DWA_03.4 Knowledge Check_DWA3.1

---

1. Please show how you applied a Markdown File to a piece of your code.

README.md > # Book Library Web App > ## User Stories

```
# Book Library Web App

This Book Library app is social media platfrom that is specifically
targeted at book enthusiast. Users are able to see a preview off the
various books. This app inlcudes basic features such as if you would
like to see a description of the book you are able to click on any book
you choose to see it, it allows you to filter the books by genre, author
or title and you are able to toggle between light and dark mode so you
can read comfortably at night.

Table of Contents (up to date)
- [Book Library Web App](#book-library-web-app)
  - [Features](#features)
  - [Requirements](#requirements)
  - [Getting Started](#getting-started)
  - [User Stories](#user-stories)


## Features

- 💚Simple and user-friendly
- 📕View a Summarized description of a book
- 📄Ability to filter books
- 🌓Ability to use light or dark mode
```

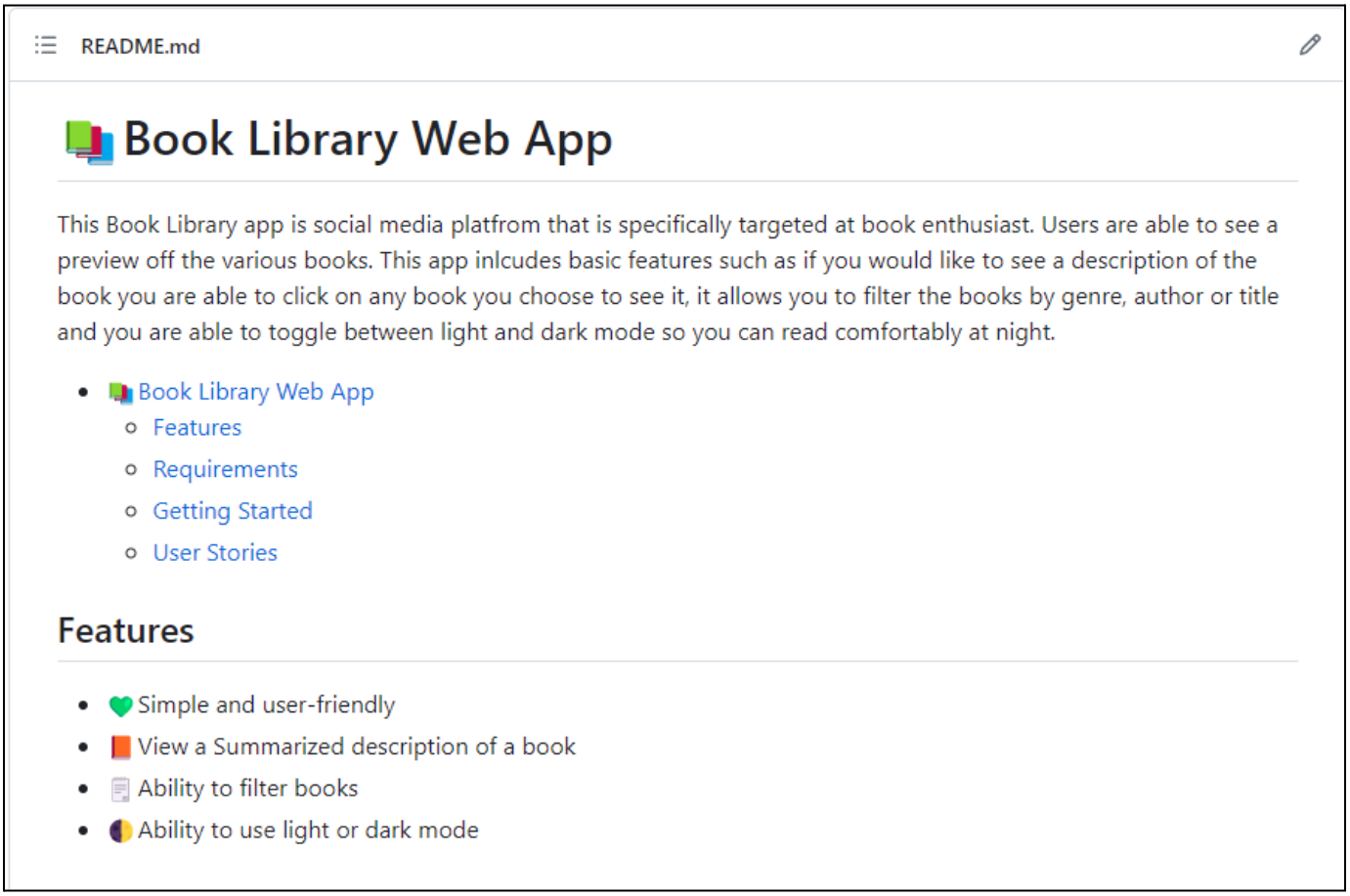
README.md > # Book Library Web App > ## User Stories

```
## Requirements

The following is required:

AN IDE like Visual Studio Code
[Basic HTML, CSS and Javascript Skills](https://developer.mozilla.org/
en-US/docs/Learn)
A browser like [Google Chrome](https://www.google.com/chrome/?brand=YTUH&
gclid=Cj0KCQjwjryjBhD0ARIsAMLvnF96_IwtQD7sHuLw-nvKEuTyy2VbsEGU8C2Sf6RZ8Cn
5VSN03UxtDDkaAh7ZEALw_wcB&gclsrc=aw.ds)

## Getting Started

1. Clone the repository

