Full Stack Development BootCamp - Day 7

Build Real-World Apps from Backend to Frontend!

Trainer:

Muhammad Maaz Sheikh
Technical Team Lead

Tech Stack Covered:

- Nest Js
- PostgreSQL
- React Js
- Real world project development

Organized By:

Iqra University

Recap Day 6

Table of Content

Section A-10: Misc. Extra Items For Full Stack Development

Login Page Implementation with JWT Auth

Login Page Implementation with JWT Auth

Goal:

- Login Page with email/userId and password
- Call your Nest JS login API using RTK Query
- Save JWT in localStorage or redux state
- Implement login/logout functionality
- Show protected route with 2-3 menu items after login
- Provide file structure with exact file paths

Login Page Implementation with JWT Auth

Step-by-Step Plan:

Step#01: Create Your React Project

- npm create vite@latest
- Press enter to proceed to your project name
- Select Framework and Variant React and Typescript

Step#02: Installed Required Packages

```
npm install @reduxjs/toolkit react-redux
npm install react-router-dom
```

Login Page Implementation with JWT Auth

Step#03: Create RTK Query for Login

Path: /src/features/auth/authApi.ts

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';
export const authApi = createApi({
 reducerPath: 'authApi',
 baseQuery: fetchBaseQuery({
   baseUrl: 'http://localhost:3000', // Replace with your NestJS API base
 }),
 endpoints: (builder) => ({
   login: builder.mutation<{ access_token: string }, { email: string; password: string }>({
     query: (credentials) => ({
       url: '/auth/login', // Your API endpoint
       method: 'POST',
       body: credentials,
     }),
   }),
 }),
});
export const { useLoginMutation } = authApi;
```

Login Page Implementation with JWT Auth

Step#04: Create Redux Slice for Auth State

Path: /src/features/auth/authSlice.ts

```
import { createSlice } from '@reduxjs/toolkit';
interface AuthState {
 token: string | null;
}
const initialState: AuthState = {
 token: localStorage.getItem('token'),
};
const authSlice = createSlice({
 name: 'auth',
 initialState,
 reducers: {
   setToken: (state, action) => {
     state.token = action.payload;
     localStorage.setItem('token', action.payload);
   },
   logout: (state) => {
     state.token = null;
     localStorage.removeItem('token');
   },
 },
});
export const { setToken, logout } = authSlice.actions;
export default authSlice.reducer;
```

Login Page Implementation with JWT Auth

Step#05: Setup Redux Store with RTK Query

Path: /src/app/store.ts

```
import { configureStore } from '@reduxjs/toolkit';
import { authApi } from '../features/auth/authApi';
import authReducer from '../features/auth/authSlice';

export const store = configureStore({
 reducer: {
   [authApi.reducerPath]: authApi.reducer,
   auth: authReducer,
 },
 middleware: (getDefaultMiddleware) =>
   getDefaultMiddleware().concat(authApi.middleware),
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Login Page Implementation with JWT Auth

Step#06: Setup Selector file to select the Token Value

Path: /src/features/auth/authSelector.ts

```
import type { RootState } from "../../app/store";



export const selectTokenValue = (state: RootState) => state.auth.token;
```

Login Page Implementation with JWT Auth

Step#07: Update your main.tsx file

Path: /src/main.tsx

Your main.tsx should look like below

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App'
import { Provider } from 'react-redux';
import { store } from './app/store';
import { BrowserRouter } from 'react-router-dom';

createRoot(document.getElementById('root')!).render(
 <StrictMode>
   <Provider store={store}>
     <BrowserRouter>
       <App />
     </BrowserRouter>
   </Provider>
 </StrictMode>,
)
```

Login Page Implementation with JWT Auth

Step#08: Protected Route Setup

Path: /src/components/protectedRoute.tsx

```
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useSelector } from 'react-redux';
import { RootState } from '../app/store';

const ProtectedRoute = ({ children }: { children: JSX.Element }) => {
 const token = useSelector((state: RootState) => state.auth.token);
 return token ? children : <Navigate to="/login" />;
};

export default ProtectedRoute;
```

Login Page Implementation with JWT Auth

Step#09: Nav Bar Setup for Menu Navigation

Path: /src/components/navBar.tsx

```
import { Link } from 'react-router-dom' ;
import { logout } from '../features/auth/authSlice' ;
import { useDispatch, useSelector } from 'react-redux';
import { selectTokenValue } from '../features/auth/authSelector' ;
const Navbar = () => {
 const token = useSelector(selectTokenValue);
 const dispatch = useDispatch();
 const handleLogout = () => {
   dispatch(logout());
 };
 if (!token) return null;
 return (
   <nav>
     <Link to="/dashboard">Dashboard</Link> |
     <Link to="/home"> Home</Link> |
     <Link to="/about"> About</Link> |
     <button onClick={handleLogout}>Logout</button>
   </nav>
 );
};
export default Navbar;
```

Login Page Implementation with JWT Auth

Step#10: Create Login Page UI

Path: /src/pages/Login.tsx (Part 1)

```
import React, { useState } from 'react';
import { useLoginMutation } from '../features/auth/authApi' ;
import { useDispatch } from 'react-redux';
import { setToken } from '../features/auth/authSlice' ;
import { useNavigate } from 'react-router-dom' ;
const Login = () => {
 const [email, setEmail] = useState('');
 const [password, setPassword] = useState('');
 const [login] = useLoginMutation ();
 const dispatch = useDispatch();
 const navigate = useNavigate();

 const handleSubmit = async (e: React.FormEvent) => {
   e.preventDefault();
   try {
     const result = await login({ email, password }).unwrap();
     dispatch(setToken(result.access_token));
     navigate('/dashboard');
   } catch (error) {
     alert('Login failed');
   }
 };
```

Login Page Implementation with JWT Auth

Step#10: Create Login Page UI

Path: /src/pages/Login.tsx (Part 2)

```
return (
   <form onSubmit={handleSubmit}>
     <h2>Login</h2>
     <input
       type="text"
       placeholder="Email or User ID"
       value={email}
       onChange={(e) => setEmail(e.target.value)}
     />
     <br />
     <input
       type="password"
       placeholder="Password"
       value={password}
       onChange={(e) => setPassword(e.target.value)}
     />
     <br />
     <button type="submit">Login</button>
   </form>
 );
};
export default Login;
```

Login Page Implementation with JWT Auth

Step#11: Create Dashboard, Home And About Pages

Path: /src/pages/Dashboard.tsx

```
const Dashboard = () => {
return <h1>Welcome to Dashboard Page!</h1>;
};
export default Dashboard;
```

Path: /src/pages/Home.tsx

```
const Home = () => {
   return <h1>Welcome to Home page!</h1>;
 };
export default Home;
```

Path: /src/pages/About.tsx

```
const About = () => {
   return <h1>Welcome to About Page!</h1>;
 };
export default About;
```

Login Page Implementation with JWT Auth

Step#12: Set Up Routing update your App.tsx

Path: /src/App.tsx

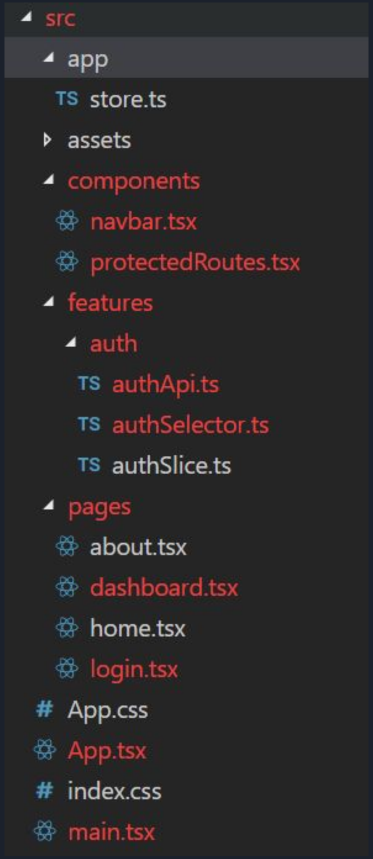
Your **app.tsx** should look like this Remove old code that was generated automatically with project

```
import { useState } from 'react'
import { Routes, Route, Navigate } from 'react-router-dom';
import Login from './pages/Login';
import Dashboard from './pages/Dashboard';
import './App.css'
import Home from './pages/home';
import About from './pages/about';
import Navbar from './components/navbar';
import ProtectedRoute from './components/protectedRoutes';
function App() {
 return (
     <div>
     <Navbar />
     <Routes>
       <Route path="/login" element={<Login />} />
       <Route path="/dashboard" element={<ProtectedRoute><Dashboard /></ProtectedRoute>} />
       <Route path="/home" element={<ProtectedRoute><Home /></ProtectedRoute>} />
       <Route path="/about" element={<ProtectedRoute><About /></ProtectedRoute>} />
     </Routes>
   </div>
 )
}export default App
```

Login Page Implementation with JWT Auth

Final Project and File Setup

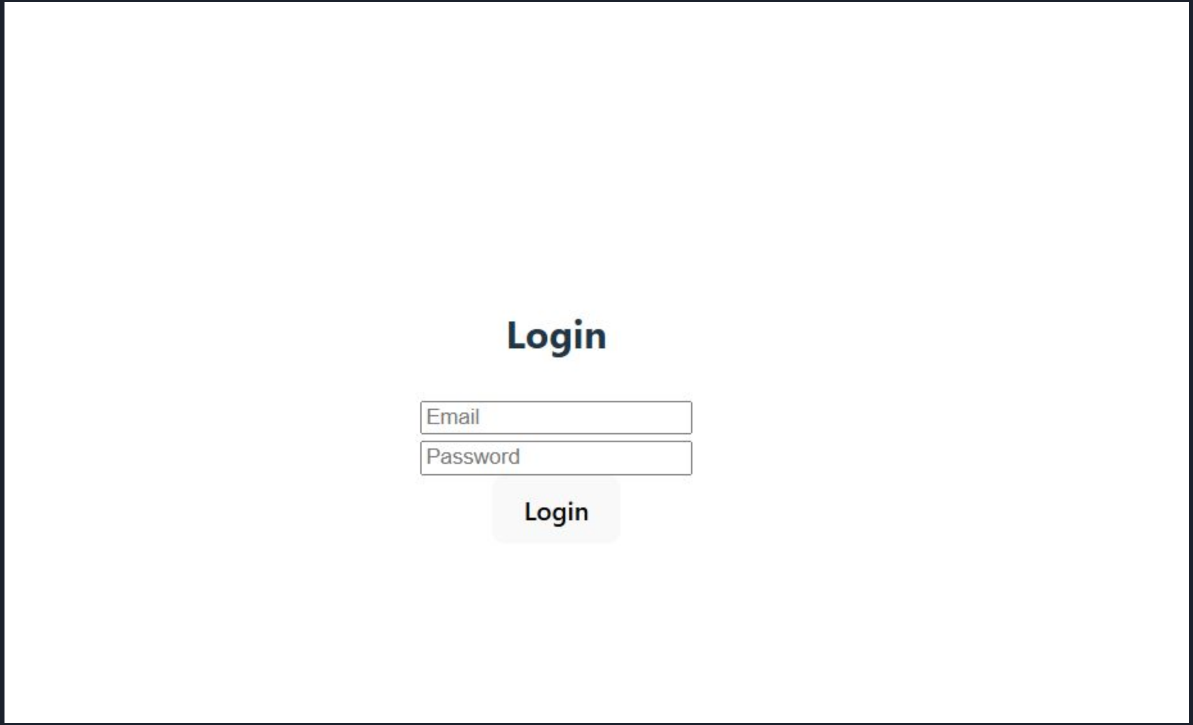
Now run npm run dev



