

## Experiment \_\_: Create a Kanban Board for Hospital Management System using Jira

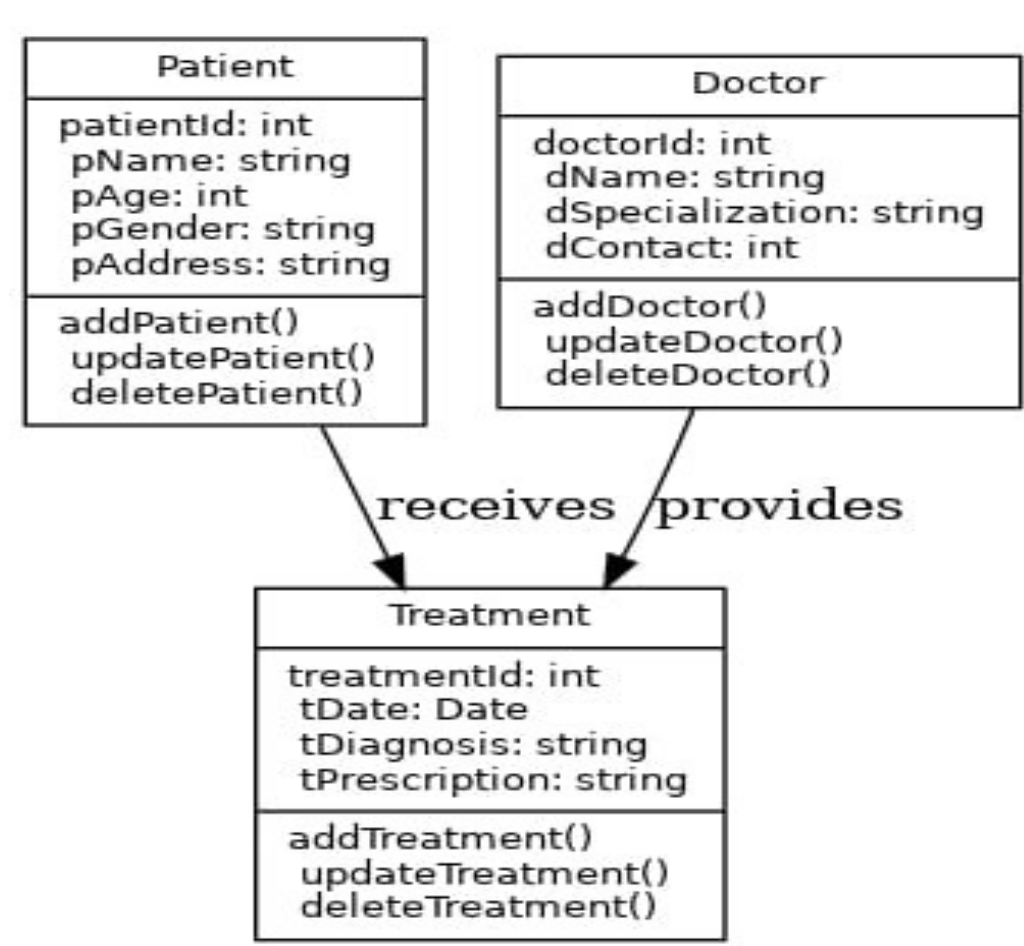
### Aim:

To create a Kanban board for a Hospital Management System using Jira, by defining projects, teams, epics, stories, tasks, and releases.

### Description:

Agile methodology emphasizes iterative and incremental development. **Kanban**, a popular Agile framework, helps visualize workflow, limit work-in-progress, and improve efficiency. Using **Jira**, hospital operations such as patient registration, doctor assignment, treatment, and billing can be managed effectively by organizing them into epics, stories, and tasks on a Kanban board.

### Class Diagram: Hospital Management System



## **Procedure:**

### **Step 1: Create a Project**

- Start Jira → Click **Projects** → **Create Project**.
- Select **Software Development** → **Kanban** → **Use Template**.
- Choose **Company Managed Project**.
- Enter project name **“Hospital Management System”** → **Click Create Project**.
- Skip recommended options. → Project created successfully.

### **Step 2: Create a Team**

#### **a) Invite Members**

- Enter email → Select name → Invite.

#### **b) Create a Team**

- Click **Teams** → **Create Team** → **Add Members** → **Click Create**.
- Team created successfully

### **Step 3: Create Epics**

- Click **Create** → **Work Type: Epic**.
- Enter Epic name (e.g., *Patient*), description, select team, start & due dates.
- Click **Create**.
- Similarly create other Epics: *Doctor*, *Treatment*.

### **Step 4: Create Stories**

- Click **Create** → **Work Type: Story**.
- Enter story name (e.g., *Add Patient*) under parent Epic (*Patient*)
- Add description, assign team, set dates → Click **Create**.
- Similarly create stories like *Update Patient*, *Delete Patient*, *Doctor Consultation*, *Treatment Details*.

### **Step 5: Create Tasks**

- Click **Create** → **Work Type: Task**.
- Enter task name (e.g., *Patient ID*) under parent Epic.
- Add description, assign team, set dates → Click **Create**.
- Similarly create tasks under *Doctor* and *Treatment*.

## Step 6: Create Version and Release

### Create Version:

- In the sidebar, select **More actions (•••)** and select **Releases**
- Click on the **"Create version"** button, typically found at the top right of the Releases page.
- **Enter Version Details:**
  - 1) **Name:** Provide an identifiable name for the version (e.g., *"Patient Module v1.0"*, *"Doctor Module v1.0"*, *"Treatment Module v1.0"*).
  - 2) **Start date (Optional):** Specify the planned start date for work related to this version.
  - 3) **Release date (Optional):** Define the target release date for the version.
  - 4) **Description (Optional):** Add a brief description outlining the scope or purpose of the version (e.g., *"Includes epics: Patient Registration, Update Patient, and Delete Patient"*).
- **Save:** Click **"Save"** to create the new version

### Releasing a Version:

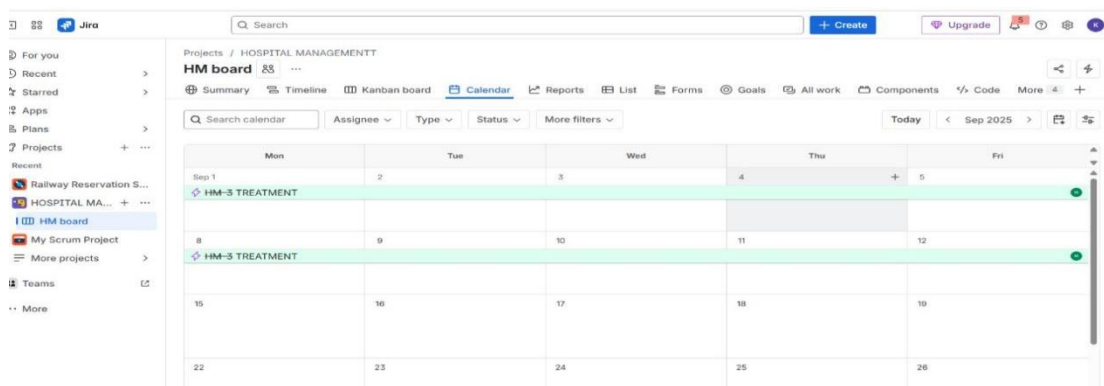
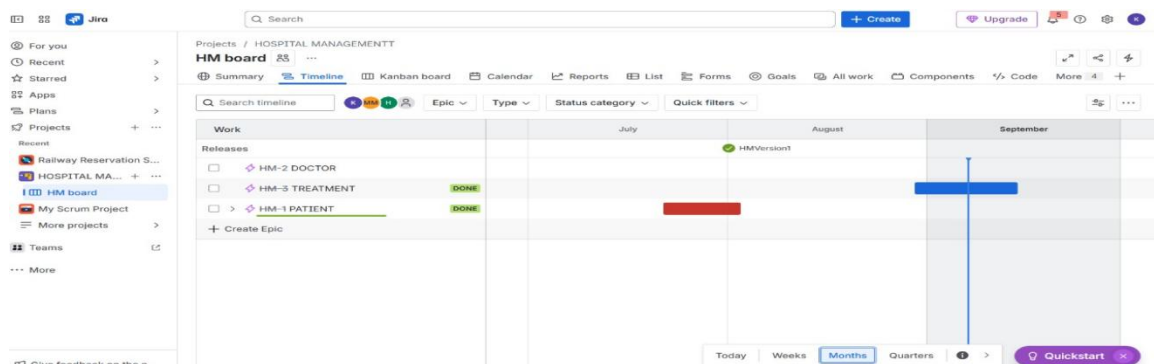
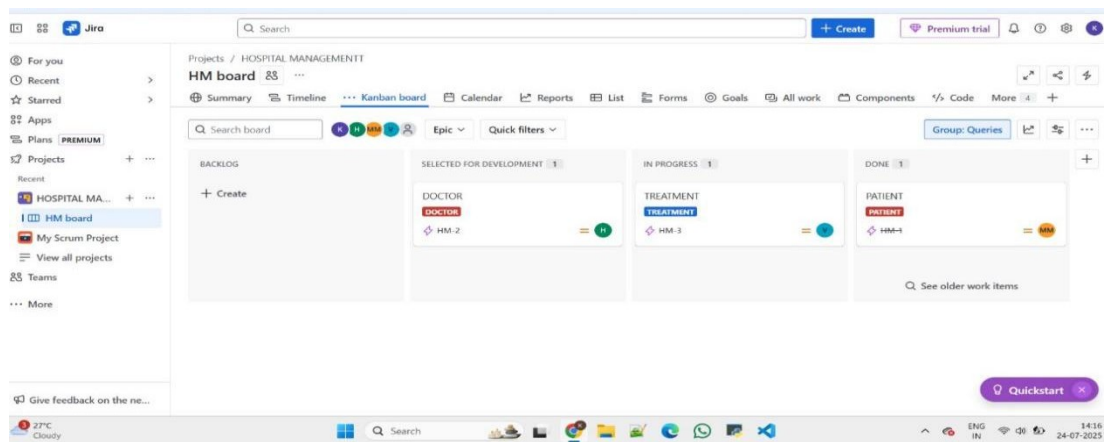
- **Initiate Release:** On the individual version's page, click the **"Release"** button. This button is usually prominent.
- **Confirm Release Details:** A dialog box will appear, allowing you to confirm or add details related to the release, such as the actual release date (e.g., *Releasing Doctor Consultation and Treatment Details*).
- **Execute Release:** Click the **"Release"** button within the dialog to finalize the release of the version.

### Work Flow Images:

The screenshot displays the Jira web application interface. The main content area shows a project board titled 'HM board' under the 'HOSPITAL MANAGEMENT' project. The board is in 'List' view and contains three issues:

Type	Key	Summary	Status	Comments	Assignee	Due date
Issue	HM-1	PATIENT	BACKLOG	Add comment	Maithili Mavinkurve	
Issue	HM-2	DOCTOR	SELECTED FOR D.	Add comment	Harika	
Issue	HM-3	TREATMENT	IN PROGRESS	Add comment	viha	

The interface includes a sidebar on the left with navigation options like 'For you', 'Recent', 'Starred', 'Apps', 'Plans', 'Projects', and 'Teams'. The top bar shows the Jira logo, search bar, and user profile. The bottom status bar shows system information like temperature, time, and date.



## Result:

By using Jira and Kanban methodology, the hospital workflow can be efficiently managed. Tasks are visualized, responsibilities are assigned, and progress is easily tracked, ensuring better coordination and timely delivery of healthcare services.

## Experiment : Create a Scrum Board for Railway Reservation System using Jira

### Aim:

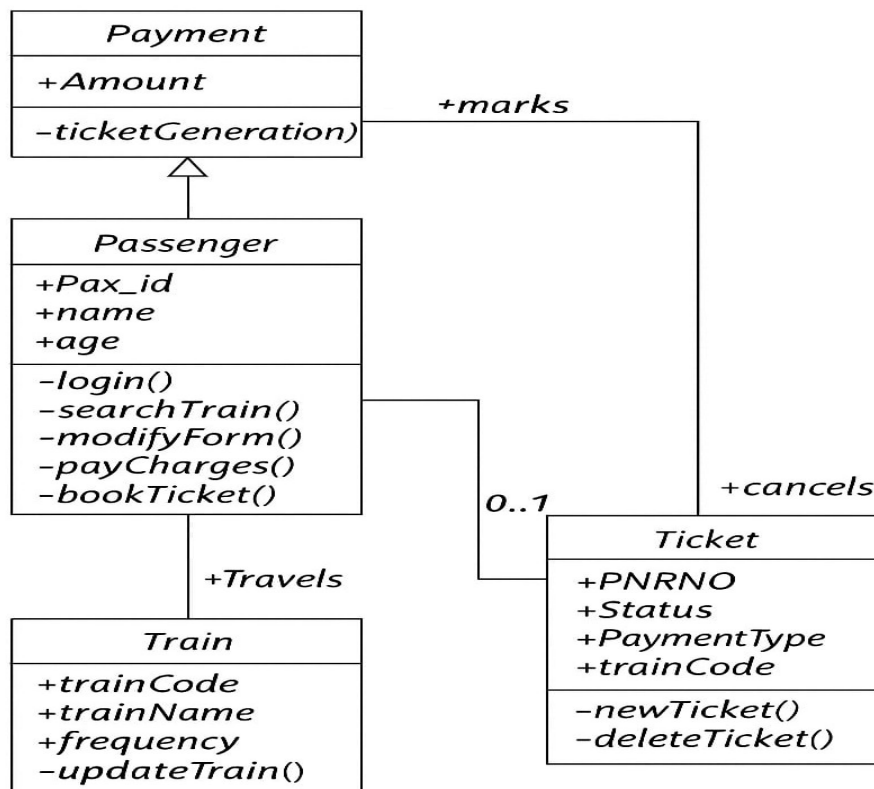
To create a **Scrum Board** for a Railway Reservation System using **Jira**, by organizing workflow into sprints and managing product backlog items such as Passenger, Train, Ticket, and Payment modules.