```
git clone https://github.com/Rehumile/Book-Library-WebApp.git
```

2. Run a localhost server

3. Open `index.html`

User Stories

- 👤As a user, I want to view a list of book previews, by title and
author, so that I can discover new books to read.
- 👤As a user, I want an image associated with all book previews so
that I can recognize a book by the cover even if I forgot the name.
- 👤As a user, I want to have the option of reading a summary of the
book so that I can decide whether I want to read it.
```

As you will from the screenshots above, this is a README markdown file. This is the readme file for the Final Project in the IWA Javascript which is a Book Library App. With the use of the markdown syntax rules:

- I used a single hash to represent the main heading.
- I used 2 hashes to represent the sub headings (eg. Features, Requirements etc.)
- I add a table of contents so users are able to navigate to whatever section that they would like to see.
- To add a link I first use square brackets and add the name that will be shown for the link and then I add normal brackets which will contain the URL address.
- To add a code snippet, I added 3 backticks at the start of the code and 3 backticks at the end.
- To add bullet points, I used a dash symbol

Here is a snippet of the readme preview:



---

2. Please show how you applied JSDoc Comments to a piece of your code.

```
/**
 * This object literal stores the settings of the colors of the dark and night mode in 'RGB' form.
 * This will update the css settings when user chooses between dark and night mode
 * @type {Object}
 */
const css = {
    day: {
        dark: '10, 10, 20',
        light: '255, 255, 255',
    },

    night :{
        dark: '255, 255, 255',
        light: '10, 10, 20',
    }
    }
```

The code snippet comes from the code from the IWA JavaScript Final Project. As you will see, I used JavaScript Documentation specifically the @type to specify the data type of the `css` object as well as a description of what the object stores and what it will be used for.

```
/**
 * A function that takes a book as an object literal and
 * converts it into an HTML element that can be appended to the DOM.
 *  Creating book elements individually prevents the JS having to re-render
 * the entire DOM every time a new book is created.
 * @param {object} props - Book object literal with book properties
 * @returns {HTMLElement} - HTML element with book details
 */
    const createPreview = (props) => {
    const {author,  id, image, title} = props


        let BookElement = document.createElement('button')
        BookElement.classList = 'preview'
        BookElement.setAttribute('data-preview', id)

        BookElement.innerHTML = /* html */ `
            <img
                class="preview__image"
                src="${image}"
            />

            <div class="preview__info">
                <h3 class="preview__title">${title}</h3>
                <div class="preview__author">${authors[author]}</div>
            </div>
        `
        return BookElement
    }
```

The second example, I used JS Doc for the `createPreview` function. The documentation includes a description of what the function does. I use the @param to specify the data type of the parameter as well as a short description of what the parameter is. I use the @return to specify what the function will return which is a HTML element

