CS5030: SOFTWARE ENGINEERING PRINCIPLES

Software Design, Modelling and Analysis

220031985, 220018772, 200007892, 190022154, 220032472

University of
St Andrews

15th November 2022

CONTENTS

THE BIRTH OF PROJECT BREEZE: BEHIND THE SCENES

Abstract. This article shows the design of a career network application oriented to university students and employer organisations. During the planning phase of the application, the team adopted an agile process and demonstrate an understanding of software design, UML (Unified Modelling Language) Diagrams, and their applications. Appropriate notations are utilised while modelling the system to their respective diagrams. In making the application, the team adhere to the different artefacts of the software lifecycle. Furthermore, the team investigated the best-suited software architecture for the application and analysed the models from a software-development perspective. Finally, the team documented the outcomes of the work in this technical report.

Keywords: Unified Modelling Language (UML), SCRUM, Agile, Service-oriented Architecture, Entity Relationship (ER), Career Network, Application Programming Interface

1 PHASE ONE - INTRODUCTION

In the given scenario, Group 18 employed by a software company is tasked with designing an application for a client that allows university students and employers to connect via the app where employers offer internships and job opportunities to the students. The name "Breeze" was given for identification of the application.

1.1 REQUIREMENTS OF THE APPLICATION

- The Application should allow universities and employer organisations to register with it.
- Both organisational contacts and students should have access to search and filter facilities to easily find information that is of interest to them.
- A recommendation facility which automatically matches students and opportunities and displays the top 10 matches each time a student or an organisational contact logs in to the application.

- **Student USE CASE**: When a university is registered with the app, each matriculated student at the said university will be able to create a profile in the application with the relevant information given below:
    - Academic Transcripts
    - Prior Work Experience
    - Prizes Won
    - Links to Portfolios or Repositories of work
    - Expected Completion Date
    - Preferences for Career Positions and Sector

Conditions:

- Each student affiliated with a registered university can view all opportunities posted by the organisation.
- Once a student reaches the completion date of their programme at the university, they no longer have access to the system.

- **Employer USE CASE:** Once an employer organisation registers with the application, they would be able to do the following:
    - Create a profile for their organisation
    - Name up to 5 specific contacts within the organisation
    - Add internship and job opportunities in their organisation

Conditions:

- The named contacts within each registered organisation can view all student profiles in the application.
- Each employer organisation is responsible for keeping the status of the opportunities it offers up to date. The status includes the following:
    - Active
    - Withdrawn
    - Filled by a student registered with the application
    - Filled by an external candidate

Developer Notes:

- All data in the system can be used to generate summary reports for the client company to monitor the take-up and effectiveness of the application.

1.2 FUNCTIONAL REQUIREMENTS OF THE APPLICATION

The following were identified as the functional requirements of Project "Breeze":

- Authentication of users whenever they log in to the system.
- The organisation must be able to post job requirements.
- The University must be able to matriculate students.
- Students must be able to search and apply for jobs.
- Organisational contacts must be able to view student information.
- The application should be able to generate summary reports for the client company to monitor the take-up and effectiveness of the application.
- The client should be able to generate reports for organisations, universities and students with information that is of interest.
- Recommendations must be provided as per optimisation (TOP 10) for organisational contacts or students.
- The search and filter functionalities must show relevant results upon being called by the organisation's contacts or students.
- The user of the application must be able to change the password after initial login.

1.3 NON - FUNCTIONAL REQUIREMENTS OF THE APPLICATION

The following were identified as the non-functional requirements of Project "Breeze":

- The application must be able to scale to incoming traffic and maintain its performance.
- Cross-platform support for the application is required.
- Every action by the users performed in the application must be recorded on an audit trail.
- Adhering to Security and Privacy Policy as stated below in this report.
- The application must have faster deployment and loading time.
- The application must be secure and reliable at all stages of its usage.

1.4 CHOOSING AGILE METHODOLOGY

As the developer, it is critical to select the appropriate agile technique and establish a solid software architecture for the application. Given the specific circumstances, only a short window of time was given for the construction of the application, and customer satisfaction is critical to the commercial success of the application.

Even during the later development process, changes in requirements should be embraced rather than pursuing "strict" requirements. Since deliveries are incremental, an agile process would allow changes in system requirements and provide constant "value" delivery to the consumer.

Therefore, the only process that could **meet the requirements** as provided by the client and our need for continuous delivery was the **Agile development process**. The section that follows explains the agile approach that is ideal for Project "Breeze".

Iterative and Incremental Development (IID) was chosen as the incremental model for development due to the flexibilities it offered:

- The software would first release, followed by improvements about opinions from the stakeholders.
- Risks are identified and mitigated during the early phases.

At one stage of development, the waterfall model was considered an ideal build model for “Project-Breeze” but was soon dropped since the client was assumed to provide feedback on a **“SPRINT”** bases during the development.

1.5 AGILE FRAMEWORKS CONSIDERED FOR EFFECTIVE COLLABORATION

KANBAN was **considered but dropped** because there is no essential oversight such as a master management team. Given the short window of time, the development of each specified requirement could lose track of time management without oversight roles, and the project could be led to mayhem and chaos as a result [1].

Extreme Programming (XP) was another agile methodology that was considered by the team. However, since Project "Breeze" is meant to be built and evolved for a longer period, the **core idea of "the simplest thing that will work" and "head-first into the code before the design process" of XP** without putting too much weight on the long-term product view did not suit the requirement of the project.

The issue the team faced with **Lean Development** was the ruthless removal of any activity that does not bring ultimate value to the product. The team **prioritised adaptability**, and Project "Breeze" offers extensibility to enhance and add new features. **The market value of "Project Breeze" is immense**, and we were not inclined to believe that the client and the user would be satisfied if only the **Minimum Viable Product was presented.** The team concluded that we did not want to approach Lean Development as it went against the requirements of software completion as stated in the business model.

Bypassing other methodologies, we conclude that **SCRUM** was the viable option given its simplicity on the **"definition of done**" used for validating requirements. Although SCRUM has its fair share of cons such as the over-segregation of features focusing on a singular part, it offered advantages to the **service-oriented architecture** by allowing the team to **independently develop services.**

1.6 AGILE ROLE-BASED STRATEGY

Software development is a **complex endeavour**, requiring **frequent and intense communication** with the team and customer. The team took an **Agile Role Based Strategy** as shown below, which helped shorten the feedback cycle of Project Breeze so that the team can learn faster. The strategy continuously seeks out ways to better understand the requirements and helped break down large features into smaller requirements, which resolves the ambiguities.

The team was able to handle ambiguities and were able to derive common conclusions by assigning roles to the team and posting queries as per role demands. The following table shows an example of how a role-based strategy was implemented. Roles were reversed on a sprint basis to understand various aspects of the requirements from an independent view.

Team	Role
220031985	Product Owner
220018772	Product Owner
200007892	Developer
190022154	Developer
220032472	Scrum Master

Table: Agile Process Role-Based Strategy (Implemented in SPRINT I)

1.7 AZURE DEVOPS INTEGRATION INTO PROJECT "BREEZE":

As concluded in the aforesaid discussion, the team decided on the SCRUM-based Agile methodology for the development. Therefore, the team required a suitable platform that could be quickly familiarised with to share and track the progress of planning, development, and discussion across the team. The Azure DevOps platform was chosen to accommodate such needs.

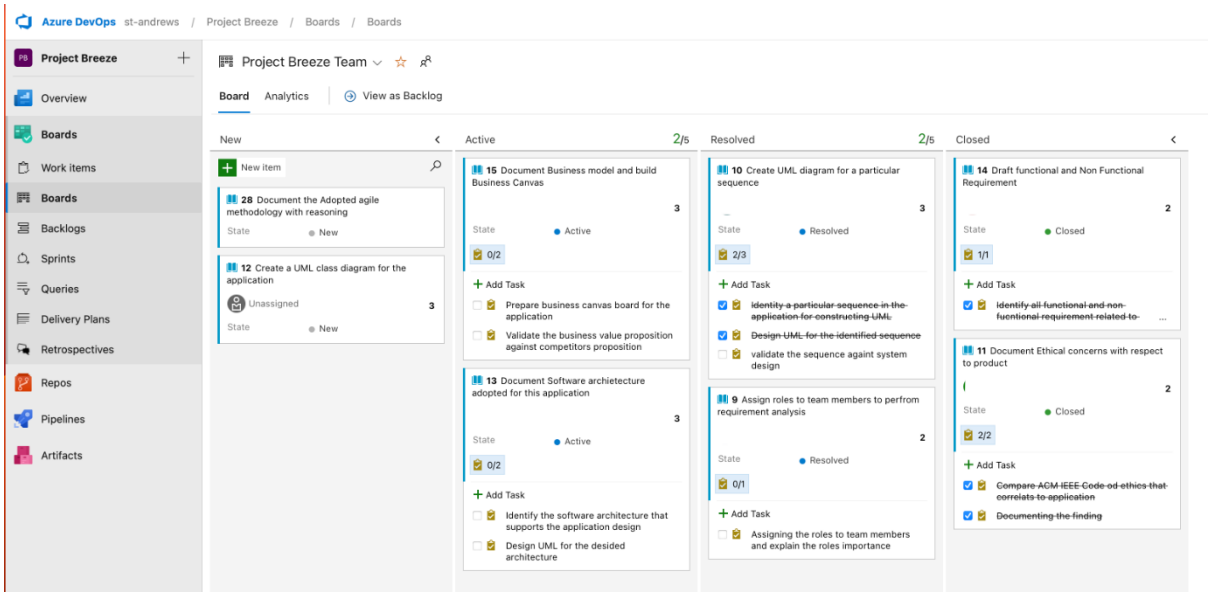

Fig 1.1: Image illustrating Azure DevOps board used in SPRINT 1

The above image illustrates the usage of the Azure DevOps board to track the progress of this assignment in the initial phase.