### Description:

Scrum is an Agile framework that organizes work into **time-boxed iterations (sprints)**. A **Scrum Board** helps visualize backlog items, sprint progress, and completed tasks.

For a **Railway Reservation System**, Scrum methodology is applied to handle operations like passenger management, train scheduling, ticket booking, and payment processing. Jira Scrum Board allows tracking of **Epics, Stories, and Tasks** for each sprint, ensuring systematic and timely development of the system.

### Class Diagram: Railway Reservation System



## **Procedure:**

### **Step 1: Create a Project**

- Start Jira → Click **Projects** → **Create Project**.
- Select **Software Development** → **Scrum** → **Use Template**.
- Choose **Company Managed Project**.
- Enter project name “**Railway Reservation System**” → **Create Project**.

### **Step 2: Create a Team**

- Invite members by entering email IDs.
- Create a team and assign members.
- Team created successfully.

### **Step 3: Create Epics (Product Backlog)**

- Create Epics for the main modules from the class diagram:
  - 1) **Passenger** (login, search train, modify form, book ticket)
  - 2) **Train** (update train, cancel train)
  - 3) **Ticket** (new ticket, delete ticket, status updates)
  - 4) **Payment** (pay charges, generate ticket, amount handling).

### **Step 4: Create Stories**

- Under each Epic, create User Stories:
  - 1) *Passenger Epic*: Add Passenger, Modify Passenger Form, Book Ticket.
  - 2) *Train Epic*: Update Train Info, Cancel Train.
  - 3) *Ticket Epic*: Generate New Ticket, Delete Ticket, Update Status.
  - 4) *Payment Epic*: Make Payment, Confirm Ticket Generation.

### **Step 5: Create Tasks**

- Break stories into smaller tasks, e.g.:
  - 1) For *Passenger*: Create Passenger ID, Validate Login, Implement Search
  - 2) For *Train*: Create Train Code, Update Frequency.
  - 3) For *Ticket*: Assign PNR Number, Track Status.
  - 4) For *Payment*: Calculate Amount, Validate Payment Type.

### **Step 6: Manage Scrum Board**

- Move backlog items into the sprint backlog.

- Start a new sprint (set sprint goal and duration).
- Move tasks across columns: *To Do* → *In Progress* → *Done*.
- Monitor burndown chart to track sprint progress.

## Step 7: Create Version and Release

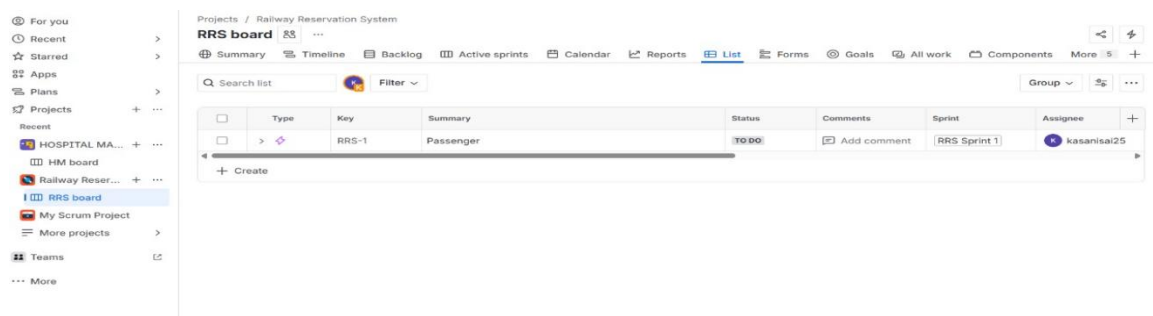
### Create Version:

- In the sidebar, select **More actions (•••)** and select **Releases**
- Click on the **"Create version"** button, typically found at the top right of the Releases page.
- **Enter Version Details:**
  1. **Name:** Provide an identifiable name for the version (e.g., *"Passenger Module v1.0"*, *"Train Module v1.0"*, *"Ticket Module v1.0"*, *"Payment Module v1.0"*).
  2. **Start date (Optional):** Specify the planned start date for work related to this version.
  3. **Release date (Optional):** Define the target release date for the version.
  4. **Description (Optional):** Add a brief description outlining the scope or purpose of the version (e.g., *"Includes epics: Add Passenger, Modify Passenger Form, and Book Ticket"* or *"Includes epics: Make Payment and Confirm Ticket Generation"*).
- **Save:** Click **"Save"** to create the new version.

### Releasing a Version:

- **Initiate Release:** On the individual version's page, click the **"Release"** button. This button is usually prominent.
- **Confirm Release Details:** A dialog box will appear, allowing you to confirm or add details related to the release, such as the actual release date (e.g., *Releasing Train Module with Update Train Info and Cancel Train stories*).
- **Execute Release:** Click the **"Release"** button within the dialog to finalize the release of the version.

### Work Flow Images:



Projects / Railway Reservation System

**RRS board**

Summary Timeline Backlog Active sprints Calendar Reports List Forms Goals All work Components More

Search backlog Version Epic Type Label Quick filters

RRS Sprint 1 29 Aug - 5 Sep (2 work items)

Passenger book the Ticket

RRS-2 Book Ticket PASSENGER DONE Aug 31

RRS-3 Search Train PASSENGER IN PROGRESS Aug 31

RRS Sprint 2 5 Sep - 12 Sep (0 work items)

Plan a sprint by dragging work items into it, or by dragging the sprint footer.

+ Create

0 work items | Estimate: 0

Projects / Railway Reservation System

**RRS board**

Summary Timeline Backlog Active sprints Calendar Reports List Forms Goals All work Components More

Search calendar Assignee Type Status More filters

Today < Sep 2025 >

Mon	Tue	Wed	Thu	Fri
Sep 1	2	3	4	5
RRS Sprint 1				
RRS-1 Passenger				RRS Sprint 2
8	9	10	11	12
RRS Sprint 2				
15	16	17	18	19
22	23	24	25	26

Projects / Railway Reservation System

**RRS board**

Summary Timeline Backlog Active sprints Calendar Reports List Forms Goals All work Components More

Search timeline Epic Type Label Status category Quick filters

Work August September October

Sprints RRS-1 Passenger

Releases

+ Create Epic

Today Weeks Months Quarters > Quickstart

## Result:

The Scrum Board for the Railway Reservation System was successfully created using Jira. By dividing work into Epics, Stories, and Tasks aligned with the modules *Passenger*, *Train*, *Ticket*, and *Payment*, the project can be developed efficiently with iterative sprint planning and tracking.



## Experiment : Create a Kanban Board for Library Management System using Jira

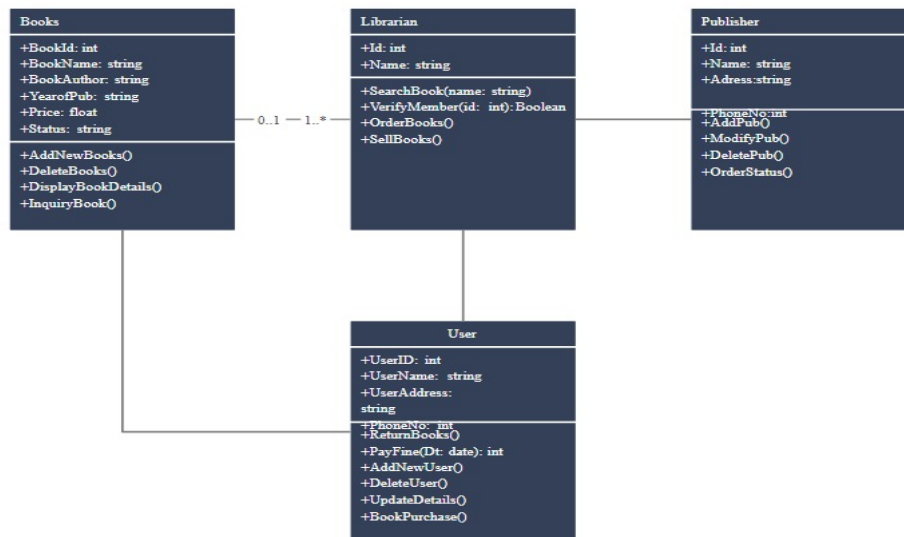
### Aim:

To create a **Kanban Board** for a Library Management System using **Jira**, by defining epics, stories, tasks, and releases to manage the library workflow effectively.

### Description:

The Library Management System is designed to manage books, librarians, publishers, and users efficiently. The system keeps track of available books, manages user memberships, handles fines, and facilitates book purchase and sales. By implementing the Kanban methodology in Jira, we can visualize the workflow, track tasks, and ensure smooth project delivery.

### Class Diagram: Library Management System



### Procedure:

#### Step 1: Create a Project

- Start Jira → Click **Projects** → **Create Project**.
- Select **Software Development** → **Kanban** → **Use Template**.
- Choose **Company Managed Project**.
- Enter project name "**Library Management System**" → **Create Project**.

## **Step 2: Create a Team**

- Invite members by entering their email IDs.
- Create a project team and assign roles (e.g., Developer, Tester, Admin).
- Team created successfully.

## **Step 3: Create Epics**

- **Books** (Add Books, Delete Books, Display Book Details, Inquiry Book).
- **Librarian** (Search Book, Verify Member, Order Books, Sell Books).
- **Publisher** (Add Publisher, Modify Publisher, Delete Publisher, Order Status).
- **User** (Add User, Delete User, Update Details, Book Purchase, Return Books, Pay Fine).

## **Step 4: Create Stories**

- Under each Epic, create user stories:
  - 1) **Books Epic:** Add new book, Delete book record, Display book details, Search/Inquiry book.
  - 2) **Librarian Epic:** Search for a book, Verify member ID, Place book order, Sell books.
  - 3) **Publisher Epic:** Add publisher details, Modify publisher record, Delete publisher, Track order status.
  - 4) **User Epic:** Add new user, Delete user, Update user details, Purchase book, Return book, Pay fine

## **Step 5: Create Tasks**

- Break stories into smaller technical tasks:
  - 1) **Books:** Implement BookID, Validate Book Status, Update Price field.
  - 2) **Librarian:** Create Search Function, Implement Verify Member Logic, Automate Order Placement.
  - 3) **Publisher:** Build Publisher ID system, Enable Modify/Delete Operations, Integrate Order Tracking.
  - 4) **User:** Generate UserID, Validate Phone Number, Build Fine Calculation, Implement Book Purchase Flow.

## **Step 6: Manage Kanban Board**

- Use Kanban board columns: **To Do** → **In Progress** → **Done**.
- Add backlog items and assign them to members.
- Move tasks across the board as progress is made.
- Track workflow efficiency to ensure smooth delivery.

## Step 7: Create Version and Release

### Create Version:

- In the sidebar, select **More actions (•••)** and select **Releases**
- Click on the **"Create version"** button, typically found at the top right of the Releases page.
- **Enter Version Details:**
  1. **Name:** Provide an identifiable name for the version (e.g., *"Books Module v1.0"*, *"Librarian Module v1.0"*, *"Publisher Module v1.0"*, *"User Module v1.0"*).
  2. **Start date (Optional):** Specify the planned start date for work related to this version.
  3. **Release date (Optional):** Define the target release date for the version.
  4. **Description (Optional):** Add a brief description outlining the scope or purpose of the version (e.g., *"Includes epics: Add Books, Delete Books, Display Book Details, and Inquiry Book"* or *"Includes epics: Add User, Delete User, Purchase Book, and Pay Fine"*)
- **Save:** Click **"Save"** to create the new version.

