COMPLETE DOCUMENTATION: React Portfolio Website Deployment

Version: 1.0
Last Updated: January 31, 2026
Project: React Portfolio Website
GitHub Repository: https://github.com/ShubhamCyberStack/portfolio-website
Live Website: http://65.2.181.108

TABLE OF CONTENTS

PART 1: UNDERSTANDING THE BASICS

PART 2: WINDOWS DEVELOPMENT SETUP

PART 3: AWS EC2 UBUNTU DEPLOYMENT

1. Prerequisites on AWS
2. Connecting to EC2
3. Installing Dependencies
4. Cloning from GitHub
5. Building for Production
6. Nginx Configuration
7. Deployment
8. Accessing Your Live Website

PART 4: TROUBLESHOOTING & MAINTENANCE

1. Common Errors and Solutions
2. How to Update Your Website
3. Best Practices

PART 1: UNDERSTANDING THE BASICS

1. WHAT IS NODE.JS?

Simple Explanation:

Node.js is like a **translator** that lets you run JavaScript code on your computer (not just in the browser).

Technical Explanation:

- **JavaScript** was originally designed to run only in web browsers (Chrome, Firefox, etc.)
- **Node.js** is a runtime environment that allows JavaScript to run on servers and computers
- Built on Chrome's V8 JavaScript engine
- Enables backend development, command-line tools, and build processes with JavaScript

Why Do We Need It?

1. **Run Development Server** - `npm start` runs a local web server
2. **Install Packages** - Downloads React and other libraries
3. **Build Production Files** - Converts React code to browser-ready code
4. **Run Scripts** - Automate tasks like testing and deployment

Real-World Analogy:

Think of JavaScript as **English**:

- **Browser** = You can only speak English in England (limited environment)
- **Node.js** = Now you can speak English anywhere in the world (any environment)

Installation:

- Download from: https://nodejs.org/
- Includes npm (Node Package Manager) automatically
- Version: LTS (Long Term Support) recommended

2. WHAT IS npm?

Simple Explanation:

npm is like the **Google Play Store for JavaScript** - it downloads and manages code libraries (packages).

Full Name:

Node **P**ackage **M**anager

What Does It Do?

Task	Command	Purpose
Install packages	`npm install`	Downloads all libraries your project needs
Add new package	`npm install react`	Adds a specific library
Run scripts	`npm start`	Runs the development server
Build project	`npm run build`	Creates production-ready files
Check version	`npm -v`	Shows npm version

Real-World Analogy:

Building a house:

- **npm** = Hardware store
- **Packages** = Bricks, cement, tools
- **package.json** = Shopping list of materials you need
- **node_modules** = Storage room with all the materials

Key Files:

package.json (Shopping List)

- Lists all packages your project needs
- Specifies versions
- Defines scripts (start, build, test)

node_modules/ (Storage Room)

- Folder containing all downloaded packages
- Contains thousands of files
- **Never edit files here manually**
- **Never push to GitHub** (too large, can be regenerated)

package-lock.json (Detailed Receipt)

- Records exact versions of every package
- Ensures everyone gets the same versions
- Auto-generated, don't edit manually

3. WHAT IS REACT?

Simple Explanation:

React is a **JavaScript library** for building user interfaces (UIs) - basically, the visual part of websites.

Created By: Facebook (Meta)

Key Concepts:

Components (Building Blocks)

Think of LEGO blocks - you create small, reusable pieces and combine them to build a complete website.

Example Component Structure:

- Header component (shows name and photo)
- About component (shows description)
- Skills component (shows skill cards)
- Footer component (shows copyright)

All components combine in App.js to create the complete website.

JSX (HTML-like Syntax in JavaScript)

Looks like HTML but it's JavaScript! React uses JSX to describe what the UI should look like.

Browser doesn't understand JSX → That's why we need a build process!

Props (Passing Data)

Props allow you to pass data from parent components to child components.

Example: App.js passes your name, title, and photo to the Header component using props.

Why Use React?

1. ✅ **Reusable Components** - Write once, use everywhere
2. ✅ **Fast Updates** - Only changes what's needed
3. ✅ **Large Community** - Tons of resources and libraries
4. ✅ **Industry Standard** - Used by Facebook, Netflix, Airbnb, etc.

4. WHAT IS THE BUILD PROCESS?

Simple Explanation:

The build process **converts your development code into optimized files** that browsers can understand and load quickly.

What Happens During Build:

```
Your React Code (Development)
        ↓
    Build Process (npm run build)
        ↓
Optimized Files (Production)
```

Detailed Steps:

Step 1: Transpilation (Convert Modern JavaScript)

Converts modern JavaScript (ES6+) that older browsers don't understand into older JavaScript (ES5) that all browsers understand.

