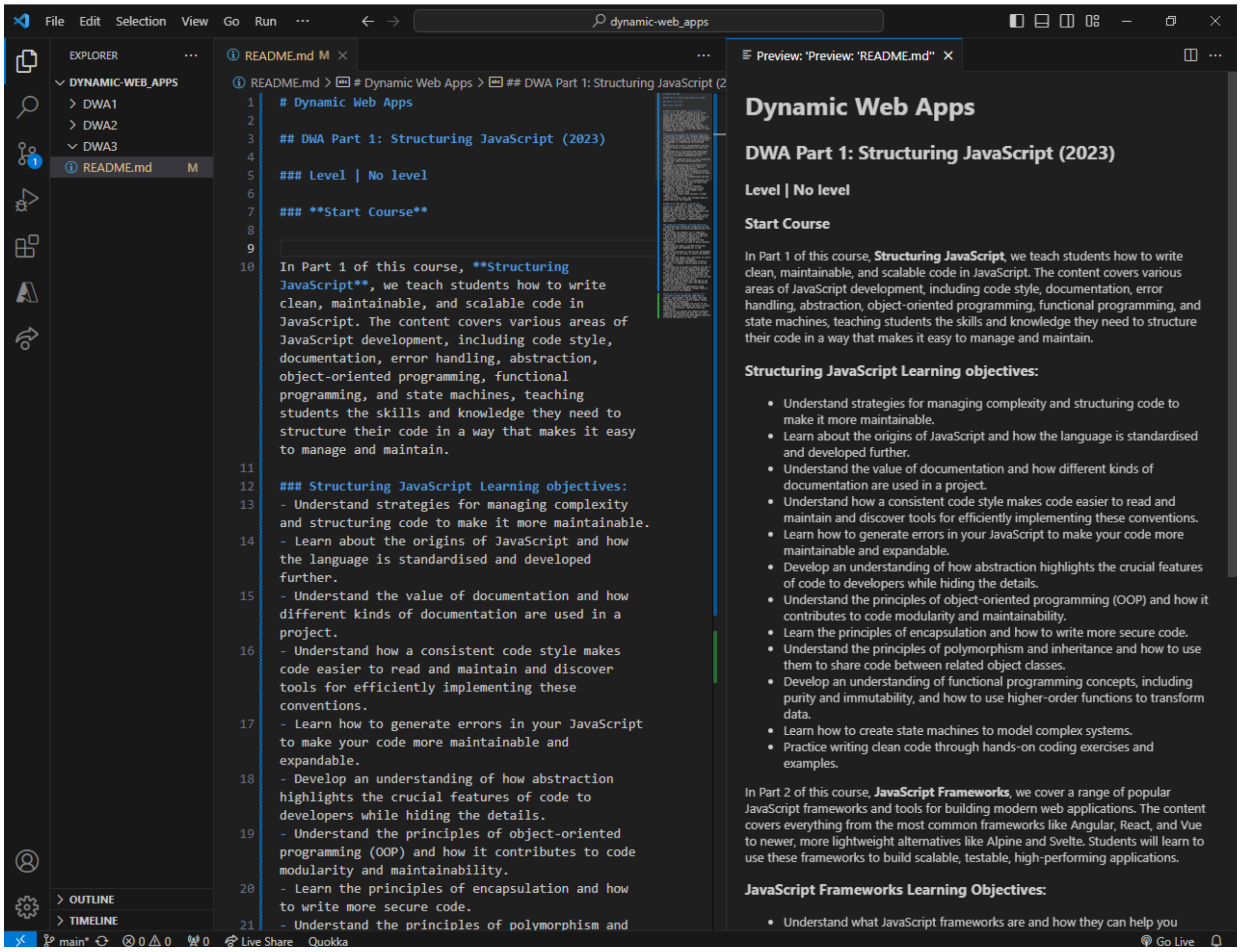
# DWA_03.4 Knowledge Check_DWA3.1

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


File Edit Selection View Go Run ···
dynamic-web_apps
EXPLORER
DYNAMIC-WEB_APPS
DWA1
DWA2
DWA3
README.md M
OUTLINE
TIMELINE
README.md M
README.md > # Dynamic Web Apps > ## DWA Part 1: Structuring JavaScript (2
Preview: 'Preview: 'README.md''
main* 0 0 0 Live Share Quokka Go Live

```
# Dynamic Web Apps

## DWA Part 1: Structuring JavaScript (2023)

### Level | No level

### **Start Course**


In Part 1 of this course, **Structuring JavaScript**, we teach students how to write clean, maintainable, and scalable code in JavaScript. The content covers various areas of JavaScript development, including code style, documentation, error handling, abstraction, object-oriented programming, functional programming, and state machines, teaching students the skills and knowledge they need to structure their code in a way that makes it easy to manage and maintain.

### Structuring JavaScript Learning objectives:
- Understand strategies for managing complexity and structuring code to make it more maintainable.
- Learn about the origins of JavaScript and how the language is standardised and developed further.
- Understand the value of documentation and how different kinds of documentation are used in a project.
- Understand how a consistent code style makes code easier to read and maintain and discover tools for efficiently implementing these conventions.
- Learn how to generate errors in your JavaScript to make your code more maintainable and expandable.
- Develop an understanding of how abstraction highlights the crucial features of code to developers while hiding the details.
- Understand the principles of object-oriented programming (OOP) and how it contributes to code modularity and maintainability.
- Learn the principles of encapsulation and how to write more secure code.
- Understand the principles of polymorphism and
```