Each member was assigned an individual user story along with corresponding tasks and a specified date of completion. This board was reviewed during the stand-up SCRUM meetings of each sprint cycle. This helped the team to understand the progression of each developer and resolve any difficulty faced by an individual.

2 PHASE TWO – PRODUCT VISION

2.1 PROJECT BREEZE'S ARCHITECTURE BUILDING BLOCKS

The group gathered to discuss the needs of the application for the client company on obtaining the specifications. The purpose of the platform was to link college students with firms offering internships and career possibilities, and the team followed such purpose as the fundamental requirement throughout the entire development.

After simplifying and cogitating the requirements over numerous iterations, the team achieved the basic architectural building blocks of the application environment (Figure 1), and the application structure (Figure 2) as shown below:

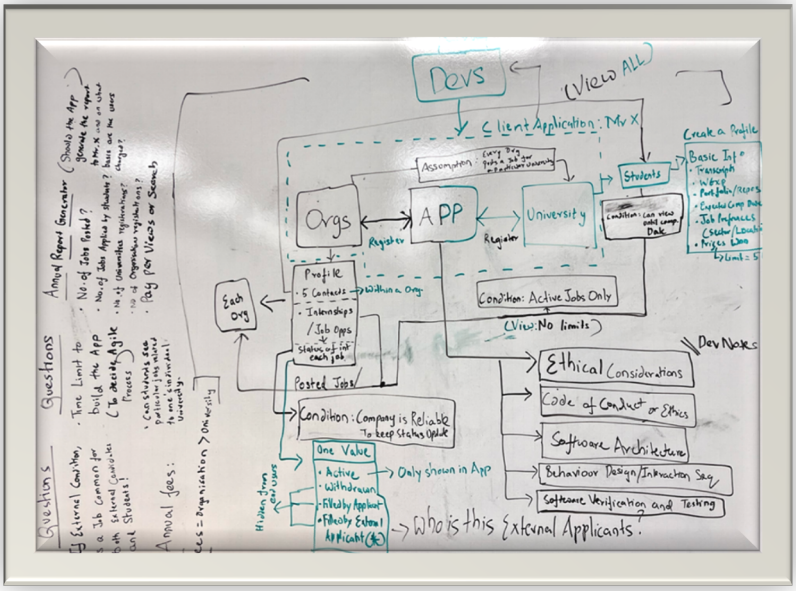

Fig 2.1: Whiteboard Illustrating the First Draft of the Structure "Alpha"

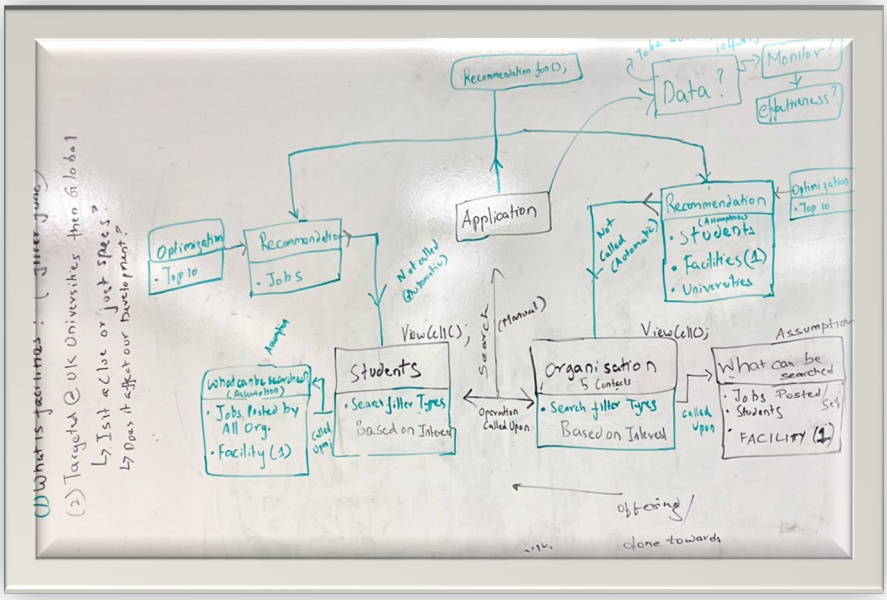

Fig 2.2: Whiteboard illustrating the First Draft of the Structure "Beta"

Further improvements were made to the basic drafts of the above structure to maintain appropriate diagrams and document the process in a precise and logical manner, as shown below:

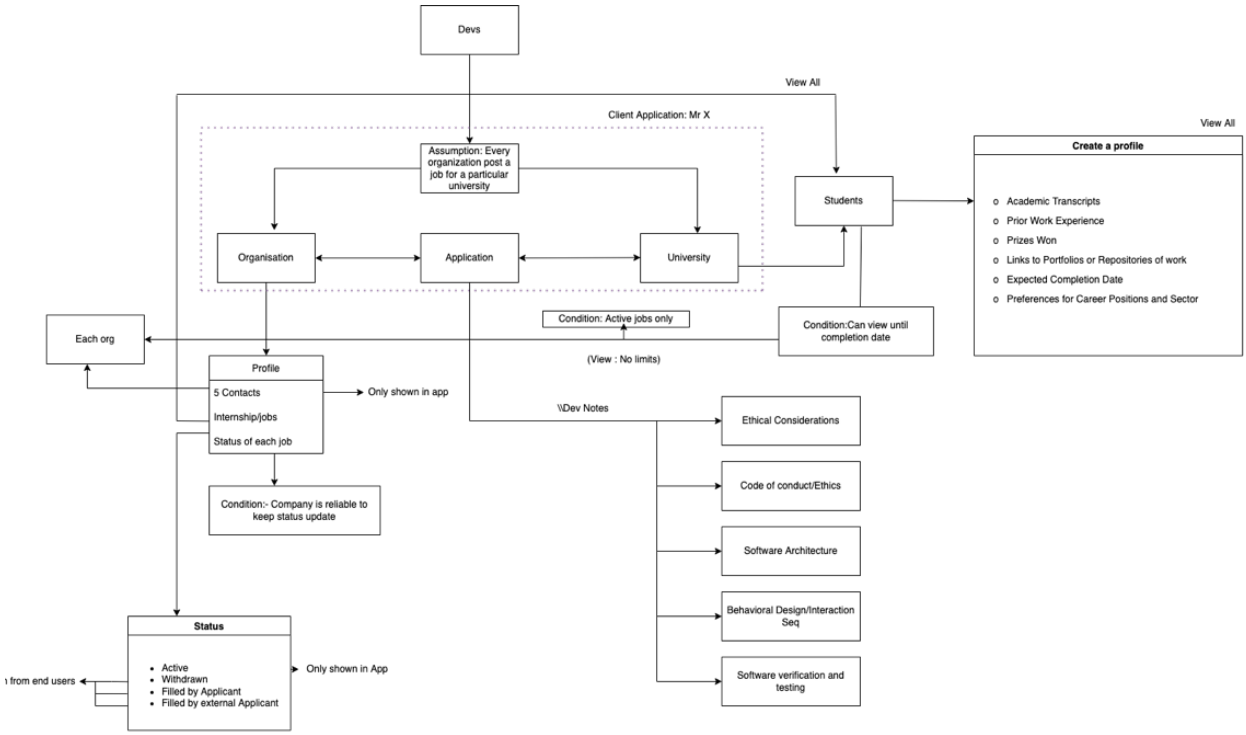

Fig 2.3: Image illustrating the "Alpha" basic architectural building block.

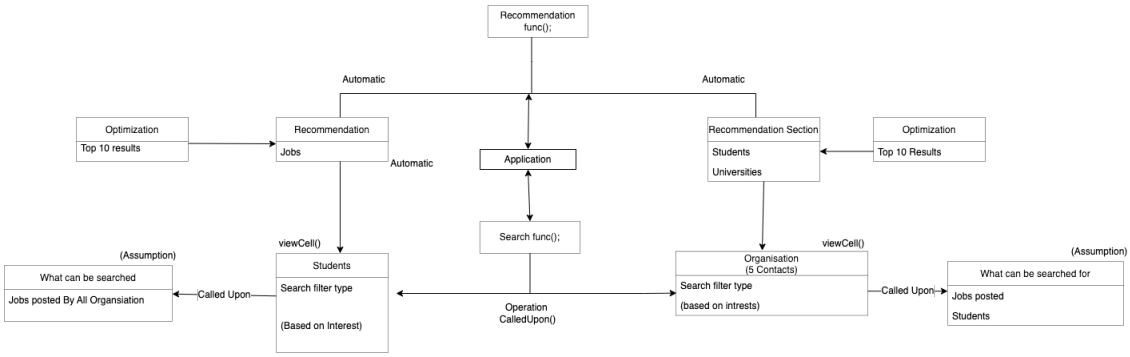

Fig 2.4: Image illustrating the "Beta" basic architectural building block.

Note: The above images do not represent the final representation of the system architecture.

