|  |  |  |  |
| --- | --- | --- | --- |
| ID and name | UC – 1: Recording a bug | | |
| Primary actor | Quality Assurance Engineer | Secondary actors | None |
| Description | A QA Engineer accesses the Bug Checking Software from either the corporate intranet or external Internet, views the Current Bug List and records a new bug by inputting a name and description of it. The status of a bug is automatically set to “Unsolved”. | | |
| Trigger | A bug was found by the QA Engineer | | |
| Preconditions | PRE – 1: QA Engineer is logged into the BCS | | |
| Postconditions | POST – 1: The bug is added to the CBL with the status “Unsolved”  POST – 2: CBL is updated to reflect the new order of bugs. | | |
| Normal flow | **1.0 Recording a bug**   1. QA Engineer opens the BCS. 2. BCS displays two lists: a list of all the currently unsolved bugs (CBL) and a list of all the solved bugs. (SBL) 3. QA Engineer inputs the name and description of a bug. (see 1.0.E1, 1.0.E2) 4. BCS opens a pop-up window asking the QA Engineer for confirmation whether the inputted data is correct. 5. QA Engineer confirms or cancels the operation and requests to modify the data. (return to 3) 6. CBL is updated with the information of the bug added by the QA and sets its status to “Unsolved”. | | |
| Alternative flows | None | | |
| Exceptions | **1.0.E1 Inputted data is incorrect**  1. BCS informs the QA Engineer either the name or the description of the bug is incorrect (or both)  2a. If the QA Engineer cancels, the current bug is not added to the CBL and the BCS terminates the use case.  2b. Else if the QA Engineer confirms, new data can be inputted and the BCS restarts the use case.  **1.0.E2 The bug has already been solved**  1. BCS informs the QA Engineer that the bug has already been solved by checking the SBL.  2a. If QA Engineer does not want to input a new bug, the BCS terminates the use case.  2b. Else if QA Engineer wants to input a new bug, the BCS restarts the use case. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID and name | UC – 2: Solving a bug | | |
| Primary actor | Programmer | Secondary actors | None |
| Description | A Programmer accesses the Bug Checking Software from either the corporate intranet or external Internet, views the Current Bug List, selects a bug and clicks a button which signals the CBL that the selected bug was solved. The solved bug’s status is set to “Solved”. | | |
| Trigger | A Programmer wants to solve a bug | | |
| Preconditions | PRE – 1: Programmer is logged into the BCS | | |
| Postconditions | POST – 1: The bug’s status is set to “Solved” and is moved from the CBL to the SBL  POST – 2: Both the CBL and the SBL are updated to reflect the new order of bugs. | | |
| Normal flow | **2.0 Solving a bug**   1. Programmer opens the BCS. 2. BCS displays two lists: a list of all the currently unsolved bugs (CBL) and a list of all the solved bugs. (SBL) 3. Programmer selects a bug from the CBL. 4. BCS opens a pop-up window asking the Programmer if the selected item is the correct one. 5. Programmer confirms or cancels the operation and requests to select another bug. (return to 3) 6. BCS updates the selected bug’s status to “Solved” and moves it from the CBL to the SBL. | | |
| Alternative flows | None | | |
| Exceptions | None | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID and name | UC – 3: Modifying a bug | | |
| Primary actor | QA Engineer | Secondary actors | None |
| Description | A QA Engineer accesses the Bug Checking Software from either the corporate intranet or external Internet, views the Current Bug List, selects a bug and clicks a button which allows them to modify the bug’s data. | | |
| Trigger | A QA Engineer wants to modify a bug | | |
| Preconditions | PRE – 1: QA Engineer is logged into the BCS | | |
