✅ Set 1:

◆ Q1. Git Scenario – Project Initialization

Objective: Start a project `InvoiceApp` and push it to GitHub.

🔧 Steps:

Initialize Git: Open the terminal in the folder and run:
mkdir InvoiceApp
cd InvoiceApp
git init
git config --global user.name "Atharva Gunjal"
git config --global user.email "your-email@example.com"
# Add your project files here
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/Atharva-Gunjal/class-test.git
git branch -M main
git push -u origin main
Or:
git push -u origin main

◆ Q2. JQL Advanced Search – Created, Due, and Resolution Filters

a. Find all issues created in the last 10 days that are still unresolved:

```
created >= -10d AND resolution = EMPTY
```

b. Show all issues that are due in the next 3 days:

```
due <= 3d
```

c. Find all bugs that were resolved in the last 5 days:

```
issuetype = Bug AND resolved >= -5d
```

Set 2:

✅ Q1. GitHub Scenario – Team Collaboration

◆ 1. Create a new GitHub repository

- Go to https://github.com
- Click **+** → **New repository**
- Enter repository name (e.g., `team-project`)
- (Optional) Add description
- Select **Public** or **Private**
- ✅ Check **Initialize this repository with a README**
- Click **Create repository**

◆ 2. Add a `README.md`

If not added during creation:

- On the repo page → Click **Add file** → **Create new file**
- Name it `README.md`
- Add project description
- Click **Commit changes**

◆ 3. Invite your team for collaboration

- Go to the repo → Click **Settings** → **Collaborators**
- Click **Invite a collaborator**
- Enter GitHub usernames/emails → Click **Add**

◆ 4. Set up branch protection on `main` to require pull requests

- Go to **Settings** → **Branches**
- Under **Branch protection rules**, click **Add rule**
- Set the pattern to `main`
- Enable:
    - ✅ Require a pull request before merging
    - ✅ Require review from at least 1 reviewer

- Click **Create** to save

✅ Q2. JQL Search – Transitions, Assignee, Due Dates

◆ a. Issues where status changed from "To Do" to "In Progress" in last 7 days

```
jql
CopyEdit
status CHANGED FROM "To Do" TO "In Progress" AFTER -7d
```

◆ b. Issues assigned to you that are overdue

```
jql
CopyEdit
assignee = currentUser() AND duedate < now() AND resolution = Unresolved
```

◆ c. Tasks that transitioned to "Done" in last 2 days

```
jql
CopyEdit
status CHANGED TO "Done" AFTER -2d
```

Set 3:

✅ Q1. Git Scenario – Branching & PR

◆ 1. Create a new branch `search-feature`

```
bash
CopyEdit
git checkout -b search-feature
```

◆ 2. Make changes and push the branch

```
bash
CopyEdit
# After editing files
git add .
git commit -m "Added search feature"
git push origin search-feature
```

◆ 3. Open a pull request and assign a reviewer

- Go to your repo on GitHub
- You'll see a prompt to open a **Pull Request** for `search-feature`
- Click **Compare & Pull Request**
- Add a **title** and **description**
- Assign a **reviewer** under "Reviewers" section

◆ 4. Merge the changes after review

- After reviewer approves, click **Merge pull request**
- Choose **Confirm merge**
- (Optional) Delete the `search-feature` branch

✅ Q2. JQL – Created/Updated/Transitioned Filter Queries

◆ a. Issues created between March 1 and March 10, 2025

```
jql
CopyEdit
created >= "2025-03-01" AND created <= "2025-03-10"
```

◆ b. Issues updated within the last 3 days

```
jql
CopyEdit
updated >= -3d
```

◆ c. Issues that transitioned from "In Progress" to "Testing" after April 1, 2025

```
jql
CopyEdit
status CHANGED FROM "In Progress" TO "Testing" AFTER "2025-04-01"
```

📓 When to use `updated >= -3d` VS `created >= -3d`

Query Type	Use it when you want to know...
`created >= -3d`	Which issues were **created** in the last 3 days
`updated >= -3d`	Which issues were **modified** (comment, status, field, etc.) in the last 3 days

Set-4

✅ Q1. Git Scenario – Resolve Merge Conflict

🔲 Situation:

You and your friend both worked on the same file (e.g., `index.html`) but on different branches.

You try to **merge** those branches, and Git says:

⚠️ Merge conflict! 😱

🔍 1. How to view the conflict

After running:

```
bash
CopyEdit
git merge teammate-branch
```

Git shows:

```
pgsql
CopyEdit
CONFLICT (content): Merge conflict in index.html
```

Now, open the file `index.html`. You'll see something like this:

```
html
CopyEdit
<<<<<<< HEAD
<h1>This is your version</h1>
```

```
=======
<h1>This is your teammate's version</h1>
>>>>>>> teammate-branch
```

These lines show **both versions**. You need to choose or mix them.

🛠️ 2. How to resolve it locally

Steps:

1. Open the file (`index.html`) in any editor (VS Code, Notepad++).
2. Decide what version is correct, or combine them.
3. Remove the `<<<<<<<`, `=======`, `>>>>>>>` lines.

