AIN SHAMS UNIVERSITY FACULTY OF ENGINEERING

International Credit Hours Engineering Programs

Object Oriented Programming

Team 22

Submitted to:

Dr. Mahmoud Ibrahim Khalil

Eng. Ahmed Hossam

Submitted by:

Name	ID
Abdelrahman Mohamed	23P0203
Hussein Mohamed Kamal	23P0416
Saif Eldin Baher	23P0153
Youssef Khaled Abdelkarim	23P0012

Contents

Business Logic of the Agile Tool Project

1. User Registration and Login Process

- Registration: When a user wants to join the system, they need to register by providing a name, email, password, and their role (either ScrumMaster, Developer, QAEngineer, Stakeholder).
- Login: Once registered, the user can log in using their email and password. If the login is successful, the user is granted access to the system. Each user can only have one active session at a time.
- Role-Based Permissions:
    - User: This is an abstract class that acts as a parent class of all users and give the base template.
    - ScrumMaster: Can create sprints, assign team members, and manage work items across sprints.
    - TechnicalStaff: This is an abstract class that acts as a parent class of Developer and QAEngineer classes.
    - Developer: Can be assigned tasks or bugs, change the status of tasks/bugs, and mark them as "READY_FOR_TEST".
    - QAEngineer: Can be assigned tasks or bugs, change the status of tasks/bugs to "DONE" once they are ready for testing.
    - Stakeholder: Can create Epics and Stories but cannot manage task statuses. They can view high level work items in the sprint (such as Epics and Stories).

2. Sprint Management (ScrumMaster Role)

- A Sprint is a time-boxed period for completing a set of tasks or work items.
- The ScrumMaster creates the sprint by specifying a name, objective, startDate, and endDate.
- The ScrumMaster also adds work items to the sprint. Their assignees are added to the sprint's team.
- Work Items (Tasks, Bugs, Epics, Stories) are associated with a Sprint, and team members are assigned to the work items according to their roles.

3. Work Item Creation (ScrumMaster, Developer, Stakeholder)

Work items are central to the Agile methodology and include the following types:

- Tasks: The smallest unit of work. These are created by the ScrumMaster or Developer, and they are typically assigned to Developers.
- Bugs: Represent issues in the system that need to be fixed. They follow the same process as tasks but are specifically for fixing defects.
- Stories: High-level features that describe a specific functionality from the user's perspective. Stakeholders or ScrumMasters can create them, and they can be assigned to a Developer and they act as a container for subtasks.

- Epics: Larger features that group multiple stories. They can be created by the Stakeholder or ScrumMaster, and they act as a container for related stories.

Each work item has a title, description, status, and assigned user. The status starts as TODO and can be updated by the user assigned to the work item.

4. Task Assignment to Developers

When a task is created, it doesn't initially have an assigned user. The ScrumMaster or Developer can assign the task to any user (usually a Developer).

1. The task is created using a template that includes:
    - Title: Describes the task (e.g., "Implement login functionality").
    - Description: A detailed explanation of the task.
    - Assigned User: Initially null, but it can be assigned later.
    - Status: Initially set to TODO.
2. Once a Developer is assigned to the task, it appears in their list of work items.

5. Task Template:

Here is the template for a Task:

- Title: String (e.g., "Develop authentication API").
- Description: String (e.g., "Develop an API to handle user login and authentication").
- Assigned User: User (e.g., a Developer assigned to the task).
- Status: Enum (TODO, IN_PROGRESS, READY_FOR_TEST, DONE).
- Created By: User (the user who created the task).

6. Changing Task or Bug Status

The main logic for status changes depends on the role of the user assigned to the task and the current status of the task:

- Developer: Can moveathem through the following states:
    - TODO → IN_PROGRESS: When the Developer starts working on a task.
    - IN_PROGRESS → READY_FOR_TEST: When the task is ready to be tested (optional step, can also be done directly to DONE).
- QA Engineer: Can only move a task to DONE once it has been marked as READY_FOR_TEST by the developer.
- Business Logic for Status Change:
    - A Developer can change a task's status from TODO to IN_PROGRESS or READY_FOR_TEST.
    - Once the task reaches READY_FOR_TEST, the Developer cannot change it directly to DONE; it needs a QAEngineer to do that.
    - A QAEngineer can change the status from READY_FOR_TEST to DONE.

Example:

- Developer Action: A Developer is assigned a task like "Develop login page". Initially, the task is TODO.
1. The Developer starts working on the task and updates its status to IN_PROGRESS.
2. Once the development is completed, the Developer changes the status to READY_FOR_TEST.
- QA Engineer Action: Once the task is READY_FOR_TEST, a QAEngineer tests it. If the task passes testing, the QAEngineer moves the status to DONE.

7. Status of Epics and Stories

Epic Status

- TODO: The Epic is created, but no stories are being worked on.
- IN_PROGRESS: At least one story is in progress.
- DONE: All stories in the Epic are complete.

Epic Status Update Logic:

- The Epic starts as TODO.
- When a story is worked on, the Epic status updates to IN_PROGRESS.
- When all stories are marked DONE, the Epic status changes to DONE.

Story Status

- TODO: The story is created, but work hasn't started.
- IN_PROGRESS: The story is being worked on by a developer.
- READY_FOR_TEST: All tasks in the story are completed and ready for testing.
- DONE: The story has passed QA testing.

Story Status Update Logic:

- A story starts as TODO.
- It changes to IN_PROGRESS when development begins.
- Once all tasks are complete, the story is set to READY_FOR_TEST, and then to DONE after QA verification.

Business Logic:

- Epics are completed when all associated Stories are DONE.
- Stories are completed when all tasks within them are finished and successfully tested

8. Work Item to User Assignment Flow

1. A work item is created, but the AssignedUser is initially null.

2. The work item is then assigned to a Developer or QAEngineer. This assignment ensures that the user has the work item in their list.
3. The user can then update the status of the work item as per their role and the current status.

9. Viewing Work Items

- Stakeholders**:** They can only view Epics and Stories in the sprint. They cannot change the status of tasks or bugs, and their role is primarily to track high-level progress and provide feedback.
- ScrumMaster**:** The ScrumMaster can manage the sprint, add users, and assign work items to team members.
- Developer/QAEngineer**:** These roles can manage the day-to-day task assignments, work item status updates, and report on progress.

