Tony Le Huynh - Work Book T3A1

GCAB012201

Q1.

Source control, also known as version control, is the practice of tracking and managing changes to software code. Source control systems are tools utilised by developer teams to accomplish this (Reference 2). Code changes can be tracked and backed up over time, where the entire revision history of code can be seen. Every modification made to a code base is tracked through source control. This allows for developers to roll back to earlier versions of the project if required or view previous changes, to fix any issues while minimising disruption for the developer team (Reference 3). This also means that source control can serve as a safety net for developers, as the source code can be protected from harm or code conflicts and allows for developers to experiment on sections of code (Reference 4). Through source control systems, multiple developers are able to work and collaborate simultaneously on the same code base, without causing any conflicts. Therefore, source control systems are important for teams working on large projects.

The source control process can vary depending on the requirements of the project as well as the capacity and procedures followed by the developer team/company. Below is an overview of things to consider in regards to the standard source control process.

1. Selecting a source control system that is suitable for the project

There are two main categories of source control systems - centralised and distributed (Reference 2). With centralised systems, the entire project is stored on a central repository located on a central server. These type of source control systems are normally used in projects where the developer team needs to share code and track changes (Reference 4). A widely used centralised source control system is Subversion.

For distributed source control systems, the entire project and all of its history is mirrored on each developer team member's computer (Reference 2). Each developer team member can access a repository from multiple locations and computers (Reference 4). Distributed source control systems are typically used in standard practice. This is because they are faster than centralised systems due to not having to account for the performance of the network. Teams also don't have to be connected to the network in order to be able to work on the project when using distributed source control systems, as every team member has a copy of the project and its history saved on their computer (Reference 2). Widely used distributed source control systems include Git and Mercurial.

Git is the distributed version control system that is the most widely used. Benefits of Git include that it was designed with performance, security and flexibility in mind as well as being well supported, open-source and free (Reference 2). Thus, Git is the most suitable choice for most projects and developer teams today.

2. Establish a repository

A central repository is to be created to store the codebase for the project as well as its version history. As part of the distributed source control system workflow, developers will clone this central repository to their local computers in order to work on the code. Central repositories can be hosted on widely used git platforms such as GitHub, BitBucket, GitLab, GitBucket and Sourceforge (Reference 1). There are also cloud platform-specific git providers like AWS CodeCommit and Google Cloud Source Repositories which are useful for teams that commit to using a single cloud platform (Reference 1).

3. Defining a branching strategy

A branching strategy must be established for the developer team, in order to manage concurrent software development. This strategy will be dependent on the project and team requirements. Source control ways of working are also tied to how teams wish to conduct their testing and deployment of project software. The two current dominant ways of working are GitFlow and Trunk-Based Development. Whilst the choice between the two is once again dependent on the team and project requirements, trunk-based development is typically considered best practice (Reference 2).

Here is an overview of GitFlow development as a branching strategy (Reference 5):

- **Establish main branch**: The “master” or “main” branch is established which represents the production-ready source code for the project. A “develop” branch is also established, serving as the integration branch for ongoing feature development
- **Cloning repository**: Each member of the developer team will create a local copy (or clone) of the central code repository
- **Creating feature branches**: The developers will create their own “feature” branch in order to work on a specific feature of the project code. This process helps to isolate their work from the main codebase. A branch is a pointer to a commit. Branches make it so it is more convenient to work on sections of code in different contexts - such as specific features or bug fixes (Reference 7).
- **Committing changes**: Overtime, developers will commit their changes to the feature branch they created. Each commit is small and entails an informative commit message.
- **Testing and reviewing**: Developers will test and review their changes locally in a test environment to ensure the desired outcomes are met
- **Pushing changes and creating a pull request**: Once the changes are ready for review, the developer will push their local feature branch to the remote repository and

create a “pull request”. This will request to merge their changes into the “develop” branch of the project codebase.

- **Peer review**: This pull request will be reviewed by other members of the developer team, where edits, changes and feedback can be suggested as per coding standards and best practices for the project.
- **Merge changes**: Once any changes and feedback are finalised, the pull request is approved and the changes are merged into the “develop” branch of the main project codebase. This will help keep the main codebase up to date with any features or bug fixes.
- **Releases and hotfixes**: As part of the GitFlow process, there are “release” branches which are for upcoming releases as well as “hotfix” branches which are for addressing urgent production issues.

GitFlow was popularised first, and is seen as a stricter development strategy where only select members can approve changes to the main code, in an effort to maintain code quality and reduce bugs (Reference 2).

Trunk-based development follows a similar flow to the process of GitFlow development outlined above, however there are some differences as well (Reference 2). Trunk-based development is a source control management practice where small, frequent updates are merged to a core “trunk” or main branch (Reference 2). It is a more open model as all developers on the team have access to the main code, which allows for teams to iterate quickly and implement Continuous Integration & Continuous Delivery (CI/CD) (Reference 2).

GitFlow as a strategy utilises multiple long-lived branches such as “master” or “main”, “develop”, as well as multiple primary branches. There are longer-lived branches and larger commits with Gitflow as opposed to trunk-based development. The feature branches created by developers are usually only merged into the main branch once the feature is complete under the Gitflow model. These long-lived branches require more collaboration to merge, as there is a higher risk of introducing conflicting updates and can lead to delayed integration of features (Reference 2). There are multiple primary branches for development, hotfixes, releases and features under Gitflow. With Gitflow as a branching strategy, there are more branches to manage meaning that there is usually more time and complexity required for planning and review by the developer team (Reference 2). GitFlow separates the development process and release management with dedicated branches for each release and hotfix

On the otherhand, Trunk-Based development is more simplified as an approach where the primary focus is on a single, shared branch called the “trunk” or “main” branch as the source of fixes and releases (Reference 2). This can make it easier to understand and follow for developers. The complexity of branching and merging is minimised with this approach, thus aiming for faster integration and reduced merge conflict. Under the trunk-based development model, the main branch is considered and aimed to always be production-ready, stable, has no issues and is ready to deploy at any commit.

With trunk-based development, there is faster and more frequent integration of changes. Developers commit directly to the trunk/main branch or merge short-lived feature branches quickly (Reference 2). These are done on a frequent and daily basis, resulting in a steady stream of commits flowing into the main code branch.

Once a branch is merged, the best practice is for it to be deleted (Reference 2). The number of active and inactive branches for repositories is kept to a minimum with trunk-based development. A large number of remote branches can be difficult to manage for developer teams.

Essentially, trunk-based development entails that development is done in small rapid batches, to quickly deliver code to production. CI/CD tools (Continuous Integration and Continuous Development) are also used to help achieve this. The benefit of this approach is that it's much easier for teams to discuss and make quick decisions when reviewing a smaller amount of code from the small frequent changes made, as opposed to large amounts of changes (Reference 2).

Feature flags are also used as part of trunk-based development. This enables developers to wrap new changes in an inactive code path and activate it at a later time, thus being able to commit new feature code directly to the main branch within a feature flag path as opposed to having to create a separate repository feature branch (Reference 2).

Due to the frequent committing of code to the main branch, automated tests are set in place to validate code quality to enable continuous integration. Once again, this is to ensure the main branch is ready to deploy at any time. Code reviews are also performed immediately as part of this trunk-based development, with the automated testing in place as a first layer for review (Reference 2).

In summary, the standard practice is to use trunk-based development as it provides the ability for regular and frequent software release, and being more suitable for rapid development cycles (Reference 2). GitFlow would be more suitable for projects with strictly scheduled and separated releases.

4. Committing code changes

As mentioned in the section previously, the standard source control practice utilising trunk-based development entails that small and frequent code commits are made, with clear and informative messages. This will allow for the progress of the project to be more easily tracked, and for bugs to be easier to be identified and fixed.

If any critical issues are found in production, these are also to be immediately addressed and resolved.

5. Code review and collaboration & Continuous Integration and Testing

As per trunk-based development practices, automated tests are to be set up to immediately review and approve code that is committed daily by developers. CI/CD tools can also be used to help automate the building, testing and deployment of the code. Once tests are passed, code is to also be immediately reviewed by other members of the developer team in order to ensure that code quality and standards are met, and that any feedback and changes can be made and addressed.

6. Merging and conflict resolution

Under the trunk-based development model, code is committed directly to the main branch. The aim is to ensure that the main branch is always production-ready and ready for deployment at any time. If there are any code conflicts that arise from the commit, these will need to be addressed.

7. Documentation and Training

As mentioned prior, the source control process can vary depending on the project and team requirements. Therefore, it is important that clear documentation is created which outlines the source control process used for the project, as well as coding standards and best practices. Along with the documentation, training can also be implemented to ensure that new and current members of the developer team are up-to-date with the source control process used by the team for the project.

Q2.

There are several important aspects of quality software.

1. **Functionality**

Functionality describes how well software meets its intended purpose, performs the tasks that the software was designed for and satisfies the requirements and specifications of the user and project (Reference 7). This is especially considered in terms of the practical use of the software, from the user's point of view (Reference 8).

