantic versioning (e.g., v1.0) for release tags
    - Every significant merge to master branch needs a new tag
5. Repository organisation:
    - Keep the repository clean by removing old branches and tags
    - Use a .gitignore file to exclude files that should not be tracked (e.g., build artefacts, logs, or IDE-specific files)
    - Organise code in a modular and maintainable manner
6. Collaboration and communication:

- Communicate with team members when making significant changes to the codebase
- Use issue trackers, project boards, or other tools to manage tasks and track progress

7. Installation Instructions

1. Prerequisites:
   - Git installed on your machine
   - A GitLab account
   - Java Development Kit (JDK) installed
2. Download and install Android Studio:
   - Visit the Android Studio download page: https://developer.android.com/studio
   - Download and install Android Studio for your operating system.
   - Follow the installation prompts and set up Android Studio with the recommended settings.
3. Clone the repository

```
# Clone the repository using HTTPS
git clone https://projects.cs.nott.ac.uk/comp2002/2022-2023/team4_project.git

# Or, clone the repository using SSH (if you've set up SSH keys)
git clone git@projects.cs.nott.ac.uk:comp2002/2022-2023/team4_project.git
```

4. Navigate to the project directory:

```
cd team4_project
```

5. Open the project in Android Studio:
   - Launch Android Studio and select "Open an existing Android Studio project."
   - Navigate to the project directory you cloned in step 4, and click "OK."
   - Android Studio will automatically import the project and download any required dependencies.
6. Install and configure Gradle:
   - Android Studio usually downloads and configures Gradle automatically.
   - If Gradle is not automatically installed, you can download it from the Gradle website: https://gradle.org/releases/
   - Extract the downloaded ZIP file to a directory of your choice.
   - Set the environment variable GRADLE_HOME to the extracted directory, and add GRADLE_HOME/bin to your PATH.
7. Build and run the project:
   - Click the green “Run” button in the toolbar or press Shift + F10 and run the project.
   - The app will be installed and launched on the selected device.