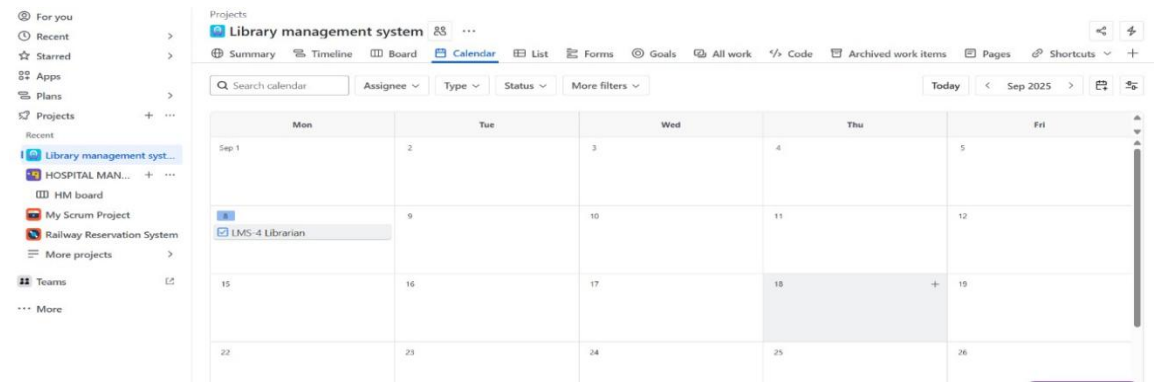
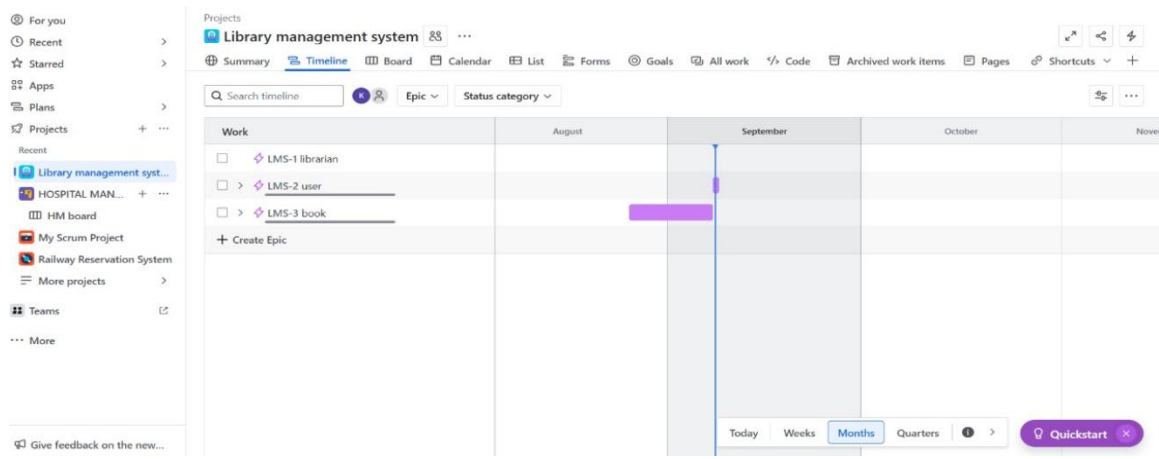
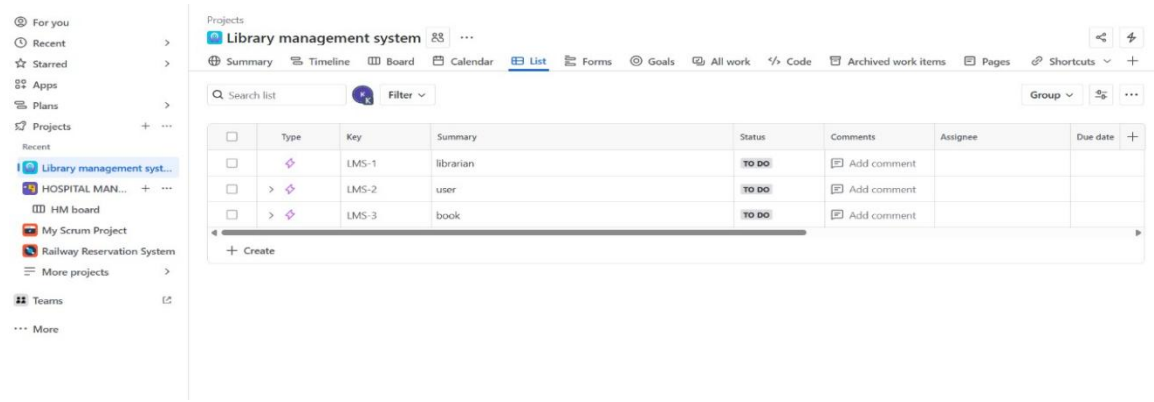
### Releasing a Version:

- **Initiate Release:** On the individual version's page, click the **"Release"** button. This button is usually prominent.
- **Confirm Release Details:** A dialog box will appear, allowing you to confirm or add details related to the release, such as the actual release date (e.g., *Releasing Publisher Module with Add/Modify/Delete Publisher and Track Order Status stories*).
- **Execute Release:** Click the **"Release"** button within the dialog to finalize the release of the version

### Work Flow Images:

The screenshot displays the Jira web interface for a project named 'Library management system'. The left sidebar shows navigation options like 'For you', 'Recent', 'Starred', 'Apps', 'Plans', 'Projects', 'Teams', and 'More'. The main content area shows the project's 'All work' tab with a table of work items. The table has columns for 'Work', 'Assignee', 'Reporter', 'Priority', 'Status', and 'Unresolved'. Three work items are listed: 'LMS-3 book', 'LMS-2 user', and 'LMS-1 librarian', all assigned to 'Unassigned' and reported by 'Mithili Mavinkurve', 'kasanisai25', and 'Harika' respectively. The status for all items is 'TO DO' and they are all 'Unresolved'. A 'Create' button is at the bottom of the table.

Work	Assignee	Reporter	Priority	Status	Unresolved
LMS-3 book	Unassigned	Mithili Mavinkurve	Medium	TO DO	Unresolved
LMS-2 user	Unassigned	kasanisai25	Medium	TO DO	Unresolved
LMS-1 librarian	Unassigned	Harika	Medium	TO DO	Unresolved



## Result:

The Kanban Board for the Library Management System was successfully created using Jira, including projects, teams, epics, stories, tasks, and releases, allowing efficient management of library operations.

## **Experiment : Create a Scrum Board for Online Job Application System using Jira**

### **Aim:**

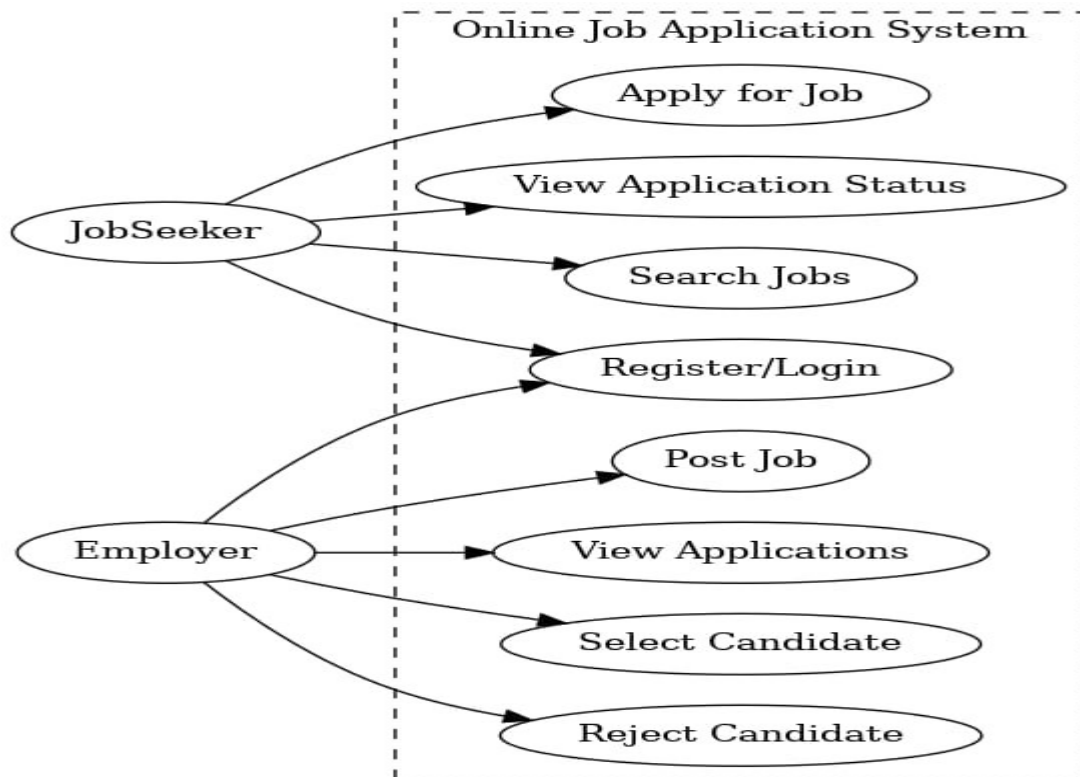
To create a Scrum Board for an Online Job Application System using Jira, by organizing workflow into sprints and managing product backlog items such as JobSeeker and Employer modules.

### **Description:**

Scrum is an Agile framework that organizes work into time-boxed iterations (sprints). A Scrum Board helps visualize backlog items, sprint progress, and completed tasks.

For the Online Job Application System, Scrum methodology is applied to handle operations like job posting, job applications, candidate selection, and tracking application status. Jira Scrum Board allows tracking of **Epics, Stories, and Tasks** for each sprint, ensuring systematic and timely development of the system.

### **Use Case Diagram: Online Job Application System**



## **Procedure:**

### **Step 1: Create a Project**

- Start Jira → Click **Projects** → **Create Project**.
- Select **Software Development** → **Scrum** → **Use Template**.
- Choose **Company Managed Project**.
- Enter project name **“Online Job Application System”** → Create Project.

### **Step 2: Create a Team**

- Invite members by entering email IDs.
- Create a team and assign members (Developer, Tester, Admin).
- Team created successfully.

### **Step 3: Create Epics (Product Backlog)**

- Create Epics for the main modules from the system:
  1. **JobSeeker** (Register/Login, Search Jobs, Apply for Job, View Application Status).
  2. **Employer** (Post Job, View Applications, Select Candidate, Reject Candidate).

### **Step 4: Create Stories**

- Under each Epic, create User Stories:

#### **JobSeeker Epic**

1. Register/Login
2. Search Jobs
3. Apply for Job
4. View Application Status

#### **➤ Employer Epic**

1. Post Job
2. View Applications
3. Select Candidate
4. Reject Candidate

### **Step 5: Create Tasks**

- Break stories into smaller technical tasks:

## JobSeeker Epic

1. Create JobSeeker ID
2. Validate Login
3. Implement Job Search Filter
4. Develop Apply for Job Form
5. Track and Display Application Status

## Employer Epic

6. Generate Employer ID
7. Build Post Job Form
8. Integrate Applications List View
9. Implement Candidate Selection Logic
10. Implement Candidate Rejection Function

## Step 6: Manage Scrum Board (Sprint Backlog)

- Move backlog items into the sprint backlog.
- Start a new sprint (set sprint goal and duration).
- Move tasks across columns: **To Do** → **In Progress** → **Done**.
- Monitor **burndown chart** to track sprint progress.

## Step 7: Create Version and Release

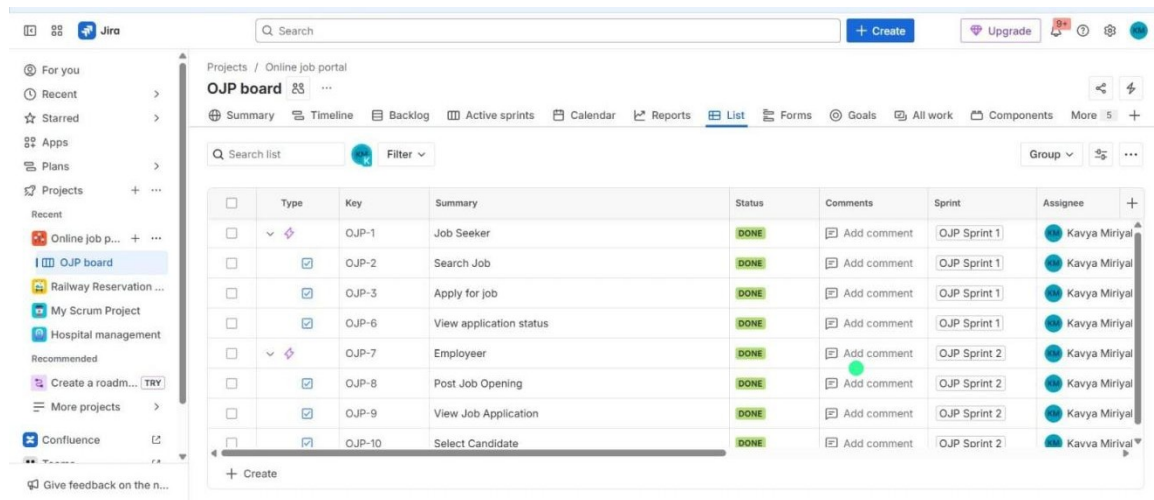
### Create Version:

- In the sidebar, select **More actions (•••)** → **Releases**.
- Click on the **"Create version"** button.
- **Enter Version Details:**
  1. **Name:** Provide an identifiable name (e.g., *"JobSeeker Module v1.0"*, *"Employer Module v1.0"*).
  2. **Start date (Optional):** Planned start date for the version.
  3. **Release date (Optional):** Define the target release date.
  4. **Description (Optional):** Briefly outline the scope (e.g., *"Includes epics: Register/Login, Search Jobs, Apply for Job, and View Application Status"*).
- **Save:** Click **"Save"** to create the version.