Example:

- Modern syntax: `const greeting = (name) => "Hello!";`
- Converted to: `var greeting = function(name) { return "Hello!"; }`

Step 2: JSX Transformation

Converts JSX (HTML-like syntax) into regular JavaScript function calls that browsers can execute.

Example:

- JSX: `<div className="header"><h1>Welcome</h1></div>`
- Converted to: `React.createElement("div", {className: "header"}, React.createElement("h1", null, "Welcome"))`

Step 3: Bundling (Combine Files)

Combines all your separate files (App.js, Header.js, About.js, Skills.js, etc.) into just 1-2 JavaScript files.

Before:

- 50+ separate JavaScript files
- Browser makes 50+ HTTP requests
- Slow loading

After:

- 2 JavaScript files (all code combined)
- Browser makes 2 HTTP requests
- Fast loading!

Step 4: Minification (Remove Unnecessary Code)

Removes all unnecessary characters:

- Whitespace and line breaks
- Comments
- Long variable names

Example:

- Before (readable, 500 KB): `function calculateTotal(price, quantity) { const total = price * quantity; return total; }`
- After (compressed, 100 KB): `function calculateTotal(p,q){return p*q}`

Step 5: Optimization

- Removes unused code (tree shaking)
- Compresses images
- Splits large files
- Adds cache busting (file hashes like `main.abc123.js`)

Build Command:

```
npm run build
```

What It Creates:

```
build/
├── index.html              # Main HTML file
├── static/
│   ├── js/
│   │   ├── main.abc123.js  # All your JS code (minified)
│   │   └── 2.def456.js     # React library (bundled)
│   └── css/
│       └── main.xyz789.css # All CSS combined and minified
└── asset-manifest.json     # List of all files
```

File Size Comparison:

Type	Development	Production	Reduction
Total Size	250 MB	500 KB	**99.8%!**
JavaScript	50+ files	2 files	Bundled
Load Time	5 seconds	0.5 seconds	**10x faster**

5. WHY CAN'T WE SKIP THE BUILD?

Problems Without Build Process:

Problem 1: Browser Doesn't Understand Modern JavaScript

- Your code uses ES6+ features (arrow functions, import statements)
- Older browsers don't understand these
- Website breaks in older browsers

Problem 2: Browser Doesn't Understand JSX

- JSX looks like: `<div className="header">Hello</div>`
- Browser sees this as a **syntax error**
- Needs to be converted to `React.createElement()` calls

Problem 3: Too Many Files

- Development: 50+ separate files
- Browser must download each file separately
- 50 HTTP requests = VERY SLOW

Problem 4: Code Too Large

- Development code has comments, long names, formatting
- 250 MB of files in `node_modules`
- Takes forever to download

With Build Process:

✅ **Modern JavaScript converted** → Works in all browsers
✅ **JSX converted** → Plain JavaScript function calls

✅ **All files bundled** → Just 2-3 files to download
✅ **Code minified** → 500 KB instead of 250 MB
✅ **Fast loading** → 0.5 seconds instead of 5 seconds

Real Example from Your Project:

Before Build (Development):

- Size: 250 MB (node_modules + source files)
- Files: 50+ JavaScript files
- Load time: 5+ seconds
- Browser: "Syntax Error: What is JSX?"

After Build (Production):

- Size: 500 KB (optimized)
- Files: 2-3 files
- Load time: 0.5 seconds
- Browser: "Perfect! I understand everything!"

6. WHAT IS GIT AND GITHUB?

Git (Version Control System)

Simple Explanation:

Git is like **Google Docs "Version History"** but for code.

What It Does:

- Tracks every change you make to files
- Lets you go back to previous versions
- Allows multiple people to work together
- Keeps a complete history of your project

Key Concepts:

Repository (Repo) = Project folder tracked by Git

- Contains a hidden `.git/` folder

- Stores all history and changes

Commit = Snapshot/Checkpoint

- Like saving a game
- Records the state of all files at that moment
- Has a message describing what changed

Branch = Parallel timeline

- Main branch: Production code (stable)
- Feature branches: Work on new features without breaking main

Common Commands:

Command	Purpose	When to Use
`git init`	Start tracking a folder	First time only (or never with create-react-app)
`git add .`	Stage files to commit	Before every commit
`git commit -m ""`	Save a checkpoint	After making changes
`git status`	See what's changed	Check before committing
`git log`	View history	See all commits
`git push`	Upload to GitHub	Share your code online
`git pull`	Download from GitHub	Get latest changes on server

GitHub (Cloud Storage for Git)

Simple Explanation:

GitHub is like **Google Drive** but for code repositories.

What It Does:

- Stores your code online (backup)
- Allows collaboration with others
- Showcases your projects (portfolio)
- Enables deployment (EC2 can clone from here)
- Free hosting for code

Git vs GitHub:

Git	GitHub
Software on your computer	Website (github.com)
Works offline	Requires internet
Version control system	Hosting service
Local repository	Remote repository

Workflow:

```
Your Computer (Git)  ↔  GitHub (Remote)
     Local Code     →     Push (upload)
     Local Code     ←     Pull (download)
```

Why Use Git & GitHub for This Project?

