

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

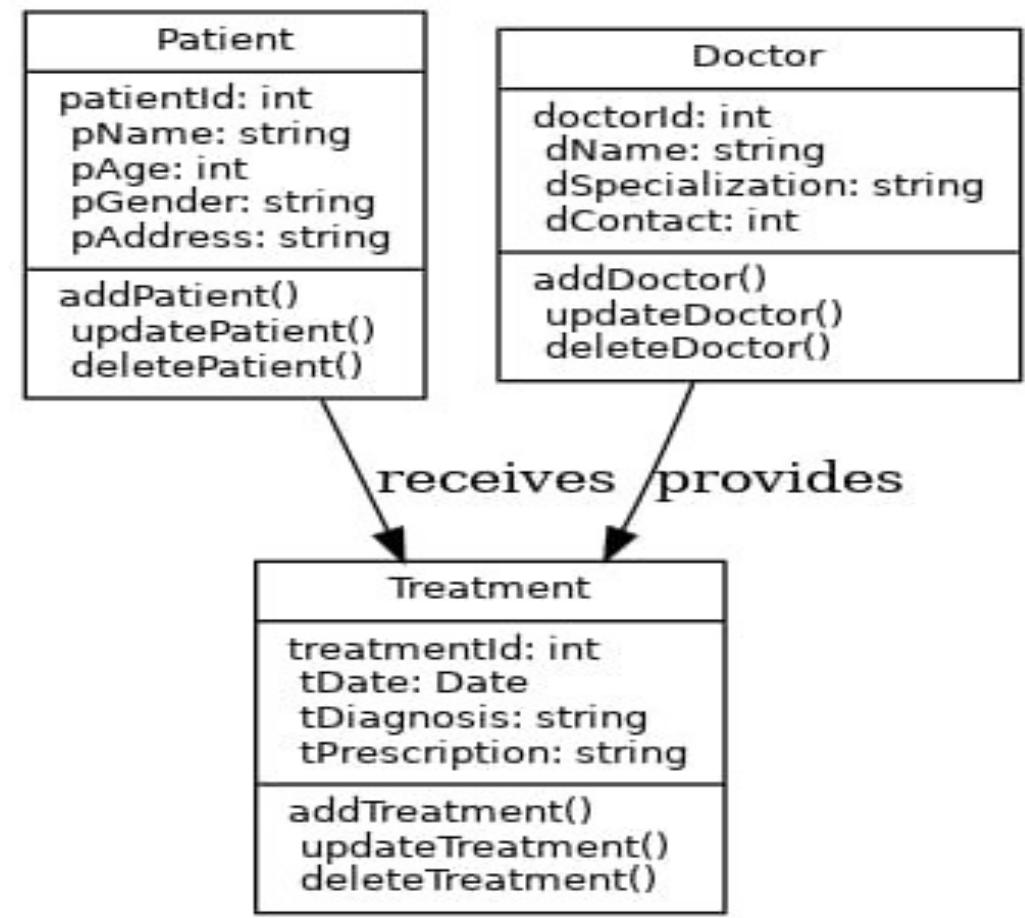
Aim:

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 shows the Jira software interface for the "HOSPITAL MANAGEMENT" project. The "HM board" is selected. The "List" view is displayed, showing the following data:

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

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:

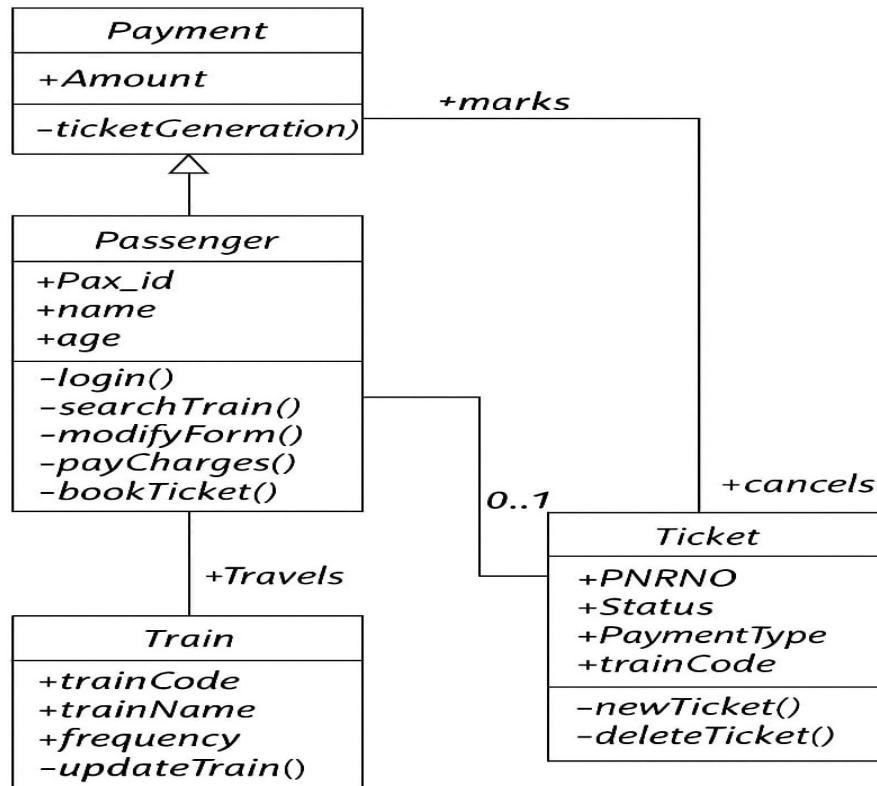
To create a **Scrum Board** for a Railway Reservation System using **Jira**, by organizing workflow into sprints and managing product backlog items such as Passenger, Train, Ticket, and Payment modules.