| Postconditions | POST – 1: The bug’s data is updated.  POST – 2: The change is reflected in the CBL. | | |
| Normal flow | **3.0 Modifying a bug**   1. QA Engineer opens the BCS. 2. BCS displays two lists: a list of all the currently unsolved bugs (CBL) and a list of all the solved bugs. (SBL) 3. QA Engineer selects a bug from the CBL. 4. BCS opens a pop-up window asking the QA Engineer if the selected item is the correct one. 5. QA Engineer confirms or cancels the operation and requests to select another bug. (return to 3) 6. BCS displays the selected bug’s data in the Name and Description text fields. 7. QA Engineer updates the desired field. (see 3.0.E1, 3.0.E2) 8. BCS opens a pop-up window asking the QA Engineer if the inputted data is correct. (see 3.0.E1, 3.0.E2) 9. QA Engineer confirms or cancels the operation and requests to modify the new data. (return to 6) 10. CBL is updated with the new information. | | |
| Alternative flows | None | | |
| Exceptions | **3.0.E1 The modified bug already exists in the CBL**  1. BCS informs the QA Engineer the newly modified bug already exists in the CBL  2a. If QA Engineer cancels the operation, the BCS terminates the use case.  2b. Else if QA Engineer confirms, the BCS restarts the use case and new data can be inputted again.  **3.0.E2 The modified bug has already been solved**  1. BCS informs the QA Engineer that the bug has already been solved by checking the SBL.  2a. If QA Engineer does not want to input a new bug, the BCS terminates the use case.  2b. Else if QA Engineer wants to input a new bug, the BCS restarts the use case. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID and name | UC – 4: Authentication | | |
| Primary actor | Engineer | Secondary actors | None |
| Description | An Engineer accesses the Bug Checking Software from either the corporate intranet or external Internet, by authenticating into the app. | | |
| Trigger | Engineer wants to log into the app. | | |
| Preconditions | PRE – 1: Engineer is employed at the company. | | |
| Postconditions | POST – 1: Engineer is logged into the app. | | |
| Normal flow | **4.0 Authentication**   1. Engineer opens the BCS. 2. BCS prompts the Engineer with a pop-up window to input their username/email and password. 3. Engineer inputs their credentials. 4. BCS performs the login operation. (see 4.0.E1) 5. Engineer confirms or cancels the operation. (return to 3, see 4.1) 6. BCS displays the window. | | |
| Alternative flows | **4.1 Engineer is a Programmer**   1. BCS displays the current and solved bug lists and two buttons: filter and solve. 2. Return to 6.   **4.2 Engineer is a QA Engineer:**   1. BCS displays the current bug list and two text fields: name and description as well as a button to add a bug. 2. Return to 6. | | |
| Exceptions | **4.0.E1 Username/Email/Password are invalid**  1. BCS informs the Engineer either the name, email or password are incorrect. (or all of them)  2a. If the Engineer cancels, the BCS terminates the use case.  2b. Else if the Engineer confirms, new data can be inputted and the BCS restarts the use case. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID and name | UC – 5: View bug list | | |
| Primary actor | QA Engineer | Secondary actors | None |
| Description | A QA Engineer accesses the Bug Checking Software from either the corporate intranet or external Internet and wants to check the current bug list. | | |
| Trigger | QA Engineer wants to check the current bug list. | | |
| Preconditions | PRE – 1: QA Engineer is logged into the BCS | | |
| Postconditions | POST – 1: The BCS displays the current bug list. | | |
| Normal flow | **5.0 View bug list**   1. QA Engineer logs into the BCS. 2. BCS displays the current bug list. | | |
| Alternative flows | None | | |
| Exceptions | None | | |