The diagrams provided a solid foundation upon which to develop the application and a clear idea of the desired outcome. The team opted to proceed with the following conducts:

- Implement an agile procedure; control the division of labour; and equally, balance contributions.

- Use software design principles, UML diagrams, and their applications for each artefact of the software development lifecycle.
- Uphold proper modelling notations that meet the standards of the application and the client.
- Create the structural and behavioural design and establish the software architecture.
- Examine the previous models.
- Providing high-quality technical documentation with a wealth of content about how Project "Breeze" was carried out.

To the left of the **Basic Whiteboard Drafts** of **Alpha** and **Beta**, we placed the queries that were to be set forth by the development team to the client to move to the next stage of software development.

The **Queries** that arose are as follows (Categorised according to the diagram):

Alpha:

- Though the requirements were first addressed to the organisation -> University - > Students, there is a mention of external candidates – Would it mention the alumni of a university or anybody out of the mentioned network?
- If the external candidate refers to a complete outside person, will there be a separate storage requirement apart from registering them to a university?
- Would the annual report generated by the client be based on,
    - No. Of Jobs Posted?
    - No. of Jobs Applied by students?
    - No. Of universities registered?
    - No. Of Organisations registered?
- Should the application generate the report for the app owner on basis of the number of Searches that the organisation or the students registered with the university do? In the case of the external candidate, who will be charged if they are not registered with a university?

Beta:

- The search and filter function has a facility, what is the meaning of this facility in terms of organisation and university? The above queries were sent to the client for further clarification on clearing the ambiguity in the requirements of The applications and moving down the development pipeline.

The reply we received from the client becomes part of the assumption, and is shown as follows:

"As far as the spec is concerned, the application only needs to store the status of opportunities offered by employers. It doesn't require you to capture the details of the successful candidate if an opportunity has been filled."
As the spec says,
"If you find the specification ambiguous or incomplete in any way, working as a group, resolve the issues in a manner consistent with the information provided and list any assumptions you make in the report. "
"So, where details are not provided in the spec, you can decide the specifics as long as they don't contradict anything already mentioned. "

FURTHER QUERIES (Assumptions) :

Q: Are the "up to 5 specific contacts" allocated to every job profile, or allocated to each organisation only?

A: 5 specific contacts are for the organisation only. But allocating contacts responsible for each opportunity could be an additional functionality.

Q: What do the "external candidates" refer to? Does the system need to store any information related to the "external candidates"?

A: The "external candidate" refers to applicants applying to the same position that are not using the system. This can include students after their completion date. They should not have access to the system.

To resolve this ambiguity the team decided that the **external candidate is not being considered** at this given point in time. Since we are following an agile methodology, we always will have room for future changes to the existing model.

Q: Does the future adaption from UK-targeted to global deployment means a potential change in the code of ethics?

A: The section for the code of ethics is mainly from the aspect of the system itself. You can use an existing code of ethics such as ACM, or you could design it from scratch.

Q: What do the "search and filter facilities" in the specs refer to?

A: It means the "functionality" of searching and filtering, not any actual physical facility.

2.2 PRODUCT STRATEGY

The product is created for the client to help the students, external applicants, and employers connect. **A scalable way to ensure informative and accurate content is to reach the target users with high user retention at a cost as low as possible.**

Our application focuses on connecting organisations directly to the students thereby improving the **process of inter-networking and resourcing**. Rather than posting and updating jobs at various competitor platforms and waiting for potential candidates to find and register for the appropriate jobs, employers can make use of the readily available student market.

This benefits employers in terms of **efficiency** and improves the students' search for employment as they are directly supported by their universities via this application. As per our market analysis, we understood that the application can help **reduce the unemployment rate** by providing relevant training and experience for university graduates.

Further analysis **recognised and indicated the faults in the competitor's application**, as listed below:

Student Perspective:

- Too much **noise** related to **political and irrelevant content**.
- Struggle to find a job as per interest.
- Missing out on information/events due to bombarding posts in the feed.

University Perspective:

- Could not track the accurate information of the student.
- Can post blogs and videos but could not get reach a wide audience.

Organization Perspective:

- Due to clickbait posts, they could not find better candidates.
- Too much noise among top companies, hard to grab candidate attention.
- Candidates filling in multiple offers at once make it difficult to filter out.
- Shortage of talent skill and accuracy.

2.3 BUSINESS MODEL

The development team created a business model that satisfied the client's needs and identified the stakeholders. At the time of writing, it is believed that the application will **initially target UK institutions and companies.** There are over 160 universities in the UK, with over 2,751,865 students, according to reports [2]; in addition, 810,316 companies were incorporated in 2021 [3], creating an **untapped economic resource pool.**

The application attempts to leverage this situation by creating a seamless and quality networking experience between job-providing organisations and university students. Students can use the application's resources thanks to its freemium business model, but the client wanted to meet the following criteria to generate revenue from the app:

- The client will charge the universities and employer organizations an **annual fee** to use the application.
- Universities are **charged a lower fee** than employers.

During the first version release, The Application is **targeted at UK universities** and employers. If successful, it will be deployed on a **global scale**.

The development team further developed the **business model canvas** for the Project code-named "Breeze" to identify the stakeholders, resources, and other segments that are related to the application.

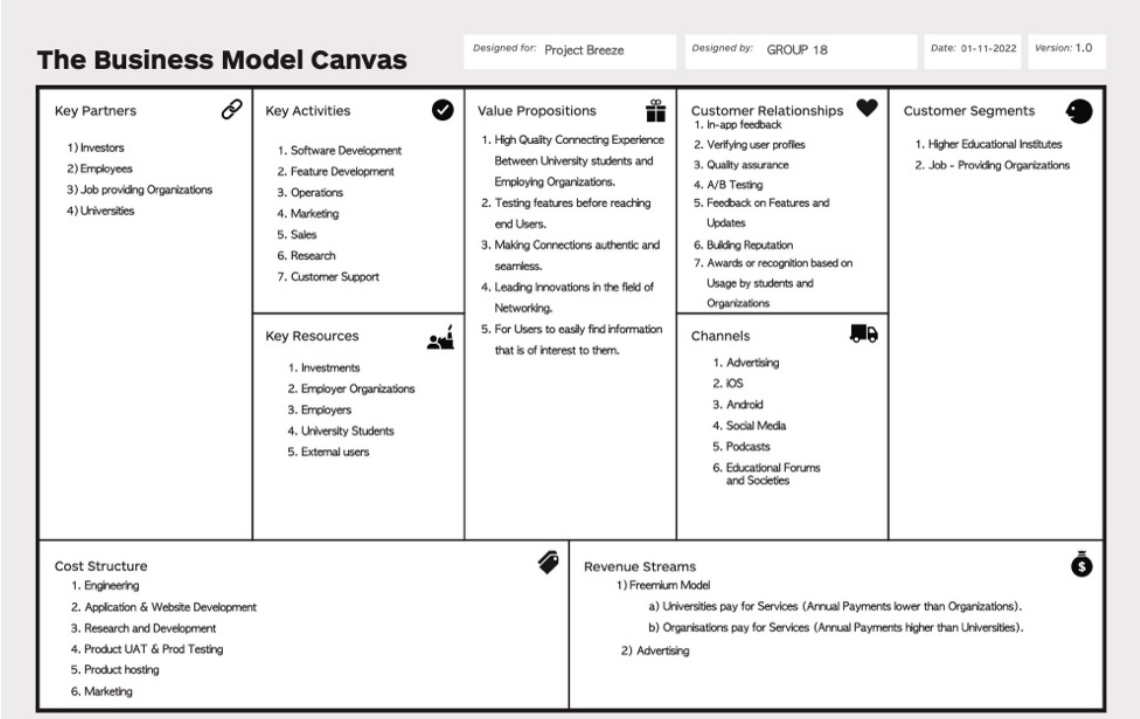

Fig 2.5: Business Model Canvas illustrating Project "Breeze" Assets

3 PHASE THREE - DESIGNING THE SOFTWARE

3.1 DEFINING THE SOFTWARE ARCHITECTURE

After a few talks about the potential future scope of the project, the initial idea of employing a monolith design was abandoned. As stated in this report, the product is anticipated to be marketed outside of the United Kingdom. The structure of the business logic that will be implemented may need to be altered in response to such future requirements. In event of the decision to build the application based on monolithic architecture, we came across the issue of scalability.

A simple case from the application perspective was the login and registration functionality. If the application was to be initially designed for the UK region, specific authentication API (Application Programming Interface) services like Microsoft

authentication library (MSAL) can be incorporated, or the usage of Google authenticator as a multi-factor authentication (MFA) vendor can be utilized.

Given a scenario of expansion into other countries, some of these MFA vendors might not be available, for example, Google services are not fully and legally available in China [4].

This could result in changes to the code base considering region-specific product restrictions and availability. If the login and registration actions can be developed and delivered as a single entity in such situations, reworking it in the future would be simpler and have no negative effects on any of the other services. Making updates will be more difficult if the original implementation uses a single code base for all the application's features.

This led the team to consider alternative approaches, which made the team weigh in on the benefits of **Service-oriented Architecture (SOA)** for our scenario. As the **need for scalability, reliability and resiliency** were considered as a primary need, we were convinced at choosing other architecture patterns over monolith.

Building software that can adapt to a **distributed and heterogeneous environment**, like in our scenario, is addressed by SOA. It is an approach to distributed system architecture that employs loosely coupled service, standard interface, and protocol to deliver seamless cross-platform integration. By giving them a common interface and integration protocol, it is utilised to combine different components [5].