Flow of Business Logic in the System

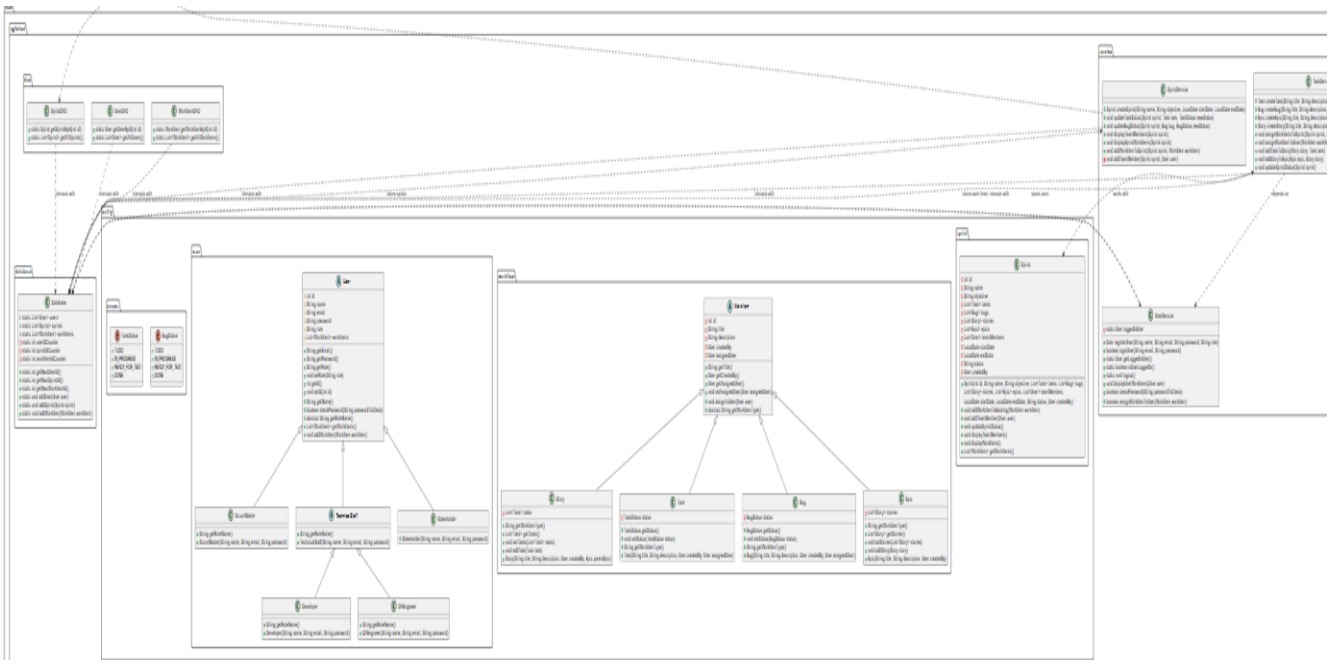
1. Sprint Creation:
   - A ScrumMaster creates a sprint, defining the sprint's objective, dates, and team members.
2. Work Item Creation:
   - Work items are created by the ScrumMaster or Stakeholder. These work items are tasks, bugs, stories, and epics that are linked to the sprint.
3. Assigning Work Items:
   - The ScrumMaster assigns work items to team members (mostly Developers or QAEngineers).
   - Work items are placed in the sprint's backlog.
4. Task Progression:
   - A Developer works on the task, changes its status from TODO to IN_PROGRESS, and then potentially to READY_FOR_TEST.
   - A QAEngineer reviews the work and changes the task's status to DONE if it's ready.
5. Sprint Status:
   - The sprint's overall status is based on the statuses of all the work items inside the sprint. If all tasks, bugs, stories, and epics are marked as DONE, the sprint status is updated to COMPLETED. Otherwise, it remains ACTIVE.

Business Logic for Task Status Change

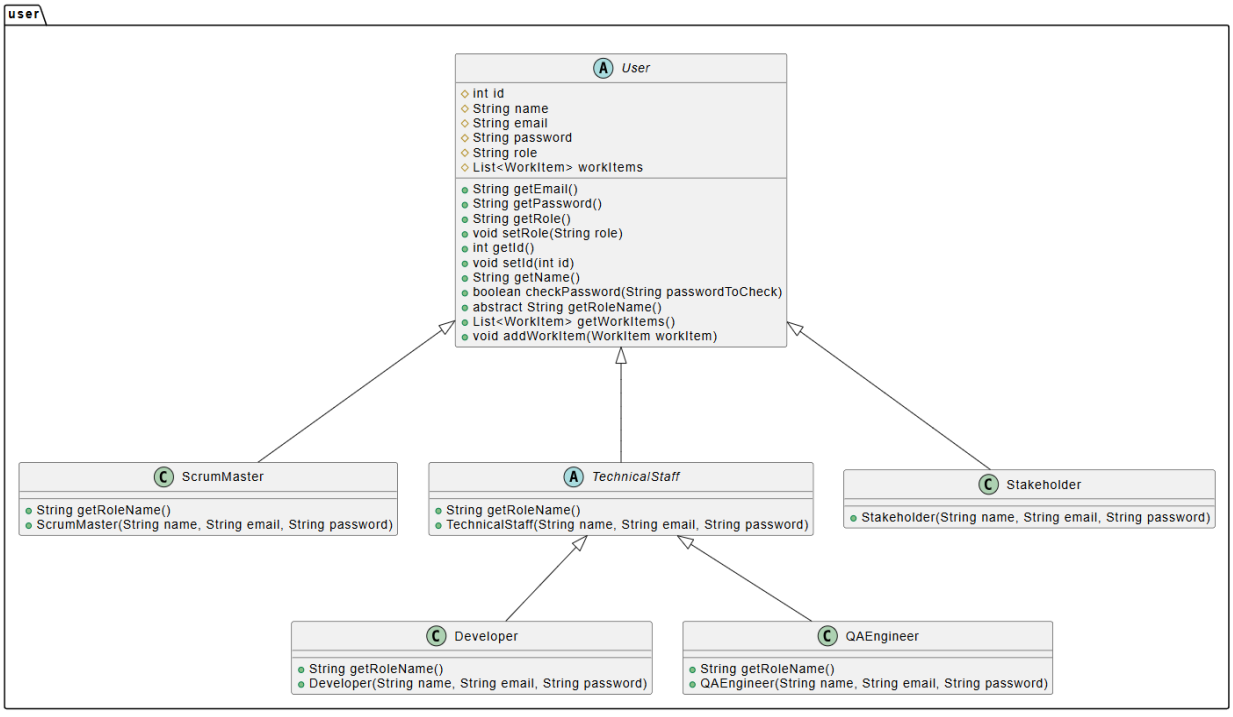
- Step 1: A task is created and assigned to a Developer.
- Step 2: The Developer updates the task status from TODO to IN_PROGRESS or READY_FOR_TEST.
- Step 3: Once the task reaches READY_FOR_TEST, the Developer cannot change it directly to DONE; only the QA engineer can do that.