**Dynamic Web Apps**

**DWA Part 1: Structuring JavaScript (2023)**

**Level | No level**

**Start Course**

In Part 1 of this course, **Structuring JavaScript**, we teach students how to write clean, maintainable, and scalable code in JavaScript. The content covers various areas of JavaScript development, including code style, documentation, error handling, abstraction, object-oriented programming, functional programming, and state machines, teaching students the skills and knowledge they need to structure their code in a way that makes it easy to manage and maintain.

**Structuring JavaScript Learning objectives:**

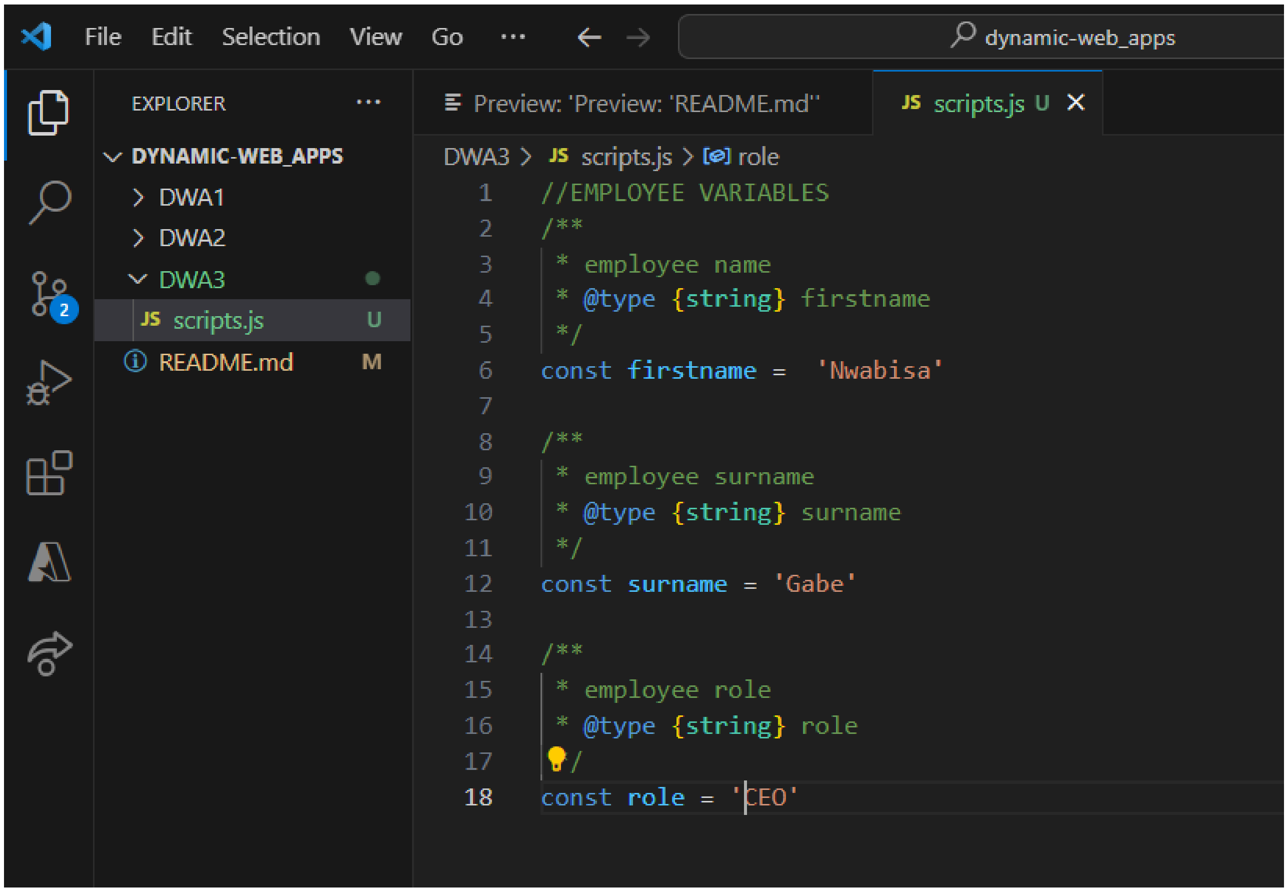
- Understand strategies for managing complexity and structuring code to make it more maintainable.
- Learn about the origins of JavaScript and how the language is standardised and developed further.
- Understand the value of documentation and how different kinds of documentation are used in a project.
- Understand how a consistent code style makes code easier to read and maintain and discover tools for efficiently implementing these conventions.
- Learn how to generate errors in your JavaScript to make your code more maintainable and expandable.
- Develop an understanding of how abstraction highlights the crucial features of code to developers while hiding the details.
- Understand the principles of object-oriented programming (OOP) and how it contributes to code modularity and maintainability.
- Learn the principles of encapsulation and how to write more secure code.
- Understand the principles of polymorphism and inheritance and how to use them to share code between related object classes.
- Develop an understanding of functional programming concepts, including purity and immutability, and how to use higher-order functions to transform data.
- Learn how to create state machines to model complex systems.
- Practice writing clean code through hands-on coding exercises and examples.