1. **Version Control**
   - Track all changes to your website
   - Undo mistakes easily
   - See who changed what and when
2. **Backup**
   - Code stored safely on GitHub
   - If laptop crashes, code is safe
   - Access from any computer
3. **Deployment**
   - EC2 clones code from GitHub
   - Easy to update: push to GitHub, pull on EC2
   - Version management (rollback if needed)
4. **Portfolio**
   - Show your code to employers
   - Demonstrate coding skills
   - Build professional reputation

7. WHAT IS AWS EC2?

AWS (Amazon Web Services)

Simple Explanation:

AWS is like **renting a computer in the cloud** that's always on and connected to the internet.

EC2 (Elastic Compute Cloud)

What Is It?

A **virtual computer** (server) running in Amazon's massive data centers around the world.

Think of it like:

- **Physical Server** = Buying a computer and keeping it in your house running 24/7
  - Costs: Computer ($500), Electricity ($50/month), Internet, Maintenance
- **EC2 Instance** = Renting a computer in Amazon's warehouse
  - Costs: $8-10/month (or FREE for 12 months!)

Key Concepts:

Instance = One virtual computer

- Has CPU, RAM, storage (like a regular computer)
- Runs an operating system (Ubuntu, in our case)
- Has a public IP address (accessible from internet)
- Can install software (Node.js, Nginx, etc.)

Instance Types:

```
t2.micro (Free Tier Eligible)
- 1 CPU core
- 1 GB RAM
- Good for small websites
- FREE for first year!

t2.small
- 1 CPU core
- 2 GB RAM
```

```
- $0.023/hour (~$17/month)

t2.medium
- 2 CPU cores
- 4 GB RAM
- $0.046/hour (~$34/month)
```

Public IP Address:

```
Example: 65.2.181.108

Your website: http://65.2.181.108
Anyone in the world can access!
```

Security Group (Firewall)

Controls who can access your server and through which ports:

```
Inbound Rules:
- Port 22 (SSH) → Your IP only (for secure login)
- Port 80 (HTTP) → Everyone (0.0.0.0/0) (for website access)
- Port 443 (HTTPS) → Everyone (for secure website - optional)
```

Why Use EC2 for This Project?

1. **Always Online**
    - 24/7 availability
    - No need to keep your laptop on
    - Professional hosting
2. **Fast Internet**
    - Data center connection
    - Quick loading worldwide
    - Low latency
3. **Scalable**
    - Start small (t2.micro - free)
    - Upgrade if traffic increases
    - Downgrade to save money

4. **Professional Experience**
    - Real production environment
    - Industry-standard cloud platform
    - Learn DevOps skills
5. **Free Tier**
    - 750 hours/month free (covers one instance 24/7)
    - Free for 12 months
    - Perfect for learning!

EC2 Data Centers:

Amazon has data centers worldwide. Your instance runs in one of them (you choose the region).

Regions:

- US East (N. Virginia)
- US West (Oregon)
- Asia Pacific (Mumbai) - closest to India
- Europe (Frankfurt)
- And 20+ more

Choosing Region:

- Pick closest to your target audience
- Lower latency = faster website
- For India: Use Asia Pacific (Mumbai)

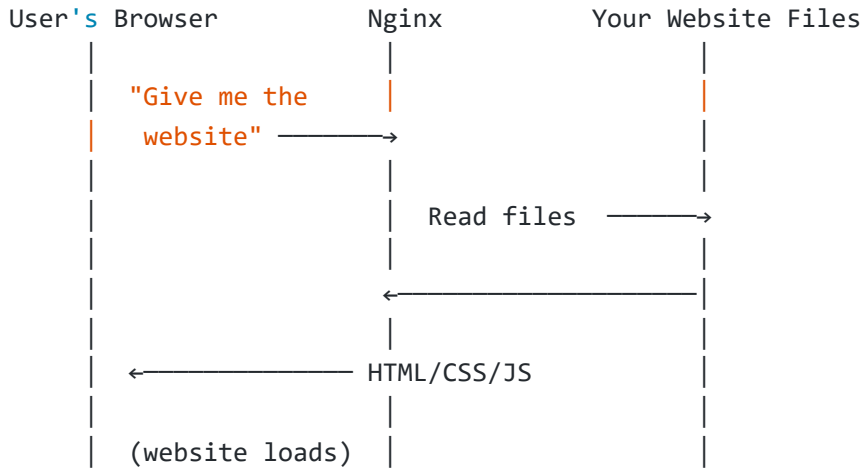
8. WHAT IS NGINX?

Simple Explanation:

Nginx is a **web server** - software that listens for website requests and sends back web pages.

Pronunciation: "Engine-X"

What Does a Web Server Do?