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:

The screenshot shows a project management application interface. The left sidebar includes sections for 'Recent', 'Starred', 'Apps', 'Plans', 'Projects' (with 'RRS board' selected), 'Teams', and 'More'. The main area displays a 'RRS board' with a 'List' view. The list table has columns for 'Type', 'Key', 'Summary', 'Status', 'Comments', 'Sprint', and 'Assignee'. One item is listed: 'RRS-1' with 'Passenger' summary, 'TO DO' status, and assigned to 'kasanisa25'. There are buttons for '+ Create' and 'Add comment'.

The screenshots illustrate the creation and configuration of a Scrum board for the Railway Reservation System (RRS) in Jira. The board is divided into three main sections: Backlog, Calendar, and Timeline.

- Backlog View:** Shows the backlog of work items. RRS Sprint 1 (29 Aug - 5 Sep) contains two tasks: "Passenger book the Ticket" (PASSENGER, DONE) and "RRS-2 Book Ticket" (PASSENGER, IN PROGRESS). RRS Sprint 2 (5 Sep - 12 Sep) contains one task: "Search Train" (PASSENGER, IN PROGRESS).
- Calendar View:** Displays the project schedule across five weeks from September 1 to September 12. It shows the start and end dates for each sprint and the tasks assigned to specific days.
- Timeline View:** Provides a detailed timeline from August to October. It highlights the duration of each sprint and the specific tasks assigned to certain dates, such as "RRS-1 Passenger" on August 29 and "RRS-2 Book Ticket" on September 5.

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

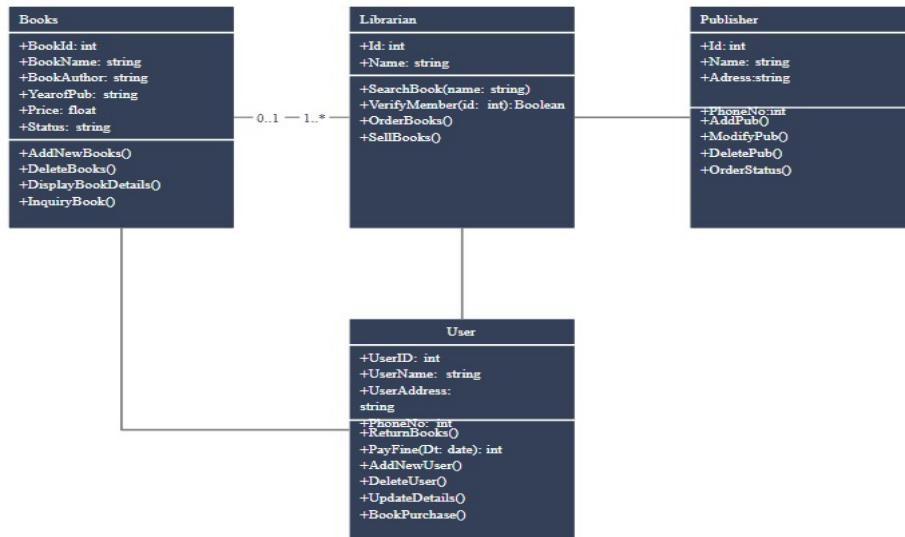
Aim:

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 shows a project management application interface. On the left, there is a sidebar with sections like 'For you', 'Recent', 'Starred', 'Apps', 'Plans', 'Projects', 'Teams', and 'More'. The 'Projects' section is expanded, showing 'Library management system' and other projects like 'HOSPITAL MAN...', 'HM board', 'My Scrum Project', and 'Railway Reservation System'. Below this is a 'Recent' section with items like 'Library management system', 'HOSPITAL MAN...', 'HM board', 'My Scrum Project', and 'Railway Reservation System'. The main area is titled 'Library management system' and contains a 'Board' view. The board has columns for 'Work', 'Assignee', 'Reporter', 'Priority', and 'Status'. There are three cards: 1. 'LMS-3 book' assigned to 'Unassigned' reporter, priority 'Medium', status 'TO DO'. 2. 'LMS-2 user' assigned to 'Unassigned' reporter, priority 'Medium', status 'TO DO'. 3. 'LMS-1 librarian' assigned to 'Unassigned' reporter, priority 'Medium', status 'TO DO'. At the bottom of the board, there is a '+ Create' button and a footer with 'Give feedback on the new' link.