### Releasing a Version:

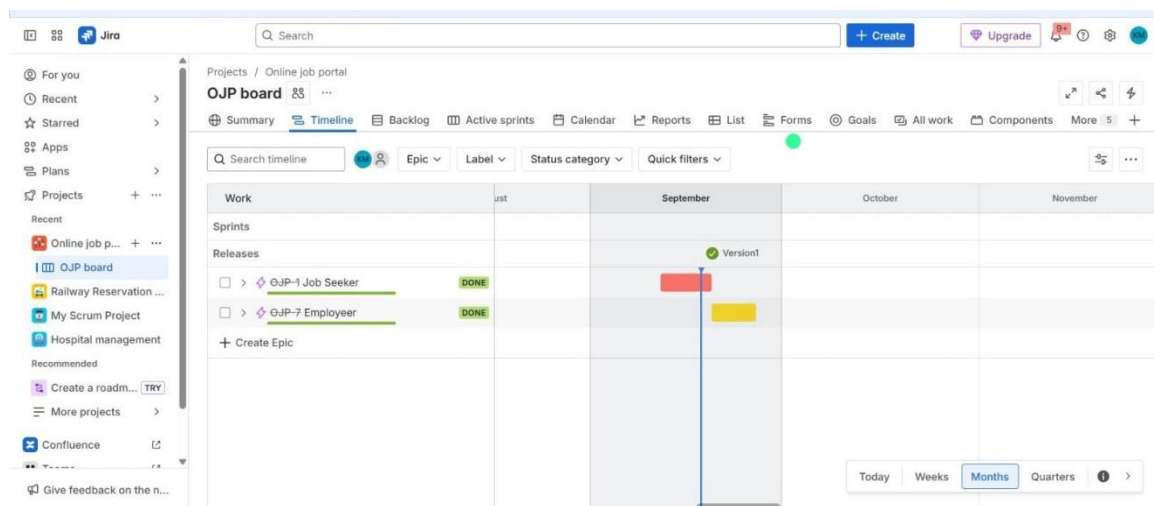
- **Initiate Release:** On the version page, click **"Release"**.
- **Confirm Release Details:** Add actual release date (e.g., *Releasing Employer Module with Post Job, View Applications, Select and Reject Candidate stories*).
- **Execute Release:** Click **"Release"** to finalize.

# Work Flow Images:

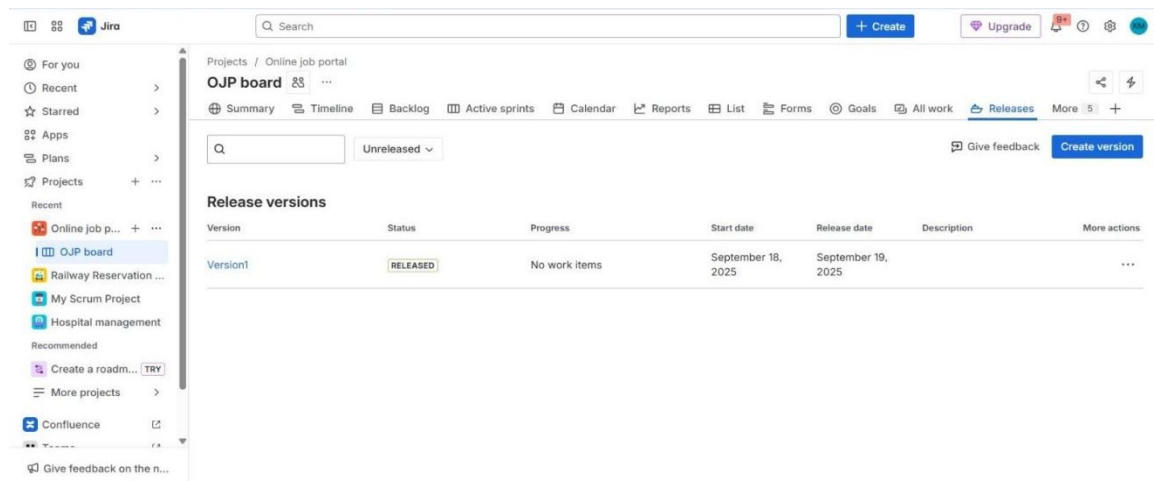


This screenshot shows the Jira interface for the 'OJP board' in the 'List' view. The left sidebar contains navigation options like 'For you', 'Recent', 'Starred', 'Apps', 'Plans', 'Projects', and 'Recent'. The main content area displays a table of issues with columns for checkboxes, Type, Key, Summary, Status, Comments, Sprint, and Assignee. All issues are marked as 'DONE' and assigned to 'Kavya Miriyal'.

	Type	Key	Summary	Status	Comments	Sprint	Assignee
<input type="checkbox"/>	▼	OJP-1	Job Seeker	DONE	Add comment	OJP Sprint 1	Kavya Miriyal
<input type="checkbox"/>	✓	OJP-2	Search Job	DONE	Add comment	OJP Sprint 1	Kavya Miriyal
<input type="checkbox"/>	✓	OJP-3	Apply for job	DONE	Add comment	OJP Sprint 1	Kavya Miriyal
<input type="checkbox"/>	✓	OJP-6	View application status	DONE	Add comment	OJP Sprint 1	Kavya Miriyal
<input type="checkbox"/>	▼	OJP-7	Employee	DONE	Add comment	OJP Sprint 2	Kavya Miriyal
<input type="checkbox"/>	✓	OJP-8	Post Job Opening	DONE	Add comment	OJP Sprint 2	Kavya Miriyal
<input type="checkbox"/>	✓	OJP-9	View Job Application	DONE	Add comment	OJP Sprint 2	Kavya Miriyal
<input type="checkbox"/>	✓	OJP-10	Select Candidate	DONE	Add comment	OJP Sprint 2	Kavya Miriyal



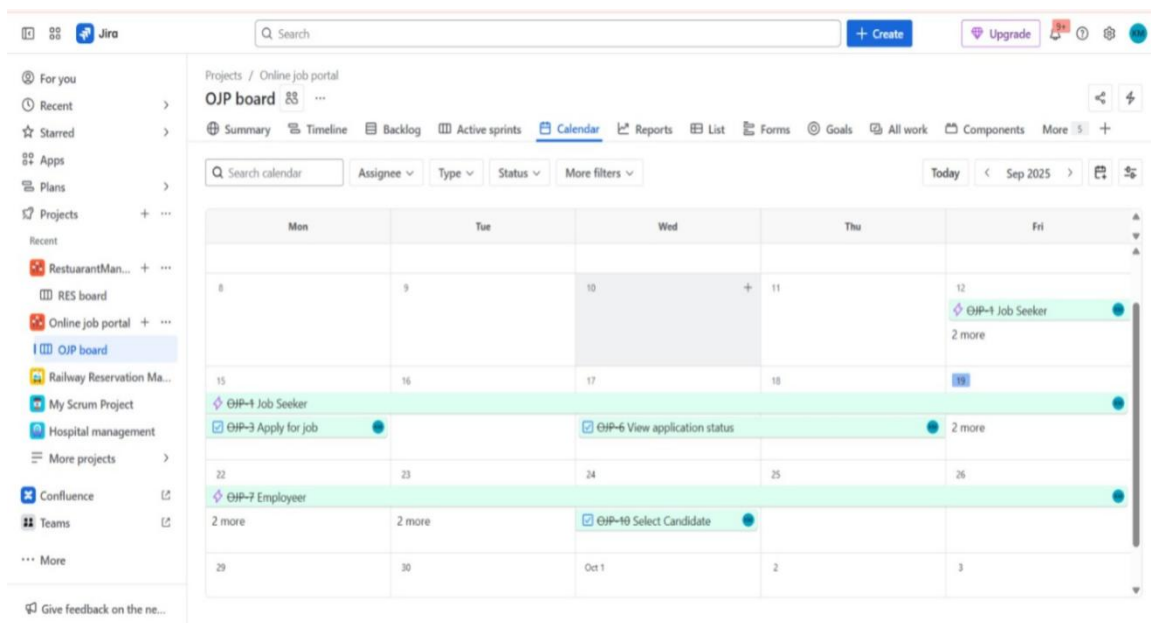
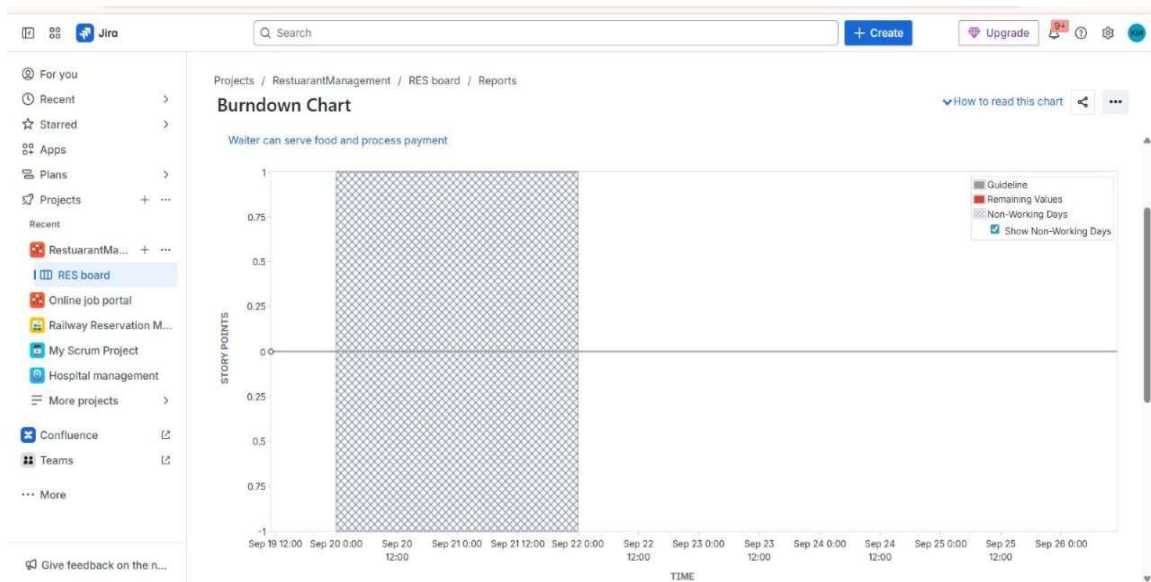
This screenshot shows the Jira interface for the 'OJP board' in the 'Timeline' view. The left sidebar is the same as the previous image. The main content area displays a timeline view with columns for 'September', 'October', and 'November'. It shows 'Sprints' and 'Releases' with a 'Version1' label. A red bar indicates a release date in September.



This screenshot shows the Jira interface for the 'OJP board' in the 'Releases' view. The left sidebar is the same as the previous images. The main content area displays a table of release versions. The 'Version1' release is marked as 'RELEASED' and has a start date of September 18, 2025.

Version	Status	Progress	Start date	Release date	Description	More actions
Version1	RELEASED	No work items	September 18, 2025	September 19, 2025		...