```
src
  app
    store.ts
  assets
  components
    navbar.tsx
    protectedRoutes.tsx
  features
    auth
      authApi.ts
      authSelector.ts
      authSlice.ts
  pages
    about.tsx
    dashboard.tsx
    home.tsx
    login.tsx
  App.css
  App.tsx
  index.css
  main.tsx
```

Login Page Implementation with JWT Auth

Output of the Application:



Database Normalization

Database Normalization

What is Normalization?

Database normalization is a database design principle for organizing data in an organized and consistent way.

Why Normalize?

- Reduce duplicate data
- Ensure logical data storage
- Improve query performance
- Maintain consistency

Database Normalization

Common Database Problems Without Normalization

The primary objective for normalizing the relations is to eliminate the below anomalies. Failure to reduce anomalies results in data redundancy, which may threaten data integrity and cause additional issues as the database increases. Normalization consists of a set of procedures that assist you in developing an effective database structure.

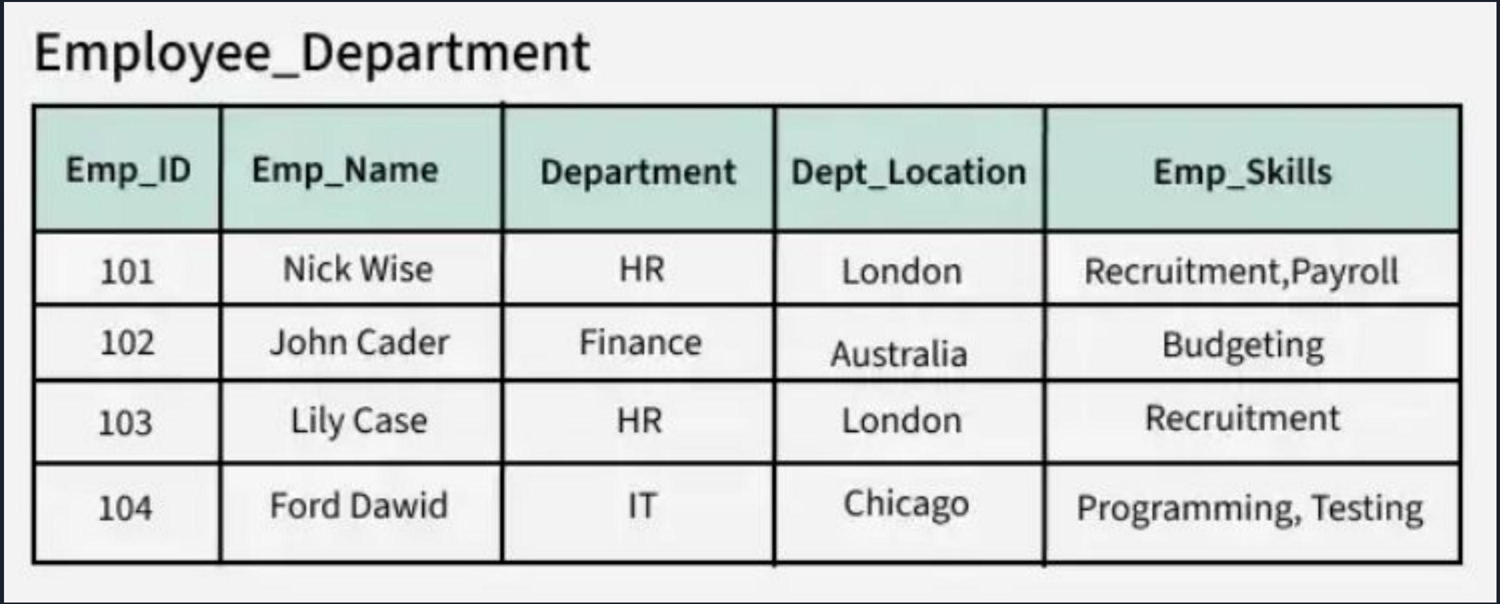
- **Insertion Anomalies**: Insertion anomalies occur when it is not possible to insert data into a database because the required fields are missing or because the data is incomplete. For example, if a database requires that every record has a primary key, but no value is provided for a particular record, it cannot be inserted into the database.

- **Deletion anomalies:** Deletion anomalies occur when deleting a record from a database and can result in the unintentional loss of data. For example, if a database contains information about customers and orders, deleting a customer record may also delete all the orders associated with that customer.

- **Updation anomalies:** Updation anomalies occur when modifying data in a database and can result in inconsistencies or errors. For example, if a database contains information about employees and their salaries, updating an employee’s salary in one record but not in all related records could lead to incorrect calculations and reporting.

Database Normalization

Example - Before Normalization



Employee_Department

Emp_ID	Emp_Name	Department	Dept_Location	Emp_Skills
101	Nick Wise	HR	London	Recruitment,Payroll
102	John Cader	Finance	Australia	Budgeting
103	Lily Case	HR	London	Recruitment
104	Ford Dawid	IT	Chicago	Programming, Testing

Database Normalization

Problems in the Employee_Department Relation

Insertion Anomaly:

If a new department is created but no employee is assigned to it yet, we cannot store its location because we need an employee record to insert.

Update Anomaly:

If the location of the HR department changes, we must update it in multiple rows (for both Nick Wise and Lily Case). If one row is missed, the data becomes inconsistent.

Deletion Anomaly:

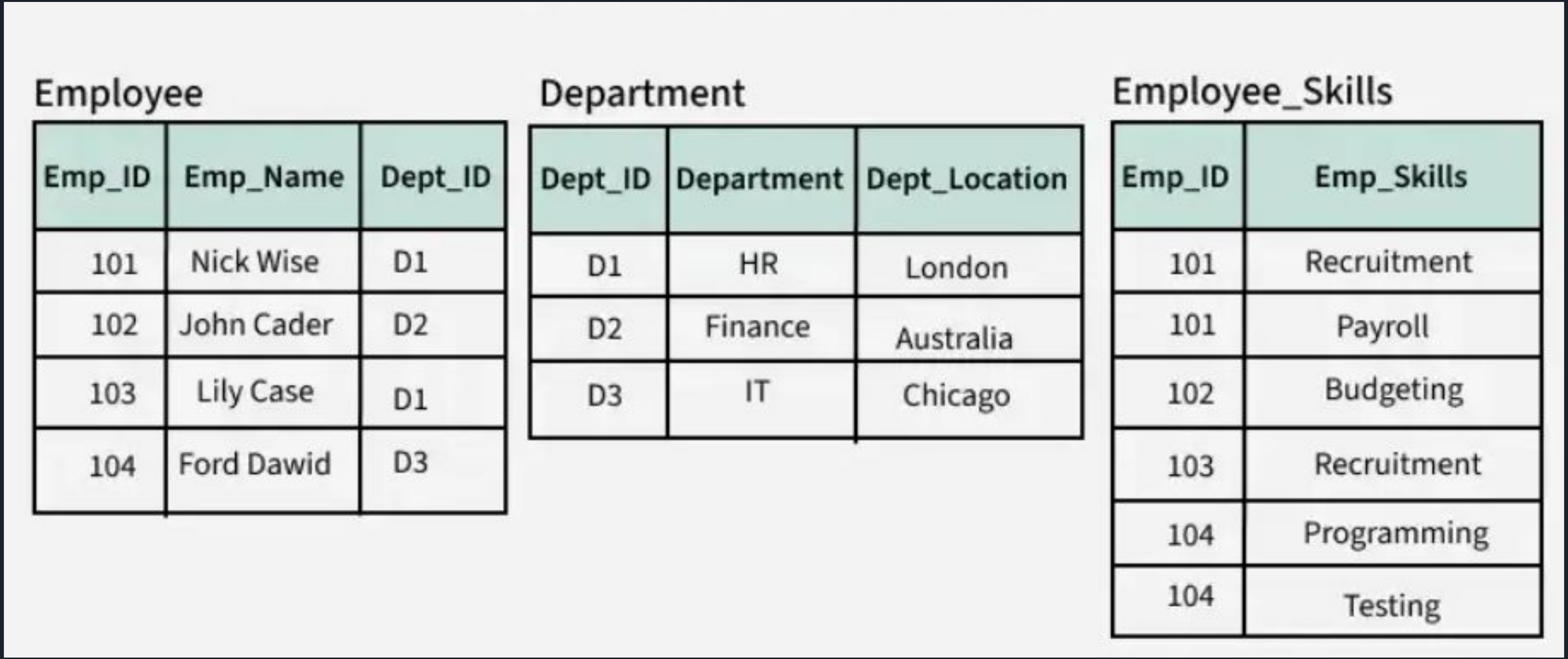
If all employees in the IT department leave, we lose the department information, including its location.

Data Redundancy:

The department location is repeated for every employee in the same department.

Database Normalization

Example - After Normalization



Employee

Emp_ID	Emp_Name	Dept_ID
101	Nick Wise	D1
102	John Cader	D2
103	Lily Case	D1
104	Ford Dawid	D3

Department

Dept_ID	Department	Dept_Location
D1	HR	London
D2	Finance	Australia
D3	IT	Chicago

Employee_Skills

Emp_ID	Emp_Skills
101	Recruitment
101	Payroll
102	Budgeting
103	Recruitment
104	Programming
104	Testing

Database Normalization

What is 1NF 2NF and 3NF?

1NF, 2NF, and 3NF are the first three types of database normalization. They stand for first normal form, second normal form, and third normal form, respectively.

There are also 4NF (fourth normal form) and 5NF (fifth normal form). There's even 6NF (sixth normal form), but the commonest normal form you'll see out there is 3NF (third normal form).

All the types of database normalization are cumulative – meaning each one builds on top of those beneath it. So all the concepts in 1NF also carry over to 2NF, and so on.

The First Normal Form – 1NF

For a table to be in the first normal form, it must meet the following criteria:

- a single cell must not hold more than one value (atomicity)
- there must be a primary key for identification
- no duplicated rows or columns
- each column must have only one value for each row in the table

Database Normalization

The Second Normal Form – 2NF

The 1NF only eliminates repeating groups, not redundancy. That’s why there is 2NF.

A table is said to be in 2NF if it meets the following criteria:

- It’s already in 1NF
- Has no partial dependency. That is, all non-key attributes are fully dependent on a primary key.