Type	Key	Summary	Status	Comments	Assignee	Due date
	LMS-1	librarian	TO DO	Add comment		
	LMS-2	user	TO DO	Add comment		
	LMS-3	book	TO DO	Add comment		

Timeline view showing tasks over time:

- LMS-1 librarian: August 28 - September 1
- LMS-2 user: September 1 - September 2
- LMS-3 book: September 2 - September 3

Mon	Tue	Wed	Thu	Fri
Sep 1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26				

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:

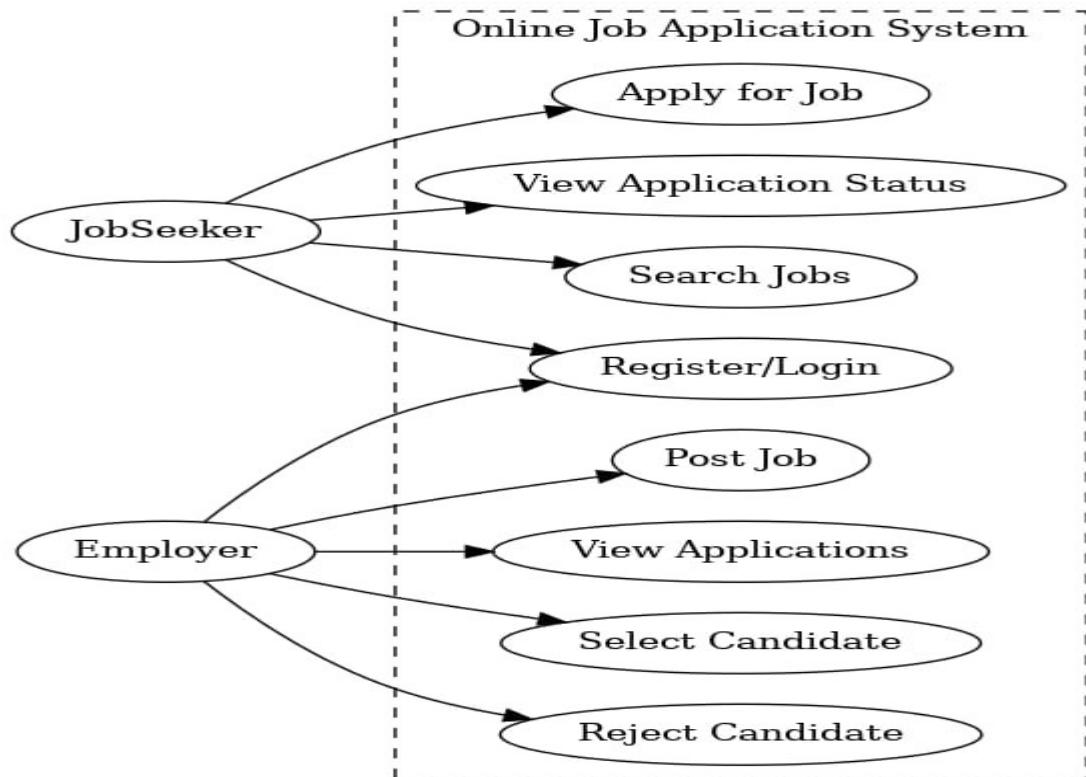
To create a Scrum Board for an Online Job Application System using Jira, by organizing workflow into sprints and managing product backlog items such as JobSeeker and Employer modules.

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 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:

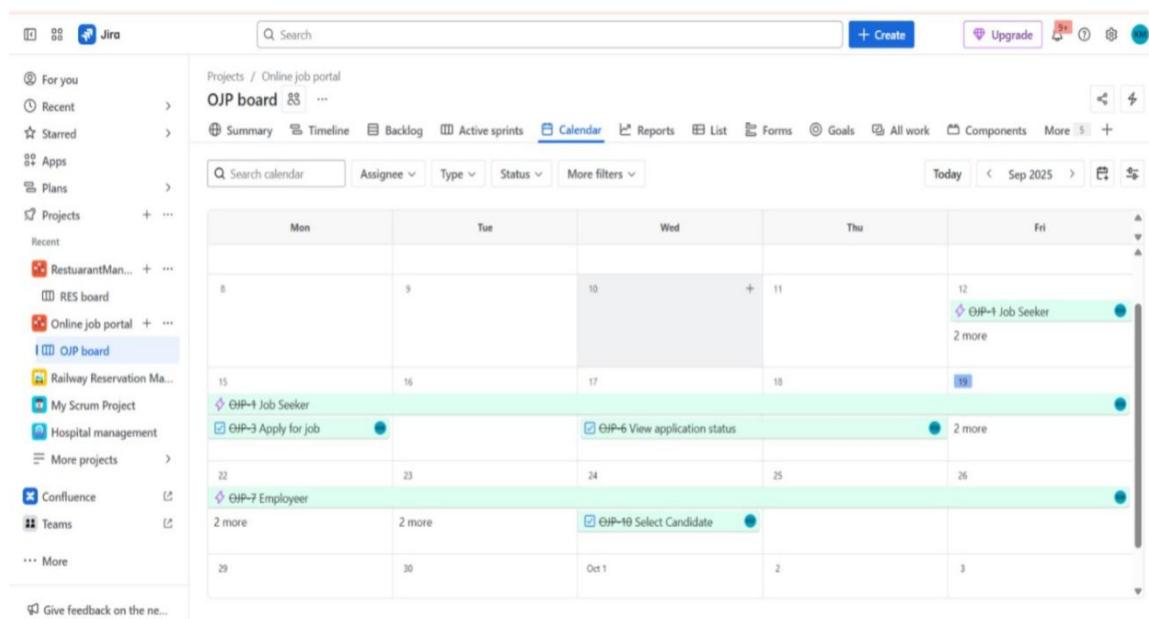
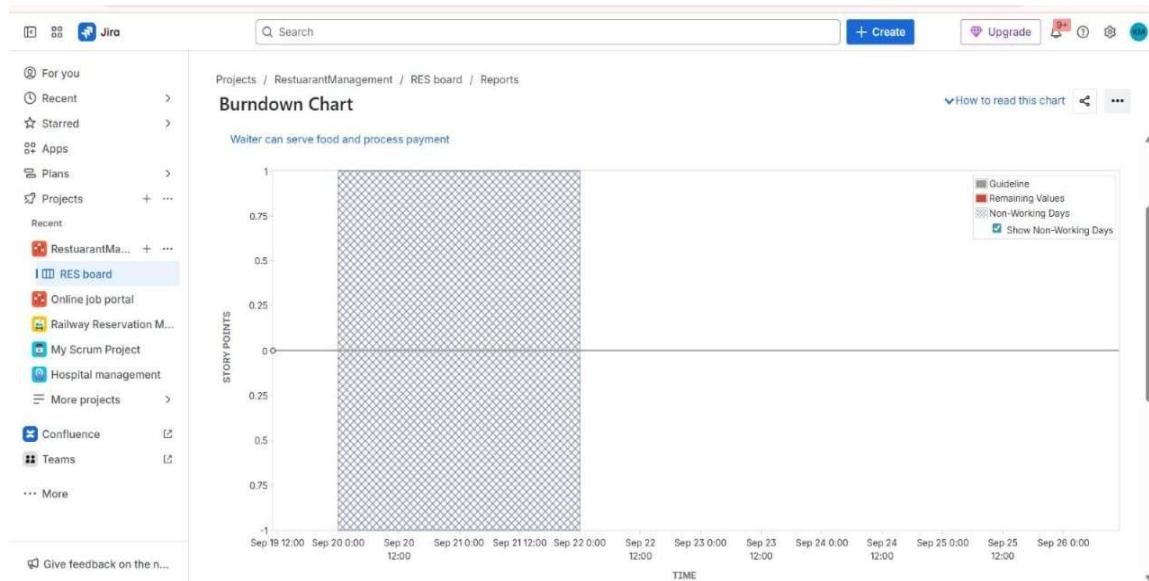
- **Initiate Release:** On the version page, click "**Release**".
- **Confirm Release Details:** Add actual release date (e.g., *Releasing Employer Module with Post Job, View Applications, Select and Reject Candidate stories*).
- **Execute Release:** Click "**Release**" to finalize.

Work Flow Images:

The screenshot shows the Jira interface for the 'OJP board'. The left sidebar includes links for 'For you', 'Recent', 'Starred', 'Apps', 'Plans', 'Projects' (with 'Online job p...' selected), 'Confluence', and a feedback link. The main area is titled 'OJP board' and shows a 'List' view. The table has columns for Type, Key, Summary, Status, Comments, Sprint, and Assignee. Work items include OJP-1 (Job Seeker, DONE, Sprint 1, Kavya Miriyal), OJP-2 (Search Job, DONE, Sprint 1, Kavya Miriyal), OJP-3 (Apply for job, DONE, Sprint 1, Kavya Miriyal), OJP-6 (View application status, DONE, Sprint 1, Kavya Miriyal), OJP-7 (Employee, DONE, Sprint 2, Kavya Miriyal), OJP-8 (Post Job Opening, DONE, Sprint 2, Kavya Miriyal), OJP-9 (View Job Application, DONE, Sprint 2, Kavya Miriyal), and OJP-10 (Select Candidate, DONE, Sprint 2, Kavya Miriyal).