Another supporting factor was that the actors in our application have varied uses, such as a student might use this application for searching and applying for opportunities posted by an employer. Given the case where the recommendation system was built as a separate service and independently maintained, and if it failed for some reason, then the student should still be able to apply for the job if he searches for a specific job using the search options which can be an independent service of its own.

This architecture elegantly handles such failures, enabling the application to be dependable. Because **SOA is a loosely linked architecture**, a problem in one system will not affect another. In SOA, all applications must be able to simultaneously receive and change data at the source level. As a result, complicated data synchronisation mechanisms are not necessary for SOA services. This strategy, though, also develops inter-service dependencies.

As stated earlier, the team adopted a SCRUM-based agile methodology for this project and building independent abstract services encourages agile working practises, with separate teams developing different parts of the application. As a result, each service may now be built, tested, and updated independently of the other connected services, which can improve the development process.

3.2 CHOOSING SOA OVER MICROSERVICE ARCHITECTURE

When comparing this with the widely used microservice architecture, we realised that in an SOA model, developers can reuse components which relate to enhancing efficiency and scalability.

However, if reusability was followed in microservice architecture then it would result in reduced agility, resilience and fault tolerance, since reusing a component will create dependencies across different services [6].

Instead, to improve efficiency and maintain high degrees of independence, developers use duplicate data or reuse code in a microservices design. Given the diversity of the application and assumption on the timeline set by the client, we decided to go with SOA architecture, as the initial investment and setup time for microservice is large and might not be suitable in our case.

SOAs make development and troubleshooting easier by making use of the benefits of utilising a common architecture. This sometimes compromises the speed of operation but favours discarding code duplication.

Since every data in the system is interdependent and every service will act upon the same data set (e.g. in the case of recommendations, the data from a job posted from the organization and the preferences added by the student are correlated and presented to the user), it was **a well-suited choice for us to adapt Service-oriented architecture.**

3.3 IDENTIFYING RELEVANT SERVICES

As per the plan we decided to divide and develop the entire application into the following services. These services have **distinct objectives** and their business logic relies on a single data source. Each of these services accesses and updates data at the source level at the same time, therefore services do not need to include complex data and synchronization models.

Each service undergoes **updating, testing, and scalability**. With SOA, each of these following services shares a common communication mechanism called an **enterprise service bus (ESB).**

Major business and domain-specific concerns are decoupled and placed in different, independent code bases. By breaking down activities into smaller processes that run independently of one another and contribute to the overall whole, SOA makes any complexity visible and easier to manage, without reducing it.

- **Service for authorization/permission management – Registration and login –** As stated in the aforesaid section on why the team chose this architecture, The client has plans to expand the application to different regions outside the UK in future, it is anticipated that the library and the vendor used for building the authentication and authorization feature vary for each country based on their restriction imposed by the government. Therefore, the app registration and login for each user are developed and maintained as a separate service.

- **Service for student and organisation contacts search/filter –** as the number of search and filter parameters are bound to change depending on several factors, as the result of user testing feedback, the search and filter action will be built as an independent service.

- **Service for Candidate and Opportunity Recommendation -** The Recommendation function will be determined by a variety of factors, including the users' location, typical salary, current market conditions, and much more. Therefore, modifications to business logic are anticipated. The suggestion criteria will be simple to update whenever this is built as a distinct feature. New features can be tested by teams, and if they do not work, they can be removed. This reduces the time it takes to deliver new features, makes changing code easier, and allows for deployment without affecting the functionality of other services.

- **Service for a job posting and candidate job application –** One of the primary features of the system is to allow students to apply to opportunities posted by the organization. This feature should be developed, maintained, and deployed independently to allow **high maintainability and testability** along with the reliability of this service, as a dedicated team can be used to actively monitor and rectify any issues if it occurs.

- **Service for report generation –** This feature is served only to a certain section of the application user and **need not be a feature** available to all the connected clients.

- **Notification service –** We realized that a notification service to perform omnichannel capabilities will be needed to inform the users about the progress of their application. This service is for **future implementation** as our database consideration does not track the response from the notification service.

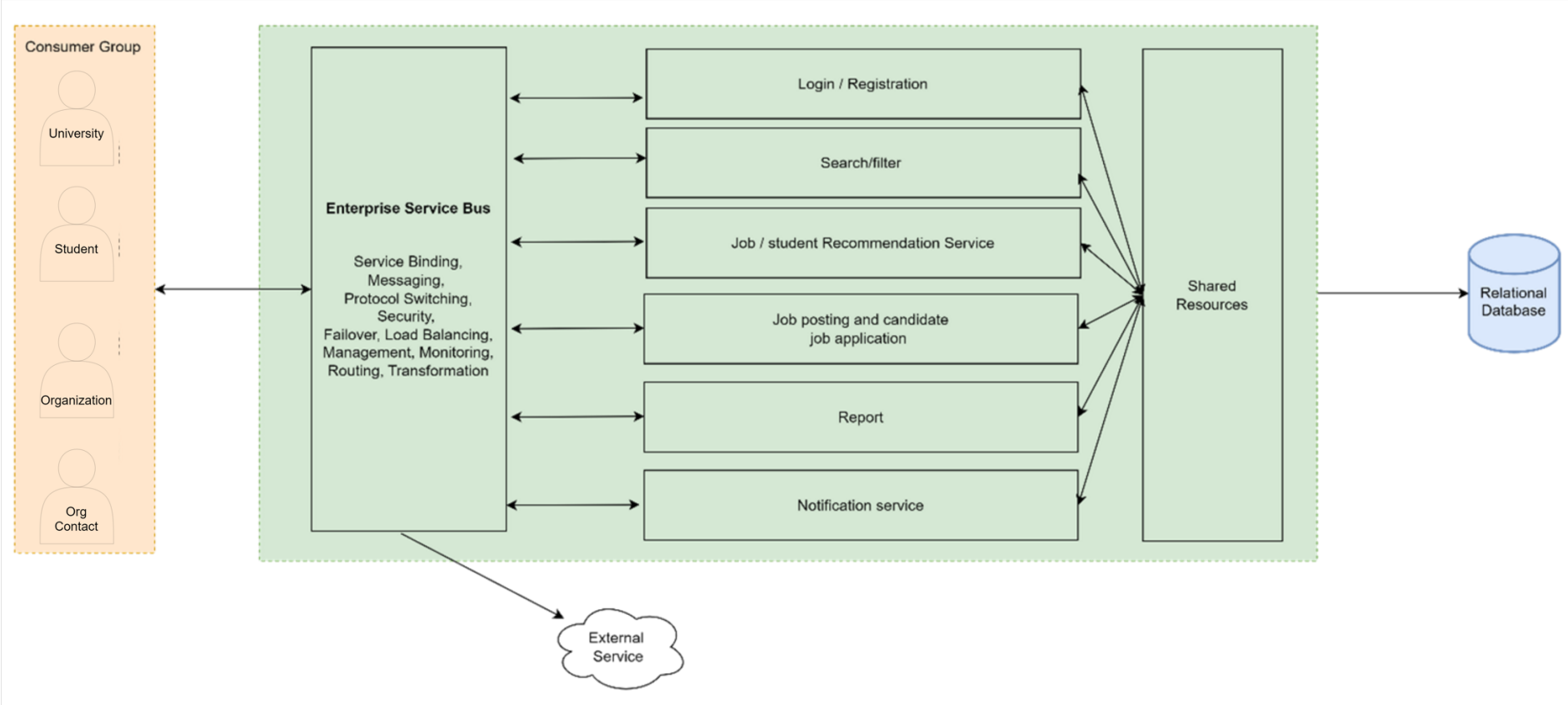

Figure 3.1: Image illustrating High-Level Service Oriented Architecture of the application

While designing the architecture, the team highlighted the hidden benefits of Service-oriented Architecture, as shown below:

- **Autonomy and loose coupling -** Because of this loose coupling and the way the services are published, development teams can save time by reusing components in other applications across the enterprise [6].
- **Interaction over components -** Each module or service will interact with each other, to perform certain data validation or for data gathering.
- **Abstraction –** The end-user or client is not required to know how the recommendation system works, therefore it would be reasonable to abstract the service level code. The client will just have to show the computed result of the recommendation done by the server.

3.4 TESTING STRATEGIES EMPLOYED FOR THE APPLICATION

As SOA is the chosen architecture for the product, there are 4 stages/tiers of testing to be performed.

Tier 1

It contains the following levels of testing:

Service Level Testing

- The services in the SOA architecture act as the building blocks of the application as they perform the business logic of the application. It is necessary to test each service as shown above using the Request-and-Response protocol.

Functional Testing

- All the services are subjected to functional testing based on their operational needs. This determines whether the response is suitable for the needs of the business.
- For instance, while the platform is recommending opportunities to a student, the recommendation service must consider any preferences the student has submitted and, after comparing them to the open positions, must return 10 recommendations. Testing this behaviour provides insight into the correct working of this service.

- Test cases are created based on the business requirements along with return statements. The processing of these test cases determines the correctness of the application and helps identify incorrect data/responses.

Security Testing

- As depicted in the architecture, a service handles the user's login procedure. This must be tested because it might cause security flaws in the programme. Since we decided to use third-party security suppliers and payment gateways for university and organisation login and subscription payments, encryption is required for data parsing.

Performance Testing

- The application is expected to be used by numerous students and business contacts who will be accessing it and requesting resources simultaneously. As a result, the services must scale to handle more traffic, and response times for all these services must be consistent under peak loads. Such scenarios can be simulated and tested to determine the performance of the system.