**Planificare pe iteratii**:

* Iteratia 1: Autentificare, Vizualizare lista bug-uri
* Iteratia 2: Rezolvare bug, Modificare bug, Adaugare bug
* Iteratia 3: Filtrare bug

|  |  |  |  |
| --- | --- | --- | --- |
| ID and name | UC – 6: Filter bug list | | |
| Primary actor | Programmer | Secondary actors | None |
| Description | A Programmer accesses the Bug Checking Software from either the corporate intranet or external Internet and wants to filter the current bug list to check for a bug. | | |
| Trigger | Programmer wants to check the current bug list. | | |
| Preconditions | PRE – 1: Programmer is logged into the BCS | | |
| Postconditions | POST – 1: The BCS displays the filtered bug. | | |
| Normal flow | **6.0 Filter bug list**   1. Programmer opens the BCS. 2. BCS displays the current bug list. 3. Programmer inputs the name of the bug that they need filtered. (see 6.0.E1) 4. BCS updates the current bug list to only show the bug(s) with the given name. | | |
| Alternative flows | None | | |
| Exceptions | **6.0.E1 Name doesn't exist**   1. BCS informs the Programmer that it couldn't return a bug with that given name. 2. BCS terminates the use case. | | |

**5. URMARIRE BUG-URI**

O firma producatoare de software pune la dispozitia programatorilor si verificatorilor sai un system prin care acestia pot sa comunice electronic. Astfel, fiecare dintre angajatii mentionati are la dispoziție un terminal prin care:

* verificatorul poate înregistra un bug, dându-i o denumire si o descriere; imediat dupa

înregistrarea bug-ului, toti programatorii vad lista bug-urilor actualizata cu obiectul nou

introdus;

* programatorul vizualizeaza lista bug-urilor; de asemenea, programatorul poate selecta un bug
* verificatorul poate modifica un bug, schimband denumirea sau descrierea; sau ambele din lista si poate declansa un buton prin care declara ca bug-ul a fost eliminat, caz în care bugul este scos din lista tuturor programatorilor.

**Use case template**

An adaptation of the standard Cockburn template will be used. The template and examples follow:

Descriptions of template fields:

* **ID and name:** Title should be descriptive and should usually begin with a verb, e.g. order, calculate, input, etc. ID can have any format but must be unique among all use cases.
* **Primary actor:** Person that wishes to accomplish a goal through the use of the system. Only a single primary actor per use case.
* **Secondary actors:** Actors that have an interest in the completion of the goal but that do not directly interact with the system.
* **Description:** Concise description of the purpose of the use case.
* **Trigger:** Condition internal or external to the system that prompts the use case to start.
* **Preconditions:** Conditions that must be true before the use case starts. Each should be labeled with an ID unique to the use case.
* **Postconditions:** Conditions that must be true after the use case ends normally. Each should be labeled with an ID unique to the use case.
* **Normal flow:** Detailed step-by-step description of the logical flow of the use case. It should describe an explicit two way interaction, with the system prompting for input and the actor responding accordingly. Each step should be numbered.
* **Alternative flows:** Flows that achieve the same goal as the normal flow but are expected to be less common or lower priority.
* **Exceptions:** Conditions that result in the normal flow ending prematurely due to an unrecoverable condition in the system. The condition that causes the flow should be clearly stated, as should be any other decisions that the actor must make in this situation.