The Third Normal Form – 3NF

When a table is in 2NF, it eliminates repeating groups and redundancy, but it does not eliminate transitive partial dependency.

This means a non-prime attribute (an attribute that is not part of the candidate’s key) is dependent on another non-prime attribute. This is what the third normal form (3NF) eliminates.

So, for a table to be in 3NF, it must:

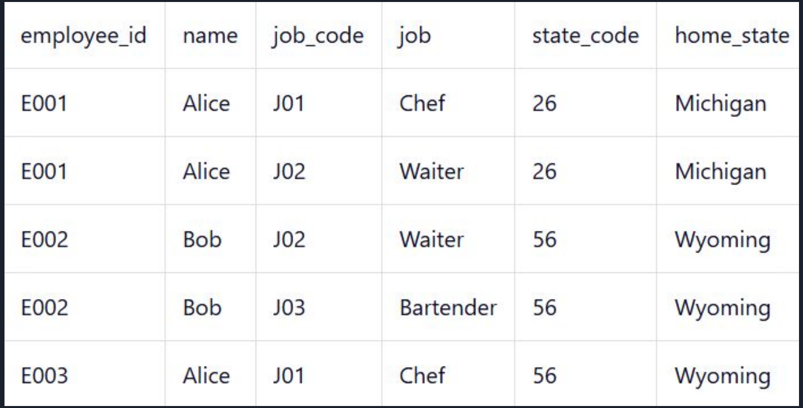
- Be in 2NF
- Have no transitive partial dependency.

Database Normalization

Examples of 1NF, 2NF, and 3NF

Database normalization is quite technical, but we will illustrate each of the normal forms with examples.

Imagine we're building a restaurant management application. That application needs to store data about the company's employees and it starts out by creating the following table of employees:



employee_id	name	job_code	job	state_code	home_state
E001	Alice	J01	Chef	26	Michigan
E001	Alice	J02	Waiter	26	Michigan
E002	Bob	J02	Waiter	56	Wyoming
E002	Bob	J03	Bartender	56	Wyoming
E003	Alice	J01	Chef	56	Wyoming

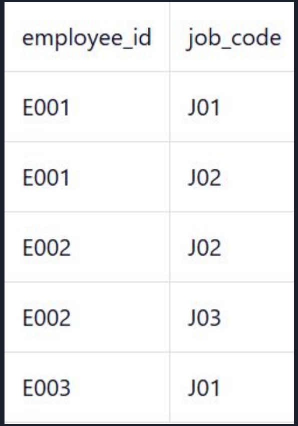
All the entries are atomic and there is a composite primary key (employee_id, job_code) so the table is in the **first normal form (1NF)**.

But even if you only know someone's **employee_id**, then you can determine their **name**, **home_state**, and **state_code** (because they should be the same person). This means **name**, **home_state**, and **state_code** are dependent on **employee_id** (a part of primary composite key). So, the table is not in **2NF**. We should separate them to a different table to make it **2NF**.

Database Normalization

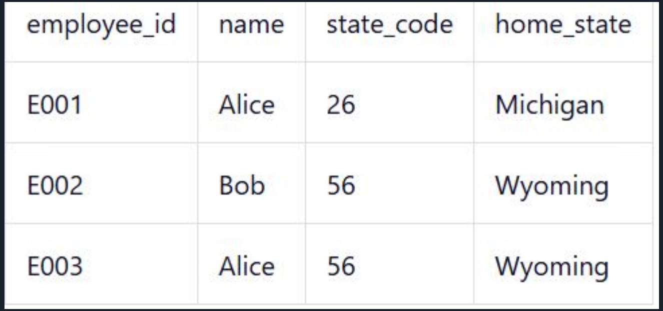
Example of Second Normal Form (2NF)

employee_roles Table



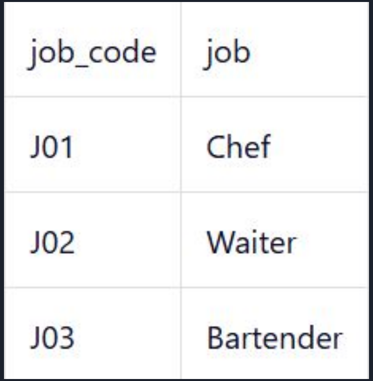
employee_id	job_code
E001	J01
E001	J02
E002	J02
E002	J03
E003	J01

employees Table



employee_id	name	state_code	home_state
E001	Alice	26	Michigan
E002	Bob	56	Wyoming
E003	Alice	56	Wyoming

jobs Table



job_code	job
J01	Chef
J02	Waiter
J03	Bartender

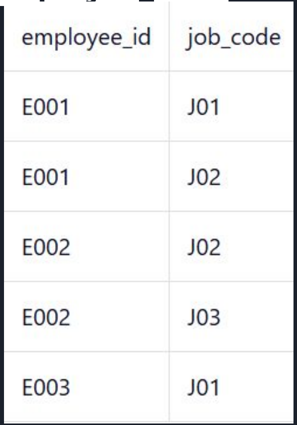
home_state is now dependent on **state_code**. So, if you know the **state_code**, then you can find the **home_state** value.

To take this a step further, we should separate them again to a different table to make it **3NF**.

Database Normalization

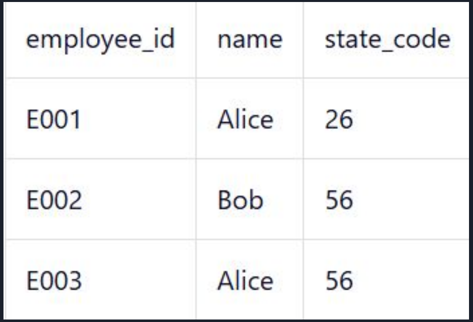
Example of Third Normal Form (3NF)

employee_roles Table



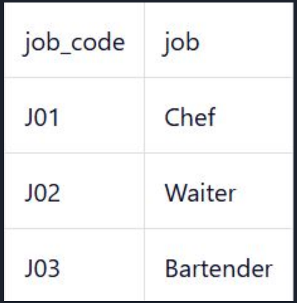
employee_id	job_code
E001	J01
E001	J02
E002	J02
E002	J03
E003	J01

employees Table



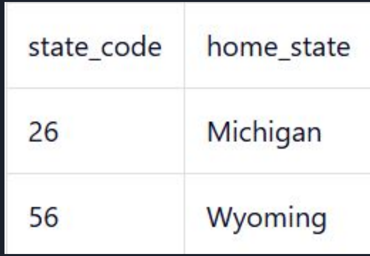
employee_id	name	state_code
E001	Alice	26
E002	Bob	56
E003	Alice	56

jobs Table



job_code	job
J01	Chef
J02	Waiter
J03	Bartender

states Table



state_code	home_state
26	Michigan
56	Wyoming

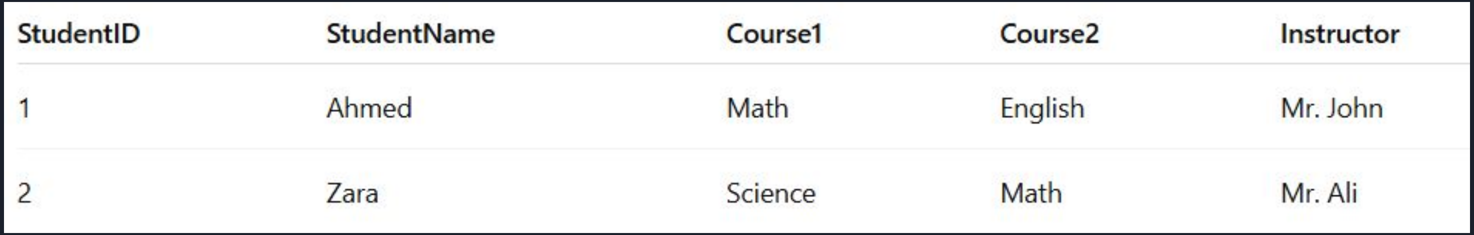
Now our database is in **3NF**.

Database Normalization

Practical Live Class Exercise:

Exercise 1: Student Enrollment

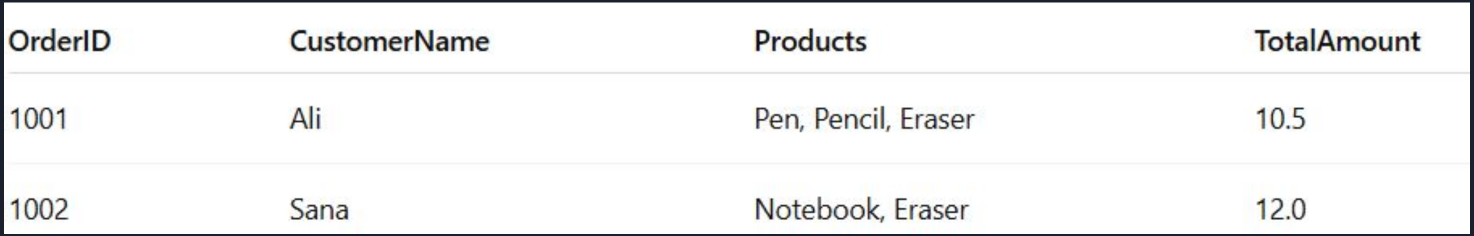
🔓 **Unnormalized**:



StudentID	StudentName	Course1	Course2	Instructor
1	Ahmed	Math	English	Mr. John
2	Zara	Science	Math	Mr. Ali

Exercise 2: Orders Table

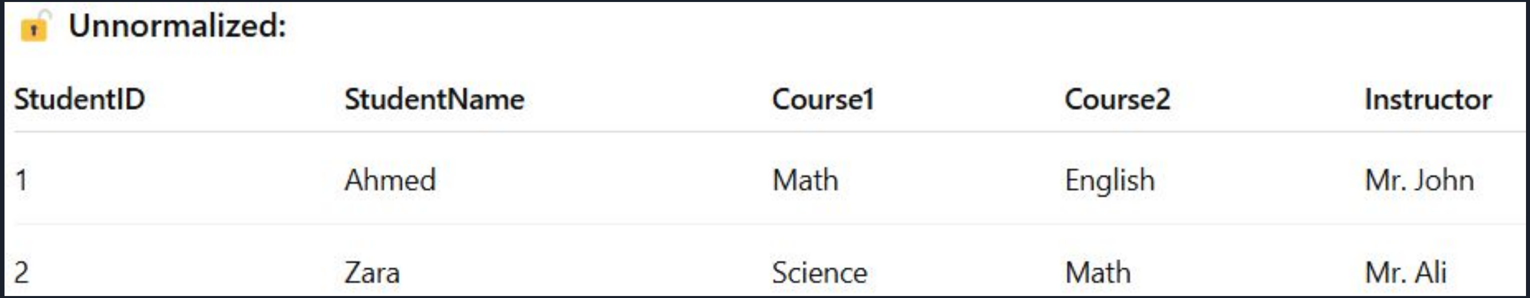
🔓 **Unnormalized**:



OrderID	CustomerName	Products	TotalAmount
1001	Ali	Pen, Pencil, Eraser	10.5
1002	Sana	Notebook, Eraser	12.0

Database Normalization

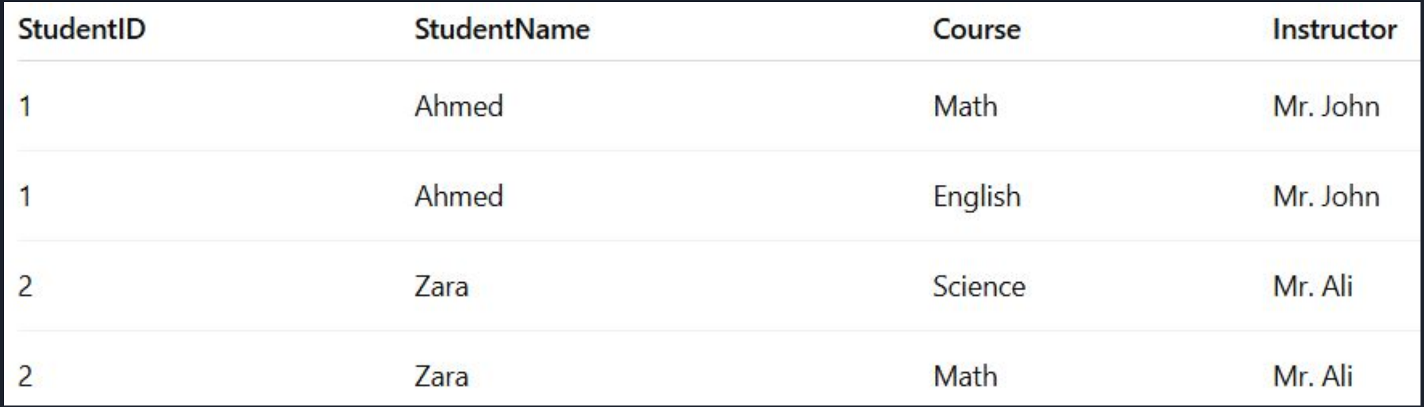
Practical Live Class Exercise# 01 Solution for Student Enrollment:



🔓 Unnormalized:

StudentID	StudentName	Course1	Course2	Instructor
1	Ahmed	Math	English	Mr. John
2	Zara	Science	Math	Mr. Ali

1NF – Flatten Course Columns:

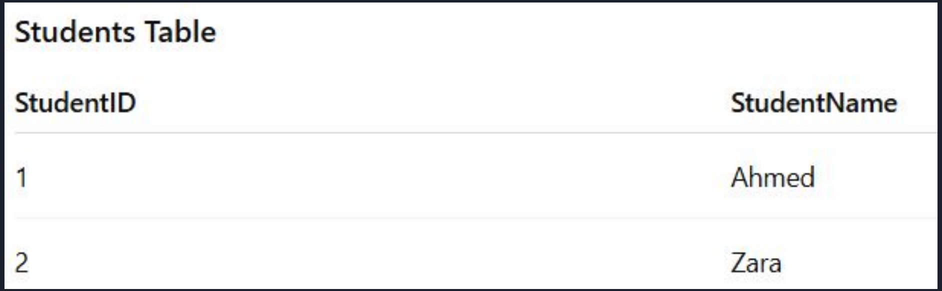


StudentID	StudentName	Course	Instructor
1	Ahmed	Math	Mr. John
1	Ahmed	English	Mr. John
2	Zara	Science	Mr. Ali
2	Zara	Math	Mr. Ali

Database Normalization

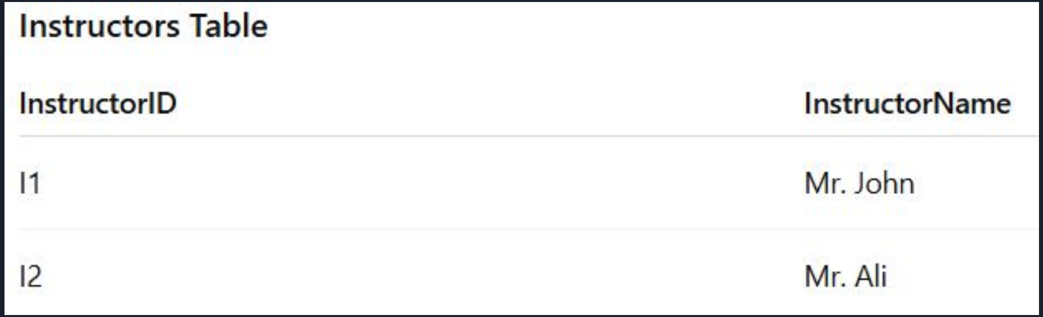
Practical Live Class Exercise# 01 Solution for Student Enrollment:

2NF – Remove Partial Dependencies:



Students Table

StudentID	StudentName
1	Ahmed
2	Zara



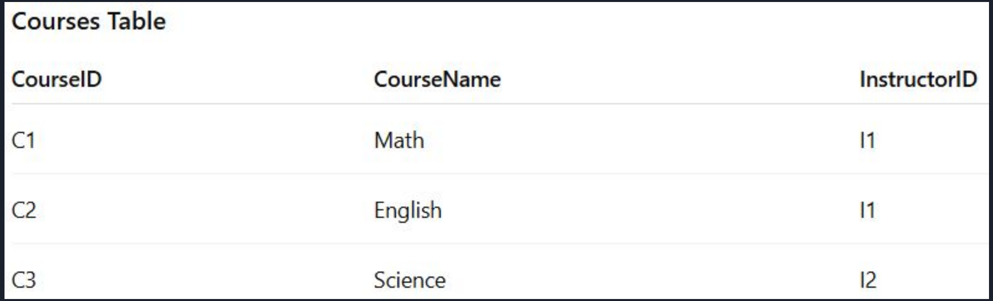
Instructors Table

InstructorID	InstructorName
I1	Mr. John
I2	Mr. Ali

Database Normalization

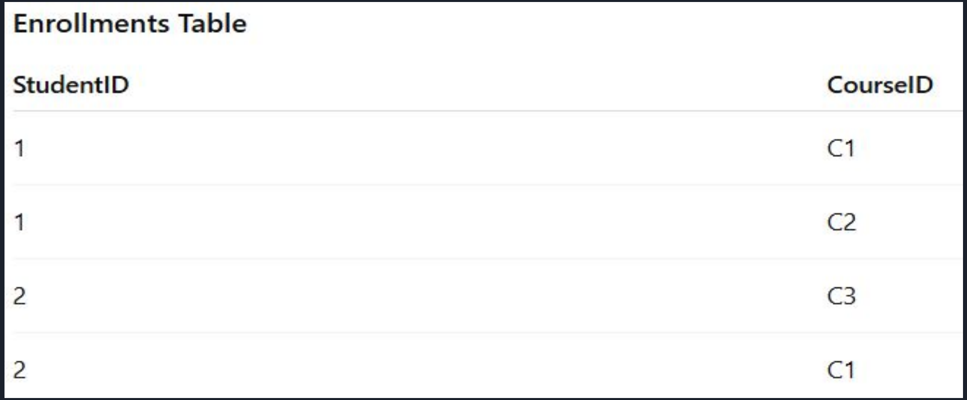
Practical Live Class Exercise# 01 Solution for Student Enrollment:

2NF – Remove Partial Dependencies:



Courses Table

CourseID	CourseName	InstructorID
C1	Math	I1
C2	English	I1
C3	Science	I2



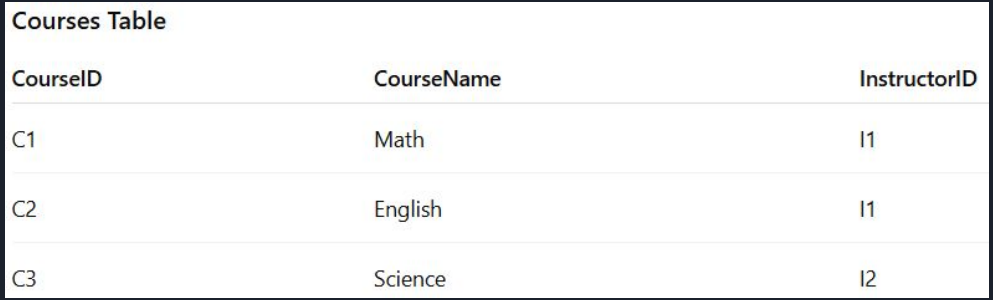
Enrollments Table

StudentID	CourseID
1	C1
1	C2
2	C3
2	C1

Database Normalization

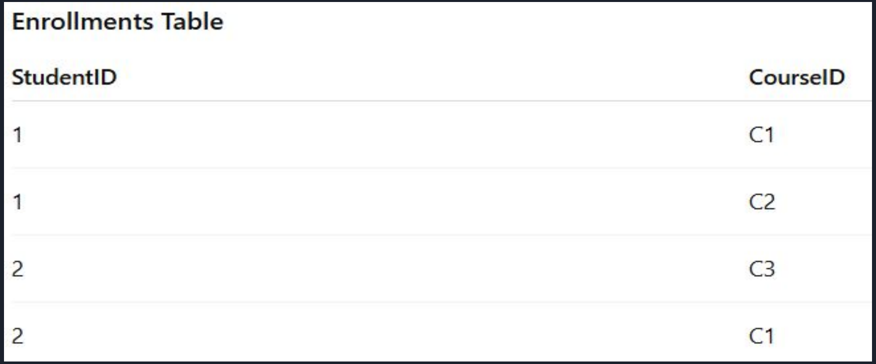
Practical Live Class Exercise# 01 Solution for Student Enrollment:

2NF – Remove Partial Dependencies:



Courses Table

CourseID	CourseName	InstructorID
C1	Math	I1
C2	English	I1
C3	Science	I2



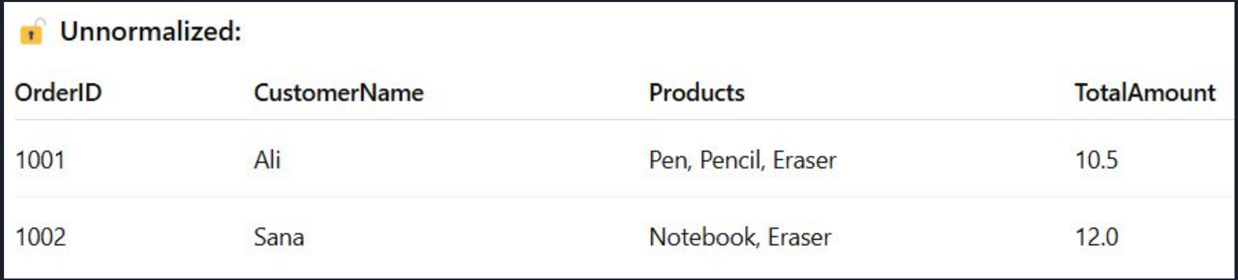
Enrollments Table

StudentID	CourseID
1	C1
1	C2
2	C3
2	C1

✅ **3NF – All dependencies resolved** ✅

Database Normalization

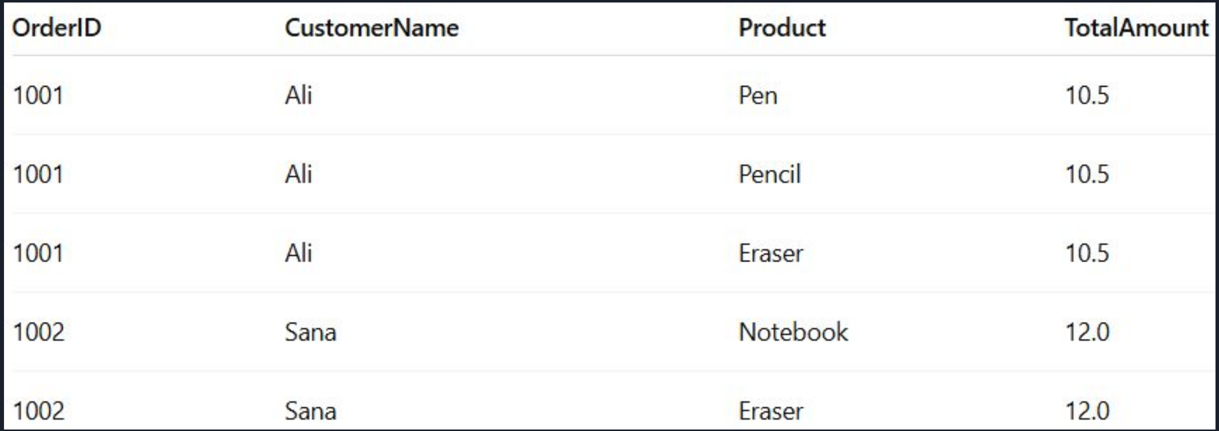
Practical Live Class Exercise# 02 Solution for Order Table:



🔓 **Unnormalized:**

OrderID	CustomerName	Products	TotalAmount
1001	Ali	Pen, Pencil, Eraser	10.5
1002	Sana	Notebook, Eraser	12.0

1NF – Atomic Products:

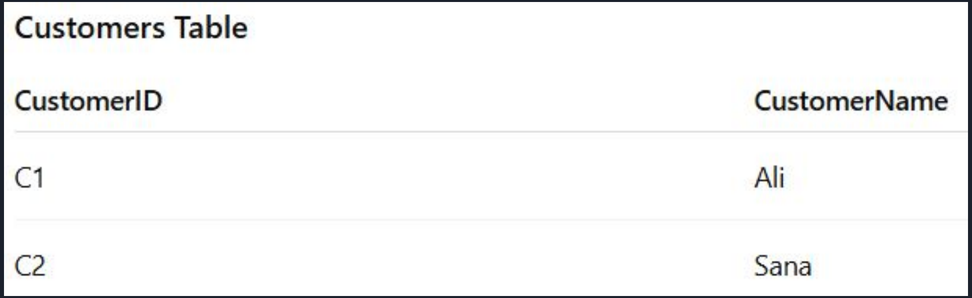


OrderID	CustomerName	Product	TotalAmount
1001	Ali	Pen	10.5
1001	Ali	Pencil	10.5
1001	Ali	Eraser	10.5
1002	Sana	Notebook	12.0
1002	Sana	Eraser	12.0

Database Normalization

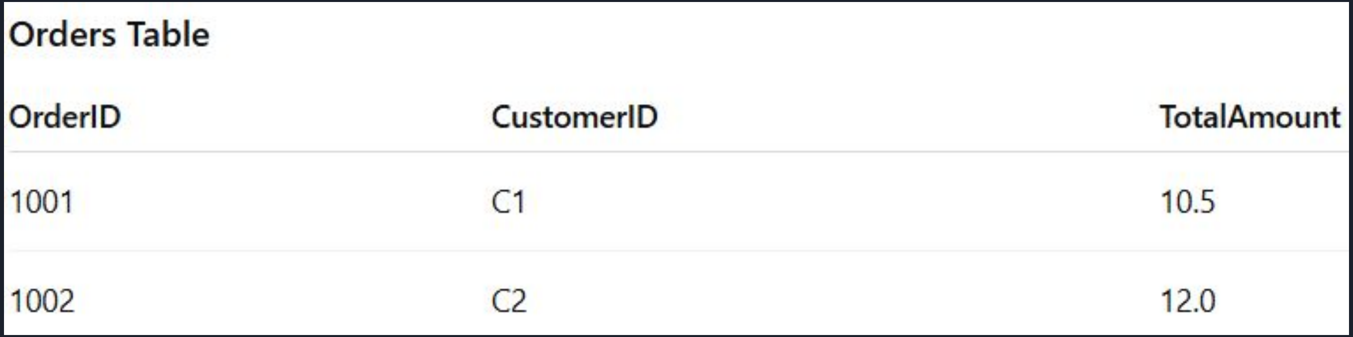
Practical Live Class Exercise# 02 Solution for Order Table:

2NF – Remove Partial Dependency (Separate Customers):



Customers Table

CustomerID	CustomerName
C1	Ali
C2	Sana



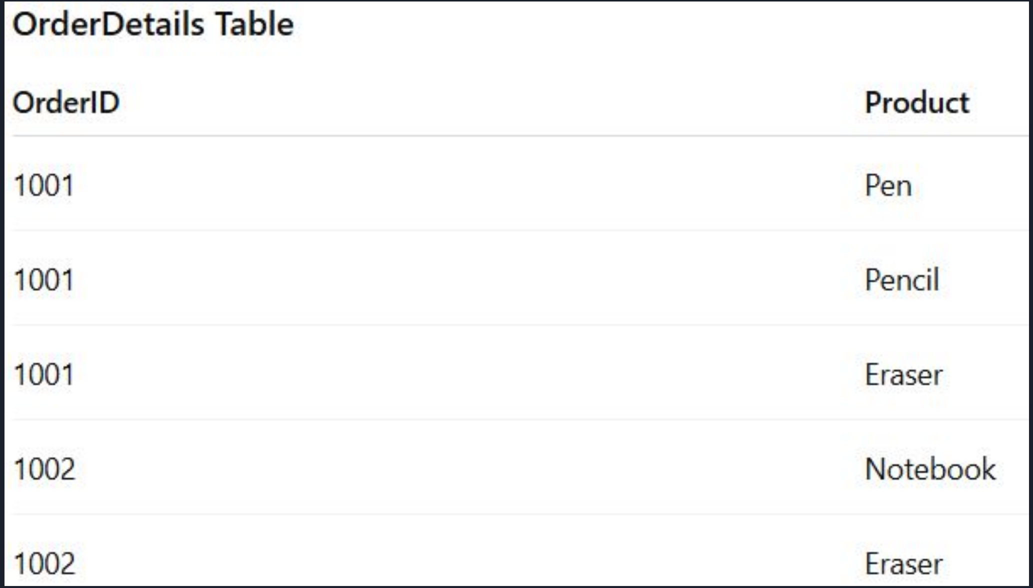
Orders Table

OrderID	CustomerID	TotalAmount
1001	C1	10.5
1002	C2	12.0

Database Normalization

Practical Live Class Exercise# 02 Solution for Order Table:

2NF – Remove Partial Dependency (Separate Customers):



OrderDetails Table

OrderID	Product
1001	Pen
1001	Pencil
1001	Eraser
1002	Notebook
1002	Eraser

3NF – TotalAmount is Derived (Move to OrderDetails or Remove):

- Keep TotalAmount as a calculated field or ensure it's based on detailed line items.

Microservices Architecture

Microservices Architecture

Introduction to Microservices

What are Microservices?

A software design pattern where the application is divided into smaller, independent services.

Each service is responsible for a specific functionality.

Communicates with other services via lightweight protocols (HTTP, Message Brokers).

Key Analogy:

"Think of microservices like departments in a company — HR, Finance, Sales — each has its own job but works together."

Microservices Architecture

Key Characteristics

Characteristics of Microservices

- Independent deployment and scalability.
- Single Responsibility per service.
- Technology agnostic (can use different stacks).
- Decentralized data management.
- Failure isolation (fault in one service doesn’t crash entire app).

Microservices Architecture

Monolith vs Microservices

Comparing Architecture Styles

Feature	Monolith	Microservices
Deployment	Single unit	Multiple independent services
Scalability	Difficult	Easy (scale per service)
Maintenance	Hard as app grows	Easier (smaller, focused services)
Technology	One stack only	Can vary between services
Team Workload	Tight coupling	Parallel work possible

Microservices Architecture

Example System Breakdown

Real-world Example: E-Commerce App

Breakdown:

- User Service – Authentication & user profiles.
- Product Service – Product management.
- Order Service – Order placement and tracking.
- Notification Service – Email/SMS notifications.

Microservices Architecture

Benefits of Microservices

Why Use Microservices?

Breakdown:

- Better scalability & performance
- Faster development and deployment
- Easier debugging and fault isolation
- Technology flexibility
- Reusability across teams/projects

Microservices Architecture

Communication in Microservices

How Microservices Talk?

- **Synchronous –** HTTP/REST API (request-response)
- **Asynchronous –** Message Brokers like Redis, RabbitMQ, Kafka (event/message-based)

How Microservices Talk?

- Built-in support for microservice architecture.
- **@nestjs/microservices** module enables communication.
- Transport layers supported:
  - ➔ Redis
  - ➔ RabbitMQ
  - ➔ Kafka
  - ➔ MQTT
  - ➔ gRPC

Microservices Architecture

Message Flow Example

Example: Order Service calls Product Service

Explanation:

- User places an order.
- Order Service needs product info.
- It sends a message (e.g., via Redis, Rabbit MQ).
- Product Service receives and replies with product data.

Microservices Architecture

Deployment Structure

Microservice Deployment

- Each service is its own app.
- Can be containerized using Docker.
- Use orchestration tools like Kubernetes or Docker Compose.
- Shared environment variables and secrets per service.

Best Practices for Microservices

- Keep services small and focused.
- Use shared libraries (common folder) for interfaces.
- Implement circuit breakers (e.g., with retry logic).
- Secure each endpoint/service.
- Use centralized logging & monitoring **tools (e.g., ELK, Prometheus).**

Microservices Architecture

When to Use Microservices

Is Microservices Right for You?

Use When:

- Your app is growing rapidly.
- Multiple teams are working in parallel.