Tier 2

This tier consists of **Process Testing**

• This evaluates the application's numerous business procedures.

• Simulators are frequently employed to produce test input data and carry out output validation.

• The data flow from various layers is checked for optimal performance after integration.

Tier 3

End-to-End Testing

- The analysed functional and non-functional requirements are tested as per the business needs
- All pertinent services will be merged once developed, and their functionality must be verified after integration.

Tier 4

Regression testing

As the client has plans to expand the features and geographical bounds of this software, updates and changes are anticipated as the product matures and might lead to the emergence of new faults and/or re-emergence of old faults. To evaluate the system stability through incremental builds and releases, regression testing is performed on all the application's functional and non-functional requirements.

3.5 ENTITY RELATIONSHIP DIAGRAM OF THE RELATIONAL DATABASE

To understand how the elements of the database interacted and relate to each other within the confinements of the system, we decided to build an Entity Relationship (ER) diagram representing the flow of information of processes across the system.

The **Entity-Relational diagram** depicts the entities, attributes, and relationships of the application at a database level with appropriate multiplicity constraints. The model assumes **four identities: university, student, organisation, and contacts.**

A university is **responsible** for matriculating students, granting them access to the application, and submitting subscription fees. A university can hold any number of students. Student profiles hold necessary information such as transcripts and position preferences as input information that are fed to the recommendation algorithm so that both students and the organisation contacts receive information tailored to their interests. The profiles are also visible to the organisation's contacts to evaluate during the application process. Students can access any number of opportunities posting, save any number of posting that is of interest or apply for any number of opportunities.

An organisation is **responsible** for creating opportunities, updating its status, and submitting subscription fees. The organisation account is used by the allocated managers within the organisation as a shared account. Each organisation is represented by at least one and up to five contacts, who can access any number of students' information and update the opportunities. Different from the organisation account, the contacts are

responsible for directly communicating with the students and reviewing applications submitted to the belonging organisation.

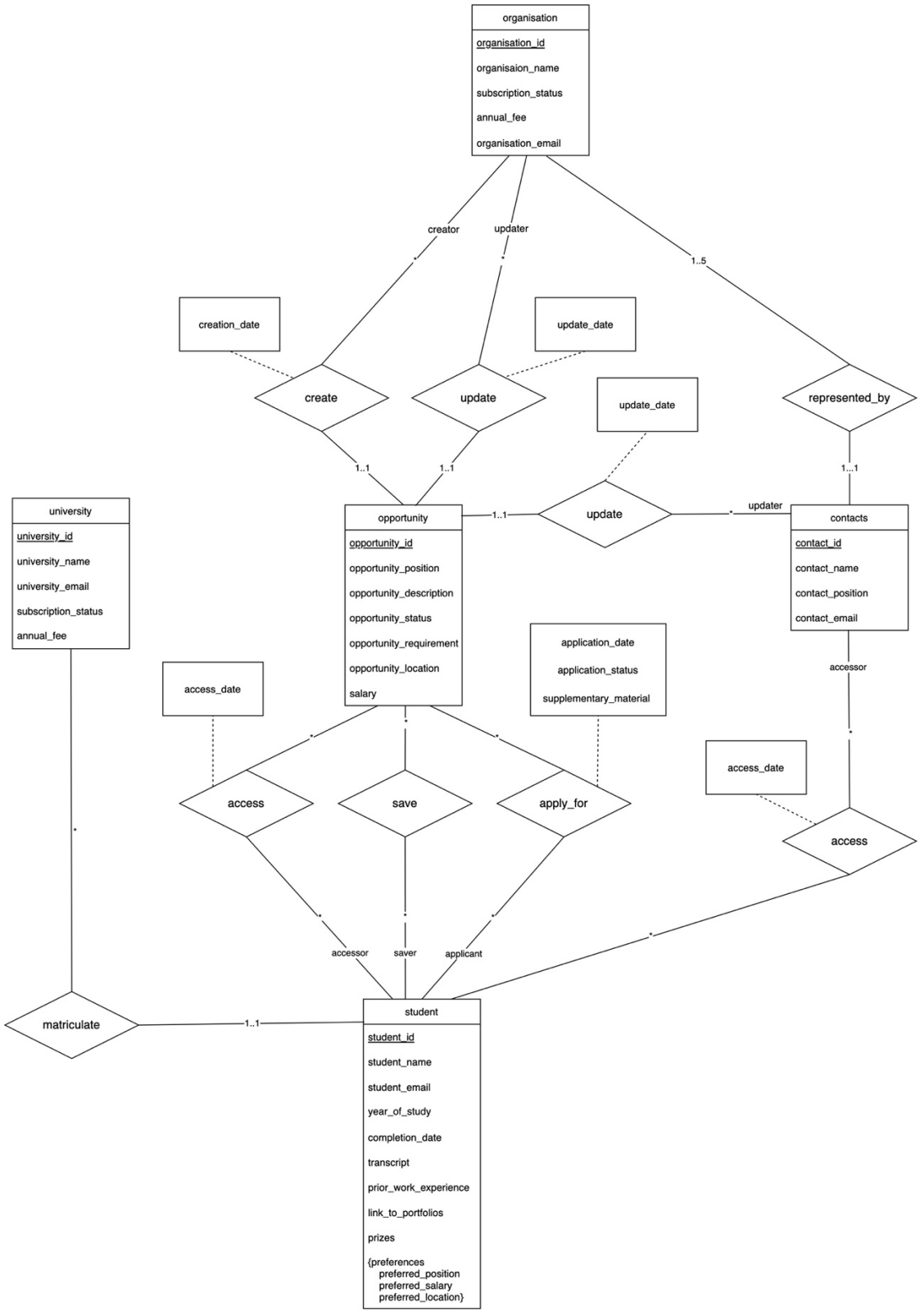

Figure 3.2: Entity Relationship Diagram representing the entities' interaction in the database

Part of the functionality is not captured by the E-R model. For example, students would only be able to access active opportunities; however, the diagram would need to include two types of opportunity entities (active and inactive opportunity) to show such differences, and this would greatly increase the complexity of the graph. It is more reasonable to include an "opportunity_status" attribute and actualise such a feature during later implementation.

Similarly, the **recommendation functionality would be implemented as algorithms** rather than stored as data, since storing recommendations at a database level means the recommendation of each entity needs to be reconsidered during the addition or update of every other entity, which brings unproportionally growing activities.

To facilitate functionalities such as recommendation and searching, the related data when a student applies for or accesses an opportunity are also stored. For example, managers at an organisation may want to know how many students have seen the posted internship, and what percentage of them are applying for this opportunity to **evaluate the attractiveness and effectiveness of the posting.**

Such scenes require each access from a student to an opportunity to be stored in the database. The application would also use the data stored to generate a summary report for the client company.

3.6 SYSTEM CONTEXT AND INTERACTIONS

The interactions between users and the system are depicted in the use case diagrams that follow. All possible system interactions are covered by the set of use cases. During the early design phases, these use case diagrams were helpful.