- The software should work exactly as described in the specifications. It should accurately produce the correct intended results consistently (Reference 9). There should be no defects in the software

- Software should work accordingly with the user's expectations and be suitable for its intended tasks (Reference 8).
- The software should also be able to interact with other systems of components seamlessly.

Functionality can be achieved through comprehensive planning and mapping out the requirements for the software. Code review as well as automated testing can also help ensure the functionality of software.

2. **Reliability**

Reliability describes the ability for the software to perform consistently and reliably. Under different environments and conditions, the software should produce the correct results consistently (Reference 11).

- Errors and exceptions should not occur, and if they do they should be handled gracefully, and proactively without causing data loss or disruption to the user experience (Reference 10).
- If there is a crash or failure, the software should also be able to recover quickly and efficiently.

3. **Usability**

This refers to the overall user experience when using the software. Usability refers to how easily the software can be used and learned (Reference 10).

- The User Interface for the software should be intuitive, clear and visually appealing
- The software should be easy to learn, and should not be difficult for users to understand the system. There should also be comprehensive documentation and onboarding resources such as tutorials for the user (Reference 10). It should be accessible for people of different types of abilities (Reference 8).
- The software should be accessible and operational when required, whilst meeting the user’s expectations and requirements (Reference 11). This is to ensure a positive user experience and satisfaction with the software.

4. **Efficiency**

Efficiency refers to how efficiently the software utilises system resources such as processor capacity, disk space, memory and CPU whilst delivering an optimal performance (Reference 8).

- Efficient software should use as minimal system and physical resources as possible whilst accomplishing its intended use (Reference 11). This is to avoid negative impact on overall system performance, where software that consumes excessive resources could

result in slowing down a user's computer. Resources such as memory, processing power, CPU, battery should be minimally used by efficient software (Reference 10).
- Efficient software will also perform tasks and respond to user input promptly, thus guaranteeing optimal user experience (Reference 8).

5. **Maintainability**

The maintainability of software refers to how easily the software can be modified, updated or maintained in the event of changing software requirements or fixing defects (Reference 9).

- It should be easy to add, modify or update software components. Quality software should be designed with modular components and flexible architecture, which can accommodate for possible future changes. Modularity is where a software system is organised into discrete, individual components that can be developed, tested and maintained independently, thus promoting the separation of concerns and minimise dependencies between components (Reference 10). This makes software much simpler to understand for developers as they can focus on and modify individual software modules without affecting the entire system (Reference 11).
- Software code should also be well-written and easily readable, as well as following established coding standards and best practices (Reference 9). This makes the software much easier to understand for developers looking to understand or make changes to the software.
- Furthermore, quality software should be easy to modify without causing any flow-on negative effects to the system or introducing any new unintended defects.

6. **Domain Boundaries**

Quality software should have domain boundaries, which represent the clear separation between different areas of concern within a software system (Reference 11).

An example of this would be where different files in a folder structure of an application have different purposes. As opposed to the case where all code is located on only one single file and the code for the features of the application are not kept separate.

Quality software has clear established domain boundaries. This allows for developers to create a more organised system, where each domain of the software can be developed, tested and maintained independently (Reference 10).

7. **Testability**

Testability refers to how easily the software can be tested, through means such as automatic or manual testing, in order to ensure software quality (Reference 7).

- A software system should be easy to test, and allow for any defects to easily be found (Reference 8).
- Quality software should be designed to facilitate testing. This includes testing processes such as unit tests, integration tests, and end-to-end tests, which work to ensure the software is working as intended and meets quality standards (Reference 8).

8. Portability

Portability refers to how usable the software is in being able to run properly on different platforms, operating systems or devices (Reference 10).

- There is a diverse range of user needs. Thus, quality software should be compatible with a wide range of different devices, configurations and operating systems (Reference 10). The software should also be easily installed and used on these devices or systems. This also means that the software should be designed with this in mind, allowing for the software to adapt to different environments easily.

9. Security

Security refers to how resistant and protected software is in the case of cyber attacks or security risks (Reference 8).

- Quality software is designed to be able to protect, prevent and block malicious or unathorised attacks which can compromise the software system (Reference 9).
- Quality software should consider and utilise the security features of authentication, authorisation, data processing, data encryption and logging in order to maintain the integrity of the software and any stored data (Reference 8).

Q3.

A MERN stack application is a web application with JavaScript as the primary programming language (Reference 12). The four main technologies utilised in a MERN stack application are MongoDB, Express, React and Node.js.

Here is an outline of the technologies of a full-stack application built using the MERN stack:

1. Frontend (React)

ReactJS is a front-end web framework & library that was built and maintained by Meta (Facebook) (Reference 1). Complex and dynamic user interfaces (UI) can be built with React through using simple components which can be reusable. These can also be connected to data on the back-end server of the application, and then rendered as HTML (Reference 13). React

provides a declarative programming style, and has great support for forms, error handling, events, lists and more (Reference 13).

2. Backend (Express & Node.js)

NodeJS is a runtime environment for using and running JavaScript outside of a web browser and on the server-side (Reference 1). It is built on the V8 engine, which provides optimal performance and scalability. Several built-in modules come with NodeJS for accomplishing tasks such as handling HTTP requests and file systems (Reference 12). Being able to write server-side code using JavaScript makes it easier for developers to transition from client-side to server-side programming.

Express is a web framework for NodeJS and runs inside a NodeJS server (Reference 13). Express provides tools and functionality for tasks such as handling HTTP responses and requests as well as URL routing, thus allowing for developers to create APIs and easily manage server-side functionality (Reference 13).

3. Database (MongoDB)

MongoDB is a non-relational, NoSQL document-oriented database that provides support for storing data in a JSON-like format (Reference 3). MongoDB comes with a flexible data model, enabling storage of unstructured data as well as providing full indexing support and replication with rich and intuitive APIs (Reference 3). The benefits of MongoDB include that (Reference 12):

- It's easy to index documents meaning that it has a fast performance,
- It is scalable and flexible
- It is able to handle and store large volumes of structured and unstructured data
- It can be run over multiple servers
- It is simple to use and setup

The structure of a MERN stack application can also be organised as shown below:

<u>Client-side (React):</u>

- Components: Reusable and dynamic front-end UI elements such as buttons, forms and navigation bars can be built using React
- Pages: Front-end views or pages of an application (which can represent different functions of sections of the application) can be built using React
- Routing: Libraries such as React Router can be utilised to navigate between different pages or views in the web application
- State Management: The application state can be handled using React

- API Calls: Data from the backend API and database can be retrieved using HTTP requests. JSON documents created in the React front end can be sent to the ExpressJS server and then stored in MongoDB (Reference 13). HTTP Requests from the React front end can also connect with ExpressJS functions which in turn interact with the MongoDB database to retrieve or update data.

Server-side (Node.js and Express):

- API Endpoints: ExpressJS can be used to create routes and controllers that handle incoming HTTP requests and return JSON data
- Middleware: Express also provides functions that can process requests and responses such as handling authentication, logging, or error handling.
- Database Connection: The MongoDB Node.js driver can be utilised to establish a connection to the MongoDB database.
- Express also provides middleware modules, such as Passport.js, which support user authentication techniques such as JWT (JSON Web Tokens) or session-based authentication as well as authorisation. This allows for additional layers of security for the MERN stack application.

Database (MongoDB):

- Database Models: Data schemas and models can be defined using MongoDB collections
- Documents: Data is stored as individual documents within collections in a JSON-like format (BSON).
- Collections: Related data is grouped into collections, much like how tables work in the case of relational databases.
- Indexing: Indexes are created for frequently queried fields. This helps to optimise database performance and allow for faster query execution.
- Aggregation: MongoDB has a built-in aggregation framework which allows for data aggregation and transformation

Q4.

There is a vast range of skills and knowledge required by developer team members, in order to create a quality web development project for a small business. Below is an overview of required skills and knowledge:

1. Quality software design and principles

This includes knowledge of what makes quality software for application. Knowledge of standard coding practices as well as application software design industry standards would prove useful in ensuring that the team is aware of what is the industry standard to follow for their software. For example, coding conventions for programming languages should be followed such as the PEP8 guideline for Python as an example.

Also it is important for the developer team to have an understanding of the aspects of quality software. These include aspects such as functionality, reliability, usability, efficiency, maintainability, testability and portability (Reference 8). This way, before coding has even begun for the project, the team has an understanding of principles to follow to create high quality software for their web application.

2. Frontend Development

Frontend development involves the creating, designing and coding of the visual aspects of the application that the user sees and interacts with - specifically the user interface (UI) and user experience (UX) (Reference 12).

HTML: Knowledge and proficiency of HTML (HyperText Markup Language) is essential as this is the standard markup language that is used to create the structure and layout of web pages (Reference 17).

CSS: A knowledge of CSS (Cascading Style Sheets) is also important. This is the style sheet language typically used for the styling, appearance and designing of web pages, such as the aspects of colours, fonts and layout, as well as responsive design for different screen sizes and devices (Reference 17).

JavaScript: JavaScript is a programming language which allows for developers to add interactivity, dynamic content and animations to the web application being built by the small business (Reference 17).

Frontend Frameworks/Libraries: Knowledge of frontend frameworks of libraries such as React, Angular or bootstrap can help streamline development and aid in the process of creating the user interface for the web application (Reference 17).