The screenshot shows the Jira interface for the 'OJP board' in 'Timeline' view. The left sidebar is identical to the previous screen. The main area displays a timeline from September to November. It shows two releases: 'Version1' (September 18-19, 2025) and 'Version2' (September 21-22, 2025). Under 'Work', there are two completed epics: 'OJP-1 Job Seeker' (DONE) and 'OJP-7 Employee' (DONE). A button for '+ Create Epic' is visible. The bottom navigation bar includes 'Today', 'Weeks', 'Months' (selected), 'Quarters', and a date range selector.

The screenshot shows the Jira interface for the 'OJP board' in 'Releases' view. The left sidebar is identical to the previous screens. The main area is titled 'Release versions' and shows a table with columns for Version, Status, Progress, Start date, Release date, Description, and More actions. One release is listed: 'Version1' (Status: RELEASED, Progress: No work items, Start date: September 18, 2025, Release date: September 19, 2025). A 'Create version' button is located at the top right of the table area.



Result:

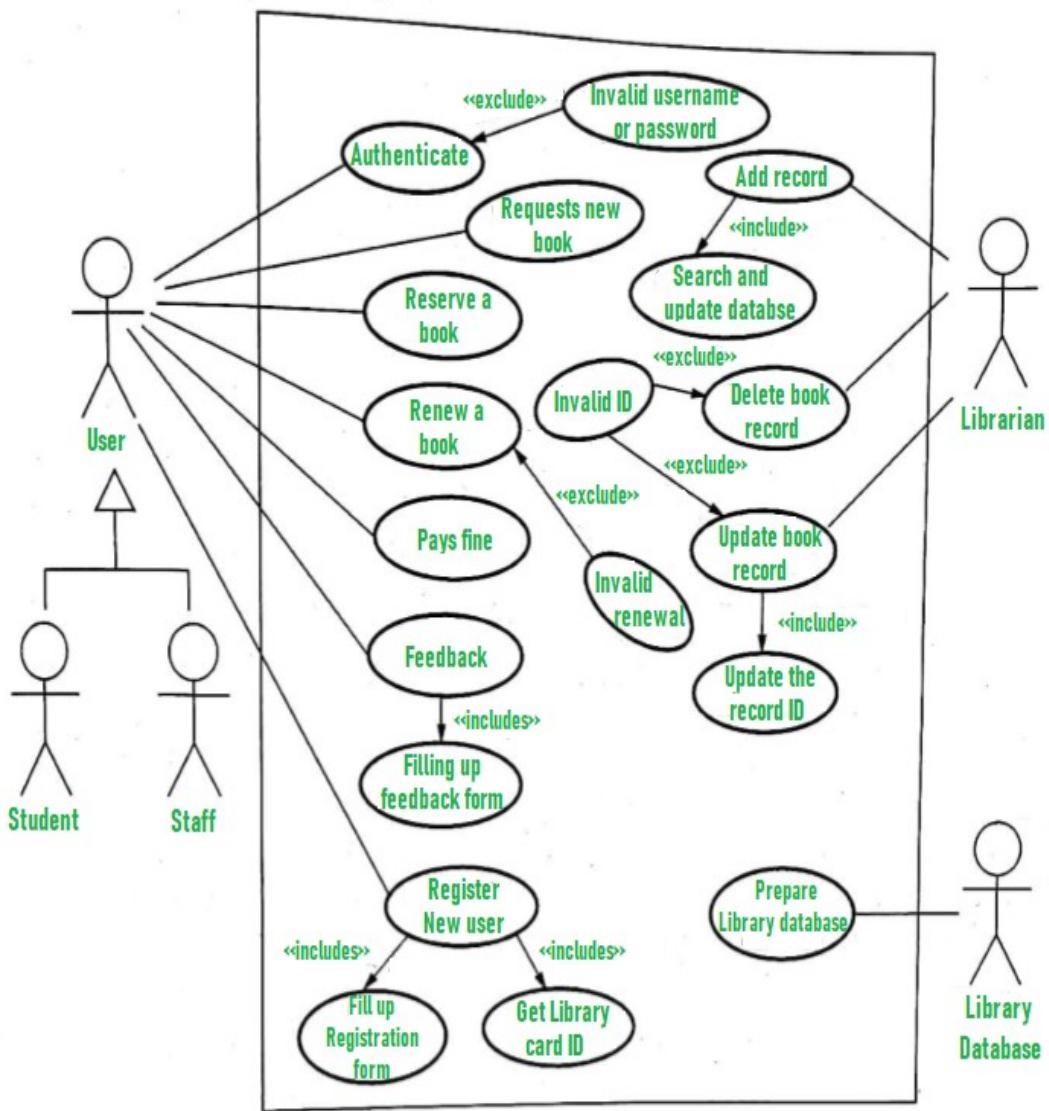
The Scrum Board for the Online Job Application System was successfully created using Jira. By dividing work into Epics, Stories, and Tasks aligned with the modules JobSeeker and Employer, the project can be developed efficiently with iterative sprint planning and tracking.