Examples

For a hypothetical *Cafeteria Ordering System*:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID and name** | UC-1: Order a Meal | | |
| **Primary actor** | Patron | **Secondary actors** | Cafeteria Inventory System |
| **Description** | A Patron accesses the Cafeteria Ordering System from either the corporate intranet or external Internet, views the menu for a specific date, selects food items, and places an order for a meal to be picked up in the cafeteria or delivered to a specified location within a specified 15-minute time window. | | |
| **Trigger** | A Patron indicates that he wants to order a meal. | | |
| **Preconditions** | PRE-1. Patron is logged into COS.  PRE-2. Patron is registered for meal payments by payroll deduction. | | |
| **Postconditions** | POST-1. Meal order is stored in COS with a status of “Accepted.”  POST-2. Inventory of available food items is updated to reflect items in this order.  POST-3. Remaining delivery capacity for the requested time window is updated. | | |
| **Normal flow** | **1.0 Order a Single Meal**   * Patron asks to view menu for a specific date. (see 1.0.E1, 1.0.E2) * COS displays menu of available food items and the daily special. * Patron selects one or more food items from menu. (see 1.1) * Patron indicates that meal order is complete. (see 1.2) * COS displays ordered menu items, individual prices, and total price, including taxes and delivery charge. * Patron either confirms meal order (continue normal flow) or requests to modify meal order (return to step 2). * COS displays available delivery times for the delivery date. * Patron selects a delivery time and specifies the delivery location. * Patron specifies payment method. * COS confirms acceptance of the order. * COS sends Patron an email message confirming order details, price, and delivery instructions. * COS stores order, sends food item information to Cafeteria Inventory System, and updates available delivery times. | | |
| **Alternative flows** | **1.1 Order multiple identical meals**   * Patron requests a specified number of identical meals. (see 1.1.E1) * Return to step 4 of normal flow.   **1.2 Order multiple meals**   * Patron asks to order another meal. * Return to step 1 of normal flow. | | |
| **Exceptions** | **1.0.E1 Requested date is today and current time is after today’s order cutoff time**  1. COS informs Patron that it’s too late to place an order for today.  2a. If Patron cancels the meal ordering process, then COS terminates use case.  2b. Else if Patron requests another date, then COS restarts use case.  **1.0.E2 No delivery times left**  1. COS informs Patron that no delivery times are available for the meal date.  2a. If Patron cancels the meal ordering process, then COS terminates use case.  2b. Else if Patron requests to pick the order up at the cafeteria, then continue with normal flow, but skip steps 7 and 8.  **1.1.E1 Insufficient inventory to fulfill multiple meal order**  1. COS informs Patron of the maximum number of identical meals he can order, based on current available inventory.  2a. If Patron modifies number of meals ordered, then return to step 4 of normal flow.  2b. Else if Patron cancels the meal ordering process, then COS terminates use case. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID and name** | UC-5 Register for Payroll Deduction | | |
| **Primary actor** | Patron | **Secondary actors** | Payroll System |
| **Description** | Cafeteria patrons who use the COS and have meals delivered must be registered for payroll deduction. For noncash purchases made through the COS, the cafeteria will issue a payment request to the Payroll System, which will deduct the meal costs from the next scheduled employee payday direct deposit. | | |
| **Trigger** | Patron requests to register for payroll deduction, or Patron says yes when COS asks if he wants to register. | | |
| **Preconditions** | PRE-1. Patron is logged into COS. | | |
| **Postconditions** | POST-1. Patron is registered for payroll deduction. | | |
| **Normal flow** | **5.0 Register for Payroll Deduction**   * COS asks Payroll System if Patron is eligible to register for payroll deduction. * Payroll System confirms that Patron is eligible to register for payroll deduction. * COS asks Patron to confirm his desire to register for payroll deduction. * If so, COS asks Payroll System to establish payroll deduction for Patron. * Payroll System confirms that payroll deduction is established. * COS informs Patron that payroll deduction is established. | | |
| **Alternative flows** | None | | |
| **Exceptions** | 5.0.E1 Patron is not a full time employee.  5.0.E2 Patron is already enrolled for payroll deduction. | | |

Extra step: Traceability

For this extra step, you will add traceability information for each use case by adding a new field to the template:

|  |  |
| --- | --- |
| Method-level traces | <fully.qualified.ClassName>#<methodName>  ... |

Any method that implements the functionality described in the normal flow, alternative flow or exceptions should be included in this field. This means that the method that is initially executed and any methods of any classes that the work is delegated to should be included.

Examples for previous use cases:

UC-1:

|  |  |
| --- | --- |
| Method-level traces | my.company.ordering.MenuWidget#dateClicked  my.company.ordering.MenuWidget#completeOrder  my.company.ordering.InventoryInterface#checkInventory  ... |

UC-5:

|  |  |
| --- | --- |
| Method-level traces | my.company.payroll.PayrollInterface#checkEligibility  my.company.payroll.RegistrationForm#confirm  ... |