README.md

Dillon Cotter T3A1 - Workbook.

1. Provide an overview and description of a standard source control process for a large project

Source control assists in managing and tracking changes to code.
You will first have to decide on which system to use, e.g. Git.

1. Create a central repository where all the project's code and documents are held.
2. Master/Main branch will hold the stable code.
3. Developers create separate branches for new features or fixes.
4. Changes from these branches are reviewd with a pull or merge request
5. Git performs automatic checks to ensure code does not conflict, but does not perform any tests on the code itself.
6. In larger projects, peer reviews are essential to ensure that the code adheres to the overall project guidlines.
7. Once checks are complete, the branch can be merged to the Main/Master branch.
8. After merginf, this code should be tested by other developers or QA members of the project team.
9. All developers should ensure to clone main/master followed by branching before any work starts, to ensure they are running the most up to date codebase.

Other considerations are to have clear lines of communcation amongst the team at all times, and that each commit has meaningful messages for the changes made, with developers also responsible for updating documentation.
It is also good practice to incorporate Backup and recovery methods to prvent potential data loss and support a disaster recovery plan.

2. What are the most important aspects of quality software?

1. Functionality
   Referring to how well the software meets the needs it was designed for. This includes suitability, accuracy, compliance and security.

2. Reliability
   How stable and dependable is the software. It includes aspects like maturity, fault tolerance, and recoverability in the event of crash or errors.

3. Usability
   How easy and pleasant the software is to use. This may be understandability, learnability, operability and attractiveness.

4. Efficiency
   This relates to the software's performance and resource usage. It includes time behaviour and resource utilisation.

5. Maintainability
   How easy is it to modify and update the software. This may encompas modularity, reusability, changeability and testability.

6. Portability How easy is it for software to transfer from one environemtn to another. This could be installability and replaceability, and how easy it is to run on various platforms.

Reference: Turing Blog Post

3. Outline a standard high level structure for a MERN stack application and explain the components

A MERN stack consists of four components:

- **M**ongoDB,
- **E**xpress.js,
- **R**eact.js, and
- **N**ode.js

1. MongoDB: is a NoSQL database that stores data in flexibile JSON-like documents, and is used for the application's data storage.
This Database is schema-less, meaning the data structure can be altered over time.
2. Express.js: is a web application framework for Node.js. It is used to build the backend of the application, handling HTTP requests and responses, simplifying the server-side development by providing a large set of features for web and mobile apps.
3. React.js: is a JavaScript library for building user interfaces, primarily for front-end. This allows developers to create large web apps that can update a data without needing to reload the page. Mostly this can be used to build UI components, making the code more readable and maintainable.
4. Node.js: is a JavaScript runtime allowing execution of JavaScript code on the server side. With Node, JavaScript can run on both server and client side, leading to a more consolidated development process.

Front-end is built with React.js, which communicates to the backend through HTTP requests.
Back-end server is setup using Node.js with Express.js handling and responding to client requests.
MongoDB serves as the database, accessed by the backend to store and retrieve application data.

This is considered a 'Full-stack' solution.

Reference: MERN Stack Explained

4. A team is about to engage in a project, developing a website for a small business. What knowledge and skills would they need in order to develop the project?

A Project team would need to be made up of Technical skills and Business skills (soft skills) **Technical Skills**

1. Web Development Fundamentals: An understanding of HTML, CSS and JavaScript, and the concept of responsive design to ensure a website works on various screen sizes and devices.
2. Back-end Development: Proficiency with a server-side language (EG. Node.js, Python etc) and frameworks (EG. Express.js or Django) and database management skills, including knowledge of SQL or NoSQL databases (EG. PostgreSQL or MongoDB)
3. Front-end Development: Experience with front-end libraries like React.js or Angular. Knowledge of client-side technologies and frameworks.
4. Version control: Familiarity with version/source control systems like Git, for tracking changes and collaboration.
5. Web Hosting and Deployment: Understanding of web hosting services and deployment processes. Knowledge of domain registration and DNS management.
6. Security: Awareness of web security pracictes to protect the site and its users.
7. SEO and Accessibility: A basic understanding of Serach Engine Optimisation to improve visibility and Knowledge of web accessibility standards to make the website usable for people with screen readers.