For example, you might keep:

```
html
CopyEdit
<h1>This is the final version we agreed on</h1>
```

✅ 3. How to commit and complete the merge

After editing and saving the file:

```
bash
CopyEdit
git add index.html
git commit -m "Resolved merge conflict in index.html"
```

🎉 Done! You fixed the conflict and completed the merge.

✅ Q2. JQL – Component, Labels, Sprint Filters

📌 a. Find all issues in the Frontend component that are not resolved:

```
jql
CopyEdit
component = Frontend AND resolution = Unresolved
```

Means: Show tasks related to Frontend that are still open.

🚨 b. Find issues labeled as urgent or production-fix:

```
jql
CopyEdit
labels in (urgent, production-fix)
```

Labels are like tags. These help filter important tasks quickly.

📅 c. Show all issues in the current sprint assigned to your team:

```
jql
CopyEdit
sprint in openSprints() AND assignee in membersOf("your-team-name")
```

💡 Replace `"your-team-name"` with your actual Jira team (e.g., `"dev-team"`).

💡 Difference between Labels vs Components

Labels	Components
Like stickers or tags	Like folders or project sections
Used for priority (e.g., urgent)	Used for code area (e.g., Frontend)
Anyone can create them	Set by project admin
You can use many at once	Usually one per issue

Set 5:

✅ Q1. GitHub Scenario – Team Collaboration

◆ Step 1: Create a new GitHub repository

1. Go to https://github.com
2. Click **"New"** or the **"+"** icon > **New Repository**
3. Enter:
   - Repository name (e.g., `group-project`)
   - Description (optional)
   - Select **Public** or **Private**
   - ✅ Check **"Initialize with README"**
4. Click **Create repository**

◆ Step 2: Add a README.md If you didn't check "Initialize with README":

1. Click **Add file > Create new file**
2. Name the file: `README.md`
3. Add some content, e.g.:

```
csharp
CopyEdit
# Group Project
This is our group collaboration repo.
```

4. Click **Commit changes**

◆ Step 3: Invite your team

1. Go to your repo > **Settings**
2. Click **Collaborators**
3. Search and **invite teammates by GitHub username**

◆ Step 4: Set up branch protection to require Pull Requests

1. Go to **Settings > Branches**

2. Under **Branch protection rules**, click **Add rule**
3. Choose `main` branch
4. Enable:
    - ✅ "Require a pull request before merging"
    - ✅ "Require approvals"
5. Click **Create**

✅ Q2. JQL – Release Readiness Queries

a. Find all issues targeted for fixVersion = "v2.0":

```
jql
CopyEdit
fixVersion = "v2.0"
```

b. List bugs in v2.0 that are still unresolved:

```
jql
CopyEdit
fixVersion = "v2.0" AND issuetype = Bug AND resolution = Unresolved
```

c. Find tasks in v2.0 that were resolved in the last 7 days:

```
jql
CopyEdit
fixVersion = "v2.0" AND issuetype = Task AND status = Done AND resolved >= -
7d
```

Set 6 — Git + JQL (with examples):

✅ Q1. Git Scenario – Tag and Release

You're ready to release version `1.0`. Follow these steps:

◆ **Step 1: Tag the release as `v1.0`** In your local Git project folder, run:

```
bash
CopyEdit
```

```
git tag v1.0
```

This creates a lightweight tag called `v1.0` pointing to the current commit.

◆ **Step 2: Push the tag to GitHub**

```
bash
CopyEdit
git push origin v1.0
```

This makes the tag visible on GitHub.

◆ **Step 3: Create a release on GitHub from the tag**

1. Go to your GitHub repository.
2. Click **"Releases" > "Draft a new release"**.
3. In the **"Tag version"** dropdown, select `v1.0`.
4. Add a title (e.g., "Version 1.0 Release") and description.
5. Click **Publish release**.

✅ Q2. JQL – Overdue & SLA Queries

Assumptions:

- You use **`duedate`** field for deadlines.
- SLA may be tracked via **labels** or custom fields (e.g., `time to resolution`).

---

a. Show all issues that are overdue by more than 2 days:

```
jql
CopyEdit
duedate <= -2d AND resolution = Unresolved
```

Shows unresolved issues with due dates more than 2 days ago.

---

b. List all issues that must be resolved within 48 hours: Option 1 (if using a label like `SLA-48h`):

```
jql
CopyEdit
labels = SLA-48h AND resolution = Unresolved
```

Option 2 (if you have a custom SLA field):

```
jql
CopyEdit
"Time to resolution" <= remaining("48h") AND resolution = Unresolved
```

---

c. Find issues with a due date within this week:

```
jql
CopyEdit
duedate >= startOfWeek() AND duedate <= endOfWeek()
```

Set 7 — Jira Scrum Project Setup + JQL Filters:

✅ Q1. Jira Scenario – Scrum Project Setup

◆ **Step 1: Create the project**

1. Go to **Jira Dashboard**.
2. Click on **"Create Project"** (usually located at the top of the screen).
3. Select **"Scrum"** as the project template.
4. Enter a **project name**, key, and description.
5. Click **Create**.