3. Web Design and User Experience (UX):

UX Best Practices: It is important that the team are familiar with the best practices for UX, as this plays an integral part in how the user interface of the website suits the requirements of the business's users and how intuitive it is to use (Reference 17).

Responsive Design: It is also important to understand responsive design principles to ensure the website can be adapted to different screen sizes and devices (Reference 17). This also includes familiarity with mobile support, ensuring a solid user experience on mobile devices when factoring in things such as viewport configurations and touch interactions. Responsive design also includes how interactive the website is to ensure optimal user experience.

Design Principles: It is important the team have an understanding of design principles for websites. This includes aspects such as colour theory, typography, and website layout (Reference 18). This is important to ensure the website created is appealing for users.

Wireframing: This skill is important for the Team dealing with the user interface (UI) and UX of the web application. Wireframes and other prototyping tools can aid in the planning and visualising of the website layout, prior to the coding of the application (Reference 1).

4. Backend Development

Backend Development refers to the server-side development and all the behind-the-scene tasks that run in a web application (Reference 19). This includes a focus on databases, back-end logic, APIs (Application Programming Interface), website architecture and servers (Reference 18).

Database Management: It is important that the team is familiar with database management systems, such as MongoDB, PostgreSQL and MySQL. This includes an understanding of relational and non-relational databases and how they work. Database management systems are important for web applications as they are used to store and retrieve data in an efficient way (Reference 18).

Server-Side Programming: It is important that the team have proficiency in a programming language that is run on the server-side, which includes languages such as Java, Python or a web framework such as Node.js for JavaScript applications (Reference 19). These languages are used for the development of the web application backend functionality, such as tasks including processing user input, authentication, interacting with the database and handling data storage.

Web Frameworks: The knowledge of using web frameworks such as ExpressJS or Flask can be useful to help simplify and streamline the development of server-side logic and API endpoints in the backend of the web application (Reference 19).

APIs: Application Programming Interfaces (APIs) are essential in the communication between the frontend and backend components of a web application. HTTP requests and verbs are made through APIs, to create, read, update and delete data and communicate with a database. Knowledge of APIs and how they work is essential for the team when creating the web application.

5. Security

It is important the IT Team have a knowledge of security principles and best practices to ensure the web application is protected from any potential threats (Reference 18). This includes principles such as authorisation, authentication, data processing, data encryption and logging. Industry standard coding practices can be utilised so that the design of the web application factors in the aspect of security. This includes things such as ensuring users are authorised and authenticated before being able to access and use the web application. Furthermore, knowledge of services outside of the web application, such as firewalls, will prove handy as an additional layer of security for the website and any data stored. Cyber threats are very important to take seriously in this modern age of the internet, and knowledge of security and how to prevent and deal with cyber attacks will be important for the IT team.

6. Server management.

It is important the Team have an understanding of how servers work as well as the fundamentals of server management. This includes things such as configuration, monitoring and maintenance (Reference 8). Knowledge of server management is important to ensure that the web application's backend infrastructure can be run reliably.

7. Version Control

Knowledge of version control (source control) as well as version control systems is important for the Team. This is so that changes to the software codebase can be tracked and managed. Team members will be able to collaborate on and individually work on the software codebase. Code changes can be tracked and backed up, thus allowing the Team to revert back to earlier versions of the code base or view the history of previous code modifications (Reference 3). The Team will need to be familiar with a version control system, such as Git, as well as a branching strategy to follow for their project, such as Gitflow or Trunk-Based Development (Reference 2). Knowledge of these version control principles will aid in the development and deployment of the web application.

8. Project Management

It is essential the Team are familiar with industry standard project management methodologies for software development. For example, Agile project management methodologies, such as

Scrum and Kanban (Reference 2). Following a software development project management methodology will ensure that the development of the web application will be organised and managed effectively.

Furthermore, it will also be important that members of the team are well-versed in using collaboration tools. This includes tools such as Trello, Asana, Jira, Confluence, Slack, Microsoft Teams and Zoom. These tools can be used to ensure that the project management can be streamlined, and that communication between team members can be clear (Reference 6). For example, Slack and Zoom can be used for Team communication channels. Trello, Asana and Jira are tools that can be used to create and assign tasks for the development of the web application.

9. Soft Skills, Problem Solving and Communication (Reference 19)

Time management skills will be required for the IT team, to ensure that deadlines are met for the delivery of the web application.

Adequate communication skills are also important for the Team. This is to ensure that there is effective collaboration and clear communication between team members, as well as any stakeholders.

Lastly, problem-solving skills will be required for the team. It is inevitable that issues will occur during the development project, and these will need to be properly identified and addressed with effective solutions in order to ensure the web application is efficiently built.

10. Testing & Quality Assurance:

Testing is crucial in ensuring that the software and code for the web application works as intended. This is a vital skill that will be required for the IT Team.

The Team should be knowledgeable in how to write and run unit tests, as these can be utilised to ensure that the individual components and functions of the web application are working as intended (Reference 18). The Team should also be knowledgeable in how to use integration and end-to-end testing, as this will assist in ensuring that the web application works according to user expectations and software requirements - as a whole (Reference 18).

Lastly, it is important that the Team is knowledgeable in the testing of different web browsers and devices. This is to ensure that the website built is compatible with various browsers and devices, and that there is a consistent user experience (Reference 18).

11. Web Hosting:

The IT Team will need to have knowledge in web hosting services. This includes being familiar with the process of deploying the web application to a hosting provider, and ensuring that the website can be accessible to the intended users (Reference 18).

12. Web Performance & Accessibility:

The IT Team will need to have knowledge in techniques that optimise web performance. This is so that a smooth user experience can be facilitated, where techniques like compression and caching play a role in optimising page load times (Reference 17). The IT Team should also be familiar with web accessibility standards to comply to, to ensure that the web application that is being developed will be accessible by people with disabilities as well (Reference 17).

Q5.

One of the projects I recently built was the Flask API Web Server Application. See below the knowledge and skills that were required for me to complete this project:

1. Version Control

For the project, I used Git as the version control system, with GitHub as the platform to host my code repository. The branching strategy used was also standard GitFlow.

I required knowledge of how to initialise and create the main branch for the project's code repository. Furthermore, I used knowledge of the processes involved in GitFlow, which included adding and tracking any changes I made to my local copy of the codebase. These changes were then committed with meaningful messages attached. Any git commits made to my application code were then pushed onto Github. The standard workflow of using Git and Github was something I needed knowledge of, in order to ensure that my application was tracked, saved, managed and backed-up using Git.

2. Agile Methodology

Whilst I was working as the sole developer for the project, without any team members, it was still important for me to employ and follow Agile project management methodology in order to ensure that the development of my application was running smoothly as planned.

As the project did not last for multiple weeks, I did not need to abide by practices included as part of Agile methodologies, such as sprints or retro meetings. However, I did use the concept of daily standup meetings. For each day that I had to code the application, I held a daily standup meeting with myself to go through what I had accomplished the day prior and what needed to be done on that day. This helped keep me accountable to the project deadlines that I had set.

I also used Trello, and needed to have knowledge on how to use this project management tool. Trello was used to create a board for the project. On this board, cards were created for managing tasks with deadlines, for storing notes that I had on specific tasks in the comments sections of the card, as well as for brainstorming where I had placed all my ideas for the application in the created cards. Different columns for the Trello board were created for my tasks, in order to help me keep organised for the project. This included columns such as 'In Progress' and 'Done'. Using Trello helped me overcome any challenges for the application, as I was able to create cards and properly keep track of any issues that needed to be addressed.

3. Time management

Time management was a skill that I needed to utilise. This was to ensure that the deadline for the delivery of the Flask API Webserver Application was met. Time management was also a useful skill for me to have in order to overcome any challenges, as poor time management would mean that any challenges that arose could have been potentially not properly addressed due to lack of time.

4. Planning, Database Schema Design and Creating ERD Diagrams

Before I commenced coding the application, I also needed to ensure that I had completed proper planning. This included features of the application that I needed to make, as well as the deadlines and software requirements attached.

Proper planning also included the fact that I had a clear vision in mind of what purpose my application was serving. I needed to properly identify the problem that the application was fixing and why it needed to be fixed. This helped to inform decisions about the development and functionality of the web application.

As part of the planning stage, I needed to create an ERD (Entity Relation Diagram) for my database. This is so that I had a clear initial idea of what relations would have been needed to be created for my database. Furthermore, I needed to have an understanding of primary and foreign keys for these relations, to ensure that the relationships between tables were properly established with no duplication, as well as the columns to include for the tables. The tool I used to create the ERD for my application was Diagrams.net. With the ERD created for my database, I used my knowledge to implement a normalised database schema design.

5. Database management system

The skills I required for this aspect was being able to use PostgreSQL and run SQL queries. I also needed knowledge of how PostgreSQL worked and how the database for my project would be designed and implemented. I needed knowledge of how data would be stored inside PostgreSQL.

PostgreSQL was the chosen relational database management system for my application, which required me to have proficiency in being able to use it.