Soft Skills

1. Communication: Ability to communicate with team members, and creating/using readable documenation.
2. Collaboration: Openness to feedback and adaptability to change.
3. Project Management: Working on prioritisation and familitarity with project management tools and methadologies (EG. Agile).
4. Business Requirements: Ability to understand/interpret the business needs of the client into technical requirements.
5. User Experience and User Interface Design: Understanding the principles of UX/UI design to create an appealing website.

5. With reference to one of your own projects, discuss what knowledge or skills were required to complete your project, and to overcome challenges

Flask API project

- Backend Development: Proficient knowledge of Python and the Flask framework was essential for building the application's logic.
- Database Management: A thorough understanding of relational databases, specifically PostgreSQL, and ORM tools like SQLAlchemy, was crucial for efficient data handling.
- API design: Skills in creating and managing various API endpoints were necessary for effective data retrieval and manipulation.
- Version control and documentation: Utilizing Git for version control and maintaining clear documentation in the README file were key practices.
- Security and Authentication: Implementing JWT for secure authentication and session management was vital to ensure user data integrity and privacy.
- Application configuration: Effective management of the application through configuration files streamlined various environmental settings.
- Dependency Management: Proper understanding and management of project dependencies, as listed in the requirements.txt, ensured smooth setup and deployment processes.
- Application architecture: Carefully structuring the application into models, controllers, and various components facilitated a modular and maintainable codebase.

6. With reference to one of your own projects, evaluate how effective your knowledge and skills were for this project, and suggest changes or improvements for future projects of a similar nature

Flask API Project

While my theoretical knowledge and basic skills were adequate, the practical implementation highlighted several areas for improvement:

- Proper Documentation and Project Management: Initially, insufficient planning led to challenges in feature implementation and code stability. Future Improvement: Adopting Agile methodologies with defined sprints and checkpoints will ensure better project tracking and flexibility in managing changes.
- Data Validation and Sanitization: Validation was primarily conducted through controllers using conditional statements, which limited thoroughness: EG.

```
# validate password complexity
password_pattern = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
if not re.match(password_pattern, password):
    return jsonify({
        "error": "Password must be at least 8 characters long and include at least 1 upperca
    }), 400
```

- Future Improvement: Integrating validation logic into the data models would standardize data integrity checks and offload responsibility from the controllers, enhancing security and maintainability.

- API Endpoint Expansion: The application's limited endpoint range restricted functionality, particularly in aggregating data by categories such as genre or director. Future Improvement: Designing more specific endpoints, such as /genres//tvshows, would improve data accessibility and application versatility by providing users with more tailored information retrieval options.

7. Explain control flow, using an example from the JavaScript programming language.

Control flow refers to the order in which individual statements, instructions or function calls are executed. In JavaScript the control flow is dictated by conditional statements, loops and function calls:

1. Conditional Statements (if-else):
Allows your code to make decisions based on certain conditions. They execute particular code if a specified condition is true and optionally define alternative blocks if code if false.
Example:

```
let number = 10;

if (number > 5) {
    console.log("The number is greater than 5.");
} else {
    console.log("The number is 5 or less.");
}
```

2. Loops (for):

Loops are used to execute a block repeatedly until a condition is met. In JavaScript typical loops include for to interrate over declared expression:
Example:

```
for (let i = 0; i < 5; i++) {
    console.log("The value of i is: " + i);
    // Expected output: "The value of i is: 0"
    // Expected output: "The value of i is: 1"
    // Expected output: "The value of i is: 2"
    // Expected output: "The value of i is: 3"
    // Expected output: "The value of i is: 4"
};
```