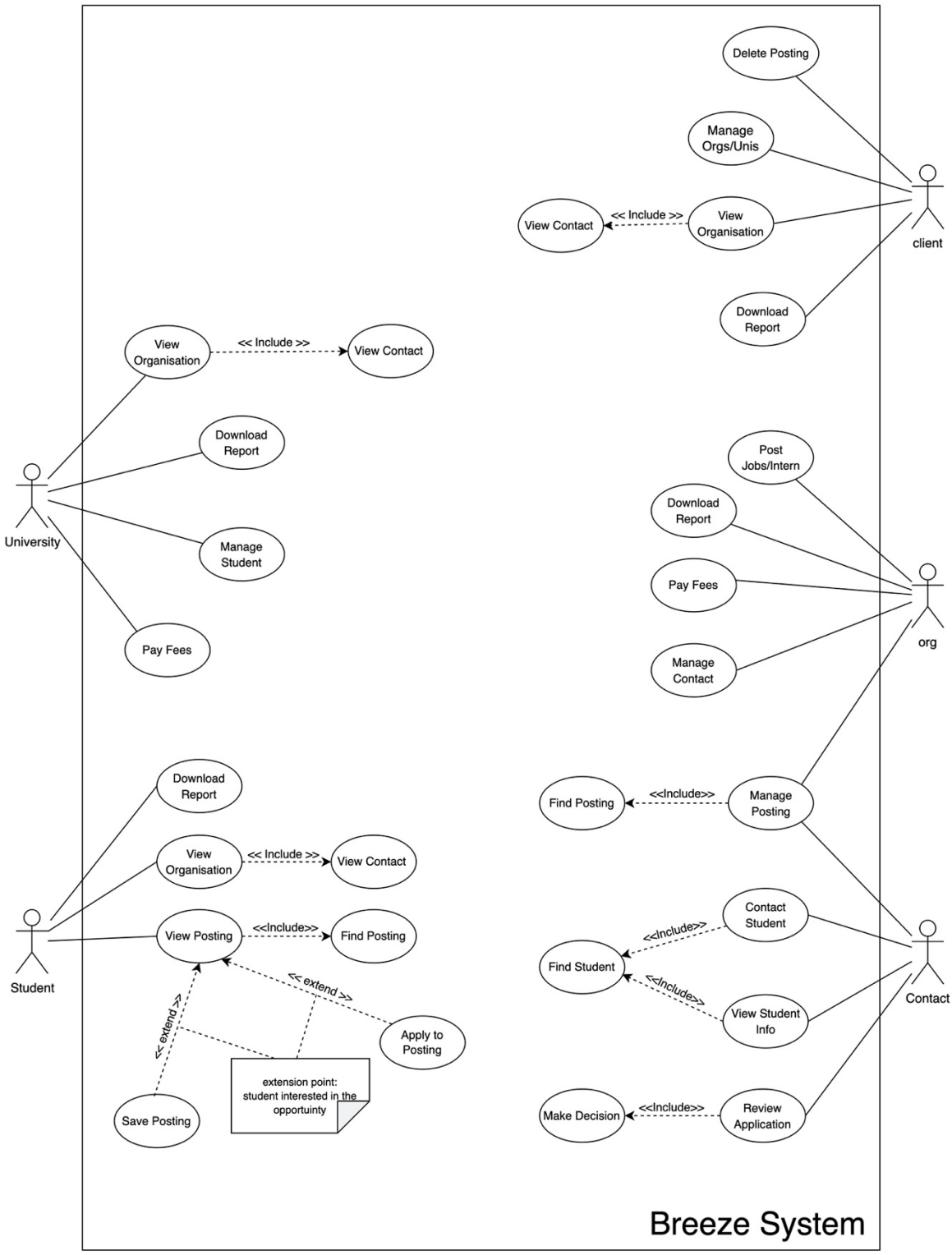

Figure 3.3: Image Illustrating Project "Breeze" Use Cases

DESCRIPTION OF APPLICATION USE CASES:

- **View Organisation** - Enables students to view details of the organisation, jobs issued, etc.
- **View Contact** - Enables users to view information about the contacts.
- **View Posting** - This enables users to view job opportunities posted by the company.
- **Find Posting** - Enables users to search for opportunities.
- **Save Posting** – Enables students to save an opportunity posting of interest.
- **Apply to Posting** – Enables students to create an application to an opportunity posting of interest.
- **Download Report** - This enables users to download reports for the corresponding group of people.
- **Manage Student** - Enables universities to manage students with certain actions, such as adding and removing.
- **Manage Posting** - Enables the client to update the information such as the status of a posting.
- **Manage organisations/Universities** - Enables clients to apply and remove via university/company.
- **Post Jobs/Intern** - Enables companies to post an opportunity.
- **View Self-posted jobs** - Enables companies to see jobs they have posted.
- **Manage Contact** - Enables companies to manage their contacts.
- **Pay Fees** - Enables companies and universities to pay the client.
- **Contact Students -** Enables company contacts to contact prospective students/candidates.
- **View Student Info** - Enables companies to view student/candidate profile information.
- **Find Students** – Enables organisations and contacts to find a specific student.
- **Manage Posting** - Enables organisations and contacts to manage and modify an opportunity posting.
- **Review Application** – Enables a contact to review the application submitted by a student.
- **Make Decision** – Enables a contact to decide on an application.

3.7 STRUCTURAL MODEL OF THE APPLICATION

The class diagram illustrates the structural design of the system from a specification perspective. The system assumes four types of accounts: University, Student, Organisation, and Contacts, each inherits from a parent Account class.

Organisations are responsible for creating Opportunities, the status of which is maintained by both the Organisations and the Contacts. Students can access the posted Opportunities and instantiate an Application when interested. The Application is reviewed by Contacts and returned to the Students.

In addition, students and contacts can access static functions from two utility classes, Search and Recommendation, to receive or find information that is of interest to them: Organisation Contacts can get Student recommendations that fitted the requirements based on the Opportunities posted by the belonging Organisation, and Students would receive recommended Opportunities that fitted their profile and preferences.

Students are also able to search for Opportunities posting with specific information and filter the list of results Opportunities, while Contacts can search for Students with certain skills or certifications mentions in the profile.

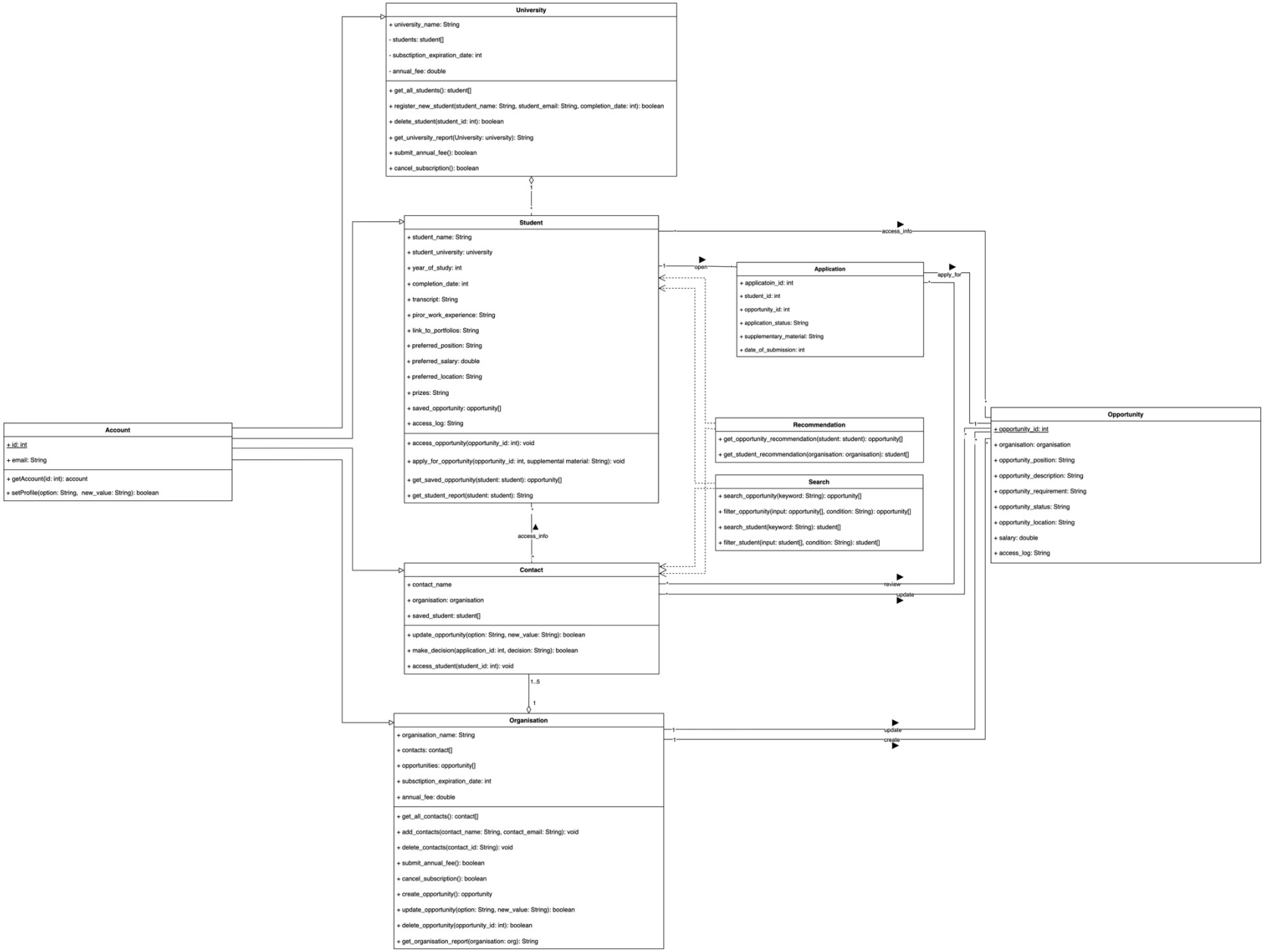

Figure 3.4: Image illustrating classes and associations in the application

3.8 SEQUENCE DIAGRAM OF STUDENTS APPLYING FOR JOB OPPORTUNITIES

The following sequence considers the job application posting by a student. The team has utilized a sequence diagram to show the interaction between students and the system. In particular, in the sequence given below, we establish the interaction of the student with the recommendation, search and filter, and opportunity services when applying for a job or internship posting. The sequence also shows the result of the application by indicating the interaction with the organization's contact.