## Result:

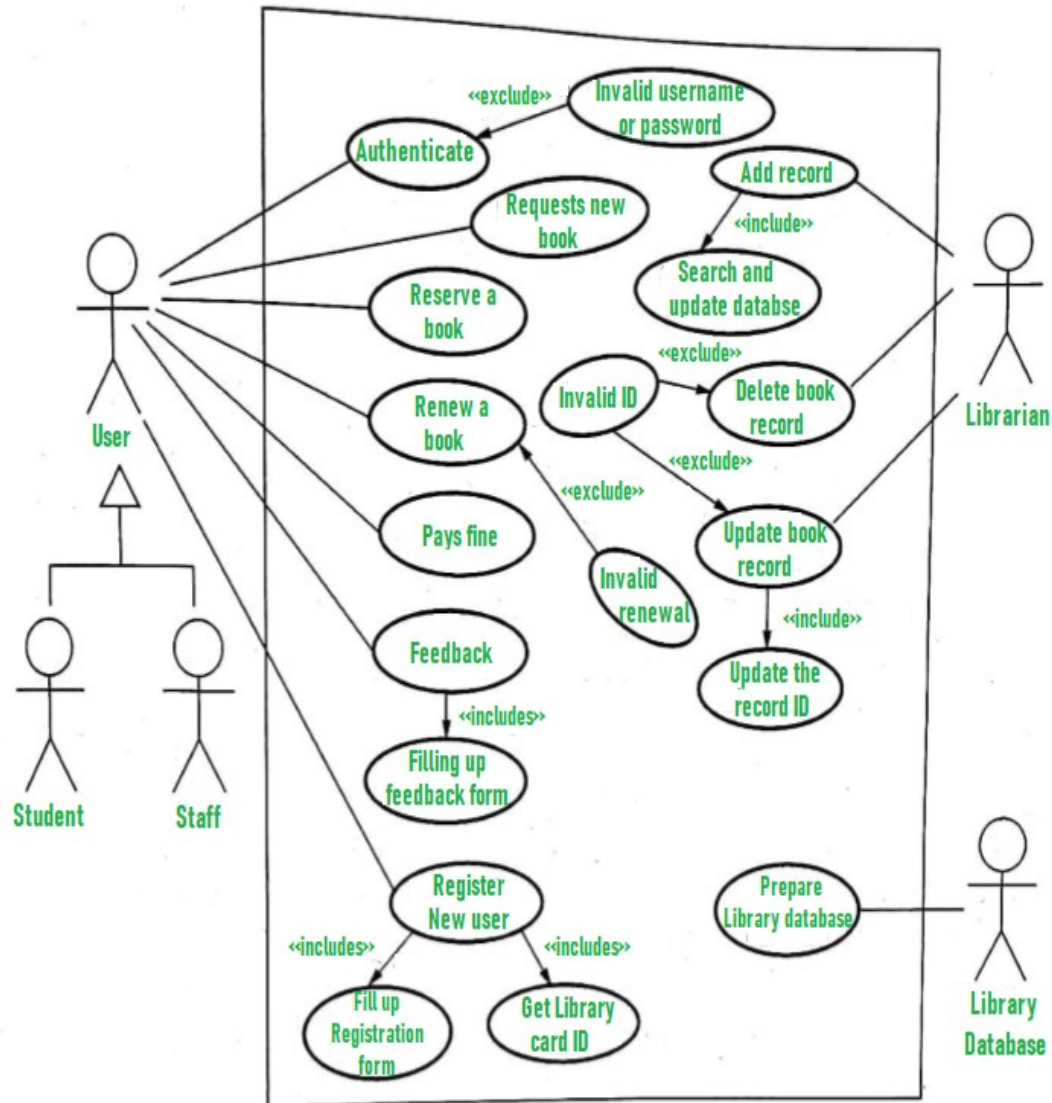
The Scrum Board for the Online Job Application System was successfully created using Jira. By dividing work into Epics, Stories, and Tasks aligned with the modules JobSeeker and Employer, the project can be developed efficiently with iterative sprint planning and tracking.

**Experiment** : Create an Agile Development Plan for the Library Management System.

**Aim**: To Create an Agile Development Plan for the Library Management System.

**Procedure**:

**Use case diagram**



Assuming 10 sprints with each sprint lasting 10 working days:

### **Sprint 1 (Days 1-10):**

- Conduct project kickoff meeting
- Develop user stories and prioritize backlog
- Create wireframes for the main screens
- Set up development environment
- Begin development of user authentication and authorization system

### **Sprint 2 (Days 11-20):**

- Complete development of user authentication and authorization system
- Begin development of book search functionality
- Begin development of book borrowing functionality
- Review wireframes with stakeholders and make necessary changes

### **Sprint 3 (Days 21-30):**

- Complete development of book search functionality
- Complete development of book borrowing functionality
- Begin development of book return functionality
- Begin development of book reservation functionality

### **Sprint 4 (Days 31-40):**

- Complete development of book return functionality
- Complete development of book reservation functionality
- Begin development of user profile functionality
- Begin development of book recommendation functionality

### **Sprint 5 (Days 41-50):**

- Complete development of user profile functionality
- Complete development of book recommendation functionality
- Begin development of book review and rating functionality
- Begin development of administrative dashboard for librarians

### **Sprint 6 (Days 51-60):**

- Complete development of book review and rating functionality
- Complete development of administrative dashboard for librarians
- Begin development of book purchase and inventory management functionality
- Begin development of fine management functionality

### **Sprint 7 (Days 61-70):**

- Complete development of book purchase and inventory management functionality
- Complete development of fine management functionality
- Begin development of reporting and analytics functionality
- Begin development of mobile application

**Sprint 8 (Days 71-80):**

- Complete development of reporting and analytics functionality
- Complete development of mobile application
- Begin development of integration with external systems (e.g. payment gateway)

**Sprint 9 (Days 81-90):**

- Complete development of integration with external systems
- Begin testing and bug fixing
- Begin user acceptance testing
- Begin documentation and training material development

**Sprint 10 (Days 91-100):**

- Complete testing and bug fixing
- Complete user acceptance testing
- Complete documentation and training material development
- Conduct system deployment
- Conduct final review and retrospective

**Result:**

This is just an example of an Agile development plan for the Library system, and the actual plan may vary depending on the specific needs of the project and the team's progress during each sprint

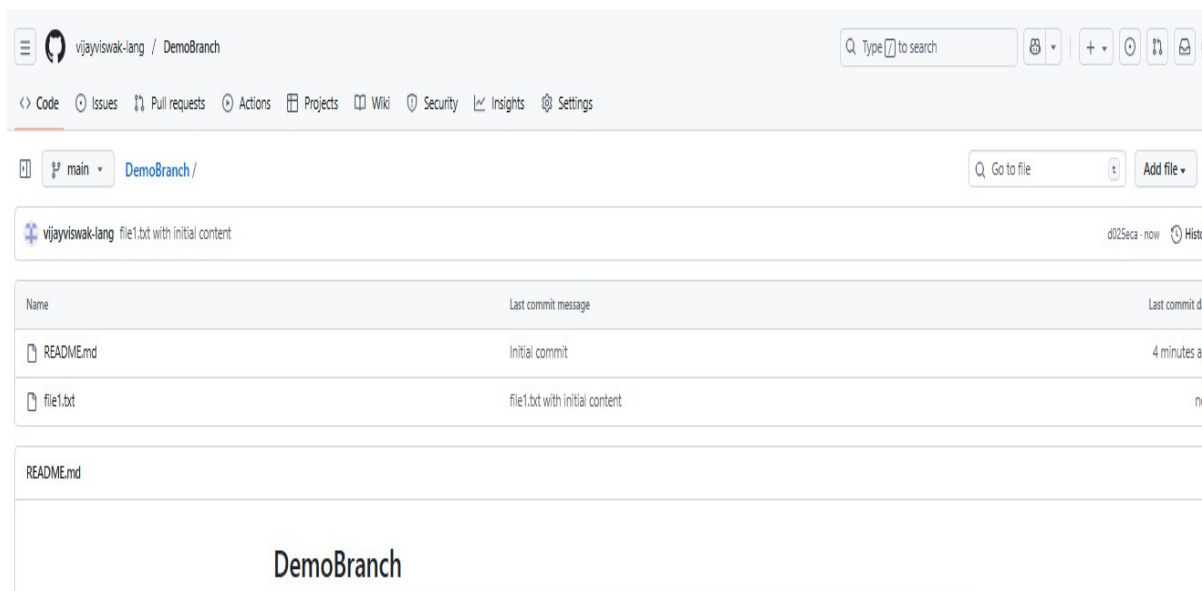
**Experiment :** To create and manage branches in a GitHub repository, perform changes independently in different branches, and then merge them into the main branch using a pull request, followed by deleting the merged branch.

**Aim:**

To understand how to create, manage, and merge branches in a GitHub repository using pull requests.

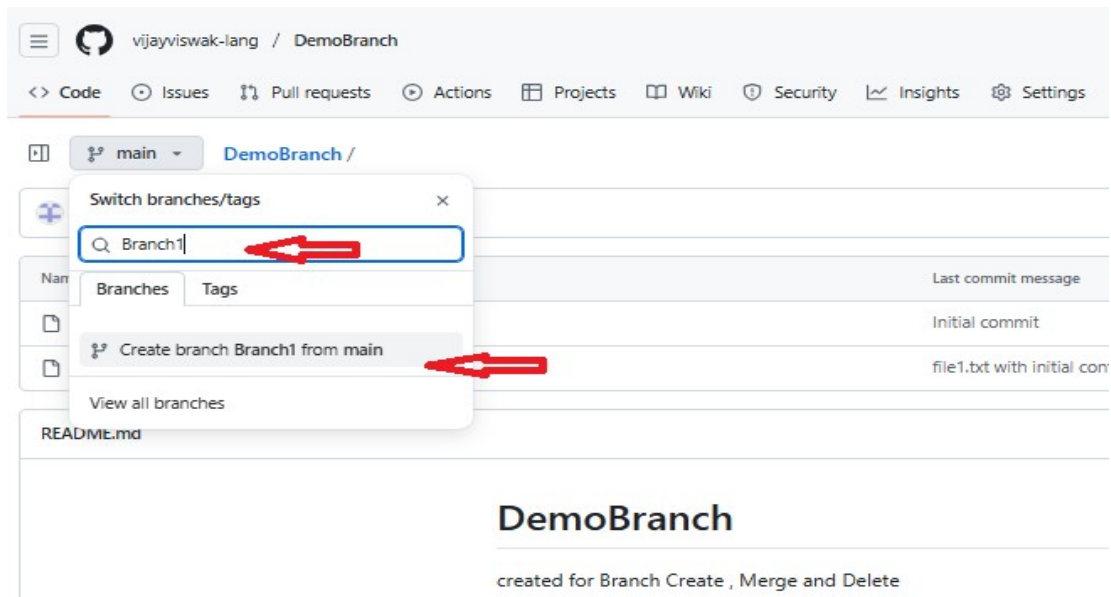
**Procedure:**

Step 1: Creating a Repository DemoBranch and create a text file in main Branch

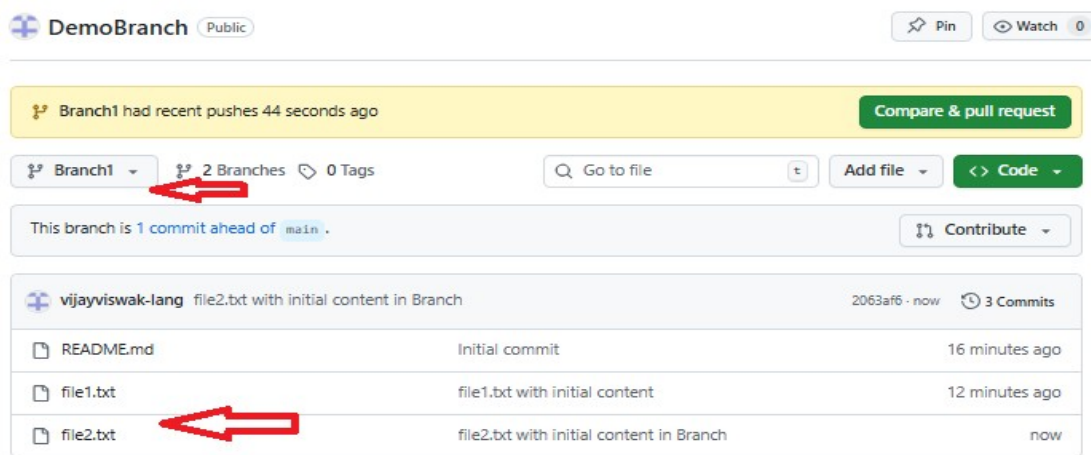


Step 2: Create a new Branch Branch1 and create a text file in Branch1 Branch

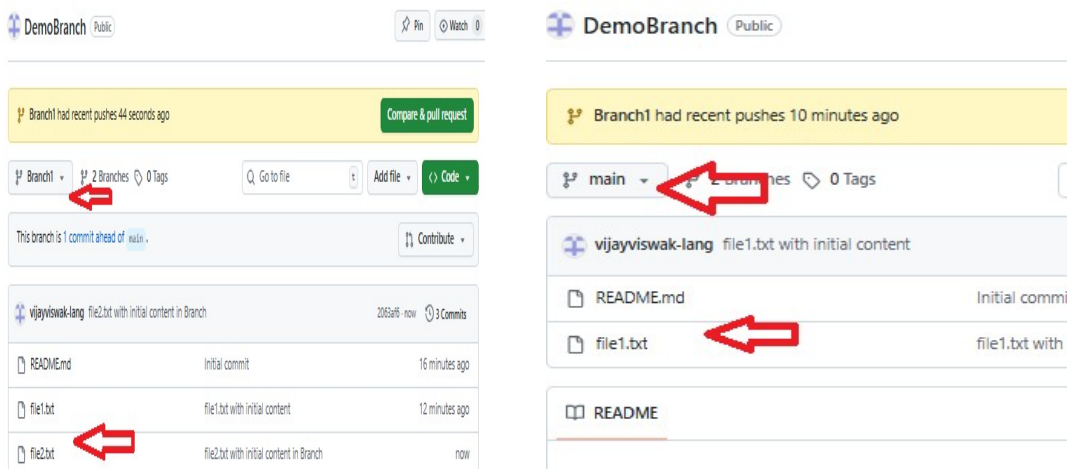
- i. Create a Branch  
In the arrow in the main box and enter a branch name and click create branch