Real-World Analogy:

- **Website Files** = Books in a library
- **Nginx** = Librarian who fetches books when you ask
- **User's Browser** = You (the reader)
- **HTTP Request** = Asking the librarian for a book
- **HTTP Response** = Librarian gives you the book

Why Use Nginx?

1. Static File Serving

Nginx is excellent at serving static files (HTML, CSS, JavaScript, images).

What happens:

1. User visits `http://65.2.181.108`
2. Nginx reads `/var/www/html/index.html`
3. Nginx sends it to user's browser
4. Browser displays your website

2. URL Routing (Critical for React!)

The Problem: React is a Single Page Application (SPA):

- Only ONE HTML file exists: `index.html`
- All pages (/about, /skills, /projects) are handled by JavaScript
- But Nginx looks for actual files!

Without special configuration:

- User visits: `http://yoursite.com/about`
- Nginx looks for: `/var/www/html/about` file
- File doesn't exist → **404 Error** ❌

With our configuration:

- User visits: `http://yoursite.com/about`
- Nginx: "File doesn't exist? Send index.html instead"
- Browser loads index.html
- React's JavaScript reads URL (/about)
- React shows About component
- **Works perfectly!** ✅

3. Performance

- **Fast** - Handles 10,000+ requests per second
- **Lightweight** - Uses little RAM (important for t2.micro)
- **Concurrent** - Many users can visit at once
- **Efficient** - Serves static files faster than Node.js

4. Security

- Rate limiting (prevent DDoS attacks)
- SSL/TLS support (HTTPS)
- Access control (block specific IPs)
- Header manipulation (security headers)

Nginx vs Other Web Servers:

Feature	Nginx	Apache	Node.js Server
Speed for Static Files	⚡⚡⚡ Very Fast	⚡⚡ Fast	⚡ Slower
RAM Usage	💚 Low (2-10 MB)	🧡 Medium (20-50 MB)	💔 High (50-200 MB)
Configuration	Simple text file	Complex XML-like	JavaScript code

Feature	Nginx	Apache	Node.js Server
Best For	Static sites, React apps	PHP sites (WordPress)	Backend APIs, dynamic apps
Concurrency	Excellent	Good	Poor (needs clustering)

Our Nginx Configuration Explained:

What the configuration does:

1. **`listen 80;`** - Listen for HTTP requests on port 80 (standard web port)
2. **`server_name _;`** - Accept any domain name or IP address (wildcard)
3. **`root /var/www/html;`** - Website files are stored here
4. **`index index.html;`** - If someone visits `/`, show `index.html`
5. **`try_files $uri /index.html;`** - If file doesn't exist, show `index.html` (React routing fix)
6. **`error_page 404 /index.html;`** - Even on 404 errors, show `index.html`

Why This Configuration?

Traditional Multi-Page Website:

```
/home → home.html
/about → about.html
/contact → contact.html
```

Each URL points to a separate HTML file.

React Single Page App (SPA):

```
/ → index.html (React shows Home component)
/about → index.html (React shows About component)
/skills → index.html (React shows Skills component)
```

Only ONE HTML file! React's JavaScript handles all page switching.

Without `try_files $uri /index.html`:

- Visit `/about` → Nginx looks for `/about` file → Not found → **404 Error** ❌

With `try_files $uri /index.html`:

- Visit `/about` → Nginx looks for `/about` file → Not found → Serves `index.html` → React handles the rest → **Works!** ✅

9. WHY `git init` WAS NOT USED

The Question:

"In Step 1.3 (Git Version Control Setup), why didn't we use `git init` ?"

Short Answer:

We didn't need to! `create-react-app` automatically initialized Git for us.

Detailed Explanation:

What Happens When You Run `create-react-app` :

```
npx create-react-app shubham-portfolio
```

Behind the scenes, create-react-app does this automatically:

1. ✅ Creates the project folder
2. ✅ Installs React and dependencies
3. ✅ Creates project structure (src/, public/, etc.)
4. ✅ **Automatically runs `git init`** ← Already done!
5. ✅ Creates `.gitignore` file
6. ✅ Makes the first commit: "Initialize project using Create React App"

So Git is **already initialized** when the project is created!

How to Verify Git is Already Initialized:

Run this command in your project folder:

```
git status
```

If you see:

```
On branch main
Your branch is up to date with 'origin/main'.
```

Or:

```
On branch master
nothing to commit, working tree clean
```

Then Git is already initialized! ✅

You can also check for the hidden `.git` folder:

```
ls -la
```

If you see `.git/` folder, Git is initialized.

When DO You Need `git init` ?