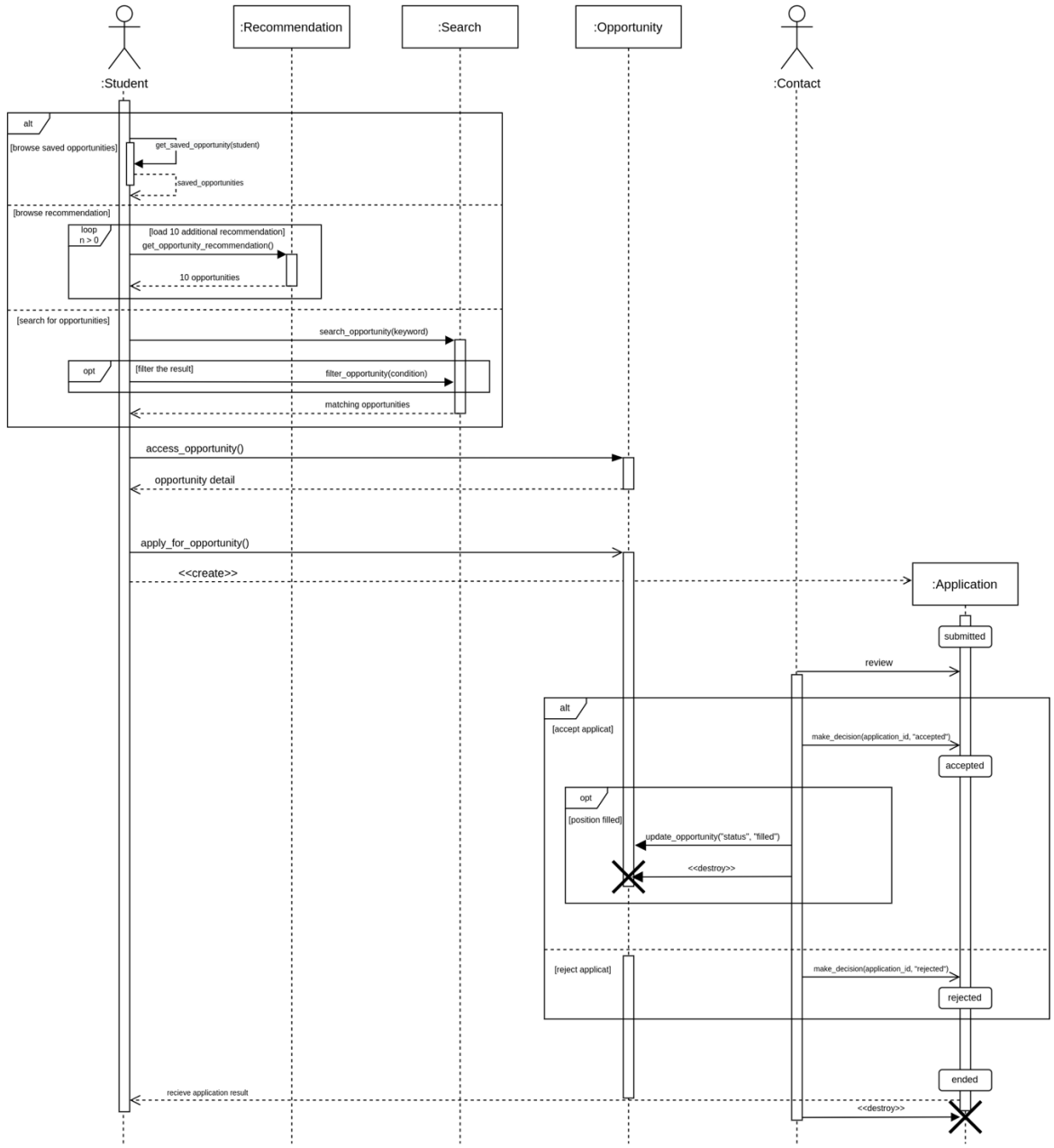

Figure 3.5: Image Illustrating the sequence Diagram for student job opportunity case

Use Case Name	Student Application
Goal in Context	A student finds an opportunity, applies for it, and receives the result.
Preconditions	1. The student has not completed the programme, has already registered an account, and has logged on to the platform. 2. There is at least one active opportunity on the platform.
Primary Actors	Student
Main Flow	1 The use case begins when the student tried to find an opportunity. 2. If the student accesses the saved opportunities, 2.1 The system gets the list of opportunities saved by the student and shows them on the platform. 3. If the student accesses the opportunity recommendation. 3.1 The system generates 10 best-matched opportunities according to the requirements of opportunity position, student information and preferences. 3.2 (loop) If the student needs to view more opportunity recommendation, the system generates 10 additional matching opportunities. 4. If the student uses the search functionality to find an opportunity with keywords, 4.1. The system generates a list of opportunities that contains the keyword within the description, position, or location. 4.2. The system shows a flited list of opportunities if the student needs to see opportunities that satisfy certain conditions. 5. The student accesses an opportunity and views the details. 6. The student opens an application to an opportunity and submits the required materials. The application status is "submitted" 7. The contacts from the organisation review the application. 8. If the contacts decided to accept the applicant, 8.1 The application is successful, and the contacts update the status of the application as "accepted." 8.2 If the position is filled and no longer accepts more applicants, the contacts update the opportunity status as "filled by student." The opportunity becomes inactive and would be hidden from other students. 9. If the contacts decided to reject the applicant, 9.1 The contacts update the status of the application as "rejected." 10. The student receives the application result. 11. The application has ended.
Postconditions	1. The student has not left the university before the application ends. 2. The organisation has not withdrawn the opportunity before the application ends. 3. The organisation and the university continue the subscription during the application.
Alternative Flows	Student Completed the Study during Application; Opportunity Withdrawn; Application Withdrawn; Subscription Expired; No Matching Opportunity.

Table 3.1: Use case of a student applying for an opportunity.

4 PHASE FOUR - PROFESSIONAL CONDUCT

4.1 ETHICAL CONSIDERATIONS

Every element of our profession is impacted by technology, including how we handle client information and interact with them. The ability of Project "Breeze" to carry out noble deeds can frequently be abused if caution is not used. This calls for ethical considerations as highlighted below where Project Breeze and its resources could cross an ethical line.

- There could be an **oversight** where the team can structure its offerings to provide better value to job seekers and employers.
- An individual could **create a false identity, misrepresent their identity, and bait other users** into sharing data. The team has taken steps to avoid **phishing attacks** to ensure such cases would not happen in the application by applying a **verification process**.
- Although summary reports are gathered by the client company and the team to track programme adoption and efficacy, they could be seen as a **vast user data collection**. All data generated by the application would be given **randomly assigned identifiers**, disguising the user and their data usage, to prevent privacy infringement.
- An individual may inadvertently post anything containing software viruses, worms, or **other harmful code**. To avoid such cases, the team had produced **a flagging condition** where users can flag such posts to bring it to the team's attention where it would be reviewed and if found in violation would be taken down from the application and an open investigation would be launched against the individual.
- Our team implemented **multi-layer security** from the very source layer of the application to **avoid** reverse engineering, decompiling, disassembling, deciphering or otherwise attempting to derive the source code for the Services or any related technology that is not open source.

4.2 CODE OF ETHICS

The rise of the internet and mobile computing are increasingly impacting our daily lives. However, **the moral line is often blurred and crossed in the desire for maximum profit and monetization**. Major technology companies have been revealed to adopt unscrupulous and unethical practices violating user privacy. Given the rapid pace of innovation and change in our sector, it is critical to keep in mind that we exist to leave an impression, and our behaviour supports this in a way that is **consistent with Project Breeze's principles.**

Our standards provide helpful advice for resolving frequent ethical dilemmas, ensuring that team members always have access to the most recent ethical information.

4.2.1 OUR CULTURE AND VALUES

Our values are the enduring principles we use to do business with integrity and win trust every day. Our culture dictates how our operating framework – who we are and how we behave.

4.2.2 PRIVACY POLICY

Privacy in the digital world was explained by Steve jobs on Stage at the D8 conference in June 2010, interviewed by Walt Mossberg and Kara Swisher [7]:

"Privacy means people know what they are signing up for, in plain English, and repeatedly. That is what it means. I am an optimist, people are smart. And some people want to share more data than other people do. Ask them. Ask them every time. Make them tell you to stop asking them if they get tired of your asking them. Let them know precisely what you are going to do with their data."

The discussion had a significant impact on our team while we brainstormed the approach of integrating privacy into our application, and we used that insight to develop the following guidelines for the application's privacy policy:

- We abide by local privacy and data protection regulations.
- Whenever we collect or process data, we clearly and accurately disclose our privacy practices.
- We respect their privacy preferences by using user data to deliver the services they have authorised.
- By creating secure goods and services, we safeguard our customers' data.

4.2.3 ZERO TOLERANCE FOR RETALIATION

It can often take bravery to always act morally upright, despite potential hardship or discomfort, even if it means standing alone. That is why we do not put up with retaliation. Here are a few examples [8]:

- Refusing to act in a way that contravenes the law, application policies, or standards, even if doing so would cost the business.
- Expressing a genuine worry about potential misbehaviour.
- Assisting with an investigation.

4.2.4 RESPECTING THE LAWS

- The application is built concerning the laws and regulations of the UK at the time of release and soon the places we operate.
- We tend to keep the application very transparent and honest with government representatives and officials
- We will respond truthfully, appropriately, and promptly to government inquires and requests for information as long it does not conflict with **Section 5.2 Privacy Policy.**
- We comply with laws that govern how our technology may be distributed and used internationally.

4.2.5 RESPECT AND PROMOTE HUMAN RIGHTS

As the team continued to discuss the app's potential ethical rules, we realised that the user is the vital component of the system that powers the application and that

upholding and supporting a user's rights is essential to using the app and is beneficial for the platform.

- We respect the rights to freedom of expression and privacy on the platform.
- When we face requirements from governments to provide user data or remove content, we verify whether the government demands are valid, binding, and otherwise comply with the rule of law.
- Respect and enrich global communities by using only voluntary labour.
- Prohibit the use of child labour and all forms of forced or compulsory labour and human trafficking by us and our affiliates, subcontractors, and extended supply chain.