Create a file in Branch1 and commit



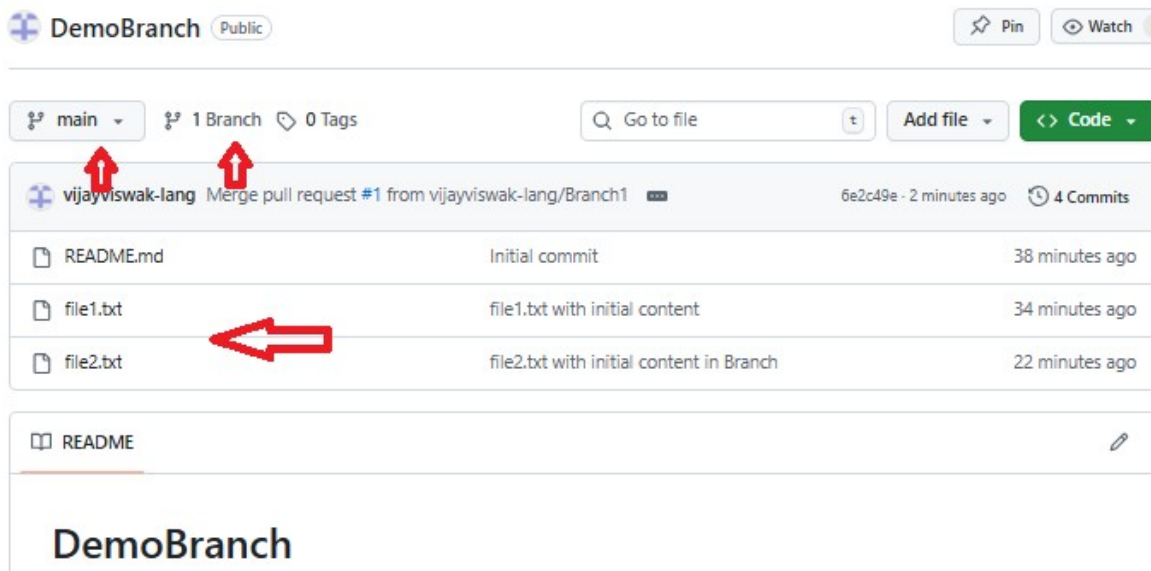
Now Main has file1.txt and Branch1 has file1.txt and file2.txt



### Step3: Merge and delete the Branch using Pull Request

- i. Click Pull request
- ii. Click new pull request
- iii. Click on Branch1
- iv. Click on create pull request
- v. Again create pull request
- vi. Click on Merge pull request
- vii. Click on confirm merge
- viii. Click on delete branch
- ix. Click code to goto Home page

Now see that there is only one branch main and it has 2 files merged from the Branch1



### Result:

Successfully created a new branch, added files independently, merged the branch with the main branch using a pull request, and deleted the merged branch. The main branch now contains both file1.txt and file2.txt after the merge.

## **Experiment** : Creating a Clone for the Repository in GitHub

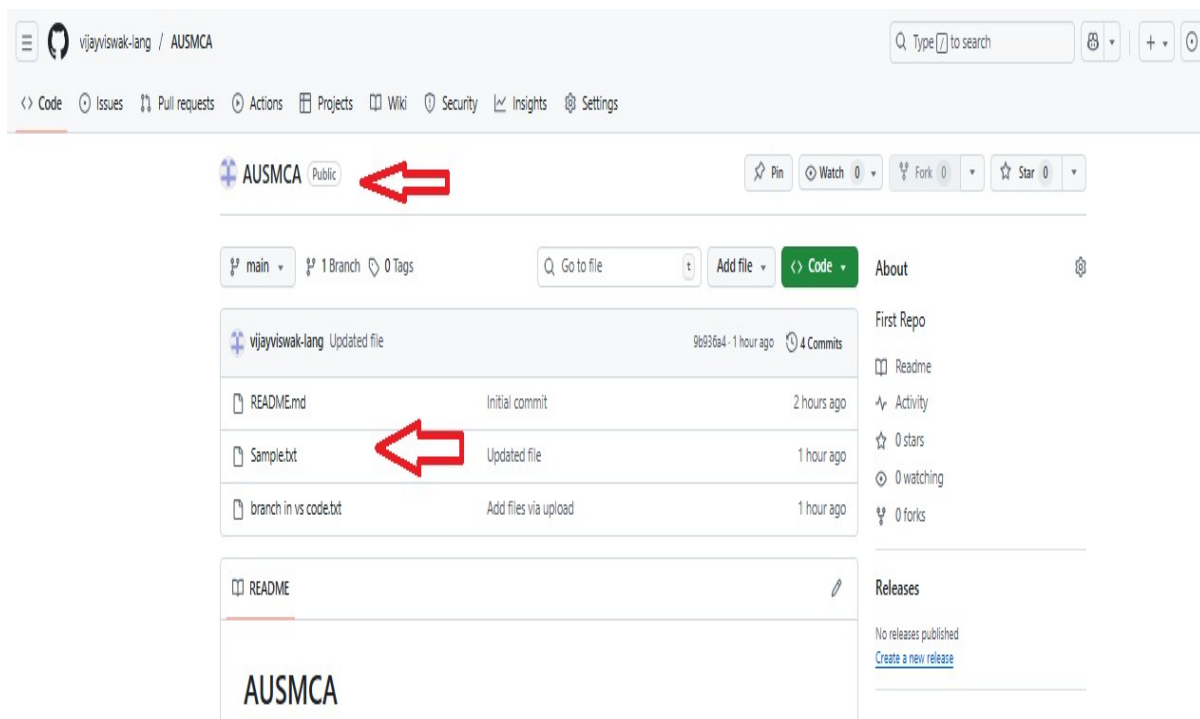
### **Aim:**

To Create a clone for the repository in Github

### **Procedure:**

#### **Step 1:**

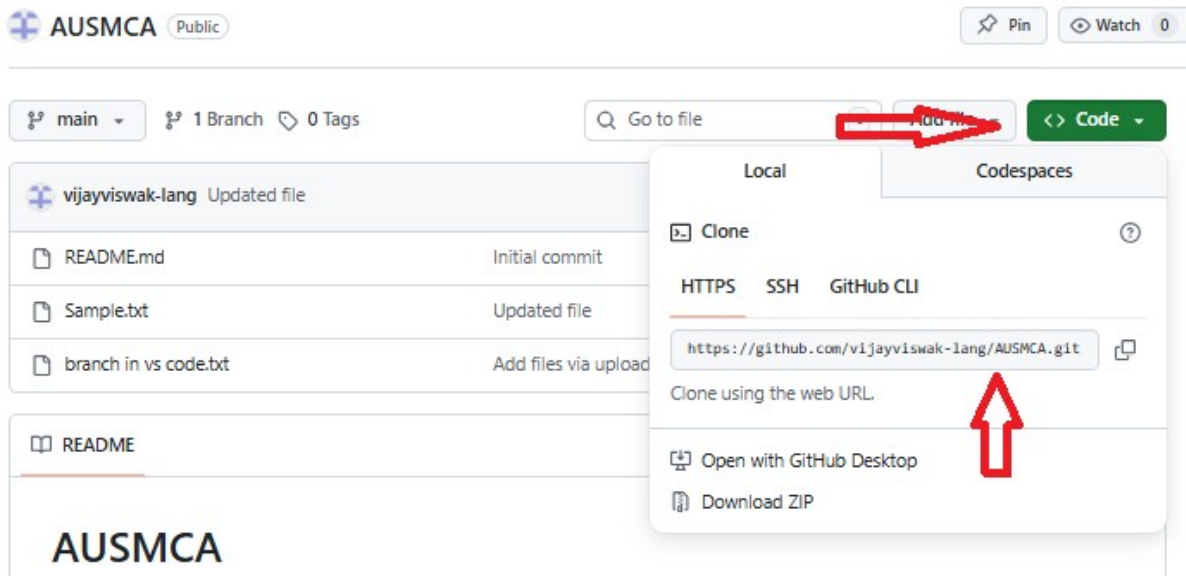
Create a repository and store some files in the repository



#### **Step2:**

- i. On GitHub, navigate to the main page of the repository.
- ii. Above the list of files, click **Code**.
- iii. Copy the URL for the repository under "HTTPS".





1. Open Git Bash.
2. Change the current working directory to the location where you want the cloned directory.
3. Type git clone, and then paste the URL you copied earlier.

4. `git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY`

5. Press **Enter** to create your local clone.

```
MINGW64:/c:/Users/Teacher/ausmca
Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$ git init
Reinitialized existing Git repository in C:/Users/Teacher/.git/

Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$ git clone https://github.com/vijayviswak-lang/AUSMCA.git
Cloning into 'AUSMCA'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 12 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (12/12), 4.63 KiB | 158.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.

Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
```

To check the repository is in the specified directory, Use the commands as follows

`cd ausmca`

`dir`

```
MINGW64:/c/Users/Teacher/ausmca
Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$
Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$
Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$
d
Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$
Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$
Teacher@VIJAYAKUMAR MINGW64 ~ (feature1)
$ cd ausmca
Teacher@VIJAYAKUMAR MINGW64 ~/ausmca (main)
$ dir
README.md Sample.txt branch\ in\ vs\ code.txt
Teacher@VIJAYAKUMAR MINGW64 ~/ausmca (main)
$
```

## **Result:**

The repository was successfully cloned from GitHub to the local system using the git clone command.

All files and folders from the remote repository are now available locally for modification and development.

## **Experiment :To create a repository in GitHub to store files and create versions.**

### **Aim:**

To Create a repository in the Git Hub to store the files and create versions

### **Procedure:**

#### **1. Steps to Create the Repository**

**Step 1:** login to the GitHub account.

**Step 2:** Click on the **new repository** option.

**Step 3:** Enter the **Name the project**, on the **ADD README** button and click **Create Repository** button.

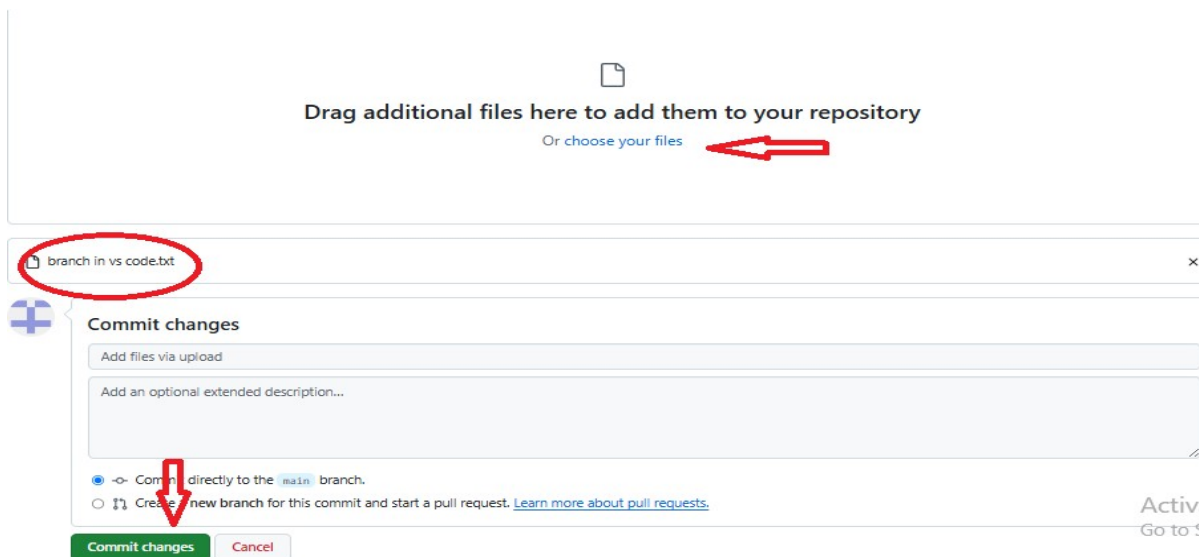
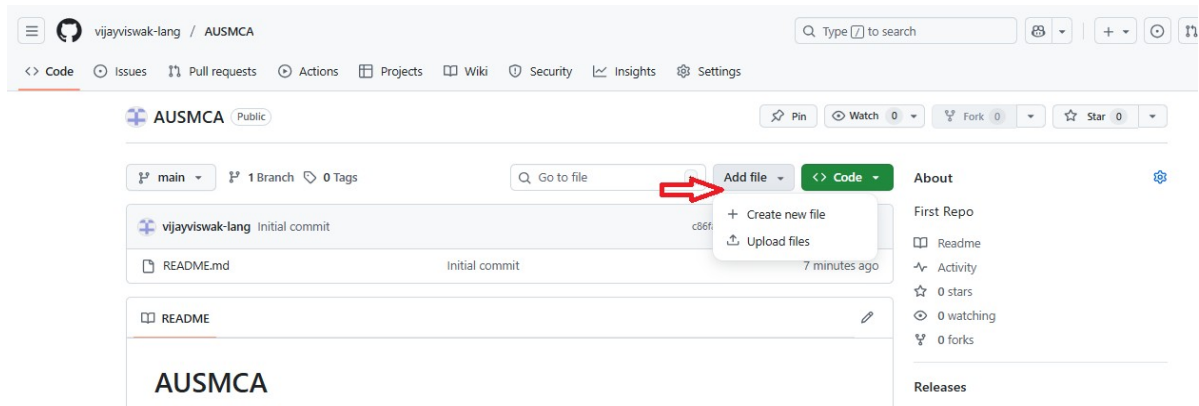
The screenshot shows the GitHub 'Create a new repository' page. At the top, there's a header with 'New repository' and a search bar. The main form is titled 'Create a new repository' and includes a note: 'Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository.](#) Required fields are marked with an asterisk (\*).' The form is divided into two sections: '1 General' and '2 Configuration'. In the 'General' section, the 'Owner' is 'vijayviswak-lang' and the 'Repository name' is 'ALUSMCA', with a red arrow pointing to the name field. Below this is a 'Description' field with the text 'First Repo' and a red arrow pointing to it. The 'Configuration' section includes 'Choose visibility' set to 'Public', 'Add README' with a toggle switch turned 'On' and a red arrow pointing to it, 'Add .gitignore' set to 'No .gitignore', and 'Add license' set to 'No license'. At the bottom right of the form is a green 'Create repository' button with a red arrow pointing to it. A watermark 'Activate Windows Go to Settings to activate Windows' is visible in the bottom right corner.