- You need high scalability or fault tolerance.

Avoid When:

- For small or simple applications.
- When team lacks experience in distributed systems.

Version Controlling - Git

Version Controlling - Git

What is Version Controlling?

Version control is the management of changes to documents, files, or any other type of data. In software development, it is essential for managing and tracking changes to the codebase, ensuring code quality, reducing errors, and improving collaboration among team members.

Without version control, managing and tracking code changes would be a difficult and error-prone task. Version control tools like Git provide a way to manage code changes, keep track of versions, and collaborate with team members. This makes it a critical component of modern software development, used by virtually all software development teams.

What is Git?

Git is a version control system that you download onto your computer. It is essential that you use Git if you want to collaborate with other developers on a coding project or work on your own project.

In order to check if you already have Git installed on your computer you can type the command git --version in the terminal.

Version Controlling - Git

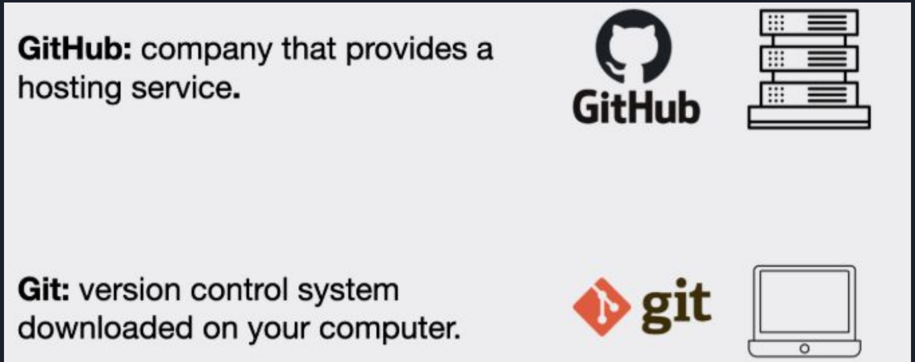
What to Use Git?

Git works on your computer, but you also use it with online services like **GitHub**, **GitLab**, or **Bitbucket** to share your work with others. These are called remote repositories.

What is GitHub?

GitHub is a product that allows you to host your Git projects on a remote server somewhere (or in other words, in the cloud).

It's important to remember that GitHub is not Git. GitHub is just a hosting service. There are other companies who offer hosting services that do the same thing as GitHub, such as Bitbucket and GitLab.


GitHub: company that provides a hosting service.
GitHub
Git: version control system downloaded on your computer.
git

Version Controlling - Git

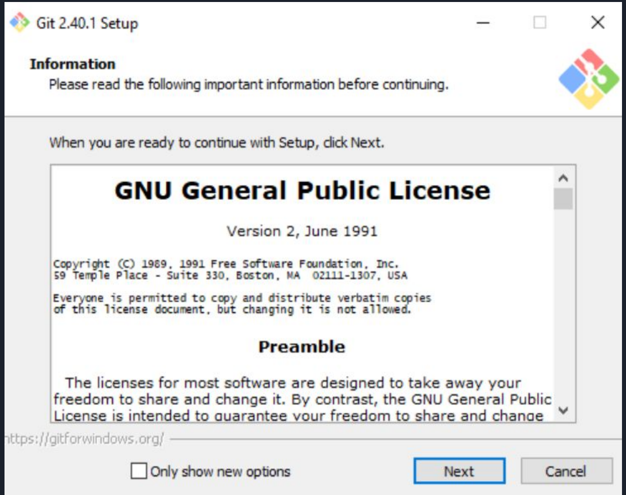
How to Get Started with Git

How to Install Git

Git is a popular version control system used by software developers to manage and track changes to code. Here are the steps to install Git:

Step 1: Download Git

To get started, go to the official Git website (https://git-scm.com/downloads) and download the appropriate installer for your operating system.



Git 2.40.1 Setup

Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

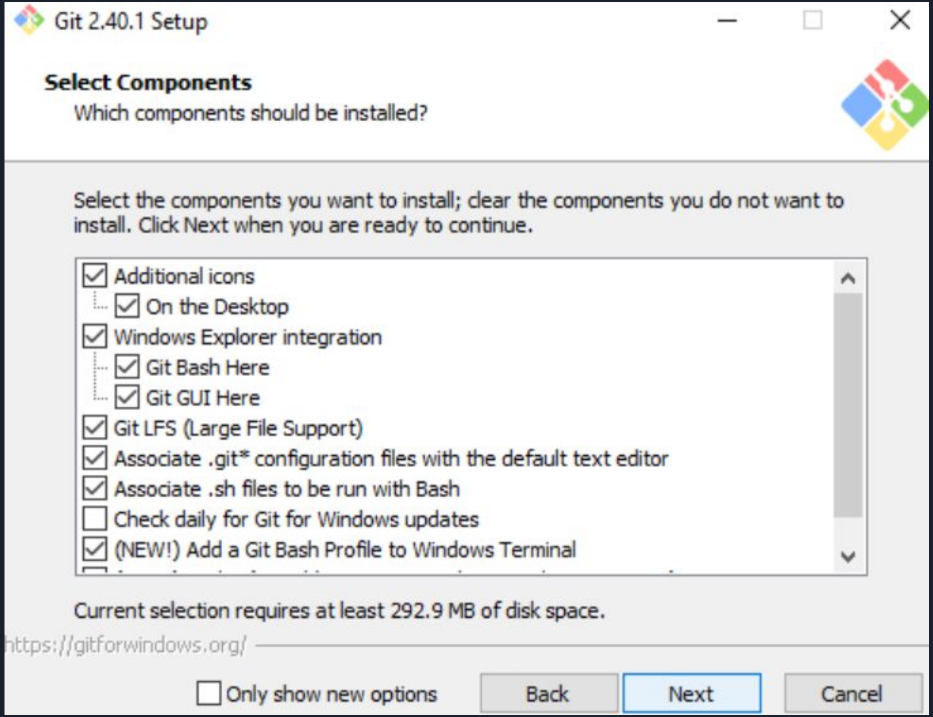
Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

Version Controlling - Git

Step 2: Run the Installer

Once the download is complete, run the installer and follow the prompts. The installation process will vary depending on your operating system, but the installer should guide you through the process.



Git 2.40.1 Setup

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Additional icons
  - [✓] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Check daily for Git for Windows updates
- [✓] (NEW!) Add a Git Bash Profile to Windows Terminal

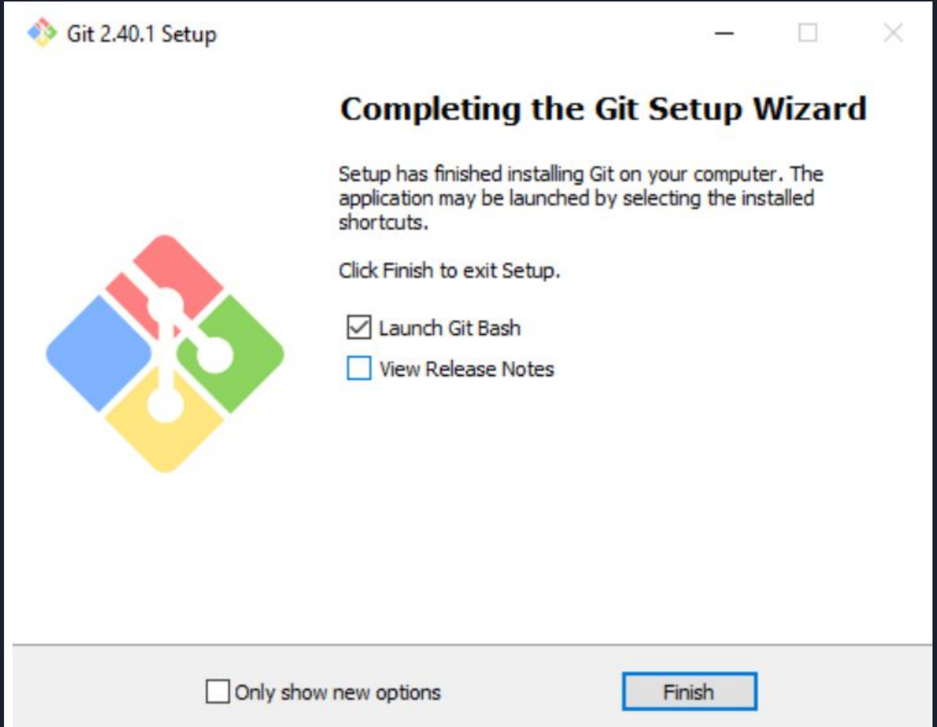
Current selection requires at least 292.9 MB of disk space.

https://gitforwindows.org/

Version Controlling - Git

Step 3: Select Installation Options

During the installation process, you'll be prompted to select various options. For most users, the default options will be sufficient, but you can choose to customize your installation if desired.



Git 2.40.1 Setup

Completing the Git Setup Wizard

Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

- [✓] Launch Git Bash
- [ ] View Release Notes

[ ] Only show new options

Version Controlling - Git

Step 4: Complete the Installation

Once you've selected your installation options, the installer will install Git on your computer. This may take a few minutes depending on your system.

Step 5: Verify the Installation

After the installation is complete, you can verify that Git has been installed correctly by opening a command prompt or terminal window and running the command **git --version**. This should display the current version of Git that is installed on your system, something like git version 2.40.1.windows.1.

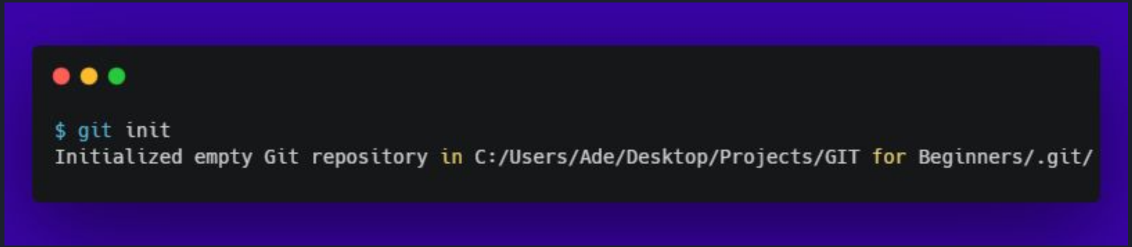
Version Controlling - Git

How to Set Up a New Git Repository

Git repositories are used to manage and track changes to code. Setting up a new Git repository is a simple process that just takes a few steps.

Step 1: Create a New Directory

The first step in setting up a new Git repository is to create a new directory on your computer. This directory will serve as the root directory of your new repository.

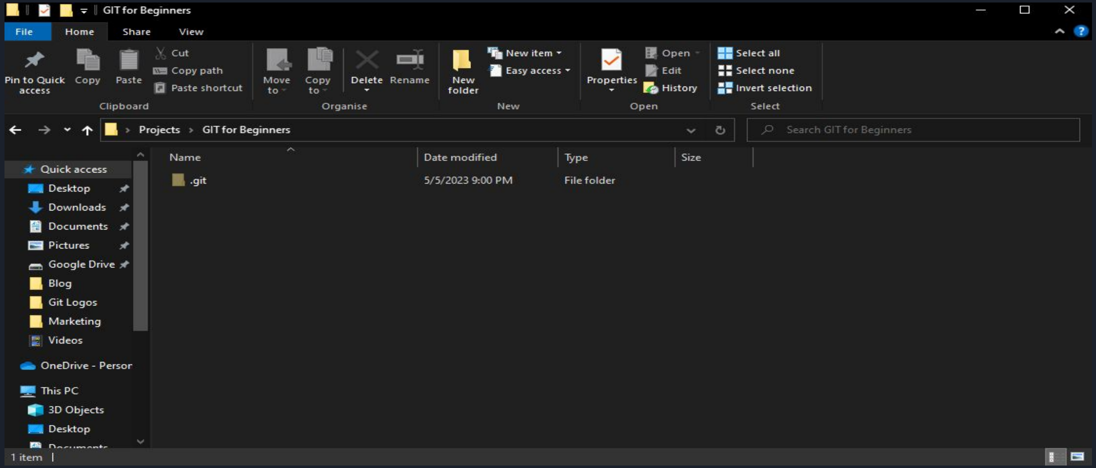