In Part 2 of this course, **JavaScript Frameworks**, we cover a range of popular JavaScript frameworks and tools for building modern web applications. The content covers everything from the most common frameworks like Angular, React, and Vue to newer, more lightweight alternatives like Alpine and Svelte. Students will learn to use these frameworks to build scalable, testable, high-performing applications.

**JavaScript Frameworks Learning Objectives:**

- Understand what JavaScript frameworks are and how they can help you

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



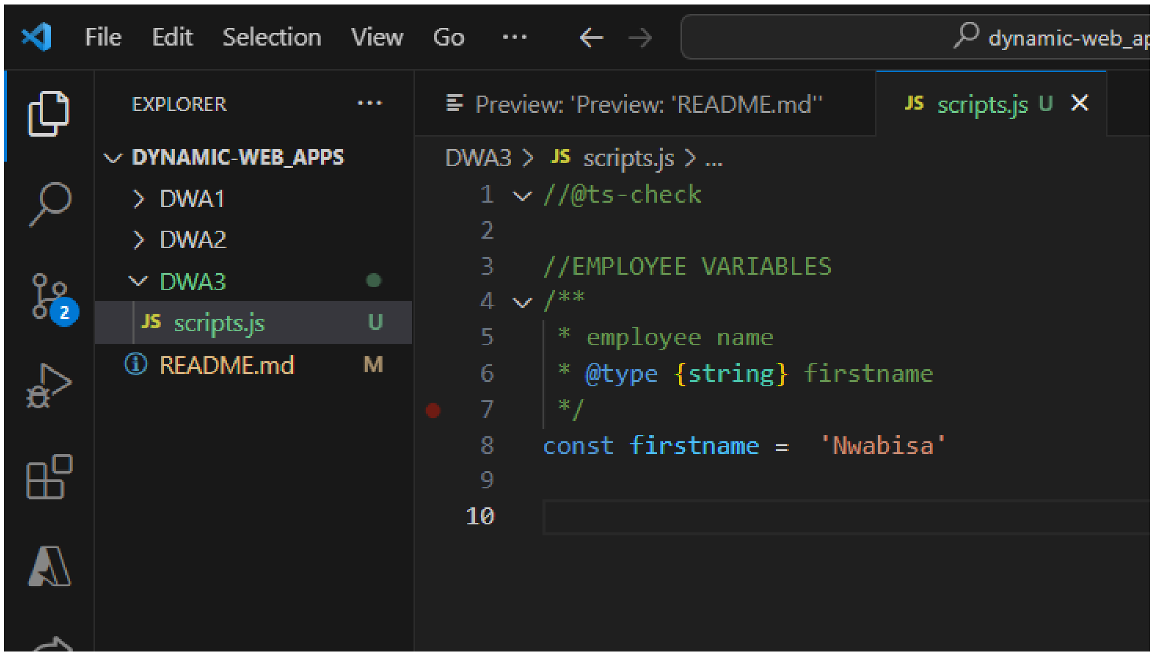
```js
//EMPLOYEE VARIABLES
/**
 * employee name
 * @type {string} firstname
 */
const firstname =  'Nwabisa'

/**
 * employee surname
 * @type {string} surname
 */
const surname = 'Gabe'

/**
 * employee role
 * @type {string} role
 */
const role = 'CEO'
```

