**Unit Test Plan**

|  |  |
| --- | --- |
| Module #: | Application #: |
| Tester: Eyenga Claude/ Mende Jennifer | Test Manager: Nguekam |
| **Module Overview : event.js, search.js, filters.js** | |
| **Gestion des événements ( classe Event, fonctions issoldout, gettagline, createevent)**  **Offre une fonctionnalité de recherche pour les filtres basés sur des prédicats**  **Fournit les informations de filtrages des événements** | |
| **Module Inputs** | |
| **-class event: id, name, ticketPrice, totalTickkets, ticketsRemaining,**  **-fonctions isSolOut, getTagline,createEvent: event,minimumTicketCount, name, price, isPopular**  **- function today, getEvents: event, events, searchPredicate** | |
| **Module Outputs** | |
| **-classe event: en retour un nouvel objet event avec les memes propriétés**  **- fonctions isSoldOut, getTagLine, createEvent : boolean(true, false), string, en retour un objet**  **- functions today, getEvents: Boolean(true, false), une liste d’objets** | |
| **Logic Flow** | |
| event.js  1. **Création d'un événement**:    * **Entrée**: id, name, ticketPrice, totalTickets, ticketsRemaining, date    * **Processus**:      + Validation des paramètres.      + Si une validation échoue, lever une exception correspondante (InvalidEventNameError, InvalidEventPriceError).      + Sinon, créer un nouvel objet Event avec les propriétés fournies.    * **Sortie**: Objet Event. 2. **Vérification si un événement est sold out**:    * **Entrée**: Objet Event.    * **Processus**:      + Vérifier si ticketsRemaining est 0.    * **Sortie**: Boolean (true si sold out, false sinon). 3. **Génération de la tagline d'un événement**:    * **Entrée**: Objet Event, minimumTicketCount, isPopular.    * **Processus**:      + Si ticketsRemaining est 0, retourner "Event Sold Out!".      + Si ticketsRemaining < minimumTicketCount, retourner "Hurry only X tickets left!".      + Si l'événement est populaire, retourner un message promotionnel spécifique.      + Sinon, retourner "Don't miss out, purchase your ticket now!".    * **Sortie**: String. 4. **Création d'un événement via createEvent**:    * **Entrée**: name, price, availableTickets.    * **Processus**:      + Valider les paramètres.      + Si une validation échoue, lever une exception correspondante (InvalidEventNameError, InvalidEventPriceError).      + Sinon, créer un nouvel objet Event.    * **Sortie**: Objet Event.   filters.js   1. **Filtrer les événements pour aujourd'hui**:    * **Entrée**: Objet Event.    * **Processus**:      + Comparer la date de l'événement à la date actuelle.    * **Sortie**: Boolean (true si l'événement a lieu aujourd'hui, false sinon).  search.js  1. **Recherche d'événements via un prédicat**:    * **Entrée**: Liste d'objets Event, fonction prédicat.    * **Processus**:      + Appliquer la fonction prédicat à chaque événement de la liste.      + Filtrer les événements qui satisfont le prédicat.    * **Sortie**: Liste d'objets Event qui satisfont le prédicat. | |
| **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 event.js  1. **Test Event Creation with Invalid Name (Too Long)**    * **Inputs**: id = 2, name = "A".repeat(201), ticketPrice = 50, totalTickets = 100, ticketsRemaining = 100, date = new Date()    * **Expected Outcome**: InvalidEventNameError thrown. 2. **Test Event Creation with Invalid Ticket Price (Negative)**    * **Inputs**: id = 3, name = "Concert", ticketPrice = -10, totalTickets = 100, ticketsRemaining = 100, date = new Date()    * **Expected Outcome**: InvalidEventPriceError thrown. 3. \*\*Test Create Event with Invalid Name (Too Long) via createEvent    * **Inputs**: name = "A".repeat(201), price = 50, availableTickets = 100    * **Expected Outcome**: InvalidEventNameError thrown.  filters.js  1. **Test Event is Not Today**    * **Inputs**: Event object with date != new Date()    * **Expected Outcome**: false 2. **Test Event is Not Within Next 7 Days**    * **Inputs**: Event object with date outside the next 7 days from today    * **Expected Outcome**: false 3. **Test Event is Not Within Next 30 Days**    * **Inputs**: Event object with date outside the next 30 days from today    * **Expected Outcome**: false  search.js  1. **Test Get Events with No Matching Predicate**    * **Inputs**: events = [Event1, Event2, Event3], searchPredicate = (event) => event.name.includes("Nonexistent")    * **Expected Outcome**: Empty array. | |
| Listin valid data selections | |
| Interface Modules   **Output Data**: Event objects, taglines, Boolean values.   **Data Input**: Parameters for event creation, list of events, search predicates.   **Internal Program Interface**: Functions calling each other within modules.   **External Program Interface**: None 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**: Jest, Mocha, Chai.   **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** | |