---

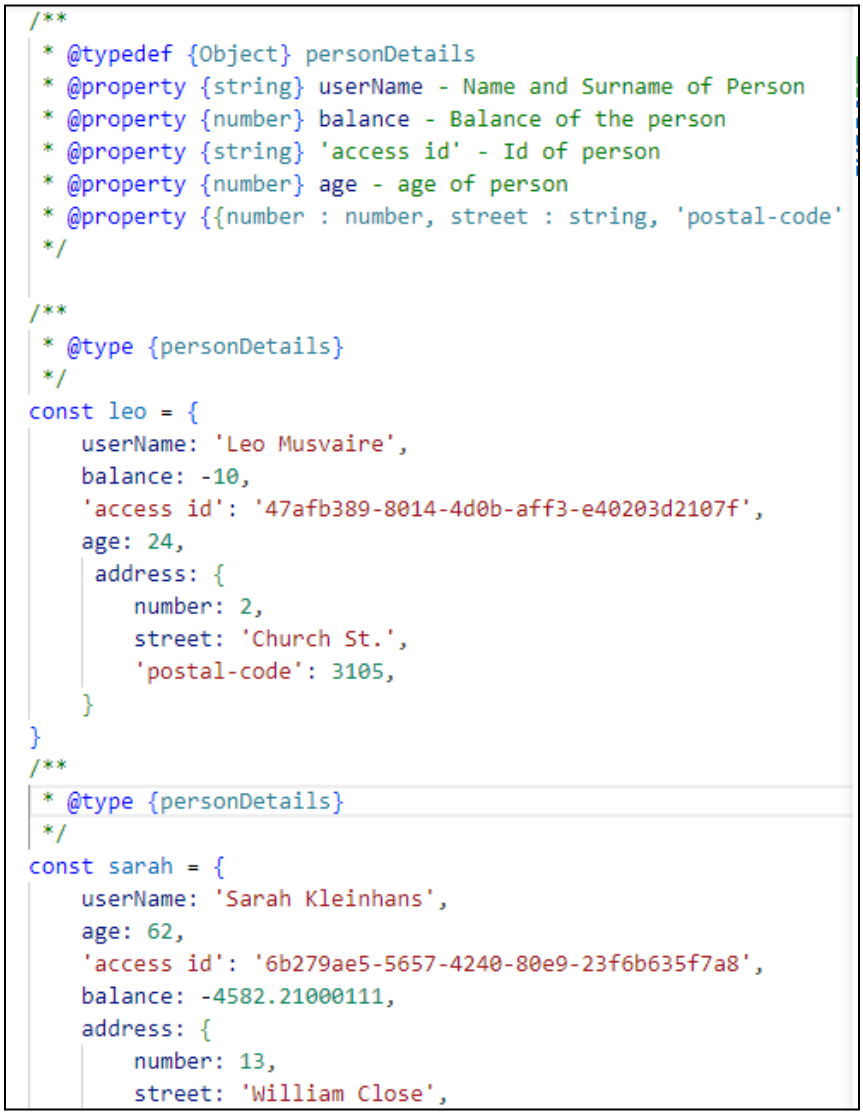
3. Please show how you applied the @ts-check annotation to a piece of your code.

```
// @ts-check
import { BOOKS_PER_PAGE, authors, genres, books, html} from './data.js'

// Data

/**
 * The variable will be used to store matches of
 * current filter settings from the books object.
 * @type {Array}
 */
let matches = books // correct
matches = genres //error

/**
 * This variable will be used to as the current page of
 * books being display and will increment by 1.
 * @type {number}
 */
let page = 1; //correct
page = 'hello' // error
```

This code snippet is also from the IWA JavaScript Final Project. Since JavaScript code will not raise an error if I change the variables, ` matches` and `page`, to different data types. I use the @ts-check to have TypeScript raise an error. For example if I assigned the variable `page` to “Hello” it would raise an error.

---

4. As a BONUS, please show how you applied any other concept covered in the 'Documentation' module.



```
/**
 * @typedef {Object} personDetails
 * @property {string} userName - Name and Surname of Person
 * @property {number} balance - Balance of the person
 * @property {string} 'access id' - Id of person
 * @property {number} age - age of person
 * @property {{number : number, street : string, 'postal-code'
 */

/**
 * @type {personDetails}
 */
const leo = {
    userName: 'Leo Musvaire',
    balance: -10,
    'access id': '47afb389-8014-4d0b-aff3-e40203d2107f',
    age: 24,
     address: {
        number: 2,
        street: 'Church St.',
        'postal-code': 3105,
    }
}
/**
 * @type {personDetails}
 */
const sarah = {
    userName: 'Sarah Kleinhans',
    age: 62,
    'access id': '6b279ae5-5657-4240-80e9-23f6b635f7a8',
    balance: -4582.21000111,
    address: {
        number: 13,
        street: 'William Close',
```

Using the code from the IWA 8 module, I created a custom type using the @typedef. The typedef is an object called `personDetails` which has the properties: userName, balance, access ID, age and address and the data type are also defined along with a description of each one. Thereafter I use the custom type on the `leo` and `sarah` objects.
