Unit Test Plan

Module #:	Application #:
Tester: Talla Donald Rodrigue	Test Manager: Ndjock
Module Overview : users.js**,** account.js purchaseHistory.js	
Handles user creation and attributes.	
Module Inputs	
-class User**: id, name,** **-fonctions** userExists, createUserId **- class** account.js **-fonctions** isValidUserName, createAccount, getPastPurchases **-class** purchaseHistory.js **-fonctions** getPurchaseHistory	
Module Outputs	
-classe users **- fonctions** userExists: boolean (true/false), createUserId: integer **- classe** account **- fonctions** isValidUserName: boolean (true/false), createAccount: object (new user account data) or error, getPastPurchases: array of strings (events) or error **- classe** purchaseHistory **- fonctions** getPurchaseHistory: object (purchase history) or error	
Logic Flow	

user.js

Test User Creation with Valid Inputs

- **Inputs**: id = 1, username = "testuser"
- **Expected Outcome**: User object created successfully.

Test userExists with Existing Username

- **Inputs**: username = "newuser1@pluralsight.com"
- **Expected Outcome**: true

Test createUserId

- **Inputs**: None
- **Expected Outcome**: 2

account.js

Test isValidUserName with Valid Username

- **Inputs**: username = "user@example.com"
- **Expected Outcome**: true

Test createAccount with Valid Username

- **Inputs**: username = "user@example.com"
- **Expected Outcome**: Object { userId: 2, username: 'user@example.com' }

Test getPastPurchases with Valid UserId

- **Inputs**: userId = 1
- **Expected Outcome**: Array of strings ["event1", "event2"]

purchaseHistory.js

Test getPurchaseHistory with Valid UserId

- **Inputs**: userId = 1
- **Expected Outcome**: Object { events: ["event1", "event2"] }

Test Data

List all test cases to be executed.

Positive Test cases

`event.js`

1. **Test Event Creation with Valid Inputs**
   - **Inputs**: id = 1, name = "Concert", ticketPrice = 50, totalTickets =

100, ticketsRemaining = 100, date = new Date()
    - **Expected Outcome**: Event object created successfully.

2. **Test Event is Sold Out**
    - **Inputs**: Event object with ticketsRemaining = 0
    - **Expected Outcome**: true

3. **Test TagLine for Sold Out Event**
    - **Inputs**: Event object with ticketsRemaining = 0, minimumTicketCount = 10, isPopular = false
    - **Expected Outcome**: "Event Sold Out!"

4. **Test Create Event with Valid Inputs via `createEvent`**
    - **Inputs**: name = "Concert", price = 50, availableTickets = 100
    - **Expected Outcome**: Event object created successfully.

`filters.js`

1. **Test Event is Today**
    - **Inputs**: Event object with date = new Date()
    - **Expected Outcome**: true

2. **Test Event is Within Next 7 Days**
    - **Inputs**: Event object with date within the next 7 days from today
    - **Expected Outcome**: true

3. **Test Event is Within Next 30 Days**
    - **Inputs**: Event object with date within the next 30 days from today
    - **Expected Outcome**: true

`search.js`

1. **Test Get Events with Matching Predicate**
    - **Inputs**: events = [Event1, Event2, Event3], searchPredicate = (event) => event.name.includes("Concert")
    - **Expected Outcome**: Array of Event objects where name includes "Concert".

Number each test case. Indicate the test to be performed and expected outcome

Negative Test Cases

`users.js`

1. **Test userExists with Non-Existent Username**
    - **Inputs**: username = "nonexistentuser@pluralsight.com"
    - **Expected Outcome**: false

`account.js`

1. **Test isValidUserName with Invalid Username**
    - **Inputs**: username = "invalidusername"
    - **Expected Outcome**: false

2. **Test createAccount with Existing Username**
    - **Inputs**: username = "newuser1@pluralsight.com"
    - **Expected Outcome**: Error "User already exists"
3. **Test createAccount with Invalid Username**
    - **Inputs**: username = ""
    - **Expected Outcome**: Error InvalidUsernameError

`purchaseHistory.js`

1. **Test getPurchaseHistory with Invalid UserId**
    - **Inputs**: userId = 999
    - **Expected Outcome**: Error "User not found"

Listin valid data selections

Interface Modules

Output Data: User objects, account data, purchase history, booleans, strings, arrays.
Data Input: Parameters for user creation, account creation, and purchase history retrieval.
Internal Program Interface: Functions within `users.js`, `account.js`, and `purchaseHistory.js`.
External Program Interface: Not specified.

Identify interfacing modules indicating the nature of the interface:

- Output data
- Data input
- Internal program interface
- External program interface

Test Tools

Unit Testing Software: Vitest
Software for Regression Testing: Same as unit testing tools located in the project directory under `tests`.

Identify software used for unit testing.
Identify names and locations of software for future regression testing.

Archive Plan

Location of Archived Data: All test data, test results, and logs will be archived in the `test-archive` directory within the project repository.
Procedures for Access: Access to the `test-archive` directory will be restricted to authorized personnel. Requests for access should be directed to the project manager.

Specify location of archived data.
Define procedures required to obtain access to this data.

Updates

Updating the Unit Test Plan: The unit test plan will be updated whenever there are significant changes to the functionality of the modules. Updates will be documented in the project change log and the test plan will be version-controlled using Git.

Identify how unit test plan will be updated.

The unit test plan will be updated