Repository will be created successfully.

#### **2. Storing the files in repository.**

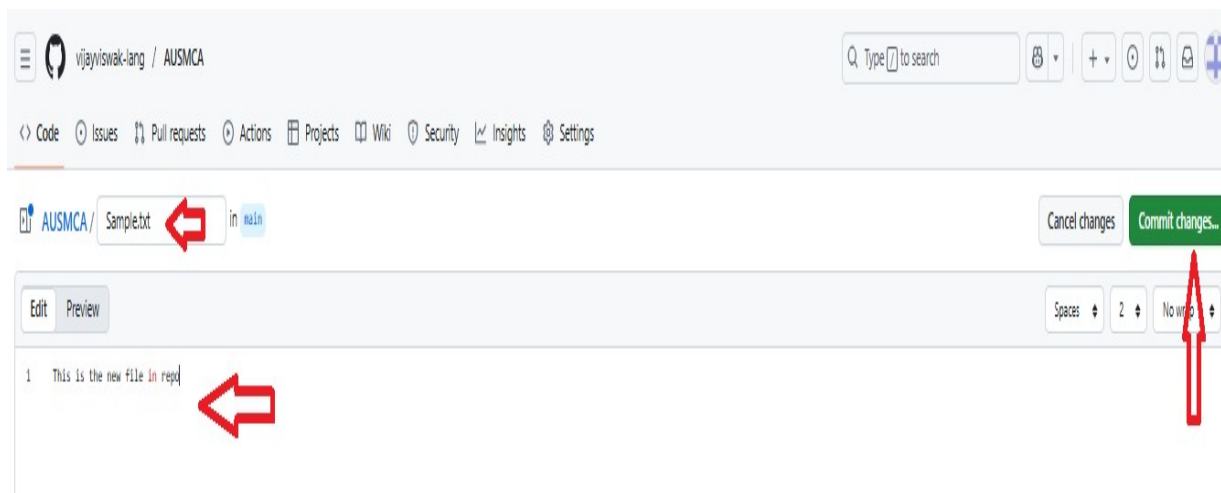
**Step 2.1:** Uploading the file

Click Add file and select upload file and click Choose your files. Select the files and click open to upload the files. After file upload click commit changes. File will be uploaded successfully.



## Step 2.2: Create a new file

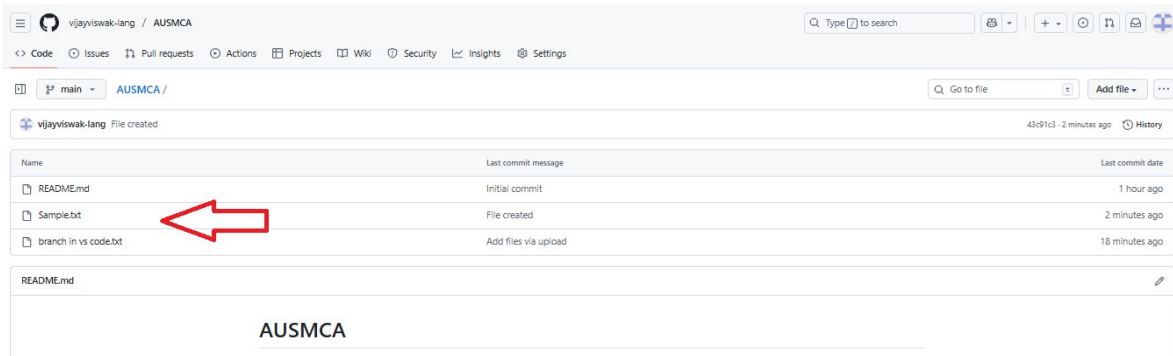
- Click Add file and select Create new file.
- Enter the file name
- Enter the content of the file
- Click Commit Changes



Enter the commit message and click Commit changes

The 'Commit changes' dialog box is shown with a close button (X) in the top right corner. It contains a 'Commit message' field with the text 'File created'. Below it is an 'Extended description' field with the placeholder text 'Add an optional extended description...'. At the bottom, there are two radio buttons: the first is selected and labeled 'Commit directly to the main branch', and the second is labeled 'Create a new branch for this commit and start a pull request' with a link to 'Learn more about pull requests'. At the very bottom are 'Cancel' and 'Saving...' buttons.

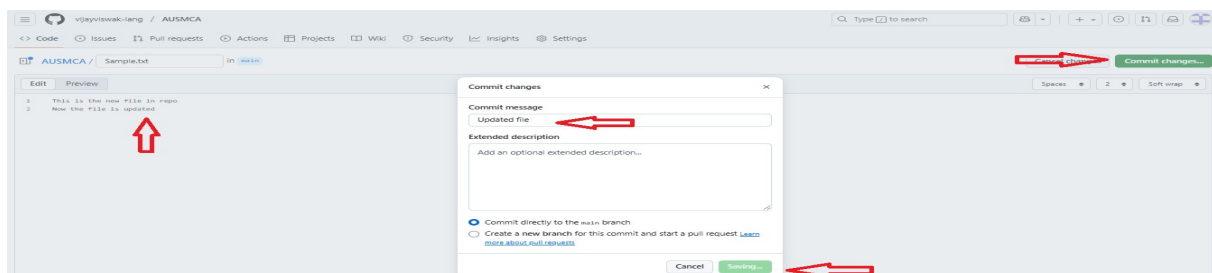
Following screen shows after storing all the files.



### 3. Creating Versions

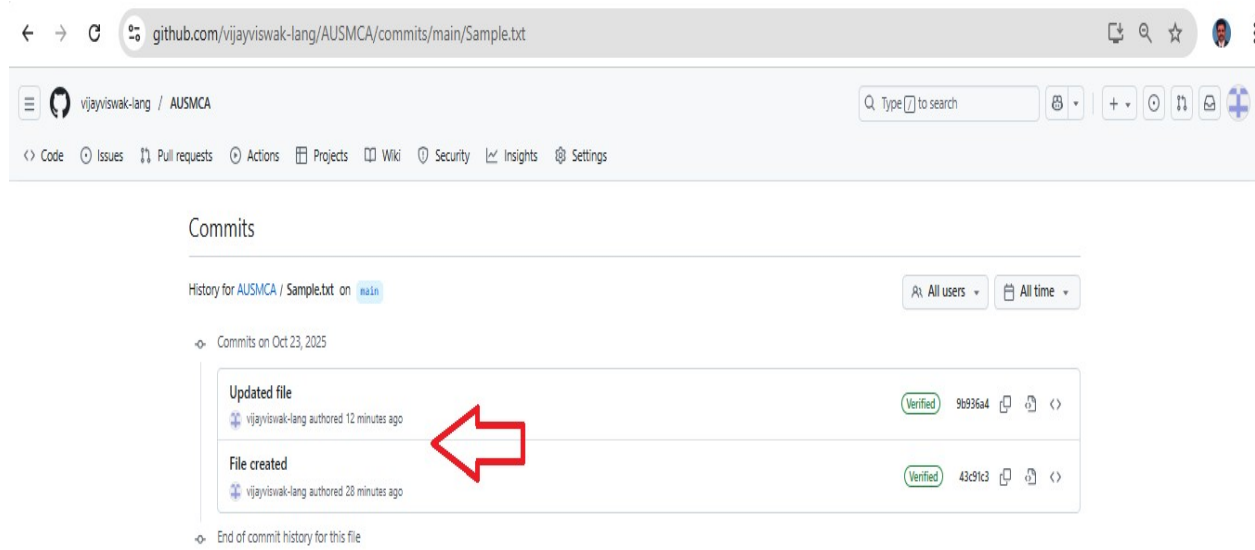
Step1: Edit the file

- i. Double click on the Sample.txt
- ii. Click the file edit button
- iii. Edit the file content
- iv. Click Commit Changes
- v. Enter Commit Message



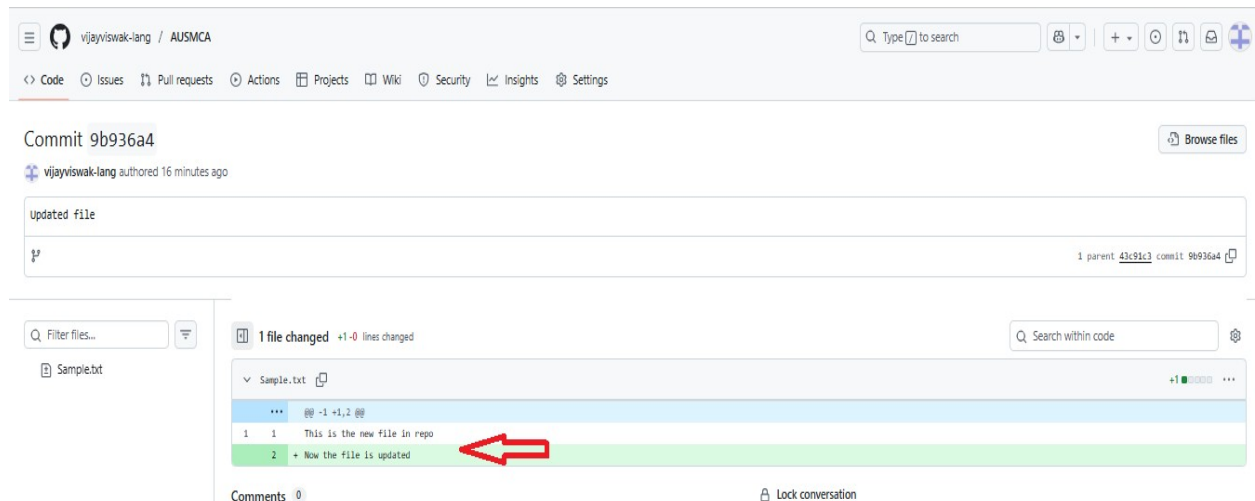
## Step 2: View History

### Click the History button



Above screen shot shows the two versions of Sample.txt ie Created and Updated

### Click on Updated file



Above screen shot shows the updated version of sample.txt

## Result:

Successfully created a **GitHub repository**, uploaded files, created new files, and maintained different versions by editing and committing changes.

The **history** feature in GitHub shows multiple versions of a file, confirming that version control has been implemented successfully.

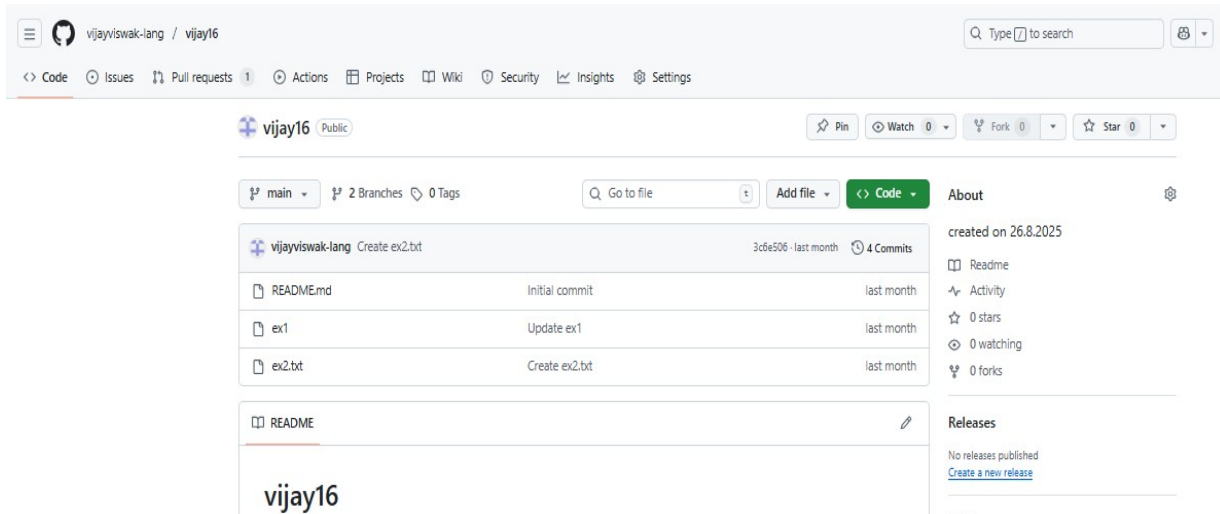
## Experiment :To create a Git Staging Environment using Git Bash.