SQL is the language used to interact with PostgreSQL, and thus it was another skill that I was required to have. Through SQL commands, I was also able to put in place an authorisation layer to ensure that only intended users were able to access the database and PostgreSQL.

Lastly, I required knowledge of using SQLAlchemy, which is the ORM (Object Relational Mapping) library that was useful for working with PostgreSQL in Python.

6. Python

It was essential that I had proficiency in the Python programming language, as the application was coded in Python and Flask is a Python web framework.

It was also important that I knew how to install Python packages as well as set up the virtual environment for the project. These tasks would be instrumental in the development of the Flask API Webserver Application.

7. Testing

The knowledge of running unit tests was also vital for my application. Tests were written by me to ensure that the code I wrote was working as intended. Testing is a crucial part of the software development process, to ensure that your code and the software application that you are working on is functioning as intended and meet the planned requirements. It also helps identify any issues, so that these can be corrected. This is also important so that any errors for my application were captured and handled gracefully.

8. Flask

To build my recent application, I was required to have knowledge of the Flask framework, including an understanding of its structure, conventions, and how to use its features to build a Python web application and APIs. Flask is a web application framework for Python.

9. MVC Architecture

MVC stands for Model View Controller. It was important that I had an understanding of how this architecture worked, so that I would have a guide to follow in terms of the architecture of my web application. As part of MVC architecture, a Flask application is separated into three main components:

1. **Model** - Where data is managed
2. **View** - How the app's data should be displayed

3. **Controlle**r - Routes commands to the model and view

Having an understanding of and following MVC architecture helped inform how I would structure the code of my Flask Webserver Application.

10. APIs

It was important that I had knowledge of API design, following REST conventions, for the creation of the API for my application. I also needed to ensure that I had an understanding of appropriate HTTP request verbs for various types of data manipulation, such as GET, DELETE, POST and PUT requests. I had to be familiar with the API endpoints of my application, with an understanding of the required data where applicable along with the expected response data. This was so that I was able to cover full CRUD (Create, Read, Update, Delete) functionality for data within the database of my application.

This would also be accomplished by using the functions and routing provided by Flask, which assist with handling HTTP requests and mapping these to specific functions in my webserver application.

11. API Testing with Postman

For the building of my Flask API Webserver Application, it was important that I was able to test my API to ensure that it was working correctly as desired. This required me to gain knowledge of how to use Postman, which is a popular third-party tool used for testing APIs.

12. Flask Extensions and third-party librariies

A wide range of Flask extensions were used to add functionality to the API Webserver Application that was built. These included extensions such as SQLAlchemy, Marshmallow, Flask-Bcrypt and Flask-JWT-Extended.

SQLAlchemy is an ORM (Object-Relational Mapping) library which allows for the Python programming language to be used for database interactions and queries, as opposed to having to use SQL. Knowledge of this library and extension was useful for interacting with the PostgreSQL database through the webserver application.

Marshmallow is a third party library for Python. Knowledge of how to use Marshmallow was required so that the schemas for the data models could be defined in the application, and also ensure that incoming data for the API was in the expected format and contained all the required fields.

Knowledge of how to use Flask-Bcrypt was important to allow for the functionality of encrypting passwords and other sensitive data in the application.

Furthermore, knowledge of how to use Flask-JWT-Extended was important for generating and verifying JSON Web Tokens (JTW), thus adding an additional layer of authorisation for security within the application.

13. Problem-solving

The soft skill of problem-solving was important in the development of the application. It is normal for unexpected issues to arise when completing a project. Problem-solving skills were used to address any issues that arose. This included doing research online, getting assistance from other developers, and also self-troubleshooting in order to solve any problems with coding.

Q6.

One of the projects I completed was creating a portfolio website for myself using HTML and CSS.

1. Version Control

Though it was my first time using version control, Git, for a software project, I did effectively use Git to track and commit the changes to my code for this portfolio website project. Multiple frequent code commits were made and pushed to the GitHub repository for the project.

However, upon further evaluation, there are some areas for improvement. The first being the messages I had used for code commits. What could be improved would be to research guidelines for meaningful messages attached to code commits. Additionally, it would be great to find industry or open-source examples, in order to get a sense of how other developers write their commit messages to properly outline what each commit is achieving.

The issue with my use of messages was that for some commits the message would not be relevant such as when I used the message “final commit of the day”. Instead, it would be more relevant to instead only refer to what that specific commit was involving. Next, some of my commit messages were too long, meaning that either I should have committed a change sooner and as a smaller change, or I needed to simplify the message to get my point across.

2. Agile Methodology

There was a complete lack of project management planning or methodology used for this portfolio project. At the time, I was unaware of Agile project management methodologies. For the portfolio project, I wrote my notes in a separate document and kept track of tasks and deadlines in my head. This of course would not be suitable at all for a case where I needed to work with other team members for the project. Furthermore, keeping track of tasks and deadlines was mentally draining, and I could have lessened the cognitive load if I instead kept

track of this using project management software. Also, there would be a possibility of forgetting tasks.

This lack of using project management methodology also resulted in me completing the project barely before the project deadline. This was due to me underestimating how long certain tasks would take, and not keeping track of tasks in progress or in completion in an organised manner.

To improve upon this, I would make sure to use project management software such as Trello. This would allow me to create a board for the project and create cards which represent the tasks to be completed. These tasks can be assigned a category and deadline for completion, and can be sorted into various columns for ease of viewing, such as ‘In progress’, ‘Blocked’ and ‘Completed’. Doing this would allow me to quickly and easily see a broad and organised overview of the status of tasks for the project. This would have most certainly allowed for the management of the project to be much more smooth and organised. Any notes or ideas that I came up with as well could be recorded on the board, which would act as a centralised location. If I were to have any team members as well, they would easily be able to collaborate with me using the Trello board.

Another thing I would do would be implement the Agile practice of daily stand-up meetings. For days dedicated to working on the project, I would implement stand-ups as a way to go through and refresh on what has been previously accomplished, and what tasks need to be completed for the day. This would work much better than keeping a track of everything in my head, which leads to mental burden and the potential for things to be missed or forgotten.

3. Planning, design principles and wireframes.

As part of the portfolio project, I needed to plan for what pages I would create on the website and what these pages would contain. In terms of the overall aesthetic of the website, I had made the decision making for this based on my own personal preferences. I experimented with randomly generating colour palettes and also being inspired off the colour scheme utilised by the website Notion. There also was not much thought put into the font and styling chosen for the website. This approach was not effective for achieving a visually pleasing website. In terms of knowledge and skill of design principles, this is most certainly an area for improvement. It would be recommended for future projects, to research website design principles such as colour theory, layout and typography. This is so that established industry-standard guidelines can be followed for the aesthetic and styling of the website, rather than the website being built based on “feel”. Those with more experience and knowledge for website design would have more insight on how to create an appealing website for users, thus this is an area that I could most certainly learn from for any future portfolio website projects.

The wireframes and mockups for the planned pages of the website were also completed based on trial and error of what I thought I wanted my website to look like. In the future, I would recommend instead to use and follow the templates provided by third party wireframe design websites, which most certainly would be more properly designed than what I had created.

Otherwise, I would recommend researching UI design or seek help from someone experienced with UI design, to ensure that effective wireframes are designed and created for the website.

4. Testing

When creating this portfolio website, I conducted no testing of the code written, other than a few manual tests to ensure that the website functionality was working as intended. Whilst manual testing did help with testing code, it did not capture all issues as it turned out that HTML validation of my website's HTML document returned several errors. As an area for improvement, I would implement software testing approaches such as unit, integration and end-to-end testing, in order to ensure that software being written was working properly as intended without any errors.

Testing was also done to see how the website would look on various devices. This was done using the "Inspect" action in Google Chrome. This did prove effective as my portfolio website did mostly work as intended on different viewports . However, there was an issue with text sizing on one of the different view sizes. This was something I had missed as I had only performed this testing through the "Inspect" action in Google Chrome on my personal computer, where the size of my personal computer screen did not provide me a completely accurate visual representation of my website. Instead, a way to improve this would be to actually physically view the website on various devices in order to properly see how the website would be displayed on these devices.

5. Quality software design principles

No principles of quality software design were followed when creating the portfolio website. This created issues for me down the line, especially in the software design aspect of maintainability. As part of maintainability, it is important that code can easily be changed and updated, should there be any need in the future to do this. To make this easier, software code should be modularised, where code is organised into individual distinct components or modules, thus promoting separation of concerns.

Separation of concerns was not fully followed for this portfolio website, as the code for the entire page of the website would be located on only one HTML document. Each page of the portfolio website was associated with one HTML document. Although the code for my portfolio website was separated between HTML documents, there was not enough separation of concerns. For example, the code for my portfolio website footer, navigation bar and header was copy and pasted across all the HTML documents. This meant that if I needed to make any changes to footer, navigation bar or header, I would need to go into each HTMl document and make the change, thus resulting in needing to make multiple changes across multiple documents. This is not ideal quality software design as any updates or changes to the software could not be made simply.