10. UML Diagram

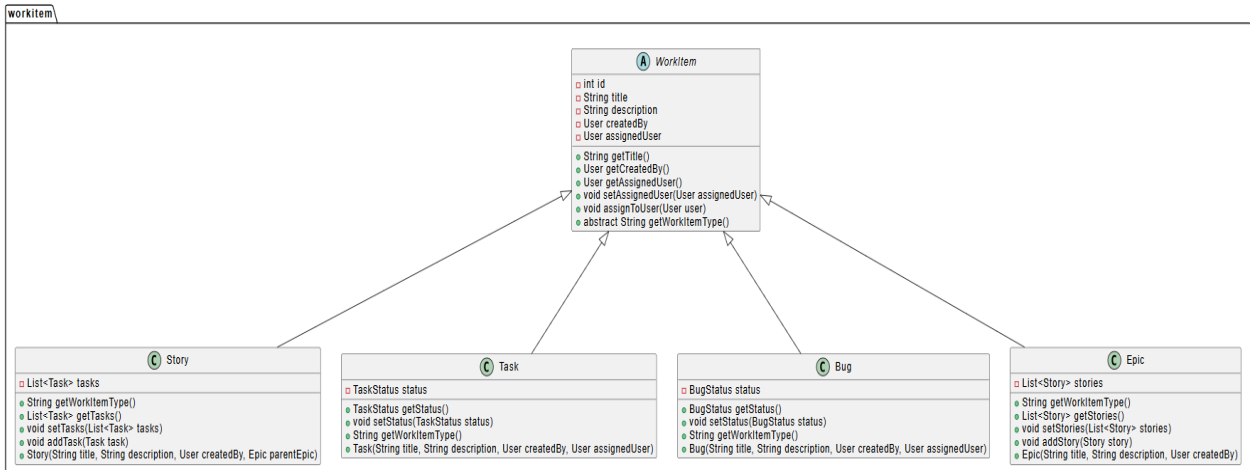
Whole diagram



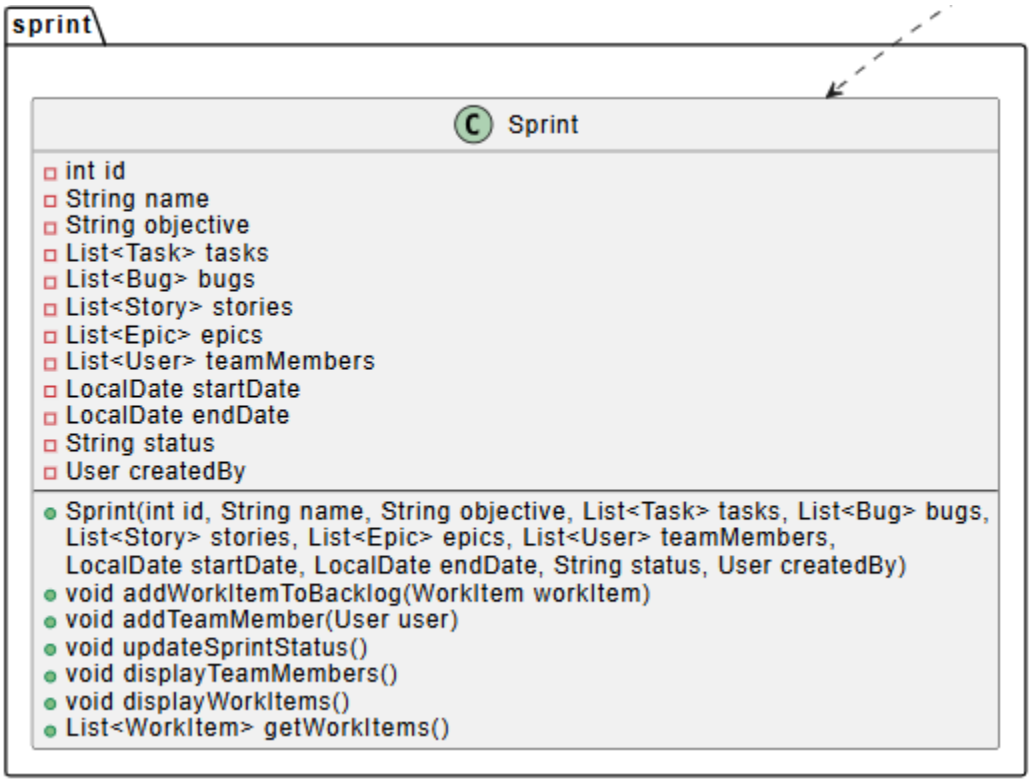
User classes



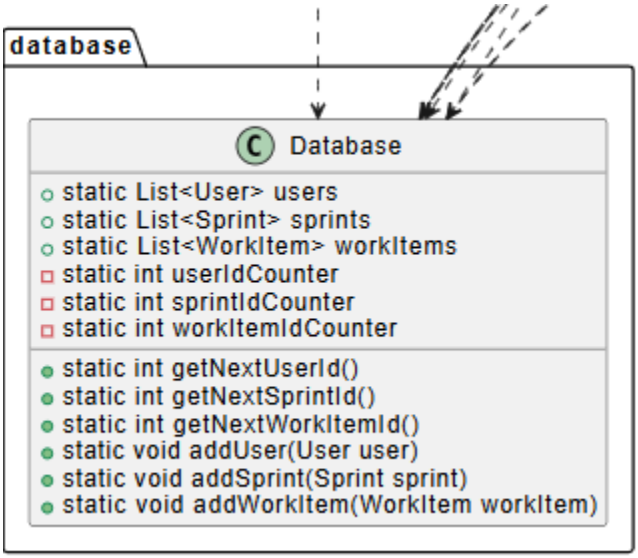
Work Item classes



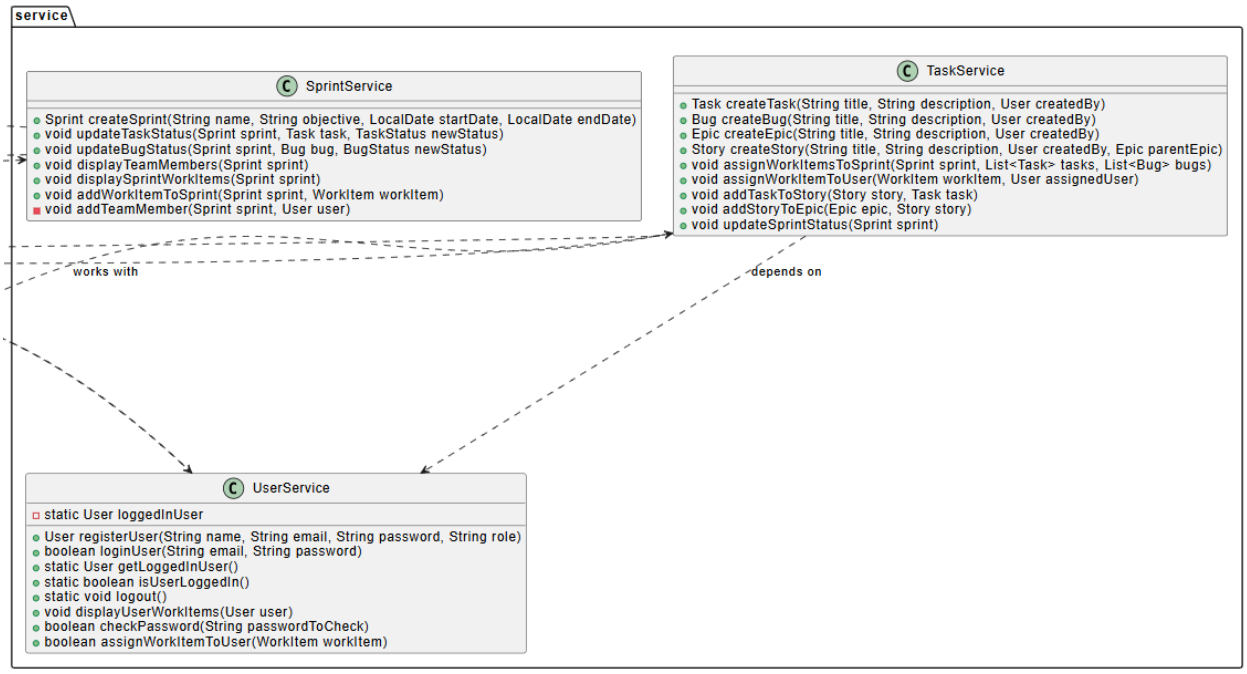
Sprint class

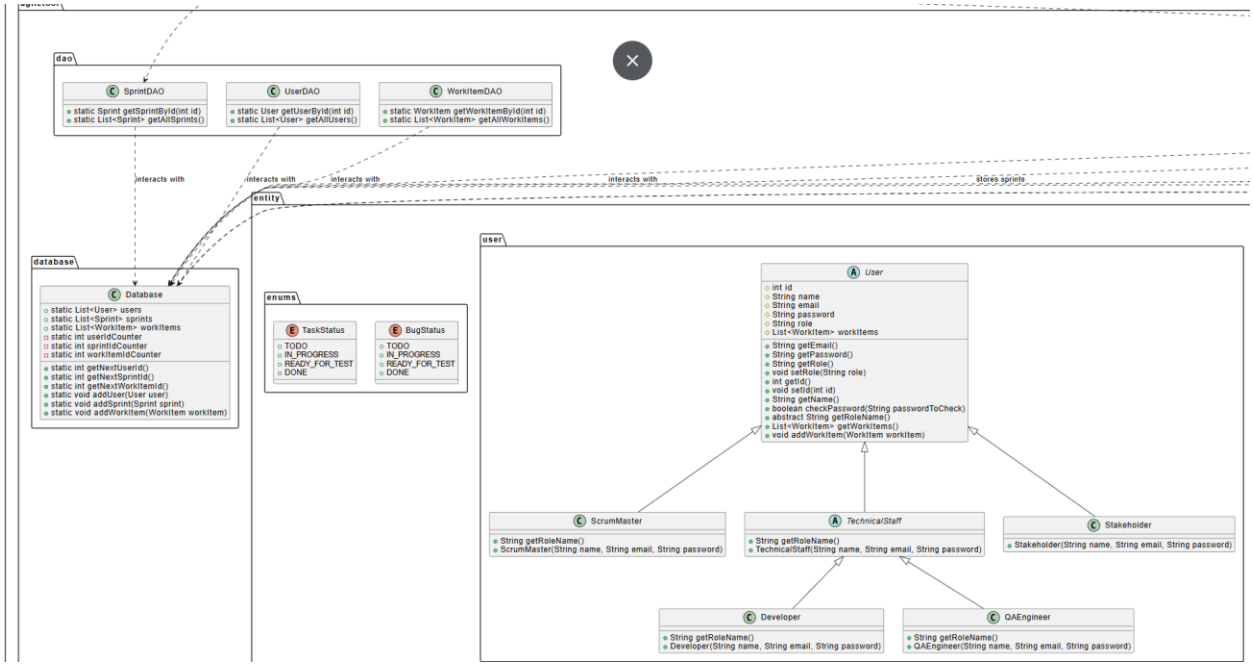


Data base class



Services Classes





Conclusion

This approach encapsulates the flow of tasks and work items in an Agile system, ensuring that each user role has clear responsibilities and constraints when interacting with work items. The business logic ensures that tasks follow a clear life cycle from creation to completion, with proper user role validation at each stage.