For of to itterate over values in an array (or string/map/set)
Example:

```
let fruits = ["apple", "bannana", "pear"];

for (const fruit of fruits) {
    console.log(fruit);
    // Expected output: "apple"
    // Expected output: "bannana"
    // Expected output: "pear"
}
```

For in that itterates over an object
Example:

```
let person = {
    name: "dillon",
    age: 33,
};

for (const detail in person) {
    console.log(`This person's ${detail} is ${person[detail]}`)
    // Expected output: "This person's name is dillon"
    // Expected output: "This person's age is 33"
}
```

3. Loops (while): This creates a loop that executes as long as the statement tested is TRUE.
Example:

```
let i = 5;

while (i > 0) {
    console.log(i);
    i - 1;
    // Expected output: 5
    // Expected output: 4
    // Expected output: 3
```

```
    // Expected output: 2
    // Expected output: 1
}
```

Another while loop exists (do-while). Both perform almost identically, however do-while tests the statement after executing the while block, wheras while tests before the block.
Example:

```
let i = 5;
do {
    console.log(i);
    i -= 1;
    // Expected output: 5
    // Expected output: 4
    // Expected output: 3
    // Expected output: 2
    // Expected output: 1
    // Expected output: 0
} while (i > 0);
```

4. Function Calls: Function calls direct flow to a different part of the program. When a function is called, the execution jumps to that function and runs its code. This means the calling of the function could occur long after the function is declared.
Example:

```
function greet (name) {
    console.log(`Hello, ${name}`);
}

greet("Dillon");
greet("Alex");
```

Reference [MDN Web Docs: Control Flow]

8. Explain type coercion, using examples from the JavaScript programming language

Type coercion refers to the automatic or implicit conversion of values from one data type to another. This can happen because variable in JavaScript are not directly bound to a specific data type.
There are two main types of coercion: implicit and explicit.

Implicit This occurs without explicit instruction when JavaScript attempts to reconcile type differences in expressions Example 1:

```
let value1 = '5'; // Stored as a string
let value2 = 10; // Stored as a number
let result = value1 + value2;
console.log(result); // Outputs: '510'
```

In this example, JavaScript has converted the value2 number to a string and concatenates it with the string '5' as '+' with a string assumes concatenation.
Example 2:

```
let value1 = '5'; // Stored as a string
let value2 = 10; // Stored as a number
let result = value1 * value2;
console.log(result); // Outputs: '50'
```

Conversly, for multiplication, JavaScript converts value1 to a number as artithmatic operations implicitly coerce numeric strings to numbers.
Example 3:

```
let value1 = '5';
if (value1) {
    console.log("It's true");
    // expected output: "It's true"
}
```

In a boolean context, non-empty strigns are implicitly coerced to being true.

Explicit
This is when a developer intentionally converts values from one type to another.
Example 1 (String to number):

```
let stringValue = '123';
let numberValue = Number(stringValue);
console.log(numberValue); // Outputs: 123
```

Example 2 (Any type to string):

```
let numberValue = 123;
let stringValue = String(numberValue);
console.log(stringValue); // Outputs: "123"
```

Example 3 (Any type to boolean):

```
let value = 'hello';
let booleanValue = Boolean(value);
console.log(booleanValue); // Outputs: true
```

Reference [MDN Web Docs: Type Coercion]

9. Explain data types, using examples from the JavaScript programming language

Data types are the classifications given to different kinds of data that we can use. JavaScript does not require a developer to declare the data type of a variable ahead of time. JavaScript has many built-in data types, which can be categorised into two main groups: Primitive and Reference types.