Instead, as a suggestion for improvement, I would modularise the code for my portfolio website more. For example, the code for my navigation bar should be located on its own separate file or document, as well as for other aspects of the page such as the header or footer. This means that if any specific function needed to be updated, then changes would only need to be made to a specific file rather than having to do this across multiple files. Furthermore, the code which is modularised on these separate files could be reused in other areas of the application, without having to rewrite the code again.

6. CSS

CSS that was used for the portfolio website did effectively work as intended. The use of flexbox was achieved properly where various page elements would change if the user's screen was minimised, or if the user hovered over a page element with their mouse. The only issue was that the text was too small for various components of the website. The improvement I would set in place for in the future would be to utilise CSS to make the text larger. Also, I would utilise CSS so that there is a minimum for the text size, and that the text could not be reduced any smaller if the user's screen was reduced, and instead shift across the page to accommodate for the reduction in screen size.

7. HTML

There were issues with the HTML code of my portfolio website. HTML validation returned errors for my HTML documents, and the HTML in my documents only contained basic semantic elements. This means that my use of HTML for my portfolio website was not up to standard practices. In the future, I would recommend that I research what proper coding practices for HTML web pages are. I did not follow any standard coding conventions or application architecture for my HTML documents, and instead simply created the features of the website as I went, with no plan. Following proper industry-standard coding practices for HTML documents would provide me a guide to follow, to ensure that I would be writing quality HTML. In my portfolio website, the only semantic HTML tags I used were <header>, <section>, <main> and <footer>. In future pratice, I would utilise more HTML5 semantic elements to ensure that my HTML documents are more accessible and easier to read.

8. Extensions

There were no extensions used when creating the portfolio website as all code was created from scratch. This was not effective as I could have created the portfolio website quicker using functionality provided by third party libraries and extensions. In the future, for a similar project, I would look into researching third-party libraries such as Bootstrap, which provide functionality to streamline the process of writing HTML and creating web pages.

Furthermore, I would recommend improvement through learning JavaScript in order to be able to use front-end libraries such as React. These front-end libraries allow for dynamic user

interfaces and web pages to be built, through using the provided components which can also be reusable.

9. Security.

The portfolio website was built with no security measures set in place. This means that my portfolio website was very prone to any cyber threats and attacks which could compromise things such as my personal information. For a future similar project, I would look into researching and establishing cyber security measures, to ensure that the website can be protected from any cyber attacks. An example of a security measure to implement would be the use of a Web Application Firewall (WAF).

TONY LE HUYNH - GCAB012201 - WORKBOOK T3A1

Q7.

Control flow in programming refers to the order in which the computer executes statements in a program (Reference 21). The default for control flow is where statements are read and executed in order from left-to-right and then top-to-bottom in a program file (Reference 20). Control flow plays a vital role in the behaviour of a program and how code is executed. in JavaScript, there are control flow structures that alter control flow by providing mechanisms to only execute blocks of code if certain conditions are met, repeat code execute and manage the flow of code execution.

There are various control structures, such as conditional statements, logical operators, comparison operators, loops and function calls, which manage and alter control flow in JavaScript (Reference 20). These control structures allow a program to make decisions on which blocks of code are to be executed as the program runs.

Here are some ways control flow can be managed and altered through using control structures in JavaScript (Reference 20):

1. Truthy & Falsy

Values that evaluate to true, when evaluated as a boolean, are known as truthy.

Values that evaluate to false, when evaluated as a boolean, are known as falsy.

Control flow can be altered based on whether statements evaluate to true or false. For example, a certain block of code can be written so that it only executes if evaluated as true.

2. Comparison operators

When comparing two values to each other, comparison values are used which return either true or false (Reference 20). Here are the comparison operators below:

```
== equal

!= not equal

\> great than

\>= greater than or equal

< less than

<= less than or equal
```

For example, the statement 2 != 3, will evaluate as true as 2 is not equals to 3. Additionally, the statement 10 > 20 will evaluate as false as 10 is not greater than 20.

As mentioned previously, comparison operators can be used to affect control flow as code can be written to only be conditionally executed if a statement evaluates as true, such as in the case where the result of comparing two values to each other via comparison operators, evaluates as true.

3. Logical operators

The three main logical operators are:

```
|| OR

&& AND

! NOT
```

The logical OR operator is represented by ||. This checks and compares two values. If either or both of the values are truthy, it returns a boolean value of true. If both the compared values are falsy, then the logical OR operator will return false.

For example:

```
2 > 3 || 3 > 4  This statement will return as false as both statements on either side of the OR operator are falsy.

3 < 4 || 1 > 3 This statement will return as true as one of the statements are truthy.
```

The logical AND operator is represented by &&. This, similar to the OR operator, checks and compares two values However in order for the AND operator to return true, both compared values must be truthy. If either of the compared values are falsy, then the statement will return as false.

For example:

```
1 > 2 && 1 < 3 This will return false as one of the statements is falsy

2 > 1 && 3 > 1 Will return true as both statements are truthy
```

The logical NOT operator, shown by the symbol !, can be used to either invert a Boolean value or the truthiness of a non-Boolean value.

For example:

```
let isHungry = true;
let opposite = !isHungry;

console.log(opposite);
// The result of this console log statement will be false.
```

4. Ternary Operator

Ternary operator accepts a conidition followed by a ? operator, after which there are two expressions which are separated by a :.

If the condition evaluates as truthy, then the first expression will be executed by the program, otherwise if the condition evalues as falsy, then the second expression will be executed instead.

For example:

```
let price = 5;
let day = "Saturday";

day == "Saturday" ? (price += 10) : (price -= 2);
/* As the condition day == "Saturday" evaluates as true, then the first expression price += 10
will be executed instead of the second expression */
```

5. Conditional Statements (if, else if, else)

Conditional statements make it so that code is only executed if specific conditions are met, with the most common conditional statement being the "if" statement (Reference 20). If the expression associated with the if statement evaluates as a truthy value, then the code within its code body will execute. Otherwise, if the expression evaluates as a falsy value, then the code inside the if statement's code body will not execute.

"else if" statements can be used after an "if" statement, to check for additional conditions. For example, if the "if" statement evaluates as falsy, the program will simply skip the code inside the "if" statement and then move to check if the "else if" statement evaluates as true and execute the code inside the "else if" statement's code boy instead.

An "else" statement can be added after an "if" statement or after a series of "else if" statements. This "else" statement will only be executed if the "if" statement, or preceeding "else if" statements all evaluate as false.

Conditional statements are instrumental in altering control flow, as whether certain sections of code are executed by the program are dependent on if specific conditions are fulfilled.

Here is an example below:

```
let age = 21;
let gender = "Male";

if (age < 18 && gender = "Male") {
    console.log('You are a boy');
} else if (age < 18 && gender = "Female") {
    console.log('You are a girl.');
} else if (age >= 18 && gender = "Male") {
    console.log('You are an adult man.');  // This line of code gets executed
} else {
    console.log('You are an adult woman.');
}
// Prints: You are an adult man
```

The first if statement will only execute and display 'You are a boy' on the console, if age is smaller than 18 and the gender is equal to "Male".

In this instance, this evaluates to false, resulting in the program skipping this line of code and checking the next "else if" statement. As the age is not smaller than 18 and the gender is not equal to "Female", the first "else if" statement is skipped by the program.

The computer will execute the second "else if" statement as this condition evaluates to truthy, resulting in 'You are an adult man' being printed to the console.

The last 'else' statement will only be executed if all the 'if' and 'else if' statements before evaluated as falsy.

6. Switch Statements

In JavaScript, switch statements is another method for control flow in a program where code is conditionally executed (Reference 20). Switch statements are useful in checking an expression against multiple case clauses. If a case matches, then the code inside that case clause is executed by the program. If none of the cases match, and a default clause is included, then the code located inside the default clause will be executed.

Here is an example below:

```
let food = "burger";

switch (food) {
  case "pizza":
    console.log("Italian");
    break;
  case "sushi":
    console.log("Japanese");
    break;
  default:
    console.log("Not sure...");
}
// Prints: Not sure...
```

For the above example, control flow is altered based on the variable 'food'. As none of the case clauses match with the expression food = 'burger', all these cases are skipped by the program and instead the code inside the default clause is executed instead. This results in an output of 'Not sure...' being logged to the console.

7. Loops (for, while, do while)

Loops are used repeatedly execute a block of code. The loop will continue to repeat until a specified condition, also known as a stopping condition, is met (Reference 20). These can be used to alter control flow, as code can be executed repeatedly based on certain conditions.

A while loop is executed as long as the specified condition evaluates to true. The loop continues to be executed until the condition evaluates to false.

See example below:

```
let i = 0;

while (i < 10) {
  console.log(i);
  i++;
}
```

For the example above, the while loop continues to be executed whilst i is less than 10. With each iteration of the while loop, the code inside the while loop is executed and the value of i is printed to the console. As well, the value of i is also incremented by a value of 1. Once the value of i is equal to 10, the while loop will break and no longer continue to execute. The result of the while loop in the example above is that the value of i is repeatedly printed multiple times with each loop iteration. The values of 0 to 9 are printed to the console before the while loop stops executing.

A do...while loop is very similar to a while loop, however the code block inside the statement is always executed at least once, before checking if the condition inside the statement is true to continue to repeatedly loop more. The difference is that a do...while loop is always executed at least once, whilst a while loop will only execute if and also as the condition inside the while loop evaluates to true.