Supported functionalities in Detail class by class

1. Entity package

1.1 User package

1.1.1: User Class (Abstract)

Purpose:
The User class is an **abstract base class** for all users in the system, providing common attributes and methods for different types of users.

Attributes

- protected int id: Unique identifier for the user.
- protected String name: User's name.
- protected String email: User's email address.
- protected String password: User's password (should be hashed in real-world systems).
- protected String role: User's role (e.g., ScrumMaster, Developer).
- protected List<WorkItem> workItems: List of work items assigned to the user.

Constructors

- public User(String name, String email, String password, String role): Initializes name, email, password, role, and an empty workItems list.

Methods

- public String getEmail(), public String getPassword(), public String getRole(): Getters for email, password, and role.
- public void setRole(String role): Sets the user's role.
- public void setId(int id), public int getId(): Getters and setters for ID.
- public String getName(): Getter for the user's name.
- public boolean checkPassword(String passwordToCheck): Compares the entered password with the stored password.
- public abstract String getRoleName(): Abstract method to be implemented by subclasses.
- public List<WorkItem> getWorkItems(), public void addWorkItem(WorkItem workItem): Methods for managing work items.

1.1.2 TechnicalStaff Class (Abstract)

Purpose
Represents a **generic technical staff** user. This abstract class can be extended by other more specific technical roles, such as SystemArchitect or InfrastructureEngineer.

Attributes
Inherits attributes from the User class.

Constructors

- public TechnicalStaff(String name, String email, String password): Initializes the role as "TechnicalStaff".

Methods

- public String getRoleName(): Returns "TechnicalStaff".
- public List<WorkItem> getWorkItems(), public void addWorkItem(WorkItem workItem): Methods inherited from User to manage work items.

1.1.3. Developer Class

Purpose:
Represents a user with the role of a **Developer**. Inherits from the User class and specifies the role as "Developer".

Attributes
Inherits attributes from the User class.

Constructors:

- public Developer(String name, String email, String password): Initializes the role as "Developer".

Methods:

- public String getRoleName(): Returns "Developer".

1.1.4. QAEngineer Class

Purpose:
Represents a user with the role of a **QA Engineer**. Inherits from the User class and specifies the role as "QA".

Attributes:
Inherits attributes from the User class.

Constructors:

- public QAEngineer(String name, String email, String password): Initializes the role as "QA".

Methods:

- public String getRoleName(): Returns "QA".

1.1.5. ScrumMaster Class

Purpose:
Represents a user with the role of a **ScrumMaster**. Inherits from the User class and specifies the role as "ScrumMaster".

Attributes
Inherits attributes from the User class.

Constructors

- public ScrumMaster(String name, String email, String password): Initializes the role as "ScrumMaster".

Methods

- public String getRoleName(): Returns "ScrumMaster".

1.1.6. Stakeholder Class

Purpose:
Represents a **Stakeholder** user in the system. Stakeholders can create Epics and Stories, but cannot change task or bug statuses.

Attributes:
Inherits attributes from the User class.

Constructors

- public Stakeholder(String name, String email, String password): Initializes the role as "Stakeholder".

Methods:

- public String getRoleName(): Returns "Stakeholder".
- public Epic createEpic(String title, String description, User createdBy): Allows stakeholders to create an Epic.
- public Story createStory(String title, String description, User createdBy): Allows stakeholders to create a Story.
- public void viewWorkItemsInSprint(Sprint sprint): Displays only Epics and Stories in the sprint.

1.2.Workietms

1.2.1. WorkItem Class (Abstract)

Purpose:
The WorkItem class is the **abstract base class** for all types of work items (e.g., Task, Bug, Epic, Story). It contains common properties and methods for managing work items within the system.

Attributes

- private int id: Unique identifier for the work item.
- private String title: The title or name of the work item.
- private String description: A description of the work item.
- private String status: Status of the work item (e.g., "TODO", "IN_PROGRESS", "DONE").
- private User createdBy: The user who created the work item.
- private User assignedUser: The user assigned to the work item.

Constructors

- public WorkItem(String title, String description, User createdBy): Initializes the title, description, and createdBy attributes. The assignedUser is set to null by default, and status is initialized to "TODO".

Methods

- public int getId(), public void setId(int id): Getter and setter for the ID.
- public String getTitle(): Returns the title of the work item.
- public String getDescription(): Returns the description of the work item.
- public String getStatus(): Returns the current status of the work item.
- public void setStatus(String status): Sets the status of the work item.
- public User getCreatedBy(): Returns the user who created the work item.
- public void setCreatedBy(User createdBy): Sets the user who created the work item.
- public User getAssignedUser(): Returns the user assigned to the work item.
- public void setAssignedUser(User assignedUser): Sets the user assigned to the work item.
- public void assignToUser(User user): Assigns the work item to a user and adds it to their list of work items.

Abstract Method

- public abstract String getWorkItemType(): An abstract method to be implemented by subclasses to return the specific type of work item (e.g., "Task", "Bug", "Epic", "Story").

1.2.2. Task Class

Purpose:
The Task class represents a **Task** work item. It extends WorkItem and provides specific behavior for tasks in the system.

Attributes

- Inherits from WorkItem.

Constructors

- public Task(String title, String description, User createdBy, User assignedUser): Initializes the task with a title, description, creator, and assigned user.

Methods

- public String getWorkItemType(): Returns "Task" to specify that this is a task work item.

1.2.3. Bug Class

Purpose:
The Bug class represents a **Bug** work item. It extends WorkItem and provides specific behavior for bugs in the system.

Attributes:

- Inherits from WorkItem.

Constructors:

- public Bug(String title, String description, User createdBy, User assignedUser): Initializes the bug with a title, description, creator, and assigned user.

Methods:

- public String getWorkItemType(): Returns "Bug" to specify that this is a bug work item.

1.2.4. Epic Class

Purpose:
The Epic class represents an **Epic** work item, which can contain multiple **Stories**. It extends WorkItem and is typically used to track large bodies of work.

Attributes

- private List<Story> stories: A list of stories associated with the epic.