### Aim:

To create a local Git repository, add files to the staging area, and learn how to stage and unstage files using Git Bash.

### Procedure:

#### Step 1: Create a repository in GitHub



#### Step 2: Creating Git Clone in Git Bash

- i. Open Git bash
- ii. Create a folder

```
Teacher@VIJAYAKUMAR MINGW64 ~ (main)
```

```
$ mkdir jan
```

- iii. Enter into the folder

```
Teacher@VIJAYAKUMAR MINGW64 ~ (main)
```

```
$ cd jan
```

- iv. Clone the repository in github

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git clone https://github.com/vijayviswak-lang/vijay16.git
```

```
Cloning into 'vijay16'...
```

```
remote: Enumerating objects: 15, done.
```

```
remote: Counting objects: 100% (15/15), done.
```

```
remote: Compressing objects: 100% (9/9), done.
```

```
remote: Total 15 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
```

```
Receiving objects: 100% (15/15), 4.37 KiB | 178.00 KiB/s, done.
```

```
Resolving deltas: 100% (2/2), done.
```

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ ls
```

```
vijay16/
```

- v. Initialize Git

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git init
```

Initialized empty Git repository in C:/Users/Teacher/jan/.git/

vi. List all the hidden files

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ ls -a
```

```
./ ../ .git/ vijay16/
```

### Step3: Creating Stages

i. Create a New File

Your new Git repository is empty.

Open notepad and enter text as follows "Vijaya Kumar"

Save this as **ten.txt** in your project folder.

ii. List Files in the Directory

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ ls
```

```
ten.txt vijay16/
```

iii. Check File Status with **git status**

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git status
```

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

```
ten.txt  
vijay16/
```

iv. Stage a File **ten.txt** with **git add**

To add a file to the staging area, use **git add <file>**:

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git add ten.txt
```

Now **ten.txt** is staged. You can check what is staged with **git status**:

v. Check Staged Files with **git status**

See which files are staged and ready to commit:

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git status
```

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

```
new file: ten.txt
```

Untracked files:

(use "git add <file>..." to include in what will be committed)

```
vijay16/
```



## vi. Stage a File ten.txt with git add

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git add vijay16
```

```
warning: adding embedded git repository: vijay16
```

```
hint: You've added another git repository inside your current repository.
```

```
hint: Clones of the outer repository will not contain the contents of
```

```
hint: the embedded repository and will not know how to obtain it.
```

```
hint: If you meant to add a submodule, use:
```

```
hint: git submodule add <url> vijay16
```

```
hint: If you added this path by mistake, you can remove it from the
```

```
hint: index with:
```

```
hint: git rm --cached vijay16
```

```
hint: See "git help submodule" for more information.
```

```
hint: Disable this message with "git config set advice.addEmbeddedRepo false"
```

## vii. Check File Status with git status

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git status
```

```
On branch main
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file: ten.txt
```

```
new file: vijay16
```

## Step 4: Unstage a File

- i. Un stage using reset

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

```
$ git reset ten.txt
```

```
Teacher@VIJAYAKUMAR MINGW64 ~/jan (main)
```

- ii. Check the status

```
$ git status
```

```
On branch main
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file: vijay16
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
ten.txt
```

## Result:

Successfully created a **Git staging environment** using Git Bash.

Files were added to the staging area using git add and removed (unstaged) using git reset.

This demonstrates the process of managing which files are prepared for commit in Git.

## **Experiment : Create, update, merge and delete a Branch in Git using Commands**

### **Aim:**

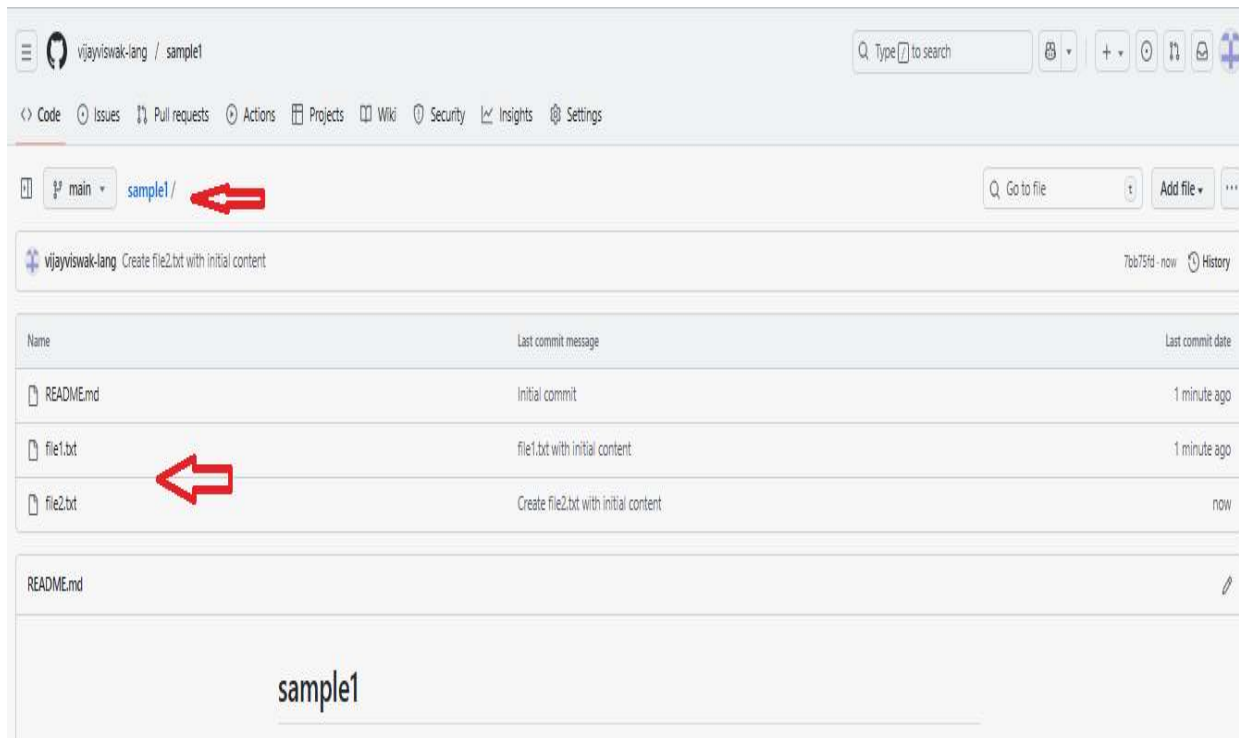
To Create, update, merge and delete a Branch in Git using Commands in Command prompt.

### **Procedure:**

#### **Branching & Merging commands**

Command	Description
git branch	List branches (the asterisk denotes the current branch)
git branch -a	List all branches (local and remote)
git branch [branch name]	Create a new branch
git branch -d [branch name]	Delete a branch
git push origin --delete [branch name]	Delete a remote branch
git checkout -b [branch name]	Create a new branch and switch to it
git checkout -b [branch name] origin/[branch name]	Clone a remote branch and switch to it
git branch -m [old branch name] [new branch name]	Rename a local branch
git checkout [branch name]	Switch to a branch
git checkout -	Switch to the branch last checked out
git checkout -- [file-name.txt]	Discard changes to a file
git merge [branch name]	Merge a branch into the active branch
git merge [source branch] [target branch]	Merge a branch into a target branch

#### **Step1:Create a repository and create two text files in GitHub**



## **Step2: create Clone for the repository in command prompt**

### **i. Open the command prompt**

Microsoft Windows [Version 10.0.18363.1556]  
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\Users\Teacher>

### **ii. Creating a clone**

```
C:\Users\Teacher>git clone https://github.com/vijayviswak-lang/sample1.git
Cloning into 'sample1'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (7/7), done.
C:\Users\Teacher>
```

### **iii. Entering into local repository**

```
C:\Users\Teacher>cd sample1
```

### **iv. Status of the repository**

```
C:\Users\Teacher\sample1>git status
On branch main
nothing to commit, working tree clean
```

Log info of repository

```
C:\Users\Teacher\sample1>git log
commit 3af8b21b8765477fe0d01d3f8da8672d5c7d04b8 (HEAD -> main)
Author: drvijayakumar <vijayviswak@gmail.com>
Date: Fri Oct 10 15:54:41 2025 +0530
    Initial Commit
```

v. To check the current branch name

```
C:\Users\Teacher\sample1>git checkout main
Already on 'main'
```

### **Step 3: New branch Creation**

i. Create a new branch using branch command

```
C:\Users\Teacher\sample1>git branch branch1
```

ii. Enter into a new branch

```
C:\Users\Teacher\sample1>git checkout branch1
Switched to branch 'branch1'
```

iii. Check the current branch

```
C:\Users\Teacher\sample1>git branch
* branch1
Main
```

iv. Append a text in a file

```
C:\Users\Teacher\sample1>echo Append in branch >> file1.txt
C:\Users\Teacher\sample1>type file1.txt
Created file1 in main
Append in branch
branch1
```

v. Update the Branch

```
C:\Users\Teacher\sample1>git commit -m "updated in branch"
// to commit the modification of a file
[branch1 ee039c6] updated in branch
1 file changed, 1 insertion(+)
```

vi. Log info of updating

```
C:\Users\Teacher\sample1>git log
commit ee039c6c0a0898ab472d09a7ff651c2af1724a30 (HEAD -> branch1)
Author: drvijayakumar <vijayviswak@gmail.com>
Date: Fri Oct 10 20:02:42 2025 +0530
    updated in branch
commit 3af8b21b8765477fe0d01d3f8da8672d5c7d04b8 (main)
Author: drvijayakumar <vijayviswak@gmail.com>
Date: Fri Oct 10 15:54:41 2025 +0530
```

## Initial Commit

- vii. To display the content of file1

```
C:\Users\Teacher\sample1>type file1.txt
```

Created file1 in main

Append in branch

branch1

- viii. Switching to main

```
C:\Users\Teacher\sample1>git checkout main
```

Switched to branch 'main'

```
C:\Users\Teacher\sample1>type file1.txt
```

Created file1 in main

- ix. Creating and entering into branch2

```
C:\Users\Teacher\sample1>git checkout -b branch2
```

Switched to a new branch 'branch2'

```
C:\Users\Teacher\sample1>git branch
```

branch1

\* branch2

main

- x. Appending file2 in branch2

```
C:\Users\Teacher\sample1>echo updated file in branch2 >> file2.txt
```

```
C:\Users\Teacher\sample1>git add .
```

- xi. Updating branch2

```
C:\Users\Teacher\sample1>git commit -m "update in branch2"
```

[branch2 211a691] update in branch2

1 file changed, 1 insertion(+)

```
C:\Users\Teacher\sample1>git log
```

commit 211a6912d898e520247310128d025ed148fab356 (HEAD -> branch2)

Author: drvijayakumar <vijayviswak@gmail.com>

Date: Fri Oct 10 20:10:34 2025 +0530

update in branch2

commit 3af8b21b8765477fe0d01d3f8da8672d5c7d04b8 (main)

Author: drvijayakumar <vijayviswak@gmail.com>

Date: Fri Oct 10 15:54:41 2025 +0530

## Initial Commit

```
C:\Users\Teacher\sample1>type file2.txt
```

created file2 in main

updated file in branch2

```
C:\Users\Teacher\sample1>git checkout main
```

Switched to branch 'main'

```
C:\Users\Teacher\sample1>type file1.txt
```

Created file1 in main

```
C:\Users\Teacher\sample1>type file2.txt
```

created file2 in main

#### **Step 4: Merging and Deleting a file**

i. Merging a branch with main

```
C:\Users\Teacher\sample1>git merge branch2
```

Updating 3af8b21..211a691

Fast-forward

file2.txt | 1 +

1 file changed, 1 insertion(+)

```
C:\Users\Teacher\sample1>type file2.txt
```

created file2 in main

updated file in branch2

ii. Deleting a branch

```
C:\Users\Teacher\sample1>git branch -d branch2 // Deleting a branch
```

Deleted branch branch2 (was 211a691).

```
C:\Users\Teacher\sample1>git branch // Branches after delete
```

branch1

\* main

```
C:\Users\Teacher\sample1>
```

#### **Result:**

Branching and merging in Git were executed successfully.

- branch1 and branch2 were created and updated independently.
- branch2 was successfully merged into the main branch.
- The merged branch (branch2) was deleted, leaving the repository clean with the main branch containing the updated files.