A for loop is used when the number of times that you want to execute a block of code is known (Reference 1).

Here is an example below:

```
for (let i = 0; i < 5; i++) {
  console.log(i); // Output: 0, 1, 2, 3, 4
}
```

A variable is first initialised where i = 0, and then a stopping conidition determines when the for loop will stop executing and the expression evaluates to false 'i < 5' and the iteration statement at the end updates the initialised iterator variable i each time the loop is completed. For the example above, each iteration of the for loop will log the value of i to the console, and the value of i is also increased by 1 with each loop iteration. The for loop stops running when the stopping condition is fulfilled and i is no longer smaller than the value of 5.

8. Functions

Functions are where reusable blocks of code are defined, and can be called with specific inputs and then return a result. These can be utilised to alter control flow through encapsulating code and executing it when needed.

```
function welcome(age) {
  console.log(`You are ${age} years of age!`);
}

greet(18); // Output: "You are 18 years of age"
```

The function above takes age as an input, and the output of the function is a statement printed to the console. In the above exampe, the function is only executed when it is called, where in this case the input is the value of 18, resulting in the output of the function executed being the logged statement "You are 18 years of age".

Q8.

The automatic or implicit conversion of values from one data type to another - is referred to as type coercion (Reference 21). Examples of conversion can be from a number data type to a string data type, or a boolean to a number etc. Implicit type coercion occurs when JavaScript automatically converts one or more values to a

different data type without the programmer having to explicitly request the conversion (Reference 12). Explicit type coercion occurs when the programmer expresses the explicit intention to convert between types by writing the appropriate code, such as Number(value) (Reference 22). In JavaScript, type coercion is typically automatically and implicitly performed by the programming language itself.

There are three main types of conversion in JavaScript which are (Reference 22):

- to string
- to number
- to boolean

1. To string conversion:

In JavaScript, when any non-string value is added to a string, the non-string value is converted to a string implicitly. For example if the string 'Blue' is added to the number 5, it will not return as an error in JavaScript. Instead, the number 5 is implicitly converted to a string value of '5' through coercion. Then both strings are concatenated with each other (Reference 12).

See this in the example below:

```
let example = "Blue" + 5;
console.log(example);
/* The output will be Blue5. This output will be in the data type string as the number 5
was converted to a string value and concatenated with the string value 'Blue'. */
```

This is also occurs with Boolean values as shown in the example below:

```
let exampleTwo = true + 8;
console.log(exampleTwo);

/* Output: true8
The boolean value of true was implicitly converted by JavaScript to a string value */
```

For the above two examples, the type coercion performed was implicit.

However it is possible for the developer to perform explicit type coercion. This can be done through using the String() function in JavaSript. See this in the example below:

```
let number = 20;
let stringExample = String(number);
console.log(stringExample);

// The output will be the string value '20'. This is because the String() function was used
// to convert the number value of 20 into a string value of '20'.
```

2. To number conversion:

In JavaScript, there are arithmetic operations available such as subtraction (-), multiplication (*), division (/) and modulus (%) (Reference 12). These arithmetic operations can only performed between numbers, meaning that any non number values can be implicitly converted into the number data type.

For the example below, the string value of '2' is automatically and implicitly converted by JavaScript, using type coercion, into an number value. This is done in all the cases in order for these arihmetic operations to be performed:

```
let a = 4 - "2";
let b = 4 * "2";
let c = 4 / "2";
let d = 4 % "2";

console.log(a);
console.log(b);
console.log(c);
console.log(d);
```

The Output is:

```
2
8
2
0
```

Each of the output values are in the data type number, where the initial string value of '2' was converted implicitly into a number data type.

This also occurs for boolean values, where the boolean value of 'true' is represented as the number value 1, and 'false' is represented as the number value 0. See examples below:

```
let a = 2 - true;
console.log(a);
/* Output is the number value 1. This is because the boolean value true was converted to the
number value 1, and 2 minus 1 is equal to 1. */

let b = 10 * false;
console.log(b);
/* Output is the number value 0. This is because the boolean value false was converted to the
number value 0. 10 multiplied by 0 is equal to 0. */
```

In order for the arithmetic operations to execute, the boolean values were implicitly converted, through type coercion, to their respective number values.

Once again, in JavaScript, the conversion of values of a number data type can be done explicitly through using the Number() function. See below:

```
let string = "10";
let numberValue = Number(string);
console.log(numberValue);
/* The output will be 10 as a number value data type. This is because the string '10' was explicitly
converted through using the Number() function, to the number value of 10. */
```

3. Booleans

The Boolean() function can be used to explicitly convert a value to a boolean. For example, Boolean(null) will return as false while Boolean({}) will return as true (Reference 22) Implicit conversion can be triggered through using the AND (&&), OR (||) or NOT (!) logical operators. These logical operators perform boolean conversions internally, however they return the value of the original operands even if they are not boolean.

4. Equality operators

The equality operator (==) can be utilised to compare values regardless of their data types (Reference 12). This once again utilises type coercion, where values will be implicitly converted by JavaScript. See below:

```
let a = 5 == "5";
console.log(a);
/* Output will be true as the string value '5' will be implicitly coerced into the
number value of 5. As 5 is equal to 5, the output will be a boolean value of true. */

let b = true == 1;
console.log(b);
/* Output will be true as the boolean value is implicitly coerced into the number
value of 1. As 1 is equal to 1, the output will be a boolena value of true. */
```

It is important to keep in mind that the strict equality operator (===) will not perform type coercion.

Q9.

Data types represent types of data that can be utilised and manipulated in various ways in a program (Reference 12). For programming languages, data types act as a classification system, which can differ from one programming language to another. Each different data type has a unique set of characteristics and operations that can be performed on it. JavaScript is a dynamically typed programming language. This means that in JavaScript, variables can receive, contain and represent different data types over time. For example, a variable in one instance can be a number and then later in the program be reassigned to contain a string instead.

In JavaScript, the typeof operator can be used to find the data type of a variable (Reference 23). For example:

```
let a = true;
let b = 12;
let c = "hello";

console.log(typeof a);
console.log(typeof b);
console.log(typeof c);
```

Output of the above example will be:

```
boolean
number
string
```

1. Primitive data types.

Primitive data types represent the basic building blocks of any programming language, and they define immutable values (Reference 21). Below are some examples

of primitive data types in JavaScript;

- Number

In JavaScript, all numbers, including integer and floating-point numeric values, are represented as the number data type (Reference 23). This can be seen in the example below:

```
let integer = 10;
let float = 3.12;

// Both variables represent the number data type, regardless of whether the numeric value
// is a whole integer or floating decimal point.
```

- String

The string (or text string) data type is represented by a sequence of characters such as "Text" (Reference 23) In JavaScript, strings can be written using single quotes (''), double quotes ("") or backticks (``). Any and all characters included inside the single quotes, double quotes or backticks in JavaScript are considered to be the string data type. Here are examples below:

```
let stringOne = "String";
let stringTwo = "Hello World 23";
let stringThree = `Password!!1010%%^&`;

// All examples above are string data types
```

- Boolean

The boolean data type represents a logical entity and can either be two values: true or false (Reference 12). Boolean values can usually be seen utilised in conditional statements and comparisons. Here are examples below:

```
let isHungry = true;
let isRaining = false;

// Both variables above represent values which are of the boolean data type
```

- Null

This data type can only have one value, which is null. The null value in JavaScript is used for referring the absence of any value or object (Reference 12). This data type is usually utilised to explicitly set a variable to have no value.

```
let emptyValue = null;
console.log(emptyValue);
// Output will be null
```

- Undefined

A variable that is declared however has not yet been initialised or assigned a value is represented as the data type undefined (Reference 12). Both its value and data type are undefined.

```
let name;
console.log(name);
// The output will be null as the variable name was not assigned a value. The variable's value and data type are both undefined.
```

- Symbol

The Symbol data type represent unique identifiers. The purpose of symbols is to create unique property keys that will not clash with keys from any other code (Reference 21). In JavaScript, the symbol data type can be defined using the Symbol() object. See here an example:

```
let key = Symbol("secret");
console.log(typeof key);
// Output of the data type will be symbol
```

- Bigint

The Bigint data type is a primitive value in JavaScript used to represent integers of arbitrary magnitude (Reference 21). Developers use this data type to store and operate on extremely large integers. The bigint data type in JavaScript can be defined using the BigInt() object:

```
let number = BigInt("432423423423423423423423423423423423423");
console.log(typeof number);
// Output of the data type will be bigint
```

2. Non-Primitve data types

Objects in JavaScript are non-primitive, mutable data types and can be used to store collections of key-value pairs. They can be used to represent more complex data structures, such as arrays, objects and classes. All JavaScript values, except primitives, are objects (Reference 23).