Constructors

- public Epic(String title, String description, User createdBy): Initializes the epic with a title, description, and creator. The stories list is initialized as an empty list.

Methods:

- public String getWorkItemType(): Returns "Epic" to specify that this is an epic work item.
- public List<Story> getStories(): Returns the list of stories associated with the epic.
- public void setStories(List<Story> stories): Sets the list of stories for the epic.
- public void addStory(Story story): Adds a story to the epic.
- public void updateStatus(): Updates the status of the epic based on the status of its stories (if all stories are done, the epic is marked as DONE).

1.2.5. Story Class

Purpose:
The Story class represents a **Story** work item. It extends WorkItem and is typically used to represent smaller units of work that contribute to an Epic.

Attributes

- private List<Task> tasks: A list of tasks associated with the story.

Constructors

- public Story(String title, String description, User createdBy, Epic parentEpic): Initializes the story with a title, description, creator, and optional reference to the parent epic.

Methods

- public String getWorkItemType(): Returns "Story" to specify that this is a story work item.
- public List<Task> getTasks(): Returns the list of tasks associated with the story.
- public void setTasks(List<Task> tasks): Sets the list of tasks for the story.
- public void addTask(Task task): Adds a task to the story.
- public void updateStatus(): Updates the status of the story based on the status of its tasks (if all tasks are done, the story is marked as DONE).

Conclusion

The **WorkItem** package defines the core work items in the system. The **WorkItem** class serves as the abstract base class, with specific implementations for **Task**, **Bug**, **Epic**, and **Story**. Each of these work items has a unique status and is managed based on its type.

- **Tasks** represent the smallest unit of work, typically assigned to a Developer.
- **Bugs** represent issues that need to be fixed and can be assigned to a Developer.
- **Epics** represent large units of work and can contain multiple Stories.
- **Stories** are smaller features that contribute to an Epic and consist of multiple tasks.

This package handles all functionality for work items, including creation, assignment, status updates, and linking between related work items (e.g., tasks within stories, stories within epics).

1.3.Sprint

1.3.1. Sprint Class

Purpose

The Sprint class represents a **Sprint** in the Agile framework. A Sprint is a time-boxed iteration in which a specific set of work items (e.g., tasks, bugs, stories, epics) is completed. The Sprint class holds various attributes related to the sprint's metadata, status, and the work items it contains.

Attributes

- private int id: A unique identifier for the sprint.
- private String name: The name of the sprint (e.g., "Sprint 1").
- private String objective: The goal or objective of the sprint (e.g., "Implement user authentication").
- private List<Task> tasks: A list of tasks associated with the sprint.

- private List<Bug> bugs: A list of bugs associated with the sprint.
- private List<Story> stories: A list of stories associated with the sprint.
- private List<Epic> epics: A list of epics associated with the sprint.
- private List<User> teamMembers: A list of team members working on the sprint.
- private LocalDate startDate: The start date of the sprint.
- private LocalDate endDate: The end date of the sprint.
- private String status: The status of the sprint (e.g., "ACTIVE", "COMPLETED").
- private User createdBy: The user (typically the ScrumMaster) who created the sprint.

Constructors

- public Sprint(int id, String name, String objective, List<Task> tasks, List<Bug> bugs, List<Story> stories, List<Epic> epics, List<User> teamMembers, LocalDate startDate, LocalDate endDate, String status, User createdBy):
    - Initializes a new Sprint object with the given id, name, objective, work items (tasks, bugs, stories, epics), teamMembers, startDate, endDate, status, and the createdBy user.
    - The lists (tasks, bugs, stories, epics, teamMembers) are initialized as empty lists if not provided.

Methods

- public int getId(), public void setId(int id): Getters and setters for the sprint ID.
- public String getName(), public void setName(String name): Getters and setters for the sprint name.
- public String getObjective(), public void setObjective(String objective): Getters and setters for the sprint objective.
- public List<Task> getTasks(), public void setTasks(List<Task> tasks): Getters and setters for the list of tasks in the sprint.
- public List<Bug> getBugs(), public void setBugs(List<Bug> bugs): Getters and setters for the list of bugs in the sprint.
- public List<Story> getStories(), public void setStories(List<Story> stories): Getters and setters for the list of stories in the sprint.
- public List<Epic> getEpics(), public void setEpics(List<Epic> epics): Getters and setters for the list of epics in the sprint.
- public List<User> getTeamMembers(), public void setTeamMembers(List<User> teamMembers): Getters and setters for the list of team members in the sprint.
- public LocalDate getStartDate(), public void setStartDate(LocalDate startDate): Getters and setters for the sprint start date.
- public LocalDate getEndDate(), public void setEndDate(LocalDate endDate): Getters and setters for the sprint end date.
- public String getStatus(), public void setStatus(String status): Getters and setters for the sprint status.
- public User getCreatedBy(), public void setCreatedBy(User createdBy): Getters and setters for the user who created the sprint.

Functional Methods

- public void addWorkItemToBacklog(WorkItem workItem):
  - Adds a work item (task, bug, story, or epic) to the sprint's backlog and adds the assigned user to the sprint's team if they are not already part of the team.
- private void addTeamMember(User user):
  - Adds a user to the sprint team if they are not already in the team.
- public void updateSprintStatus():
  - Updates the status of the sprint based on the status of its work items (tasks, bugs, stories, and epics). The sprint is marked as COMPLETED if all its tasks and bugs are done; otherwise, it is marked as ACTIVE.
- public void displayTeamMembers():
  - Displays all team members working in the sprint.
- public void displayWorkItems():
  - Displays all work items in the sprint (tasks, bugs, stories, epics).
- public List<WorkItem> getWorkItems():
  - Returns a combined list of all work items (tasks, bugs, stories, epics) in the sprint.

Status Logic

The **status of a sprint** is determined based on the progress of its work items (tasks, bugs, stories, epics):

- **ACTIVE**: The sprint is ongoing, and work items are still in progress.
- **COMPLETED**: All work items (tasks, bugs, stories, and epics) are marked as "DONE."

Conclusion

The **Sprint** class is the core component that represents a sprint in the Agile framework. It organizes various work items (tasks, bugs, stories, and epics) and tracks the progress of the sprint. It includes methods for adding and managing work items, team members, and updating the sprint status based on work item completion.