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

---

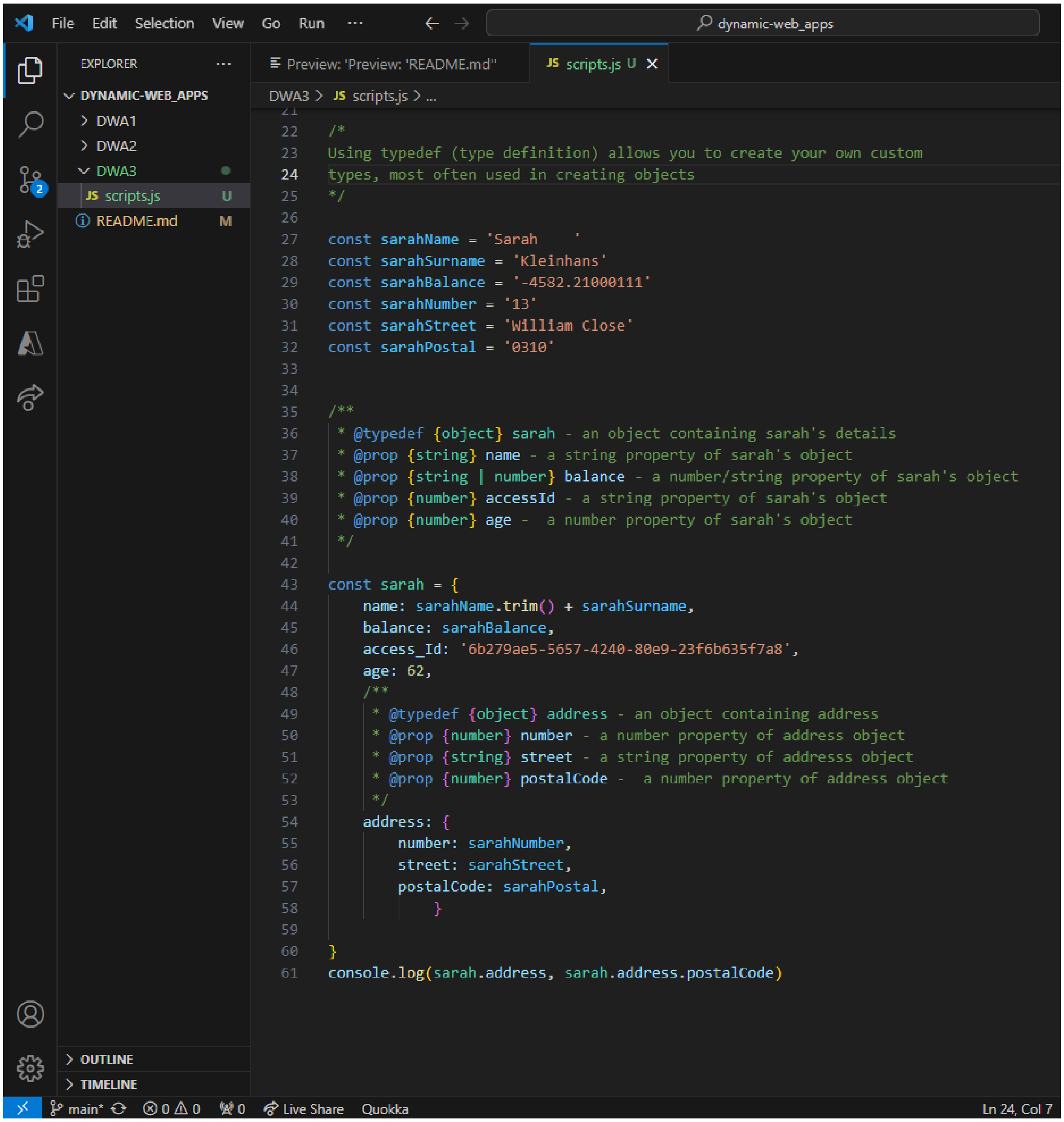


```
//@ts-check

//EMPLOYEE VARIABLES
/**
 * employee name
 * @type {string} firstname
 */
const firstname =  'Nwabisa'

```

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



```javascript
/*
Using typedef (type definition) allows you to create your own custom
types, most often used in creating objects
*/

const sarahName = 'Sarah    '
const sarahSurname = 'Kleinhans'
const sarahBalance = '-4582.21000111'
const sarahNumber = '13'
const sarahStreet = 'William Close'
const sarahPostal = '0310'


/**
 * @typedef {object} sarah - an object containing sarah's details
 * @prop {string} name - a string property of sarah's object
 * @prop {string | number} balance - a number/string property of sarah's object
 * @prop {number} accessId - a string property of sarah's object
 * @prop {number} age -  a number property of sarah's object
 */

const sarah = {
    name: sarahName.trim() + sarahSurname,
    balance: sarahBalance,
    access_Id: '6b279ae5-5657-4240-80e9-23f6b635f7a8',
    age: 62,
    /**
     * @typedef {object} address - an object containing address
     * @prop {number} number - a number property of address object
     * @prop {string} street - a string property of addresss object
     * @prop {number} postalCode -  a number property of address object
     */
    address: {
        number: sarahNumber,
        street: sarahStreet,
        postalCode: sarahPostal,
            }

}
console.log(sarah.address, sarah.address.postalCode)
```