```
$ git init
Initialized empty Git repository in C:/Users/Ade/Desktop/Projects/GIT for Beginners/.git/
```

Version Controlling - Git

Step 2: Initialize Git

.git file



Name	Date modified	Type	Size
.git	5/5/2023 9:00 PM	File folder	

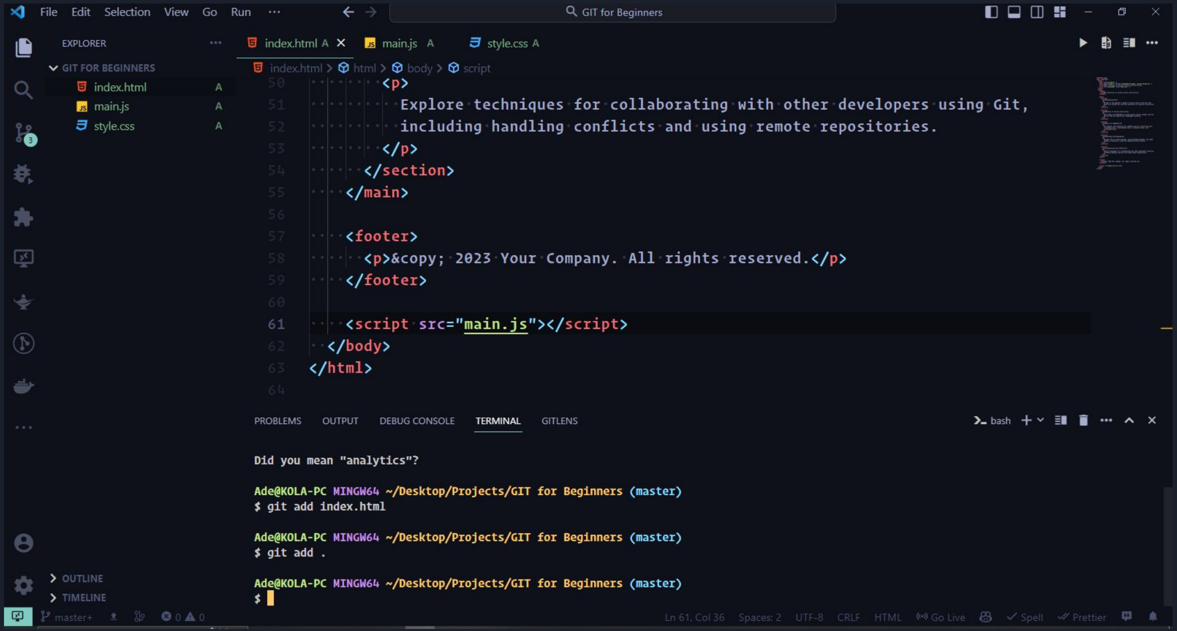
Once you have Git installed, the next step is to initialize a new repository. To do this, navigate to the root directory of your project in the command line or terminal and run the command git init. This will create a new .git directory in your project's root directory, which is where Git stores all of its metadata and version control information.

Once you've initialized the repository, you can start tracking changes to your project and making commits. It's important to note that you only need to initialize a repository once for each project, so you won't need to repeat this step for subsequent commits or changes.

Version Controlling - Git

Step 3: Add Files

After initializing your Git repository, the next step is to start tracking changes to your project by adding files to the staging area.



