**Lesson 1 – Course Kick-off & Tooling**

**Lesson Objectives**

By the end of this session, students will be able to:

1. Set Up Flutter in VS Code
2. Create & Run a Flutter App
3. Use Git via VS Code’s Terminal
4. Publish to GitHub & Invite Teammates
5. Collaborate as a Contributor
6. Follow Naming Conventions

**Required Materials**

* Laptop or PC with admin rights
* Internet access
* VS Code (with Flutter & Dart extensions) or Firebase Studio
* Git installed and configured (name & email)
* A GitHub account for remote hosting

*FYI: Check the Flutter and Git Installation Guide before you begin*

**1 Setting Up Your Tools**

1. **Open VS Code**
   * Click the VS Code icon.
   * You’ll see a big window with menus at the top and a blank area.
2. **Open the Terminal in VS Code**
   * From the menu, choose **View → Terminal**.
   * A “black box” appears at the bottom. This is where you type commands.
3. **Check Flutter Is Ready**
   * In the terminal, type:

flutter --version

* **What it does:** tells you which Flutter and Dart versions you have.
* If it shows numbers, you’re all set! If it says “not found,” ask your teacher to help install Flutter.

1. **Enable Web Support**
   * In the same terminal, type:

flutter config --enable-web

* **What it does:** lets your Flutter app run in a browser like Chrome.

1. **See Your Devices**
   * Type:

flutter devices

* **What it does:** lists where you can run your app—your computer browser or phone.

**2 Creating Your First Flutter App**

1. **Open VS Code & Install Extensions**
   * Launch VS Code.
   * Go to Extensions (🧩 icon), search for **Flutter**, and click **Install** (this also adds the Dart extension).
2. **Start a New Flutter Project**
   * Press **Ctrl + Shift + P** (or **F1**) to open the Command Palette.
   * Type and select **Flutter: New Project**.
   * Choose **Application** when prompted.
   * Enter your project name: my\_oop\_app.
   * Pick a folder where it will live.
   * VS Code will scaffold a new Flutter app for you.
3. **Select Your Device (Browser)**
   * In the bottom-right corner of VS Code’s status bar, click where it shows your current device (it may say “No Devices”).
   * From the dropdown, select **Chrome (web)**.
4. **Run the App**
   * Click the green ▶️ **Run** button at the top of the Debug pane (Ctrl+Shift+D on Windows).
   * Your default browser (Chrome) will open automatically to localhost:xxxx and display the starter counter app.
5. **Hot Reload in VS Code**
   * Make a small edit in lib/main.dart (e.g., change the title in MyHomePage(title: ' Flutter Demo Home Page')).
   * Click the circular 🔄 **Hot Reload** button in the debug toolbar floating above VS Code.
   * Your browser updates instantly—no restart needed!

**3 Uploading Your First Flutter App to GitHub**

**1. Initialize Git in Your Project**

1. Open your Flutter project in VS Code.
2. Open the terminal (**View → Terminal**).
3. In the terminal, type:

git init

*This starts Git tracking in your folder.*

**2. Prepare Your First Commit**

1. See what changed:

git status

*Shows all new files waiting to be saved.*

1. Add everything:

git add .

*Tells Git “I want to save all files.”*

1. Save your snapshot:

git commit -m "Initial Flutter app setup"

*Records your first version with a message you choose.*

**3. Create & Link Your GitHub Repo**

1. On GitHub.com, click **New Repository**.
   * Name it my\_oop\_app.
   * **Do not** check “Initialize with a README.”
   * Click **Create Repository**
2. Back in VS Code’s terminal, copy the lines GitHub shows under “…or push an existing repository from the command line”:

git remote add origin https://github.com/<*your-username*>/my\_oop\_app.git

git branch -M main

git push -u origin main

*These commands:*

* Tell Git where your online repo lives.
* Rename your branch to main.
* Send your code up to GitHub.

**4. Confirm Your Code Is Online**

* Go to your GitHub repo page (https://github.com/<*your-username*>/my\_oop\_app).
* You should see all your Flutter files listed.

**4 Inviting Your Teammates**

1. On your GitHub repo page, click **Settings** (near the top).
2. In the left sidebar, choose **Access** > **Collaborators > Manage People**.
3. Click **Add people**.
4. Enter each teammate’s GitHub username or email, and click **Add**.
5. They’ll receive an invitation—once they accept, they can clone and push to your repo.

**5 Contributing Changes as a Collaborator**

When you’re added as a co‑member (collaborator) on the group’s GitHub repo, follow these steps in VS Code’s terminal to contribute safely without owning the repo.

**1. Clone the Shared Repository**

1. **Get the clone URL:** On GitHub, go to the repo page and click **Code → HTTPS**, then copy the URL.
2. **Clone locally:** In VS Code’s terminal, run:

git clone https://github.com/<*your‑leader*>/my\_oop\_app.git

*This makes a local copy on your computer.*

1. **Enter the folder:**

cd my\_oop\_app

**2. Create Your Own Feature Branch**

Always work on a separate branch so you don’t overwrite others:

git checkout -b feature/<*yourname*>-change

Example: git checkout -b feature/stephenbalo-add-student-profile

* **checkout -b** makes and switches to a new branch named feature/<*yourname*>-change.

**3. Make and Save Your Changes**

1. **Edit files:**
   * Open lib/main.dart, README.md, or any file you’re assigned.
2. **Stage your edits:**

git add .

1. **Commit with a clear message:**

git commit -m "feat: add student-profile widget"

1. **Synchronize Before You Push**

Because someone else might have pushed new commits, you must pull in their work first:

git pull --rebase origin feature/<yourname>-change

This fetches new commits from GitHub and reapplies your commits on top of them.