Primitive Types
These are basic, immutable data types. They do not have properties or methods (with the exception of `null` and `undefined`, which are special).
The main primitive data types in JavaScript are:

1. Number: Represents both integer and floating numbers.

```
let num1 = 10; // Integer
let num2 = 5.5; // Floating point
```

2. String: Represents sequences of characters

```
let text = "Hello, world!";
```

3. Boolean: Represents a logical entity and can have two values: `true` and `false`

```
let isCoderAcademyFun = true;
```

4. Undefined: Represents a variable that has been decalred, but not assigned a value.

```
let testVariable;
console.log(testVariable); // Outputs: undefined
```

5. Null: Represents the intentional absence of any object value. usually assigned to a variable as a representation of no value

```
let emptyVariable = null;
```

6. Symbol: Represents a unique value, not equal to any other value.

```
let uniqueSymbol = Symbol('symbol');
```

7. BigInt: Represents integers larger than 2^53 - 1, the largest number JavaScript can reliably represent

```
let largeNumber = BigInt(12345678901234567890123456789012345678901234567890);
```

Reference Types
These are objects that store collections of data or more complex entities.
These are different to primatives where they are mutable and can have properties and methods.

1. Object: The most basic reference type, can store collections of data and more complex entities.

```
let person = {
    firstName: "Dillon",
    lastName: "Cotter"
};
```

2. Array: A Type of object to store multiple values in a single variable

```
let colours = ['Red', 'Blue', 'Green'];
```

3. Function: A callable object that executes a code block

```
function greet(name) {
    return "Hello, " + name
}
```

Reference: [MDN Data Structures]

10. Explain how arrays can be manipulated in JavaScript, using examples from the JavaScript programming language

Arrays in JavaScript can be manipulated to add and remove elements. It can also sort, slice, and combine.

Add:

```
let fruits = ["apple", "orange"];
fruits.push("strawberry"); // Output: ["apple", "orange", "strawberry"]
fruits.unshift("mango"); // Output: ["mango", "apple", "orange", "strawberry"]
```

Remove:

```
// from the above fruits array
fruits.pop(); // Output: ["mango", "apple", "orange"]
fruits.shift(); // Output: ["apple", "orange"]
```

Sort:

```
let nums = [4, 2, 5, 1, 3];
numbers.sort((a, b) => a - b);
// Outputs: [1, 2, 3, 4, 5]
```

Slice:

```
let fruits = ["mango", "apple", "orange", "strawberry"];
let favourites = fruits.slice(1, 3);
/// Output: ["apple", "orange"]
```

Combine:

```
let moreFruit = ["pear", "kiwi"];
let combinedFruit = favourites.concat(moreFruit);
// Output: ["apple", "orange", "pear", "kiwi"]
```

There are also more advanced manipulations possible:
Map: Which applies a function to each element and returns a new array

```
let numbers = [1, 2, 3, 4];
let squares = numbers.map(num => num * num);
// Output: [1, 4, 9, 16];
```

Filter: Which removes elements based on a condition

```
// continued code from map example
let evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers);
// Output: [2, 4]
```

Splice: Which removes or replaces elements and also adds new elements at a declared index.

```
let months = ['Jan', 'Feb', 'Mar', 'May'];
months.splice(3, 0, 'Apr');

console.log(months);
// Output: ['Jan', 'Feb', 'Mar', 'Apr', 'May'];
```

Reference MDN Arrays

11. Explain how objects can be manipulated in JavaScript, using examples from the JavaScript programming language

Objects in JavaScript are used to store collections of various data (and various data types). Some was to manipulate objects include:
- Create an Object:

```
let student = {
    firstName: "dillon",
    lastName: "cotter",
    age: 35
}
```

- Adding Properties:

```
student.school = "coder academy"
// adds a new property 'school' to 'student'
```

```
// object is now:
// student {firstName:  "dillon", lastName: "cotter", age: 35, school: "coder academy"}
```

- Modifying Properties:

```
student.age = 33;
// modifies the 'age' of 'student'
// object is now:
// student {firstName:  "dillon", lastName: "cotter", age: 33, school: "coder academy"}
```

- Removing Properties:

```
delete student.school;
// removes the 'school' from 'student'
// object is now:
// student {firstName:  "dillon", lastName: "cotter", age: 33}
```

- Iterating over properties:

```
for (let key in student) {
    console.log(`This student's ${key} is ${student[key]});
    // expected output:
    // "This student's firstName is dillon"
    // "This student's lastName is cotter"
    // "This student's age is 33"
}
```