```
        <p>
          Explore techniques for collaborating with other developers using Git,
          including handling conflicts and using remote repositories.
        </p>
      </section>
    </main>

    <footer>
      <p>&copy; 2023 Your Company. All rights reserved.</p>
    </footer>

    <script src="main.js"></script>
  </body>
</html>
```

```
Did you mean "analytics"?

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git add index.html

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git add .

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$
```

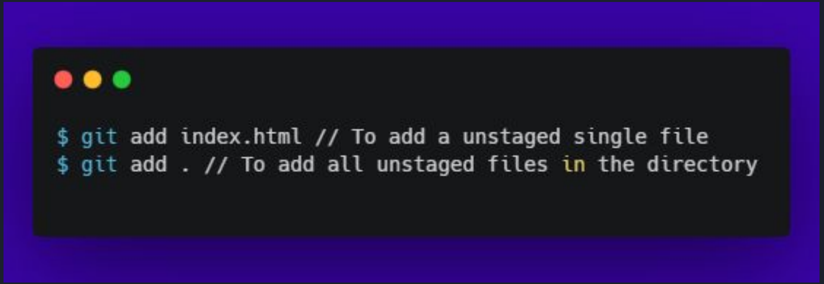
Version Controlling - Git

Step 3: Add Files (cont.)

Staging files

To do this, use the command git add <filename> to add each file to the staging area. You can also use the command git add . to add all of the files in the current directory and its subdirectories to the staging area at once.

Also, as you see in the graphic above, there's a label of (master) after the ~/Desktop/Projects/GIT for Beginners. The (master) signifies the current branch for the project. This is the default branch for all projects that initialize Git.



```
$ git add index.html // To add a unstaged single file
$ git add . // To add all unstaged files in the directory
```

Once a file is added to the staging area, it's ready to be committed to the repository. It's important to note that adding files to the staging area doesn't actually commit them – it just prepares them for the commit. You can continue to add and modify files as needed before making a commit

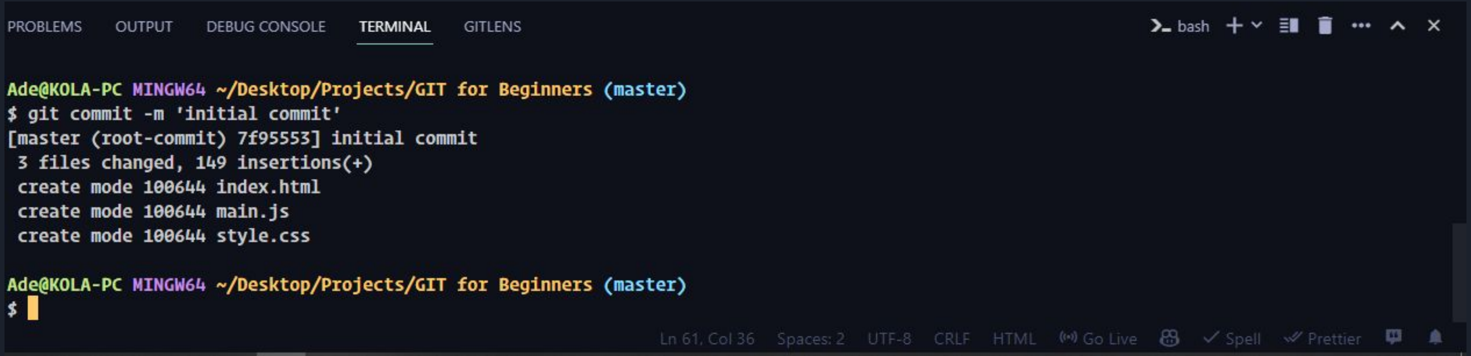
Version Controlling - Git

Step 4: Commit Changes

After adding files to the staging area, the next step is to commit the changes to your repository using the **git commit** command.

When committing changes, it's important to provide a clear and descriptive message that explains what changes you made in the commit. This message will be used to track the changes in the repository's history and will help other contributors understand the changes you made.

To commit changes, use the command **git commit -m 'commit message'** , replacing '**commit message**' with a clear and descriptive message that explains the changes made in the commit. Once committed, the changes will be saved to the repository's history and can be tracked, reverted, or merged with other branches as needed.



```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git commit -m 'initial commit'
[master (root-commit) 7f95553] initial commit
 3 files changed, 149 insertions(+)
 create mode 100644 index.html
 create mode 100644 main.js
 create mode 100644 style.css

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ 
```

Git Commit

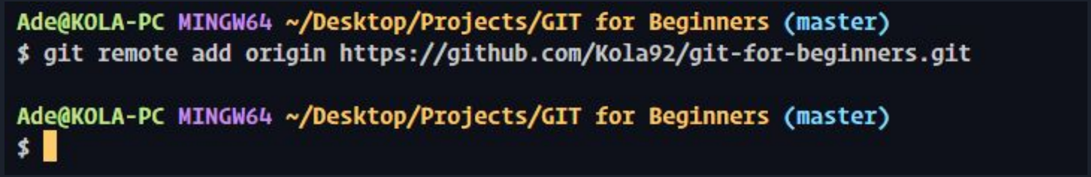
Version Controlling - Git

Step 5: Connect to a Remote Repository

To share your changes with other developers or collaborate on a project, you can connect your local repository to a remote repository using Git.

A remote repository is a copy of your repository that is hosted on a server, such as GitHub, GitLab, or BitBucket, and allows multiple contributors to work on the same codebase.

To connect to a remote repository, use the **git remote add** command followed by the URL of the remote repository.



```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git remote add origin https://github.com/Kola92/git-for-beginners.git

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ 
```

For example, to connect to a GitHub repository, you would use the command git remote add origin <repository URL>. Before you can even connect to the remote repository, you need to create it..

Now we will follow the step# 06 to create a repository on Git Hub

Version Controlling - Git

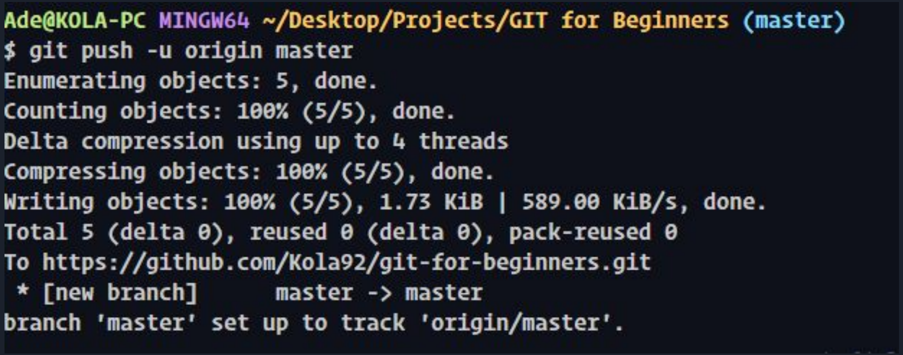
Step 6: Create Repository on GitHub

Follow link below to create your repository on Git Hub:

https://scribehow.com/viewer/How_to_Create_a_New_Repository_on_GitHub__OGEKiV2UT42dB8Kre8KfCg

Step 7: Once Connected Push Your Changes

Once connected, you can push your changes to the remote repository using this **git push -u <default branch>** command. This command is often used when pushing changes for the first time to establish the relationship between the local branch and the remote branch.

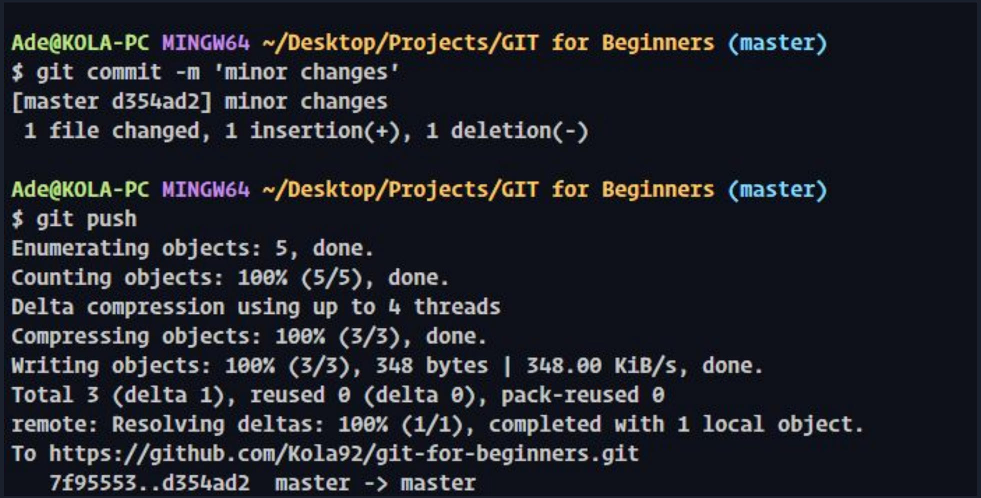