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

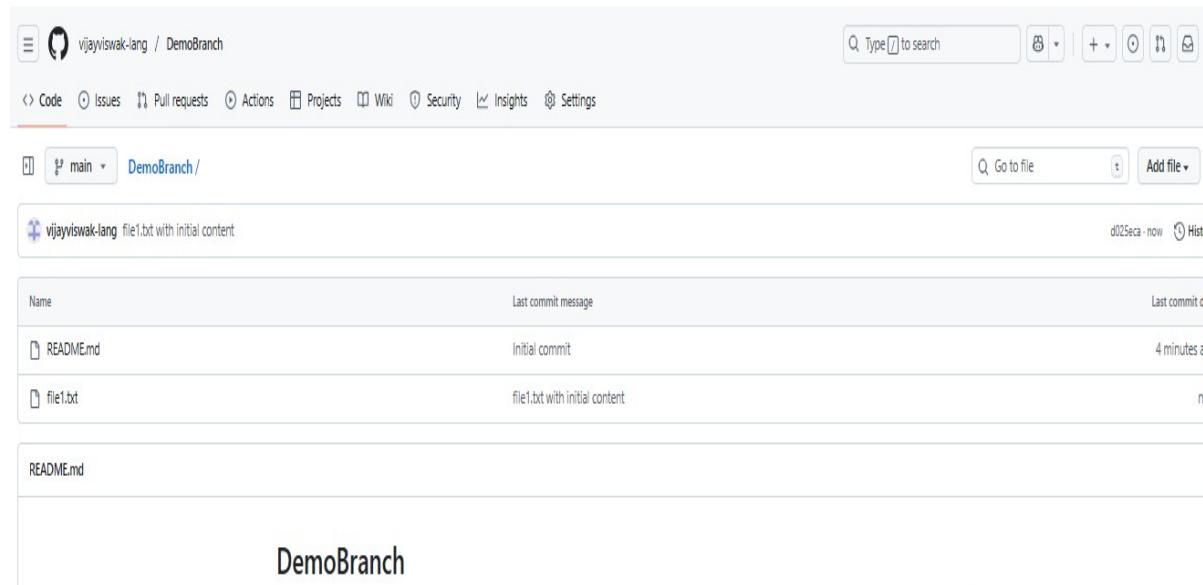
Experiment : To create and manage branches in a GitHub repository, perform changes independently in different branches, and then merge them into the main branch using a pull request, followed by deleting the merged branch.

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



The screenshot shows a GitHub repository page for 'vijayviswak-lang / DemoBranch'. The main branch is selected ('main'). The repository contains two files: 'README.md' and 'file1.txt'. The commit history shows an initial commit for 'file1.txt' made 4 minutes ago. The commit message is 'file1.txt with initial content'. The commit hash is d025eca. There is also a link to the history of the file.

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

The screenshot shows a GitHub repository named "vijayviswak-lang / DemoBranch". A modal window titled "Switch branches/tags" is open, with a search bar containing "Branch1" highlighted by a red arrow. Below the search bar, there are tabs for "Branches" and "Tags". A list of branches is shown, with the first item being "Create branch Branch1 from main", which is also highlighted by a red arrow. The main repository page below the modal shows a section titled "DemoBranch" with the text "created for Branch Create , Merge and Delete".

Create a file in Branch1 and commit

The screenshot shows the GitHub repository "vijayviswak-lang / DemoBranch" with "Branch1" selected. The commit history for Branch1 is displayed, showing two commits. The first commit is "file1.txt with initial content" (16 minutes ago) and the second commit is "file2.txt with initial content in Branch" (now). Both commits were made by "vijayviswak-lang". The main repository page shows a section titled "DemoBranch" with the text "created for Branch Create , Merge and Delete".

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

The screenshot shows two side-by-side GitHub repository pages. The left page is for "vijayviswak-lang / DemoBranch" with "Branch1" selected, showing commits for "file1.txt" and "file2.txt". The right page is for the same repository with "main" selected, showing a single commit for "file1.txt". Red arrows point to the "Branch1" dropdown on the left and the "main" dropdown on the right, indicating the current active branch.

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

The screenshot shows a GitHub repository named "DemoBranch". At the top, there are buttons for "Pin" and "Watch". Below the header, there are navigation links for "main", "1 Branch", and "0 Tags". A search bar with "Go to file" and a "Code" button are also present. The main content area displays a commit history:

File	Commit Message	Time
README.md	Initial commit	38 minutes ago
file1.txt	file1.txt with initial content	34 minutes ago
file2.txt	file2.txt with initial content in Branch	22 minutes ago

A red arrow points to the "file2.txt" entry, and another red arrow points to the "Merge pull request #1" entry in the commit history.