1.4.Enums

1.4.1. TaskStatus Enum

Purpose
The TaskStatus enum represents the possible states of a **Task**.

Values

- TODO: Task is not started.
- IN_PROGRESS: Task is being worked on.
- READY_FOR_TEST: Task is completed and ready for testing.
- DONE: Task is fully completed.

Usage
Used in the Task class to track and update the status of tasks.

1.4.2. BugStatus Enum

Purpose:
The BugStatus enum defines the possible statuses for a **Bug**.

Values

- TODO: Bug has been reported but not yet addressed.
- IN_PROGRESS: Bug is being fixed.
- READY_FOR_TEST: Bug is fixed and ready for testing.
- DONE: Bug is resolved and closed.

Usage
Used in the Bug class to manage the status of bugs.

2. DAO package

2.1. SprintDAO Class

Purpose:

The SprintDAO (Data Access Object) class is responsible for managing the **CRUD (Create, Read, Update, Delete)** operations related to **Sprint** objects. It interacts with the Database class to store and retrieve sprint data.

Attributes

- **No specific attributes** are defined in this class. All operations are static and directly interact with the Database class, which holds the list of sprints.

Methods

- public static Sprint getSprintById(int id):
  - Purpose: Retrieves a sprint from the Database.sprints list by its ID.
  - Parameters: **int id - the ID of the sprint to retrieve.**
  - Returns**:** Sprint - the sprint object with the given ID, or null if not found.
  - Usage**:** This method allows the system to fetch a specific sprint by its unique ID.
- public static List<Sprint> getAllSprints():
  - Purpose: **Returns the entire list of sprints stored in the Database.sprints list.**
  - Returns: **List<Sprint> - a list containing all sprint objects.**
  - Usage: **Used** to retrieve all sprints for viewing or processing.

Conclusion:

The SprintDAO class serves as a simple data access layer for **sprints**. It allows for adding new sprints, retrieving a sprint by ID, and fetching all sprints. It interacts directly with the Database class, which acts as the data storage mechanism for sprints in the system.

2.2 UserDAO Class

Purpose:

The UserDAO (Data Access Object) class is responsible for managing all data access operations related to User objects. It serves as the intermediary between the business logic layer and the data storage system (Database class), handling CRUD operations for user entities in the system.

Attributes

- **No specific attributes** are defined in this class. All operations are static and directly interact with the Database class, which holds the list of sprints.

Methods:

1. **public static User getUserById(int id)**
- **Purpose**: Retrieves a specific user from the system based on their unique identifier.
- **Parameters**: int id - The unique ID of the user to retrieve
- **Returns**: User object if found,null if no user exists with the given ID

2. public static List<User> getAllUsers()
- **Purpose**: Retrieves a complete list of all registered users in the system.
- **Parameters**: None
- **Returns**: List<User> containing all user objects, Empty list if no users exist in the database

2.3 WorkItem DAO Class

Purpose:

The WorkItemDAO class manages all data operations for WorkItem objects in the system. It provides the essential data access layer for work items (which could include tasks, bugs, user stories, or features), enabling creation, retrieval, and management of work items throughout their lifecycle.

Attributes:

- Static Implementation: All methods are static, requiring no class instantiation
- Centralized Data Access: Direct interface to Database.workItems collection

Methods:

1. public static WorkItem getWorkItemById(int id)
- Purpose: Fetches a specific work item using its unique identifier.
- Parameters: int id - The unique ID of the work item to retrieve
- Returns: WorkItem object if found, null if no work item exists with the given ID

2. public static List<WorkItem> getAllWorkItems()
- Purpose: Retrieves a comprehensive list of all work items in the system.
- Parameters: None
- Returns: List<WorkItem> containing all work item objects, Empty list if database contains no work items

3. Database package

3.1. Database Class

Purpose:

The Database class serves as an **in-memory data storage** for the application. It holds the lists for users, sprints, work items, and counters for generating unique IDs. This class simulates the persistence layer, storing the application's data as static lists.

Attributes:

- public static List<User> users: A static list that holds all user objects.
- public static List<Sprint> sprints: A static list that holds all sprint objects.
- public static List<WorkItem> workItems: A static list that holds all work item objects (tasks, bugs, stories, epics).
- private static int userIdCounter: A static counter for generating unique user IDs.
- private static int sprintIdCounter: A static counter for generating unique sprint IDs.
- private static int workItemIdCounter: A static counter for generating unique work item IDs.

Methods:

- **ID Generation Methods:**
  - public static int getNextUserId():
    - **Purpose:** Generates and returns the next unique user ID.
    - **Returns:** int - The next available user ID.
  - public static int getNextSprintId():
    - **Purpose:** Generates and returns the next unique sprint ID.
    - **Returns:** int - The next available sprint ID.
  - public static int getNextWorkItemId():
    - **Purpose:** Generates and returns the next unique work item ID.
    - **Returns:** int - The next available work item ID.
- **Add Methods:**
  - public static void addUser(User user):
    - **Purpose:** Adds a user to the users list and assigns it a unique ID.
    - **Parameters:** User user - The user to be added.
  - public static void addSprint(Sprint sprint):
    - **Purpose:** Adds a sprint to the sprints list and assigns it a unique ID.
    - **Parameters:** Sprint sprint - The sprint to be added.
  - public static void addWorkItem(WorkItem workItem):
    - **Purpose:** Adds a work item (task, bug, story, epic) to the workItems list and assigns it a unique ID.
    - **Parameters:** WorkItem workItem - The work item to be added.

Conclusion:

The Database class serves as a simple in-memory storage solution for users, sprints, and work items in the application. It provides methods to add entities to the static lists and generates unique IDs for users, sprints, and work items. This class simulates a persistent data layer but stores data only for the duration of the program's execution.

4. Database package

4.1. SprintService Class

Purpose:

The SprintService class provides business logic and operations for managing **sprints**. It handles the creation of sprints, updates, and other operations such as adding team members and work items.

Attributes:

- **No specific attributes** defined. All methods are static and directly interact with the Database and other system components.