- Object

An object is a collection of key-value pairs (or name-value pairs) separated by commas. (Reference 23). These values can store any other data types, including objects. JavaScript objects are defined using curly braces {}. Here is an example of an object below:

```
let person = {
  name: "Emma Watson",
  age: 25,
  isWizard: true,
};
```

- Array

An array is type of object that is used to store ordered collections of data. Arrays in JavaScript are defined using square brackets [] where array items are separated by commas (Reference 23). These can hold elements of various different data types. Here are examples of arrays below:

```
let ages = [8, 23, 35, 48, 99];
let names = ["Harry", "Kevin", "Sam", "Xavier"];
let mixedData = [56, "string", false, { key: "value" }];
```

Q10.

Arrays are a data structure in JavaScript and can be defined using square brackets []. Arrays are used to store collection of multiple elements under a single variable name (Reference 21). Below are some various ways that Arrays can be manipulated using JavaScript:

1. Elements can be added to an array

- The 'push()' method can be utilised to add a new element to an array (Reference 23). See example below:

```
let names = ["Billy", "Joe", "Sam"];
names.push("Tom"); // The push() method adds a new element "Tom' to the names array
console.log(names); // Output will be ["Billy", "Joe", "Sam", "Tom"]
```

- The 'unshift()' method can be utilised to add a new element to the beginning of an array, which 'unshifts' the previous elements (Reference 23). See example below:

```
let colours = ["Blue", "Green", "Orange"];
colours.unshift("Yellow"); // The unshift() method adds a new element "Yellow" to the start of the colours array
console.log(colours); // Output will be ["Yellow", "Blue", "Green", "Orange"]
```

2. Elements can be removed from an array

- The 'pop()' method can be utilised to remove the last element from an array in JavaScript (Reference 23). See example below:

```
let names = ["Billy", "Joe", "Sam"];
names.pop();
console.log(names);
// Output will be ["Billy", "Joe"], as the last element "Sam" was removed through the pop() method
```

- The 'shift()' method can be used to remove the first element of an array. This results in all the other elements in the array shifting to a lower index (Reference 23). See example below:

```
let colours = ["Blue", "Green", "Orange"];
colours.shift();
console.log(colours); // Output will be ["Green", "Orange"], as the shift() method removed the first element "Blue".
```

- The 'splice()' method can be used to remove and then replace elements in an array, where new elements are inserted in their place (Reference 23). See example below:

```
let colours = ["Blue", "Green", "Orange"];
colours.splice(1, 2, "Replaced", "New");
console.log(colours); /* Output will be ["Blue", "Replaced", "New"], as the splice()
method removed the second and third elements "Green" and "Orange", and then replaced them
with the elements "Replaced" and "New". */
```

3. Array elements can be modified

- By referring to the index position of an element in the array, code can be written to replace the element at that particular index position with a new value. See example below:

```
let colours = ["Blue", "Green", "Orange"];
colours[0] = "Pink";
console.log(colours);
/* Output will be ["Pink", "Green", "Orange"] as the first element of the
colours array was set to the value "Pink" instead. */
```

4. Arrays can be merged together

- The concat() method can be utilised to merge (or concatenate) existing arrays into a newly created array (Reference 23). See example below:

```
let fruits = ["Banana", "Orange"];
let vegetables = ["Broccoli", "Spinach"];

let food = fruits.concat(vegetables);
console.log(food);
/* Output will be ["Banana", "Orange", "Broccoli", "Spinach"] as the fruits and vegetables
arrays were concatenated with each other to form the new food array. */
```

5. Arrays can be iterated over using loops

- See example below using a for of loop on an array:

```
let colours = ["Blue", "Green", "Orange"];
for (let colour of colours) {
  console.log(colour);
}
```

6. Arrays can be sorted

- The sort() method can be used to sort an array alphabetically (Reference 23). See example below:

```
let names = ["Xavier", "George", "Tom", "Arthur", "Zamira"];
names.sort();
console.log(names);
// This will output the alphabetically sorted array ["Arthur", "George", "Tom", "Xavier", "Zamira"]
```

7. Arrays can be reversed

- The reverse() method can also be used to reverse the elements in an array (Reference 23). See example below:

```
let names = ["Xavier", "George", "Tom", "Arthur", "Zamira"];
names.reverse();
console.log(names);
// This will output the names array sorted in reverse order as ["Zamira", "Arthur", "Tom", "George", "Xavier"]
```

8. Arrays can also be filtered

- The filter() method can be utilised. All the elements that pass a test implemented by a provided function will be inserted into a newly created array (Reference 23). See example below:

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8];
let evenNumbers = numbers.filter(
  (number) => number % 2 === 0
); /* The number array is filtered according
to the provided function. The function notes that each element in the array must be divisible by 2
and not have a remainder after. */
console.log(evenNumbers); // [2, 4, 6, 8] is the output of the newly created function, from the filter() method.
```

Q11.

In JavaScript, objects can be defined by key:value (or name:value) pairs (Reference 23). Objects are created by using curly brackets {}. There are many various ways in which objects can be manipulated using JavaScript. Please see the following examples.

1. Object creation

Below is an example of how an object is created (Reference 23).

```
let person = {
  firstName: "Zane",
  lastName: "Smith",
  age: 25,
};
```

2. Object properties

Object properties can be accessed in JavaScript (Reference 23).. Based on the previous example, see below how object properties can be accessed:

```
console.log(person.firstName); // This will return 'Zane' as the 'firstName' key was referenced in the person object
console.log(person.age); // This will return 25 as the 'age' key was referenced in the person object
```

3. Adding object properties

Additional object properties can be added (Reference 23).. See below an example of how this can be done:

```
let person = {
  firstName: "Zane",
  lastName: "Smith",
  age: 25,
};
person["gender"] = "Male";
console.log(person);
/* This will return as {firstName: 'Zane', lastName: 'Smith', age: 25, gender: 'Male'}. It can be seen
that the key-value pair gender: 'Male' was added to the end of the object. */
```

4. Updating object properties

Object properties can also be changed or updated by referencing the particular property (Reference 23).. See below an example of how this can be done:

```
let person = {
  firstName: "Zane",
  lastName: "Smith",
  age: 25,
};
person["firstName"] = "Tom";
console.log(person);
/* This will return as {firstName: 'Tom', lastName: 'Smith', age: 25}, where the first property of firstName
was updated to have the value as 'Tom' instead. */
```

5. Deleting object properties.

Object proeprties can be deleted by using the delete keyword (Reference 23).. See below an example how this can be done:

```
let student = {
  name: "Sam",
  team: "Giraffes",
  grade: 10,
};

delete student.team;
console.log(student);
// This will output as {name: 'Sam', grade: 10}, where the object property of team was deleted.
```

6. Checks

JavaScript can be used to check if a property exists within an object (Reference 23).. See below an example of how this can be done:

```
let student = {
  name: "Sam",
  team: "Giraffes",
  grade: 10,
};

console.log("name" in student); // This will output as true
console.log("height" in student); // This will output as false as this property is not in the object
```

7. Loops

Loops can be used to iterate over the properties of an object (Reference 23).. See below an example of how this can be done:

```
let student = {
  name: "Sam",
  team: "Giraffes",
  grade: 10,
};

for (let key in student) {
  student[key] = "Updated";
}
console.log(student);
/* This will return as {name: 'Updated', team: 'Updated', grade: 'Updated'} as all the values of
the properties in the object were replaced with the value 'Updated'. */
```

8. Methods

A method can be created for an object which can then be invoked later on. By creating and defining methods for objects, deveopers can invoke these methods once more in other parts of their program (Reference 23).. See example below:

```
let student = {
  name: "Sam",
  team: "Giraffes",
  grade: 10,
  greeting: function () {
    return "My name is " + this.name + " and I am in grade " + this.grade;
  },
};
console.log(student.greeting());
/* The output of invoking the greeting() method which was created for the student object is
the string statement saying My name is Sam and I am in grade 10. /*
```

9. Merging

Two or more objects can be merged into a single object by using the Object.assign() method. (Reference 23).

```
let objectOne = { a: 1, b: 2 };
let objectTwo = { c: 3, d: 4 };
let merged = Object.assign({}, objectOne, objectTwo);
console.log(merged);
/* This will output a new object which is a result of merging objectOne and
objectTwo. The result is { a: 1, b: 3, c: 3, d:4 } */
```

10. Freezing

Object.freeze() is a method that freezes an object, thus ensuring that no further changes can be made to the object at all (Reference 21). See example below:

```
let student = {
  name: "Sam",
  team: "Giraffes",
  grade: 10,
};

Object.freeze(student);
student.name =
  "Toby"; /* This line of code should not execute as the student object has
now been frozen. The name property will not change to "Toby" due to the prior
use of Object.freeze. */
console.log(
  student.name
); /* This line of code will output 'Sam' as the student object
was frozen and can no longer be modified. */
```

Q12.

JSON stands for JavaScript Object Notation, and is a format for storing and transporting data (Reference 23). A key feature of JSON is that data is represented as key-value pairs, where each key is a string, and each paired value can be in other data type formats such as strings, numbers, booleans, objects and arrays (Reference 23). There are various ways in which JSON can be manipulated using JavaScript:

1. Accessing JSON data.

JSON data can be accessed through using dot notation (Reference 24). Here is an example on accessing the property of a JSON object:

```
let employee = {
  name: "Andy",
  age: 37,
  location: "Gold Coast",
  is_manager: false,
};

console.log(employee.name);
// The output of this will be Andy as the name property of the employee JSON object was referenced and accessed
```

2. Changing JSON data

Properties of a JSON object can be changed through dot notation (Reference 24). Here is an example on changing a property of a JSON object:

```
let employee = {
  name: "Andy",
  age: 37,
  location: "Gold Coast",
  is_manager: false,
};

employee.name = "Samantha";
console.log(employee);
/* The output of this will be { name: "Samantha", age: 37, location: "Gold Coast", is_manager: false }, where
the name key-value was changed to "Samantha". */
```

3. Adding data to JSON objects

Developers can add data to a JSON object through dot notation, where the property name is used to add the new data (Reference 24). Here is an example of this:

```
let employee = {
  name: "Andy",
  age: 37,
  location: "Gold Coast",
  is_manager: false,
};

employee.email = "email@email.com";
console.log(employee);
/* The output of this will be { name: "Andy", age: 37, location: "Gold Coast", is_manager: false,
email: "email@email.com" }. The property of "email" with the value "email@email.com" was added
to the JSON employee object */
```

4. Deleting data from JSON objects

A property can be deleted from a JSON object through using the delete operator (Reference 24). Here is an example of this below:

```
let employee = {
  name: "Andy",
  age: 37,
  location: "Gold Coast",
  is_manager: false,
};

delete employee.location;
console.log(employee);
/* The output of this is { name: "Andy", age: 37, is_manager: false }. The property "location" was
removed from the JSON employee object by using the delete operator */
```

5. Parsing JSON data

JSON data that is received from an external API or server usually is delivered in the form of a JSON string. The process of parsing involves converting a JSON string into a JavaScript object (Reference 24). Here is an example of how JSON data can be parsed by using the JSON.parse() method:

```
let jsonString = '{"name": "Andy", "is_manager": false, "employee_id": 23}';

let javascriptObject = JSON.parse(jsonString);
console.log(javascriptObject);
/* The output of this is in the form of a JavaScript Object as { name: "Andy", is_manager:
false, employee_id: 23 }. The original JSON string was parsed through using the JSON.parse() method,
which converted the JSON string format into a JavaScript Object format. */
```

6. Stringifying JSON data

In order to send JSON data back to a server or external API, JavaScript objects will need to be converted into a JSON string through a process called stringifying (Reference 24). This is achieved through using the JSON.stringify() method, where the method takes a JavaScript object as an argument and returns a JSON string. Here is an example below of this:

```
let computerObject = {
  brand: "Dell",
  year: 2017,
  isWindows: true,
};

let jsonString = JSON.stringify(computerObject);
console.log(jsonString);

/* The output of this is '{"brand":"Dell","year":2017,"isWindows":true}'. It can be shown that
the JSON.stringify method was used where the JavaScript Object called computerObject was
converted into a JSON string data format instead and stored to the variable named jsonString */
```

In summary, JSON.parse() is a method used for parsing JSON strings and converting them to JavaScript objects, whilst JSON.stringify() is the opposite where JavaScript objects are taken as arguments for the method and then converted into JSON strings. JSON objects can also have their properties modified, as well as have properties added or deleted (Reference 23).

Q13.

```
// Class has been defined with the name Car
class Car {
  /* The constructor of the class takes the argument - brand. In this case, the data type
  of brand is a string argument. */
  constructor(brand) {
    /* The carname property of the Car object is set to the value of the brand argument.
    In standard practice, usually the property name as well as the argument are usually
    set as the same name for consistency. A recommended change for this code would be to
    have the argument for the function be called "brand" and the property called brand
    as well.*/
    this.carname = brand;
  }
  // This line defines a method called present() for the Car class
  present() {
    /* The present() method returns a string which includes the carname property
    (which is set to the brand argument) */
    return "I have a " + this.carname;
  }
}
/*A class has been defined with the name Model. This class extends the Car class.
Model is a child or subclass of the Car class through use of the extends keyword.
This means that Model inherits all the properties and methods from the Car class. */
class Model extends Car {
  /* Constructor of the Model class takes two arguments - brand and mod. Brand will be
  a string data type value where as the mod argument is an integer value for this
  class. Brand represents the brand of the car where examples of these are shown below
  in the makes array such as "Ford" and "Holden". mod refers to the model year of the
  car, as shown in the models array below which contains the integer values of 1980
  up to 2019. */
  constructor(brand, mod) {
    /* Call the constructor of the parent class (Car) with the brand argument. The brand
    argument is passed to the constructor of the Car class using the super keyword. This
    calls the constructor of the parent class (Car) with the brand argument and sets the
    carname property of the Model object to brand. */
    super(brand);
    /* The model property of the Model object is set to the value of the mod argument. In
    standard practice, usually the property name and the arguments have the same name
    for consistency. A recommended change for this code simply be to rename the name of
    the argument to model, as mod is a bit unclear, and keep the property name as model. */
    this.model = mod;
  }
  // The line below defines a method called show() for the Model class
  show() {
    /* This show method returns a string. Through extending the Car class, the Model class
    is able to inherit the present() method of the Car class. This allows the show()
    method of the Model class to call the present() method using the 'this' keyword
    to refer to the current instance of the Model object. This means that the show()
    method is able to access both the carname property from the Car class and the
    model property from the Model class, and these properties are included in the
    returned string. */
    return this.present() + ", it was made in " + this.model;
  }
}
/* Variable called makes has been initialised to store an array of various string
values representing car makes. Line is also missing a semi colon at the end,
and it is best to keep this consistent with the rest of the code. */
let makes = ["Ford", "Holden", "Toyota"];

/*  Variable below called models has been initialised to store an array of numbers.
This represents the range of numbers from 1980 to 2019, which should represent
car model years. 'Array.from()' is a built-in method in JavaScript that creates a
new array from an existing array. 'new Array(40)' creates a new array with 40 empty slots,
and then Array.from() creates a new array from that empty array with each element being
the index 'i' plus 1980. The resultant 'models' array contains 40 elements. The first
element is 1980, the second element is the number 1981, and so on, up to the 40th element
```

```
which will be the number 2019. Line is also missing a semi colon at the end, best to
keep this consistent with the rest of the code. */
let models = Array.from(new Array(40), (x, i) => i + 1980);

/* Function has been defined with the name randomIntFromInterval and it takes
two arguments - min and max */
function randomIntFromInterval(min, max) {
  // min and max included

  /*This function returns a random integer between the input arguments min and
max (inclusive). Math.floor() is a built-in JavaScript function that returns
the largest integer less than or equal to a given number. For example, Math.floor(5.7)
would return 5. Math.random() is another built-in JavaScript function that
generates a random decimal number between 0 and 1 (exclusive). Together,
these two functions can be used to generate a random integer within a specified
range. The randomIntFromInterval function takes two arguments, min and max,
which represent the minimum and maximum values for the random integer to be generated. */
  return Math.floor(Math.random() * (max - min + 1) + min);
}

/* For Of loop is used to iterate over the models array. The initialised variable
for the for of loop is called model. In JavaScript, for of loops can be used to
loop through the values of iterable objects such as arrays and strings. */
for (model of models) {

  /* This line of code selects a random value from the makes array and sets this
  value equal to the variable called make. The randomIntFromInterval function
  is called with the arguments min set to 0 and max set to makes.length - 1. It
  would be best practice if the make variable was initialised using 'let'.
  Furthermore, the lines of code in this for loop do not all have a semi
  colons at the end, and it would be to include these for consistency. */
  make = makes[randomIntFromInterval(0, makes.length - 1)];

  /* Select a random value from the models array and set this value equal to the
variable named model. It would be best practice if this model variable was
initialised using 'let'. The randomIntFromInterval function is called with
the arguments min set to 0 and max should be set to models.length - 1. There
is an issue with this line of code below as it should be models.length-1 and
not makes.length-1. The length of the makes array is much shorter than the
models array. If makes.length is used in this line of code, then the random
value selected from the models array will only be between the first 3 elements
in the models array, when the models array in fact has 40 elements. */
  model = models[randomIntFromInterval(0, makes.length - 1)];

  /* Variable named mycar is initialised. It is set to equal a new Model object.
The Model object takes two arguments which are equal to the make and model
variables created in the lines of code previously in this for of loop. */
  mycar = new Model(make, model);

  // The show() method of the mycar object is invoked and this is logged to the console.
  console.log(mycar.show());

  /* The output of this 'for of' loop should result in a number of string statements
logged to the console equal to the number of elements in the 'models' array
(40 in this case). The 'show()' method returns a string value. */
}
```

REFERENCES

(1) Ed Coder Academy

(2) Atlassian (https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development)

(3) AWS

(4) GitLab

(5) Build5Nines

(6) CoderAcademy

(7) Altexsoft

(8) Codoid

(9) CaktusGroup

(10) Silas Reinagel

(11) Software Testing Help

(12) Geeksforgeeks

(13) MongoDB

(14) Indeed

(15) Manifera

(16) TPP Technology

(17) Frontend Masters

(18) Coursera

(19) Guru99

(20) Code Academy

(21) MDN

(22) freecodecamp

(23) W3Schools

(24) Vegibit