You **only** need to run `git init` in these situations:

1. **Manual project** (not using create-react-app or any tool)

   ```
   mkdir my-project
   cd my-project
   git init  # ← Need this
   ```

2. **Plain HTML/CSS project**

   ```
   mkdir simple-website
   cd simple-website
   git init  # ← Need this
   ```

3. **Existing project without Git**
   - Project folder exists but no `.git` folder
   - Run `git init` to start tracking
4. **After cloning** (but this is automatic)
   - When you `git clone` , Git is already initialized in the cloned folder

What If You Run `git init` Again?

If you accidentally run `git init` in a folder that already has Git:

```
git init
```

You'll see:

```
Reinitialized existing Git repository in /path/to/project/.git/
```

This is usually harmless but unnecessary. It just re-initializes the existing repository without losing your history.

Complete Git Workflow Comparison:

Manual Project (Needs `git init`):

```
# Step 1: Create folder manually
mkdir my-project
cd my-project

# Step 2: Initialize Git (REQUIRED!)
git init

# Step 3: Create files
touch index.html

# Step 4: Add and commit
git add .
git commit -m "Initial commit"
```

Create-React-App Project (Git Already Initialized):

```
# Step 1: Create React app (git init happens automatically!)
npx create-react-app my-react-app
cd my-react-app

# Step 2: Git is ALREADY initialized! Just add changes
git add .
git commit -m "My changes"
```

```
# Step 3: Push to GitHub
git push origin main
```

Why Create-React-App Does This:

Benefits of automatic Git initialization:

1. **Convenience** - One less step for developers
2. **Best Practice** - Version control from day one
3. **Safety** - All changes tracked from the start
4. **Professional** - Industry standard workflow

Facebook (creators of React and create-react-app) knows that:

- All professional projects use Git
- Setting up Git manually is tedious
- Developers might forget to initialize Git
- Better to do it automatically!

How to Check Git History:

To see the initial commit create-react-app made:

```
# View all commits
git log

# View commits in one line (easier to read)
git log --oneline
```

You'll see:

```
abc1234 (HEAD -> main) Your latest commit
def5678 Fixed bug
xyz9012 Initial project using Create React App  ← create-react-app made this
```

Summary:

Scenario	Need `git init` ?	Why?
Using `create-react-app`	❌ NO	Automatically initialized

Scenario	Need `git init` ?	Why?
Manual HTML/CSS project	✅ **YES**	Must initialize manually
Cloning from GitHub	❌ **NO**	Already a Git repository
Starting from scratch	✅ **YES**	Must initialize manually
Existing untracked project	✅ **YES**	Must start tracking

Key Takeaway:

In our project, we skipped `git init` because create-react-app already did it for us!

This is a feature, not a mistake. It's one of the many conveniences that create-react-app provides to make React development easier.

PART 2: WINDOWS DEVELOPMENT SETUP

SECTION 1: PREREQUISITES INSTALLATION

1.1 Install Node.js and npm

Step 1: Download Node.js

1. Open browser
2. Go to: **https://nodejs.org/**
3. Click **"Download LTS"** (Long Term Support)
   - LTS version is more stable and recommended
   - Example: v20.11.0 LTS

Step 2: Install Node.js

1. Run the downloaded `.msi` file
2. Follow installation wizard (click Next → Next)
3. Accept license agreement
4. Choose default installation path: `C:\Program Files\nodejs\`

5. **Important:** Keep "Automatically install necessary tools" **CHECKED**
6. Click **Install**
7. Wait 2-3 minutes for installation
8. Click **Finish**

What gets installed:

- Node.js runtime
- npm (Node Package Manager)
- Additional build tools

Step 3: Verify Installation

1. Open **Command Prompt** or **PowerShell** or **Git Bash**
2. Check Node.js version:

   ```
   node -v
   ```

   Expected output: `v20.11.0` (or similar)
3. Check npm version:

   ```
   npm -v
   ```

   Expected output: `10.2.4` (or similar)

Troubleshooting: PowerShell Execution Policy Error

If you see this error:

```
npm : File C:\Program Files\nodejs\npm.ps1 cannot be loaded because
running scripts is disabled on this system.
```

Solution 1: Fix PowerShell

1. Open **PowerShell as Administrator**
   - Press `Windows + X`
   - Click "Windows PowerShell (Admin)"
2. Run this command:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

3. Type `Y` and press Enter
4. Test: `npm -v`

Solution 2: Use Different Terminal

- Use **Command Prompt** (cmd) instead
- Or use **Git Bash** (install Git first)
- Both work without policy issues

1.2 Install Git

Step 1: Download Git

1. Go to: **https://git-scm.com/download/win**
2. Download starts automatically (64-bit version)

Step 2: Install Git

1. Run the downloaded `.exe` file
2. Installation wizard opens - follow these settings:

Important Settings:

- **Installation Location:** Default (`C:\Program Files\Git\`)
- **Select Components:** Keep all defaults checked
- **Default Editor:** Choose "Visual Studio Code" (if installed) or keep default
- **PATH Environment:** Choose "**Git from the command line and also from 3rd-party software**"
- **HTTPS Backend:** Choose "**Use the OpenSSL library**"
- **Line Ending:** Choose "**Checkout Windows-style, commit Unix-style**"
- **Terminal Emulator:** Choose "**Use MinTTY**" (recommended)
- **Default git pull:** Choose "**Default (fast-forward or merge)**"
- **Credential Helper:** Choose "**Git Credential Manager**"
- **Extra Options:** Keep defaults

3. Click **Install**
4. Wait 2-3 minutes
5. Click **Finish**

What gets installed:

- Git version control system
- Git Bash (Unix-like terminal for Windows)
- Git GUI (graphical interface - optional)
- Git Credential Manager

Step 3: Verify Installation

1. Open **Git Bash**
   - Press `Windows` key
   - Type "Git Bash"
   - Click on "Git Bash"
2. Check Git version:

```
git --version
```

Expected output: `git version 2.43.0.windows.1` (or similar)

Step 4: Configure Git (First Time Only)

Set your identity (used in commit history):

```
# Set your name (use your real name)
git config --global user.name "Shubham Saini"

# Set your email (use your GitHub email)
git config --global user.email "your.email@example.com"

# Verify configuration
git config --list
```

Why configure:

- Every Git commit records who made the change
- Your name and email appear in commit history
- Required for pushing to GitHub

1.3 Install Visual Studio Code (VS Code)

Step 1: Download VS Code

1. Go to: **https://code.visualstudio.com/**
2. Click "**Download for Windows**"
3. Download is about 90 MB

Step 2: Install VS Code

1. Run the downloaded `.exe` file
2. Accept license agreement
3. Choose installation location (default is fine)

Important Options - Check these:

- ✅ Add "Open with Code" action to Windows Explorer file context menu
- ✅ Add "Open with Code" action to Windows Explorer directory context menu
- ✅ Register Code as an editor for supported file types
- ✅ Add to PATH (recommended)

4. Click **Install**
5. Wait 1-2 minutes
6. Click **Finish**

What VS Code provides:

- Code editor with syntax highlighting
- Integrated terminal
- Git integration
- Extensions marketplace
- Debugging tools

Step 3: Install Recommended Extensions

1. Open VS Code
2. Press `Ctrl + Shift + X` (opens Extensions panel)
3. Search and install these extensions:

Essential Extensions:

Extension	Purpose	Publisher
ES7+ React/Redux/React-Native snippets	React code shortcuts	dsznajder

Extension	Purpose	Publisher
Prettier - Code formatter	Auto-format code	Prettier
GitLens	Enhanced Git features	GitKraken
Auto Rename Tag	Auto-close HTML tags	Jun Han
Path Intellisense	Auto-complete file paths	Christian Kohler

How to install:

- Click on extension name
- Click "Install" button
- Wait for installation
- Repeat for all extensions

1.4 Create GitHub Account

Step 1: Sign Up

1. Go to: **https://github.com/**
2. Click **"Sign up"** button (top right)
3. Enter your email address
4. Create a strong password
5. Choose a username (e.g., `ShubhamCyberStack`)
6. Verify you're not a robot (solve puzzle)
7. Click **"Create account"**

Step 2: Verify Email

1. Check your email inbox
2. Find email from GitHub
3. Click verification link
4. Complete profile setup

Step 3: Create Personal Access Token (For Git Authentication)

Why needed:

- GitHub stopped accepting passwords for Git operations

- Personal Access Token (PAT) is the new authentication method
- Use PAT as password when pushing to GitHub

How to create:

1. Log in to GitHub
2. Click your profile picture (top-right)
3. Click **Settings**
4. Scroll down left sidebar
5. Click **Developer settings**
6. Click **Personal access tokens → Tokens (classic)**
7. Click **"Generate new token" → "Generate new token (classic)"**
8. Fill in details:
   - **Note:** "Portfolio Project Token"
   - **Expiration:** 90 days (or choose custom)
   - **Select scopes:** Check ✅ **repo** (Full control of private repositories)
9. Scroll down and click **"Generate token"**
10. ⚠️ **COPY THE TOKEN IMMEDIATELY!**
    - Token looks like: `ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
    - You'll only see it once!
    - Save it in a text file or password manager

When to use:

- When Git asks for password while pushing
- Paste this token (not your GitHub password)
- Token acts as your password for Git operations

SECTION 2: CREATING THE REACT PROJECT

2.1 Choose Project Location

Decide where to store your project on your computer.