Methods:

- public Sprint createSprint(String name, String objective, LocalDate startDate, LocalDate endDate):
  - **Purpose:** Creates a new sprint with the provided name, objective, start date, and end date. Only accessible by a logged-in **ScrumMaster**.
  - **Parameters:** name, objective, startDate, endDate.
  - **Returns:** The created Sprint object.
  - **Usage:** Called by a ScrumMaster to create a new sprint.
- public void updateTaskStatus(Sprint sprint, WorkItem workItem, String newStatus):
  - **Purpose:** Updates the status of a task in the sprint, based on the role of the logged-in user (either **Developer** or **QA Engineer**).
  - **Parameters:** Sprint sprint, WorkItem workItem, String newStatus.
  - **Usage:** Called when a task status needs to be updated (e.g., from TODO to IN_PROGRESS).
- public void updateBugStatus(Sprint sprint, WorkItem workItem, String newStatus):
  - **Purpose:** Updates the status of a bug in the sprint, based on the role of the logged-in user (either **Developer** or **QA Engineer**).
  - **Parameters:** Sprint sprint, WorkItem workItem, String newStatus.
  - **Usage:** Called when a bug status needs to be updated.
- public void addWorkItemToSprint(Sprint sprint, WorkItem workItem):
  - **Purpose:** Adds a work item (task, bug, story, or epic) to the sprint's backlog.
  - **Parameters:** Sprint sprint, WorkItem workItem.
  - **Usage:** Used to add a work item to a sprint's backlog.

Private Method:

- private void addTeamMember(Sprint sprint, User user):
  - **Purpose:** Adds a user to the sprint team if not already a member. This method is **private** and is called within the class by other methods, such as addWorkItemToSprint, to add the assigned user to the sprint team when a work item is added.
  - **Parameters:** Sprint sprint, User user.
  -
- public void displaySprintWorkItems(Sprint sprint):
  - **Purpose:** Displays all work items (tasks, bugs, stories, and epics) in the sprint.
  - **Parameters:** Sprint sprint.
  - **Usage:** Displays the work items in a sprint.

4.2. UserService Class

Purpose:

The UserService class handles the logic for user management, including registration, login, and work item assignment. It also provides methods for checking if a user is logged in and managing user roles.

Attributes:

- private static User loggedInUser: Holds the currently logged-in user.

Methods:

- public User registerUser(String name, String email, String password, String role):
  - **Purpose:** Registers a new user in the system based on the role (e.g., Developer, ScrumMaster, QA).
  - **Parameters:** name, email, password, role.
  - **Returns:** The created User object or null if the email is already registered.
  - **Usage:** Used during the user registration process.
- public boolean loginUser(String email, String password):
  - **Purpose:** Logs the user in by verifying the email and password.
  - **Parameters:** email, password.
  - **Returns:** true if login is successful, false otherwise.
  - **Usage:** Called to log a user in.
- public static User getLoggedInUser():
  - **Purpose:** Retrieves the currently logged-in user.
  - **Returns:** The logged-in User object.
  - **Usage:** Used to fetch the details of the user currently logged in.
- public static boolean isUserLoggedIn():
  - **Purpose:** Checks if any user is logged in.
  - **Returns:** true if a user is logged in, false otherwise.
  - **Usage:** Used to verify if the user is logged in before performing any user-specific operations.

- public boolean assignWorkItemToUser(WorkItem workItem):
  - **Purpose:** Assigns a work item to the currently logged-in user.
  - **Parameters:** WorkItem workItem.
  - **Returns:** true if the work item is successfully assigned, false otherwise.
  - **Usage:** Used to assign tasks or bugs to users.

4.3. TaskService Class

Purpose:

The TaskService class handles the business logic related to **task** creation, assignment, and updating. It manages work items (specifically tasks and bugs) within the system and allows stakeholders, scrum masters, and other users to perform task-related actions.

Attributes:

- **No specific attributes** are defined. Methods operate directly on the Task, Bug, Story, and Epic objects.

Methods:

- public Task createTask(String title, String description, User createdBy):
  - **Purpose:** Creates a new task with the specified title, description, and creator.
  - **Parameters:** title, description, createdBy.
  - **Returns:** The created Task object.
  - **Usage:** Used by users (including stakeholders) to create new tasks.
- public Bug createBug(String title, String description, User createdBy):
  - **Purpose:** Creates a new bug.
  - **Parameters:** title, description, createdBy.
  - **Returns:** The created Bug object.
  - **Usage:** Used to create bugs in the system.
- public Epic createEpic(String title, String description, User createdBy):
  - **Purpose:** Creates a new epic without assigning it to a user.
  - **Parameters:** title, description, createdBy.
  - **Returns:** The created Epic object.
  - **Usage:** Used by users (typically ScrumMasters) to create epics.
- public Story createStory(String title, String description, User createdBy, Epic parentEpic):
  - **Purpose:** Creates a new story associated with a given epic.
  - **Parameters:** title, description, createdBy, parentEpic.
  - **Returns:** The created Story object.
  - **Usage:** Used by users (usually ScrumMasters or stakeholders) to create stories and associate them with epics.
- public void addTaskToStory(Story story, Task task):
  - **Purpose:** Adds a task to a story. Only accessible by **ScrumMaster**.
  - **Parameters:** Story story, Task task.

  - **Usage:** Adds tasks to a story and updates the story's status accordingly.
- public void addStoryToEpic(Epic epic, Story story):
  - **Purpose:** Adds a story to an epic. Only accessible by **ScrumMaster**.
  - **Parameters:** Epic epic, Story story.
  - **Usage:** Adds stories to epics and updates epic status as needed.
- public void updateSprintStatus(Sprint sprint):
  - **Purpose:** Updates the sprint's status based on the completion of its tasks and bugs.
  - **Parameters:** Sprint sprint.
  - **Usage:** Ensures that the sprint's status is updated to COMPLETED or ACTIVE based on the status of work items.

Conclusion

The service classes (SprintService, UserService, TaskService) encapsulate the core business logic of the system. They provide methods for:

- Managing users (registration, login, work item assignment).
- Managing sprints (creation, team members, work items).
- Managing work items like tasks, bugs, epics, and stories (creation, updates, assignments).

These service classes handle operations that are crucial for the system's functionality and ensure that business rules are enforced during each action. The services also interact with the data layer (Database) and help manage the flow of information in the system.