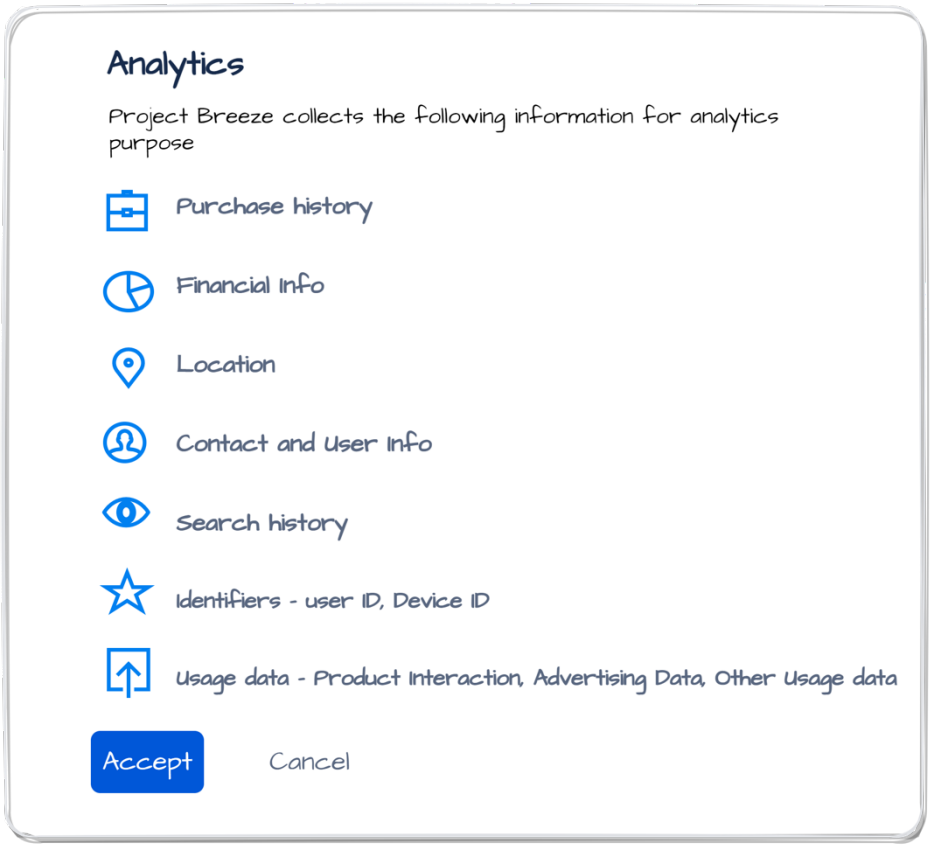

Figure 4.1: Low fidelity wireframe illustration demonstrating the user analytic data presented to a user of the system.

4.2.6 BUILDING TRUST

Trust is one of the crucial factors connecting consumers and businesses; it is necessary to establish and **maintain a long-term relationship** with our audience and aids in

providing better experiences, boosting loyalty, and attracting unaccustomed users. The team discussed how trust **reflects the company's culture** and derived the following:

- We listen carefully to feedback and questions from investors and other stakeholders.
- We make sure that any communications about our products and services are honest and accurate.
- Our advertising and promotional material are accurate and minimised false claims.

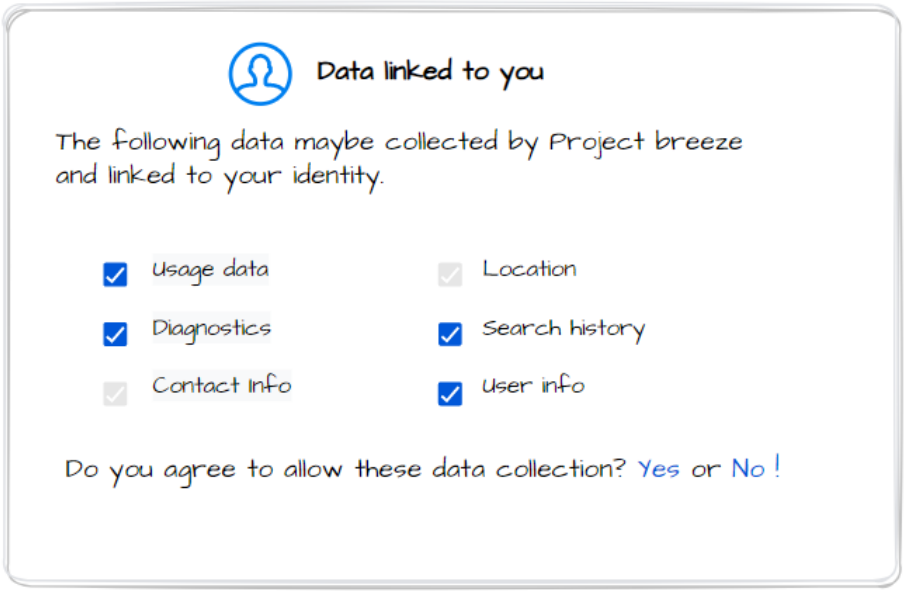

Figure 4.2: Low fidelity wireframe illustration of demonstrating data that is collected from the student indicating App Transparency.

4.3 CONDUCT AS A SOFTWARE ENGINEER

The above sections outline how the code of ethics rules and symbolises the application and corporate morals. The team decided to create a separate piece that only **reflects on the developer as an individual** since with tremendous power comes great responsibility, and sometimes one can forget the way [10].

Security and Privacy are extremely important in software development, we avoided building Project Breeze's code with backdoors or lax security that could be used to track or restrict what users do. Since our programme depends significantly on user

data from employers, colleges, and students, we were obligated to take all reasonable security precautions.

The phrase "**technical debt**," which every software engineer dreads, came up during Project Breeze's development. We worked to ensure that it would be as transparent as possible so that future releases would be based on informed decisions to repay technical debt as quickly as feasible.

Even though the team iterated and pushed final commits, **hidden flaws** will inevitably reappear at some point leading to the application design suffering. The team concluded that any drawbacks after deployment must be identified, resolved, and learnt from as early as possible to prevent repeating them in the future.

As software engineers and users, we believe that **data collection should always be transparent and under the users' control.** Moving forward, the inbuilt policy was to show the data collected by the application to a user and he/she can always choose to withdraw any of the data collected or share them with the developers.

5 PHASE FIVE – DEVELOPMENT ANALYSIS:

5.1 SYSTEM DESIGN ANALYSIS:

The team led a detailed analysis of the given requirements and was able to design a software product that catered to the needs of the project. We have stated in the report that we could not fully quantify the need for notification services as the complexity grew with structuring the other element of this project and therefore, we stated the notification status can be an additional implementation for the client.

We tried to adhere to the code of ethics and have not majorly compromised on quality over quantity. The development is refined from technical debt and there were multiple feedback sessions to properly structure the design until the complete satisfaction of the team.

The team had tried to produce making independent services that are not termed as hard–dependencies or shared resources in terms of failures. If given extended time, the team would like to understand or reconfigure the architecture that contains minimal failure models and high-security layers not that the present architecture is at risk.

5.2 REFLECTIONS ON THE PROCESSES USED:

Role-based Agile process allowed the team to better under the requirements, as the analysis was done from the perspective of the client and developer. Each team member played one of these roles and posted queries based on their understanding. This helped the team to resolve certain ambiguities and come to a common conclusion.

The adoption of the agile process and usage of the Azure DevOps board along with its wiki feature to store our findings helped us track our progress on daily basis. The goal in each sprint was to complete the cycle as soon as possible with useable segments on prioritized works, and the Azure board was reviewed during our scrum stand-up calls in a way helping the team resolve blockers faced by individuals.

Every feedback posted on an individual's work was considered constructive criticism and taken positive reinforcement and completed by each team member within the said time.

Every individual demonstrated their findings to the team and was able to collect feedback for making improvements to their understanding and adding value to the report in a structured manner.

6 SUMMARIES OF CONTRIBUTIONS:

Report Section	Contributor
• Phase 1 and phase 2 analysis • Business canvas • Code of ethics • ANALYSIS	220031985
• Product strategies • Functional and Non-functional requirements • Business model	220018772
• ER Modelling • Use case diagram and its description • Weighing pros and cons of agile frameworks	200007892
• System architecture • Phase 1 and phase 2 analysis • Ethical considerations	220032472
• Structural class diagram • Behavioural sequence diagram • ER modelling	190022154

REFERENCES

[1] **Kanban Disadvantages,** https://www.kodalia.com/blog/article/kanban-disadvantages/, Accessed on 30-11-2022

[2] **HESA Student Records**, https://www.hesa.ac.uk/data-and-analysis/students/whos-in-he, Accessed on 30 – 11 – 2022.

[3] **Companies register activities: 2020 to 2021,** https://www.gov.uk/government/statistics/companies-register-activities-statistical-release-2020-to-2021/companies-register-activities-2020-to-2021, Accessed on 31-10-2022.

[4] **Multi-factor Authentication (MFA) workarounds for China**, https://www.coloradocollege.edu/offices/its/guides/mfa-china.html, Accessed on 29-10-2022

[5] **What Is SOA? Service Oriented Architecture**, https://www.geektonight.com/service-oriented-architecture-soa/, Accessed on 28-11-2022

[6] **SOA vs Microservices,** https://www.ibm.com/cloud/blog/soa-vs-microservices, Accessed on 1-11-2022

[7] **Steve Jobs at the D8 Conference**, https://youtu.be/39iKLwlUqBo, Accessed on 9-11-2022

[8] **LinkedIn Standard of Business Conduct,** https://socialimpact.linkedin.com/content/dam/me/linkedinforgood/en-us/banner/global-stan31dards-of-business-conduct-2017.pdf, Accessed on 9-11-2022

[9] **Unified Modelling Language, v2.5.1,** https://www.omg.org/spec/UML/2.5.1/PDF, Accessed on 14-11-2022

[10] **Code of Ethics**, https://ethics.acm.org/code-of-ethics/software-engineering-code/, Accessed on 9-11-2022