If there are conflicts, Git will ask you to resolve them—once fixed,  
 git rebase --continue.

**4. Push Your Branch Upstream**

git push -u origin feature/<*yourname*>-change

* **-u origin** tells Git to remember where to push next time.
* Now your branch is on GitHub under the main repo.

**5. Open a Pull Request (PR)**

1. On GitHub, you’ll see a banner “Compare & pull request.”
2. Click it, or go to **Pull requests → New pull request**.
3. Choose your branch (feature/<*yourname*>-change) as the source, and **main** as the target.
4. **Title:** Short summary, e.g. “Add student‑profile widget.”
5. **Description:**
   * What you changed
   * Why it helps
6. Click **Create pull request**.

**6. Keep Your Branch Up to Date**

If others merge into **main** while you’re working:

1. **Switch back to main:**

git checkout main

1. **Download the latest updates:**

git pull origin main

1. **Go back to your branch:**

git checkout feature/<*yourname*>-change

1. **Merge main into your branch:**

git merge main

1. **Resolve any merge conflicts**, stage, and commit. Then push again:

git add .

git commit -m "<*your message*>"

git push

**Tips:**

A good, consistent naming convention makes it easy to see *what* each branch and commit is for at a glance. Here’s a simple guideline you can follow:

**1. Branch Naming**

Use the format:

<type>/<short‑hyphenated‑description>

* **type** – what kind of work you’re doing. Common types:
  + feature – brand‑new functionality
  + fix – a bug fix
  + docs – documentation changes
  + chore – maintenance (e.g., build tweaks, dependency bumps)
* **description** – a few words (all lowercase, hyphens instead of spaces) describing the work

**Examples**

* feature/home-screen
* fix/login-button-alignment
* docs/update-readme
* chore/upgrade-flutter-sdk

**2. Commit Message Style**

Follow the **Conventional Commits** pattern:

<type>(<scope>?): <short‑imperative‑sentence>

1. **type** – same kinds as branches (feat, fix, docs, chore, plus refactor, test, etc.)
2. **scope** *(optional)* – the area of the code you’re touching (e.g. main.dart, auth, README)
3. **subject** – a brief, imperative‑mood description of *what* changed

* **Lowercase** for type and scope
* **No period** at the end of the subject
* **Keep it under ~50 characters** if possible

**Examples**

* feat(home\_screen): add centered “Hello, OOP!” text
* fix(auth): correct password validation logic
* docs(readme): add setup instructions for web
* chore(deps): bump flutter to latest stable

**Why This Works**

* **Branches** tell you *where* you’re working (feature/... tells teammates “this is new work”)
* **Commits** tell you *what* changed and *why* (“fix(auth): correct password validation logic” immediately explains the problem you solved)

Sticking to these conventions keeps your repo organized, makes code reviews smoother, and helps everyone on the team understand history at a glance.

**Group Task 1 (3–4 Members)** *Practice: branching, simple UI change, collaboration workflow, pull‑request drafting.*

**1. Group Roles & Setup**

|  |  |
| --- | --- |
| **Role** | **Primary Tasks** |
| **Member 1** | – Create the feature branch in Git  – Push initial branch to GitHub |
| **Member 2** | – Modify lib/main.dart to display “Hello, OOP!” centered |
| **Member 3** | – Update README.md with setup, run, and branch instructions |
| **Member 4** | – Draft the Pull Request title & description on GitHub  – Prepare the group report structure |

If only 3 members, combine “Member 4” with another role.

**2. Step‑by‑Step Instructions**

1.      **Clone the Repository from (Member 1)**

git clone https://github.com/your-leader/my\_oop\_app.git

cd my\_oop\_app

*FYI: You can skip this one if you have already created this repository from Lesson 1*

2.      **Create & Push the Feature Branch (Member 1)**

git checkout -b feature/home-screen

git push -u origin feature/home-screen

*Now the branch exists on GitHub for everyone to use.*

3.      **Fetch & Switch to the Feature Branch (Members 2–4)** Each of you, on your own device, do:

git clone https://github.com/your-leader/my\_oop\_app.git

cd my\_oop\_app

git fetch

git checkout feature/home-screen

*This pulls down and checks out the shared branch.*

4.      **Implement the UI Change (Member 2)**

·   Open lib/main.dart in VS Code.

·   Replace the Scaffold body with:

Center(

  child: Text(

'Hello, OOP!',

style: TextStyle(fontSize: 24),

  ),

)

·   Save, then run to verify:

flutter run -d chrome

5.      **Stage, Commit & Push All Changes (Member 2)**

git add .

git commit -m "feat(main.dart): add Hello, OOP! centered text"

git push -u origin feature/home-screen

*Pushes Member 2’s work back up to feature/home-screen.*

6.      **Update Documentation (Member 3)**

·   Edit README.md to include:

## Setup & Run (Web)

1. flutter config --enable-web

2. flutter run -d chrome

## Branch & Commit

- git checkout feature/home-screen

- git add .

- git commit -m "feat: add Hello, OOP! home screen"

- git push

·   Save your changes.

7.      **Pull, Stage, Commit & Push All Changes (Member 3)**

git pull

git add .

git commit -m "docs(readme): add terminal instructions"

git push -u origin feature/home-screen

*Pushes everyone’s combined work back up to feature/home-screen.*

8.      **Open the Pull Request (Member 4)**

·   On GitHub, go to your repo → **Pull requests → New pull request**.

·   Base: main ← Head: feature/home-screen

·   **Title:** feat: add Hello, OOP! home screen

·   **Description:**

o   **What:** Centered greeting in Flutter UI and updated README.

o   **Why:** Practice branching, collaboration, and documentation.

·   Click **Create pull request**.