This sets up your Scrum project, which includes Scrum boards, sprints, and backlog.

◆ **Step 2: Set up a board and sprints**

1. After the project is created, go to the **Board** settings.
2. Choose **Scrum** board.
3. Define the **columns** for the board, such as "To Do", "In Progress", "Done".
4. Set up **Sprints**:
    - Click on **Backlog**.
    - Create a **Sprint** by clicking **Create Sprint**.
    - Add backlog items (user stories, tasks, etc.) to this sprint.

◆ **Step 3: Add backlog items**

1. Go to the **Backlog** section of your board.
2. Click on **Create Issue** and add **User Stories**, **Tasks**, or **Bugs**.
3. Ensure that you define **Story Points** and assign each item to the right sprint.

◆ **Step 4: Start and manage a sprint**

1. After adding items to your sprint, click on **Start Sprint**.
2. Set the **Sprint duration** (e.g., 2 weeks).
3. Manage the sprint during execution:
    - **Track progress** with the Scrum board.
    - **Reassign tasks**, update statuses, or add new tasks as needed.
4. At the end of the sprint, perform a **Sprint Review** and **Sprint Retrospective** to discuss the completed work and improve for the next sprint.

✅ Q2. JQL – Team Assignment and Status Filters

a. Find all unresolved tasks assigned to Team-A:

```
jql
CopyEdit
assignee in (Team-A) AND resolution = Unresolved
```

This query returns all tasks that are assigned to Team-A and are still unresolved.

---

b. Find issues currently in "Blocked" status for more than 2 days:

```
jql
CopyEdit
status = Blocked AND updated <= -2d
```

This query finds all issues in "Blocked" status that haven't been updated in over 2 days.

---

c. List all bugs assigned to your name and in "In Review":

```
jql
CopyEdit
assignee = currentUser() AND status = "In Review" AND type = Bug
```

Set-8

Q1. GitHub Scenario – Fork and Contribute

You want to contribute to an open-source project. Here's a simple guide:

1. **Fork the Repository**:
   - On GitHub, go to the project you want to contribute to.
   - Click the **Fork** button (usually in the top-right corner of the page).
   - This will create a copy of that repository in your own GitHub account.
2. **Clone Your Fork**:
   - After forking, go to your own GitHub profile and find the forked repository.
   - Click the **Code** button and copy the **URL** (HTTPS or SSH).
   - Open your terminal/command prompt, go to the directory where you want to store the project, and run the following command:

     ```
     bash
     CopyEdit
     git clone https://github.com/your-username/repository-name.git
     ```

3. **Create a Branch, Make Changes, and Push**:
   - After cloning the repository, go into the project directory:

     ```
     bash
     CopyEdit
     cd repository-name
     ```

   - Create a new branch (a separate workspace for your changes):

     ```
     bash
     CopyEdit
     git checkout -b my-feature-branch
     ```

   - Make changes to the code in your editor (e.g., add a new feature or fix a bug).
   - After changes are done, save them and run:

     ```
     bash
     CopyEdit
     git add .
     git commit -m "Description of the changes"
     ```

   - Push your changes to your GitHub fork:

     ```
     bash
     CopyEdit
     git push origin my-feature-branch
     ```

4. **Open a Pull Request**:
   - Go to your forked repository on GitHub.
   - You'll see a message saying "Compare & Pull Request." Click it.
   - Add a description of what changes you've made.
   - Click **Create Pull Request** to request the original repository's owner to review and merge your changes.

Q2. JQL – Time-Sensitive Filters & Workload Tracking

These are **JQL queries** (Jira Query Language) used to filter issues based on time-related conditions. Here's a simple explanation:

1. **Find issues created this week**:

   ```
   jql
   CopyEdit
   created >= startOfWeek()
   ```

   - This will show all issues created **from the start of this week** (Monday) to now.
2. **Find all issues resolved in the previous quarter**:

   ```
   jql
   CopyEdit
   resolved >= startOfQuarter(-1) AND resolved <= endOfQuarter(-1)
   ```

   - This query finds issues that were resolved **in the previous quarter**.
   - `startOfQuarter(-1)` is the start of the previous quarter, and `endOfQuarter(-1)` is the end of the previous quarter.
3. **Show unresolved issues due in the next 7 days, assigned to your team**:

   ```
   jql
   CopyEdit
   resolution = Unresolved AND due <= 7d AND assignee in (Team-A)
   ```

   - This will list all unresolved issues that are **due within the next 7 days** and **assigned to your team** (Team-A).

Time-Sensitive Functions in JQL:

- **startOfWeek()**: This function gets the **first day of the current week** (usually Monday).
- **startOfQuarter(-1)**: This gets the **first day of the previous quarter**. If you use `-1`, it gives you the previous quarter (e.g., if today is April 2025, this will return January 1, 2025).
- **<= 7d**: This is a **relative date filter**. It shows issues with a **due date within the next 7 days**.