Recommended Location:

```
C:\Users\shubh\Documents\Projects\
```

Alternative Locations:

```
C:\Projects\
D:\MyProjects\
C:\Users\shubh\Desktop\WebProjects\
```

In Terminal:

```
# Open Git Bash or Command Prompt

# Navigate to Documents folder
cd ~/Documents

# Create Projects folder
mkdir Projects

# Enter Projects folder
cd Projects
```

Verify location:

```
# Check current directory
pwd
```

Should show: `/c/Users/shubh/Documents/Projects`

2.2 Create React Application

Command:

```
npx create-react-app shubham-portfolio
```

What this command does:

1. **`npx`** - npm package runner (comes with npm)
    - Downloads and runs create-react-app temporarily
    - No need to install create-react-app globally
2. **`create-react-app`** - Official React project scaffolding tool

- Creates complete React project structure
- Installs all necessary dependencies
- Sets up build tools (Webpack, Babel)

3. **`shubham-portfolio`** - Your project name
   - Creates folder with this name
   - Can be any name (use lowercase, no spaces)

What happens:

```
Creating a new React app in C:\Users\shubh\Documents\Projects\shubham-portfolio.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[████████████████████████████] 100%

✓ Success! Created shubham-portfolio at C:\Users\shubh\Documents\Projects\shubham-portfolio

Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies.

We suggest that you begin by typing:

  cd shubham-portfolio
  npm start

Happy hacking!
```

Time taken: 2-5 minutes (depends on internet speed)

What gets installed:

- React library

- React-DOM (connects React to browser)
- React-Scripts (build tools, dev server, testing)
- 1000+ dependencies (in node_modules)

2.3 Navigate into Project

```
# Enter project folder
cd shubham-portfolio

# List all files (including hidden)
ls -la
```

You should see:

```
.git/                   # Git repository (hidden folder)
node_modules/           # All dependencies (250+ MB)
public/                 # Static files (HTML, images)
src/                    # Source code (your code goes here)
.gitignore              # Files Git should ignore
package.json            # Project configuration
package-lock.json       # Exact dependency versions
README.md               # Project documentation
```

2.4 Open in VS Code

Method 1: From Terminal

```
# Open current folder in VS Code
code .
```

The `.` (dot) means "current directory"

Method 2: From VS Code

1. Open VS Code
2. Click **File** → **Open Folder**
3. Navigate to `C:\Users\shubh\Documents\Projects\shubham-portfolio`
4. Click **Select Folder**

VS Code opens with:

- Left sidebar: File explorer
- Main area: Empty (no files open yet)
- Bottom: Integrated terminal (if not visible, press `Ctrl + ` `)

2.5 Start Development Server

In VS Code Terminal:

If terminal not visible:

- Press `Ctrl + ` ` (backtick)
- Or: **Terminal** → **New Terminal** from menu

Run:

```
npm start
```

What happens:

1. **Compiling...**

   ```
   Compiling...
   ```

2. **Success!**

   ```
   Compiled successfully!

   You can now view shubham-portfolio in the browser.

     Local:            http://localhost:3000
     On Your Network:  http://192.168.1.5:3000

   Note that the development build is not optimized.
   To create a production build, use npm run build.

   webpack compiled with 1 warning
   ```

3. **Browser opens automatically**

- Goes to: `http://localhost:3000`
- Shows spinning React logo
- Text: "Edit src/App.js and save to reload."

Keep this terminal running!

- Don't close it
- Don't press `Ctrl + C` (stops the server)
- Server watches for file changes and auto-reloads browser

What's running:

- **Webpack Dev Server** - Serves your React app
- **Hot Module Replacement** - Updates browser without full refresh
- **Error Overlay** - Shows errors in browser
- Runs on port 3000

SECTION 3: PROJECT STRUCTURE EXPLANATION

3.1 Root Level Files and Folders

```
shubham-portfolio/
├── .git/                  # Git version control data
├── node_modules/          # All dependencies (don't touch!)
├── public/                # Static files
├── src/                   # Source code (your code)
├── .gitignore             # Files Git ignores
├── package.json           # Project configuration
├── package-lock.json      # Dependency versions
└── README.md              # Project documentation
```

Detailed Explanation:

`.git/` (Hidden Folder)

- Created automatically by create-react-app
- Contains Git version control data
- Stores all commits, branches, history
- **Never modify this folder manually**
- To see it: `ls -la` (shows hidden files)

`node_modules/` (250+ MB)

- Contains all installed packages
- 1000+ folders (React, Babel, Webpack, etc.)
- **Never edit files here**
- **Never push to GitHub** (too large)
- Can be regenerated with `npm install`
- Listed in `.gitignore`

`public/` Folder

- Contains static files
- Files here are copied as-is to build
- No processing by Webpack
- Use for: images, fonts, favicon

`src/` Folder

- Contains your source code
- React components go here
- Processed by Webpack (bundled, minified)
- This is where you write code!