```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.73 KiB | 589.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Kola92/git-for-beginners.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Version Controlling - Git

Git Subsequent Remote Push

However, for subsequent push changes, use the command git push without specifying any additional arguments. Git will attempt to push changes from the current local branch on your local machine (computer) to the corresponding branch on the remote repository. It assumes that the local branch and the remote branch have the same name.

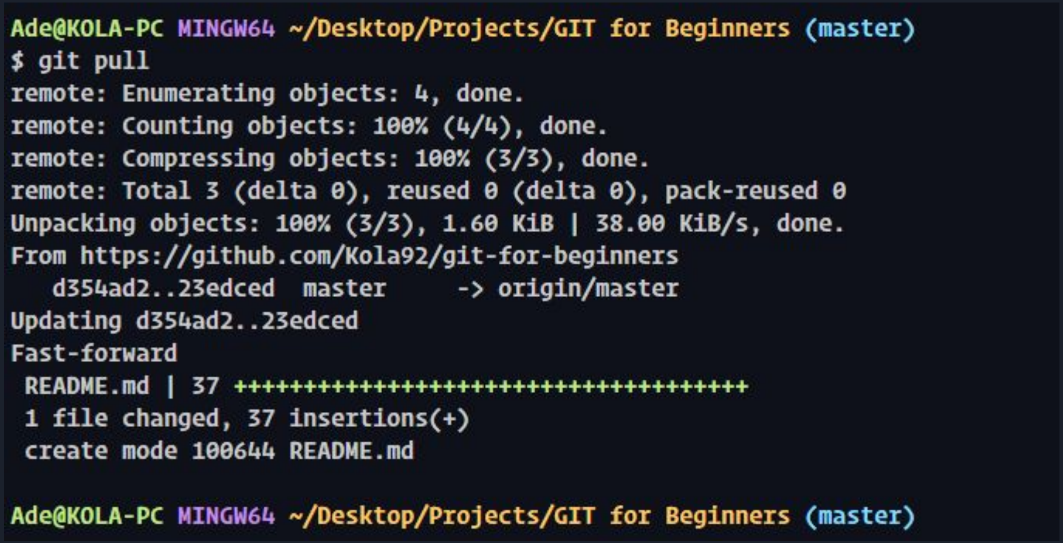


```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git commit -m 'minor changes'
[master d354ad2] minor changes
 1 file changed, 1 insertion(+), 1 deletion(-)

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 348 bytes | 348.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kola92/git-for-beginners.git
   7f95553..d354ad2  master -> master
```

Version Controlling - Git

Git Pull

The **git pull** command fetches the latest changes made by other contributors from a remote repository and automatically merges them into the current branch. By connecting to a remote repository, you can collaborate with other developers and contribute to open-source projects.



```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 1.60 KiB | 38.00 KiB/s, done.
From https://github.com/Kola92/git-for-beginners
   d354ad2..23edced  master     -> origin/master
Updating d354ad2..23edced
Fast-forward
 README.md | 37 +++++++++++++++++++++++++++++++++++++
 1 file changed, 37 insertions(+)
 create mode 100644 README.md

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
```

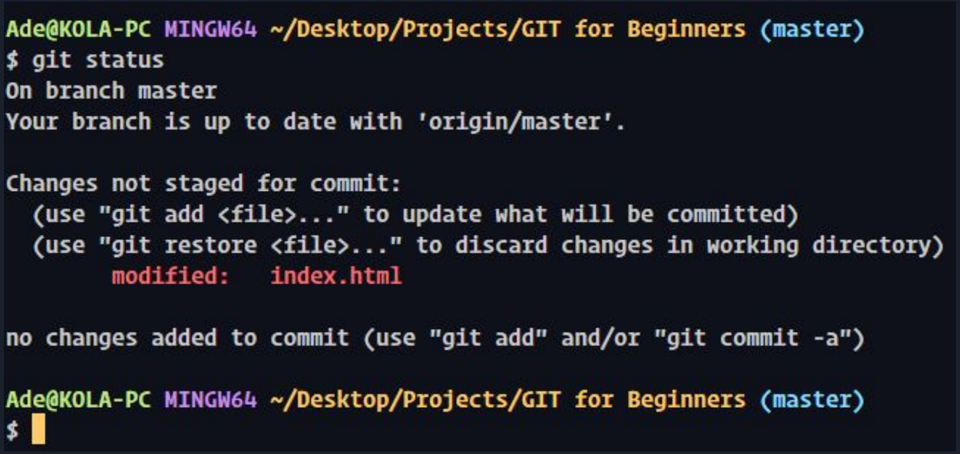
Version Controlling - Git

Basic Commands to Create and Commit Changes

Once you've set up a new Git repository and added some files to it, you'll need to commit changes to your repository. Here are the basic commands to create and commit changes in Git.

Step 1: Check the Status

Before committing changes, you should check the status of your repository to see what changes have been made. To do this, run the command git status in a terminal or command prompt window.

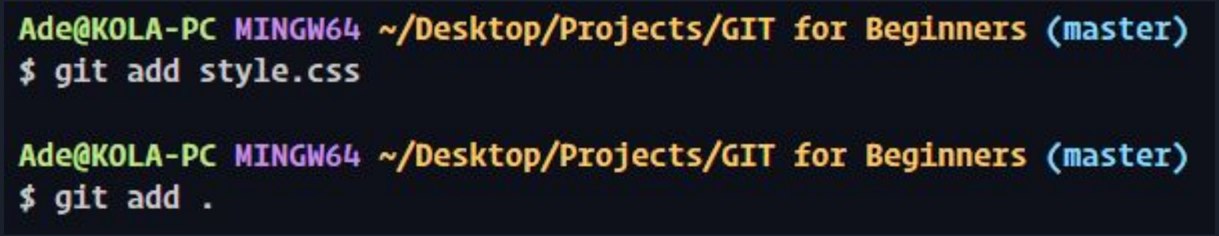


```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$
```

Version Controlling - Git

Step 2: Stage Changes

To commit changes, you'll need to stage them first using the **git add** command. This tells Git which files to include in the next commit. You can stage all changes by running the command **git add .** or stage specific changes by running the command **git add <filename>** .



```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git add style.css

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git add .
```

When you stage changes, Git takes a snapshot of the files at that moment in time. This snapshot includes all of the changes you've made since the last commit.

Staging changes allows you to carefully review your changes before committing them. You can stage changes in small chunks and commit them separately, or stage all changes and commit them together. This gives you more control over the changes you make to your codebase and helps you keep track of what changes have been made over time.

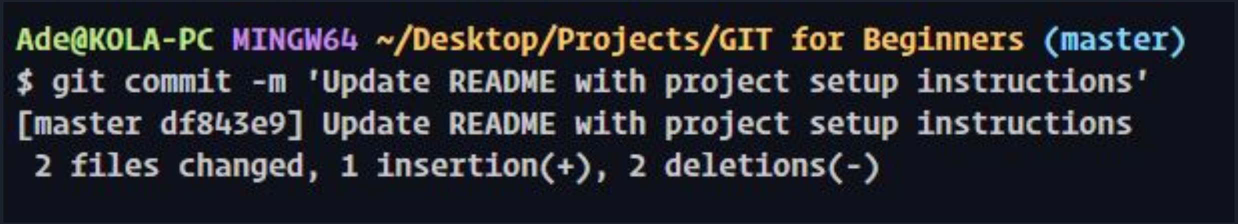
By staging changes in Git, you can ensure that your commits accurately reflect the changes you've made to your codebase.

Version Controlling - Git

Step 3: Commit Changes

Once you've staged your changes, you can commit them to your repository using the **git commit** command. This creates a new snapshot of your repository with the changes you made.

The commit is a snapshot of the changes made then, and it includes a reference to the previous commit in the branch's history. This allows developers to track the changes made to the code over time, collaborate with other developers, and roll back to previous versions of the code if necessary.



```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git commit -m 'Update README with project setup instructions'
[master df843e9] Update README with project setup instructions
 2 files changed, 1 insertion(+), 2 deletions(-)
```

You'll need to include a commit message to describe the changes you made using the -m flag. For example, git commit -m Added new featurethe "Added new feature“ part is what the commit is called.

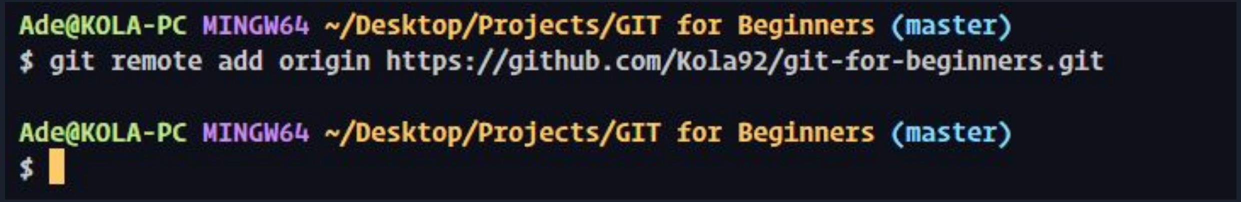
By including a clear and concise commit message like "Added new feature," other developers can quickly understand the purpose of the commit and what changes were made. This makes collaboration and code maintenance easier.

Version Controlling - Git

Step 4: Push Changes

If you’re working on a team or want to share your changes with others, you can push your changes to a remote repository using this **git push** command. This uploads your changes to a shared repository that others can access.

To push changes to a remote repository, you’ll first need to add a remote URL using the **git remote add** command. This tells Git where to push your changes.



```
Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ git remote add origin https://github.com/Kola92/git-for-beginners.git

Ade@KOLA-PC MINGW64 ~/Desktop/Projects/GIT for Beginners (master)
$ 
```

Interview Questions/Answers