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

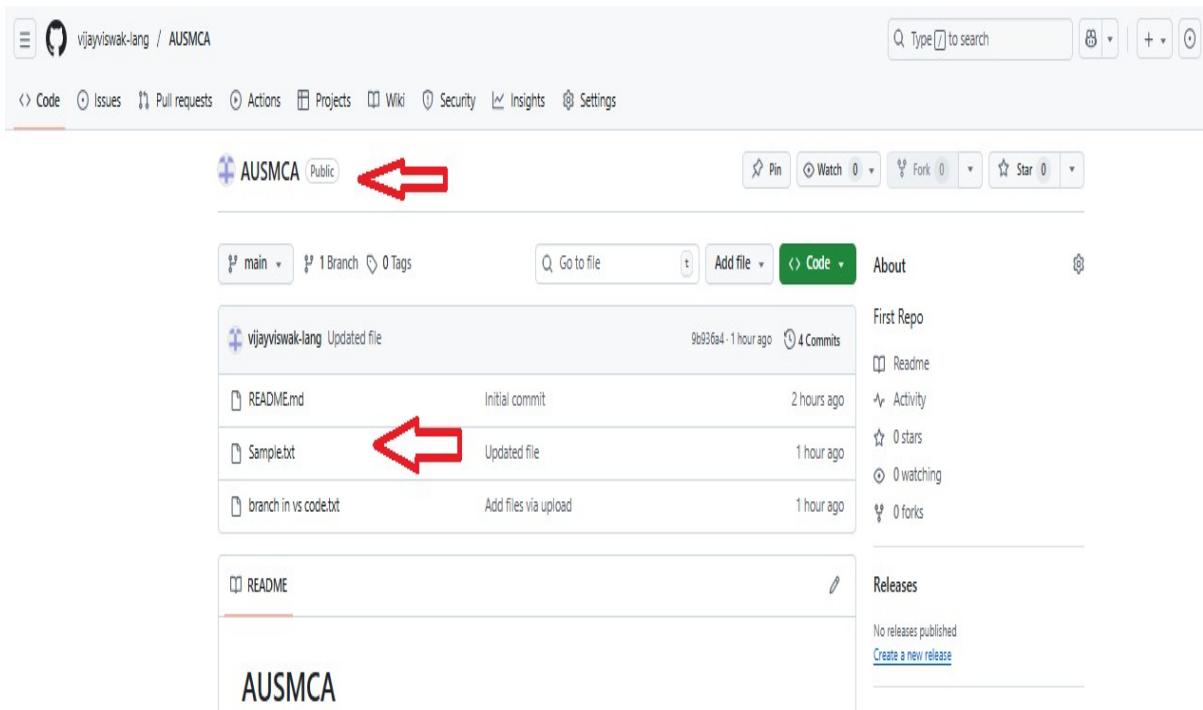
Aim:

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".

A screenshot of a GitHub repository page for 'AUSMCA'. The page shows a list of files: README.md (Initial commit), Sample.txt (Updated file), and branch in vs code.txt (Add files via upload). Below the files, there is a section titled 'AUSMCA'. On the right side, there is a 'Clone' button with three options: HTTPS, SSH, and GitHub CLI. A red arrow points to the 'HTTPS' link, which is highlighted with a red box. Another red arrow points upwards from the 'Clone' button towards the 'AUSMCA' section.

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.

A screenshot of a terminal window titled 'MINGW64:/c/Users/Teacher/ausmca'. The terminal shows the following command and its execution:

```
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`

The screenshot shows a terminal window titled "MINGW64:/c/Users/Teacher/ausmca". The command history is as follows:

```
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)
$
```

Red arrows highlight the directory path "ausmca" and the file "code.txt".

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 'Create a new repository' page on GitHub. The 'General' section is selected. A red arrow points to the 'Repository name' field, which contains 'AUSMCA'. Another red arrow points to the 'Description' field, which contains 'First Repo'. A third red arrow points to the 'Create repository' button at the bottom right of the form.

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.

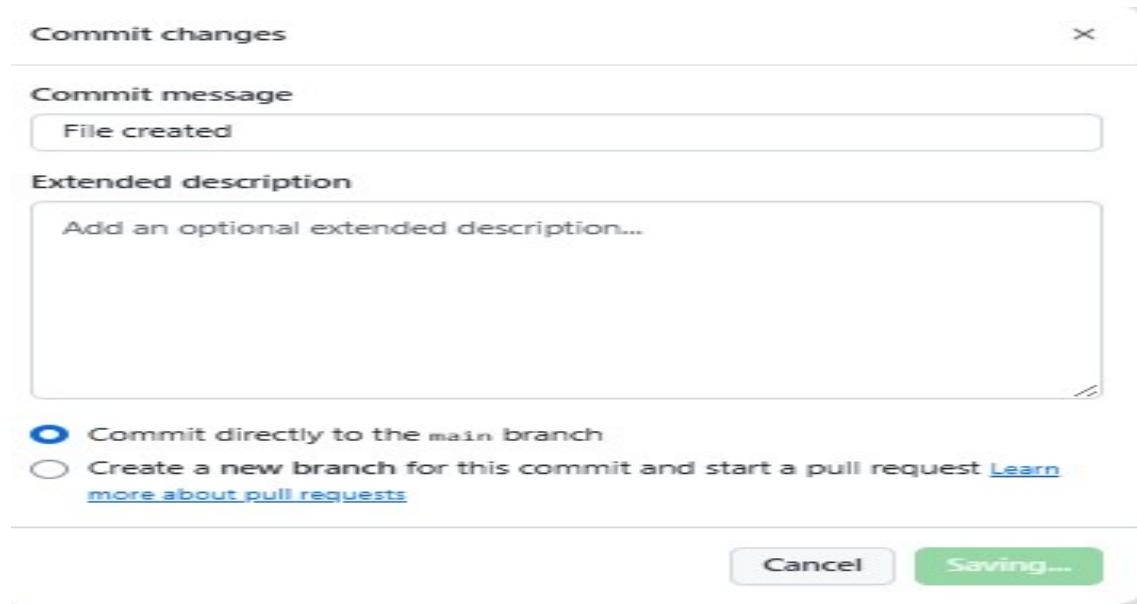
The screenshot shows a GitHub repository named 'AUSMCA'. At the top, there's a search bar and several navigation links: Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, the repository details are shown: 'main' branch, 1 Branch, 0 Tags. A search bar says 'Go to file'. To its right is a red arrow pointing to the 'Add file' button. The repository has 1 commit from 'vijayviswak-lang' with the message 'Initial commit' and a timestamp of '7 minutes ago'. On the right side, there's an 'About' section with 'First Repo', 'Readme', 'Activity', '0 stars', '0 watching', and '0 forks'. Below that is a 'Releases' section. The main content area shows a file named 'README' with the text 'AUSMCA'. Below it is a section titled 'Drag additional files here to add them to your repository' with a 'choose your files' link. A red arrow points to this link. The bottom part of the screenshot shows the 'Commit changes' dialog. It has a file list containing 'branch in vs code.txt' (circled in red). It includes fields for 'Add files via upload' and 'Add an optional extended description...'. Below these is a radio button group: 'Commit directly to the main branch.' (selected) and 'Create new branch for this commit and start a pull request.' A red arrow points to the second option. At the bottom are 'Commit changes' and 'Cancel' buttons.

Step 2.2: Create a new file

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

This screenshot shows the 'Commit changes' dialog from the previous step. The file 'branch in vs code.txt' is listed. The 'Create new branch for this commit and start a pull request.' radio button is selected. At the bottom, there are 'Cancel changes' and 'Commit changes...' buttons. Red arrows point to the 'Commit changes...' button and the 'Cancel changes' button. The bottom part of the screenshot shows the repository page again, with a file named 'Sample.txt' in the 'main' branch. The file content is '1 This is the new file in repd'. Red arrows point to the file name 'Sample.txt' and the file content area.

Enter the commit message and click Commit changes



Following screen shows after storing all the files.

vijayviswak-lang / AUSMCA

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main / AUSMCA /

vijayviswak-lang File created 43d91c3 · 2 minutes ago History

Name	Last commit message	Last commit date
README.md	Initial commit	1 hour ago
Sample.txt	File created	2 minutes ago
branch in vs code.txt	Add files via upload	18 minutes ago

README.md

AUSMCA

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

Commit changes

Commit message

Updated file

Extended description

Add an optional extended description...

Commit directly to the main branch

Create a new branch for this commit and start a pull request [Learn more about pull requests](#)

Cancel Saving...

Step 2: View History

Click the History button

The screenshot shows the GitHub repository 'vijayviswak-lang / AUSMCA' with the 'Sample.txt' file selected. The 'Commits' section displays two entries:

- Updated file**: authored 12 minutes ago (verified commit 9b936a4)
- File created**: authored 28 minutes ago (verified commit 43c91c3)

A red arrow points from the text 'Above screen shot shows the two versions of Sample.txt ie Created and Updated' to the 'File created' entry.

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

Click on Updated file

The screenshot shows the commit details for commit 9b936a4. The commit message is "updated file". The commit details show one file changed, Sample.txt, with the following content:

```
@@ -1 +1,2 @@
1   1 This is the new file in repo
2 + Now the file is updated
```

A red arrow points from the text 'Above screen shot shows the updated version of sample.txt' to the second line of the commit message.

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

The screenshot shows a GitHub repository page for the user 'vijay16'. The repository name is 'vijay16' and it is public. The commit history shows four commits:

- vijayviswak-lang Create ex2.txt (Initial commit, last month)
- ex1 (Update ex1, last month)
- ex2.txt (Create ex2.txt, last month)
- vijay16 (Commit message, last month)

The repository has 2 branches and 0 tags. It includes a README file. The 'About' section indicates it was created on 26.8.2025. The 'Releases' section shows no releases published.

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

vijayviswak-lang / sample1

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main sample1 /

vijayviswak-lang Create file2.txt with initial content 7bb75fd · now History

Name	Last commit message	Last commit date
README.md	Initial commit	1 minute ago
file1.txt	file1.txt with initial content	1 minute ago
file2.txt	Create file2.txt with initial content	now

README.md

sample1

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.