- Merging objects:

```
let contactDetails = {
    phone: "0412345678",
    email: "14582@coderacademy.edu.au"
};
let fullProfile = {...student, ...contactDetails};
// creates a new object called full profile with properties:
// {firstName:  "dillon", lastName: "cotter", age: 33, phone: "0412345678", email: "14582@codera
```

Reference MDN Working with Objects

12. Explain how JSON can be manipulated in JavaScript, using examples from the JavaScript programming language

JSON (JavaScript Object Notation) is a text-based format, easy for humans to understand, and easy for computers to understand. It is commonly used to exchange data between a web server and client, as well as storing and transporting data.
Due to JSON and JavaScript objects being very similar, manipulation of data is quite straightforward.
It can be manipulated in a few ways:

Convert JavaScript Object to JSON:

```
let student = {
    firstName: "Dillon",
    lastName: "Cotter",
    age: 33
}
let jsonStudent = JSON.stringify(student);
// Output: '{"firstName": "Dillon", "lastName": "Cotter", "age": 33}'
```

Convert JSON to JavaScript Object:

```
let jsonStudent = '{"firstName": "Dillon", "lastName": "Cotter", "age": 33}';
let student = JSON.parse(jsonStudent);
// Output: {firstName: "Dillon", lastName: "Cotter", age: 33}
```

Manipulate JSON Data:

```
student.school = "Coder Academy";
student.age = 30;
delete student.lastName;

let updatedJsonStudent = JSON.stringify(student);
// Output: '{"firstName": "Dillon", "age": 30, "school": "Coder Academy"}'
```

Reference MDN Working with JSON

13. For the code snippet provided below, write comments for each line of code to explain its functionality. In your comments you must demonstrates your ability to recognise and identify functions, ranges and classes

```
/**
 * Defines a class called 'Car' to model a car's basic properties
 *
 * @class Car
 * @typedef {Car}
 */
class Car {
    /**
     * Creates an instance of Car.
     *
     * @constructor
     * @param {string} brand The brand name of the car.
     */
    constructor(brand) {
      this.carname = brand; // 'carname' property holds the brand of the car
    }
    /**
     * Returns a string representation of the Car object
     *
     * @returns {string}
     */
    present() {
      return 'I have a ' + this.carname;
    }
}


  /**
   * Defines a class called 'Model' that extends the 'Car' to include model year information
   *
   * @class Model
   * @typedef {Model}
   * @extends {Car}
   */
class Model extends Car {
    /**
     * Creates an instance of Model.
     *
     * @constructor
     * @param {string} brand The brand name of the car
     * @param {number} mod The model year of the car
     */
    constructor(brand, mod) {
      super(brand); // Call to parent class 'Car' constructor
      this.model = mod; // 'model' propery holds the model year
    }
    /**
     * returns a string that includes both the make and the model year
     *
     * @returns {string} Description of the car with model year
     */
    show() {
      return this.present() + ', it was made in ' + this.model;
    }
}

// Initialises an array of car makes
let makes = ["Ford", "Holden", "Toyota"]
// Initialises an array of model years from 1980 to 2019
let models = Array.from(new Array(40), (x,i) => i + 1980)


/**
 * Function to generate a random integer within a defined range [min, max]
 *
 * @param {number} min the minimum value in the range
 * @param {number} max the maximum value in the range
 * @returns {number} A random integer between min and max (inclusive)
 */
function randomIntFromInterval(min,max) { // min and max included
```

```
    return Math.floor(Math.random()*(max-min+1)+min);
}

// Loop through each model year
for (model of models) {
    // Randomly choose a make from the 'makes' array
    make = makes[randomIntFromInterval(0,makes.length-1)]
    // Randomly choose a model year from the 'models' array
    model = models[randomIntFromInterval(0,makes.length-1)]

    // Create a new Model object with the random make and model year
    mycar = new Model(make, model);
    // display the car's description
    console.log(mycar.show())
}
```

References: MDN Classes