`.gitignore` File

- Lists files Git should not track
- Includes: node_modules, build, .env files
- Pre-configured by create-react-app

`package.json`

- Project configuration file
- Lists dependencies (React, react-dom, etc.)
- Defines scripts (start, build, test)
- Project metadata (name, version)

`package-lock.json`

- Records exact versions of all dependencies
- Ensures everyone installs same versions
- Auto-generated, don't edit manually

- Committed to Git

README.md

- Project documentation
- Written in Markdown format
- Displays on GitHub repository page
- Explains how to run the project

3.2 package.json Explained

Location: `package.json` (root level)

Purpose: Project configuration and dependency management

Key Sections:

1. Metadata

```
"name": "shubham-portfolio"      // Project name
"version": "0.1.0"                // Version number
"private": true                   // Not published to npm
```

2. Dependencies

```
"dependencies": {
  "react": "^18.2.0",             // React library
  "react-dom": "^18.2.0",         // React DOM bindings
  "react-scripts": "5.0.1"        // Build tools
}
```

Version Syntax:

- `^18.2.0` - Caret: Install 18.2.0 or newer minor version
- `~18.2.0` - Tilde: Install 18.2.0 or newer patch version
- `18.2.0` - Exact: Install exactly 18.2.0

3. Scripts

```
"scripts": {
  "start": "react-scripts start",      // Development server
```

```
    "build": "react-scripts build",      // Production build
    "test": "react-scripts test",        // Run tests
    "eject": "react-scripts eject"       // Eject from CRA (advanced)
  }
```

How to run scripts:

```
npm start      # Runs: react-scripts start
npm run build  # Runs: react-scripts build
npm test       # Runs: react-scripts test
```

4. ESLint Configuration

```
"eslintConfig": {
  "extends": ["react-app"]    // React linting rules
}
```

5. Browserslist

```
"browserslist": {
  "production": [">0.2%", "not dead"],
  "development": ["last 1 chrome version"]
}
```

Defines which browsers to support (affects transpilation)

3.3 public/ Folder

Location: `public/` (root level)

Contents:

```
public/
├── index.html         # Main HTML template
├── favicon.ico        # Browser tab icon (16x16)
├── logo192.png        # PWA icon (192x192)
├── logo512.png        # PWA icon (512x512)
├── manifest.json      # PWA configuration
└── robots.txt         # Search engine instructions
```

index.html (Most Important File)

Purpose: Single HTML file that loads your entire React app

Key Elements:

1. The Root Div:

```
<div id="root"></div>
```

- React renders everything inside this div
- Empty initially
- React takes over and fills with components
- Don't add anything inside this div manually

2. Meta Tags:

- Viewport settings for mobile responsiveness
- Theme color for mobile browsers
- Description for SEO

3. Links:

- Favicon
- Apple touch icon
- Manifest file

What happens:

1. Browser loads `index.html`
2. Browser loads JavaScript files (injected during build)
3. React finds `<div id="root">`
4. React renders your entire app inside it

Other Files:

`favicon.ico`

- Small icon in browser tab
- 16x16 pixels
- Replace with your own logo

`logo192.png` and `logo512.png`

- Used for Progressive Web App (PWA)
- Icons for home screen when "Add to Home Screen"
- Can be replaced

`manifest.json`

- PWA configuration
- Defines app name, icons, colors
- Used when installing app on mobile

`robots.txt`

- Instructions for search engine crawlers
- Controls what search engines can index
- `User-agent: *` - Allow all
- `Disallow:` - Nothing blocked

3.4 src/ Folder (Where You Code)

Location: `src/` (root level)

Default Contents:

```
src/
├── App.js              # Main App component
├── App.css             # App styles
├── App.test.js         # App tests
├── index.js            # Entry point (DON'T MODIFY)
├── index.css           # Global styles
├── logo.svg            # React logo
├── reportWebVitals.js  # Performance monitoring
└── setupTests.js       # Test configuration
```

index.js (Entry Point)

Purpose: Connects React to the DOM

What it does:

1. Imports React library

2. Imports ReactDOM (connects React to browser)
3. Imports App component
4. Finds `<div id="root">` in index.html
5. Renders `<App />` component inside root div

Key Line:

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Do NOT modify this file - it works perfectly as-is

App.js (Main Component)

Purpose: The main component of your application

Structure:

- Imports (React, CSS, components)
- Function definition (component logic)
- Return statement (JSX - what to display)
- Export statement (make available to other files)

This is where your website code goes!

- In our project, we replaced this with our portfolio structure
- Imports all components (Header, About, Skills, etc.)
- Arranges them in order

CSS Files:

`index.css` - Global styles

- Applies to entire app
- Body styles, fonts, resets
- Imported in index.js

`App.css` - App-specific styles

- Styles for App component
- Component-specific CSS
- Imported in App.js

Component CSS files (we created these):

- `Header.css`, `About.css`, etc.
- Each component has its own CSS
- Keeps styles organized and modular

Other Files:

`App.test.js`

- Unit tests for App component
- Uses Jest testing framework
- Run with: `npm test`

`reportWebVitals.js`

- Performance monitoring
- Measures: FCP, LCP, FID, CLS, TTFB
- Optional feature

`setupTests.js`

- Test environment configuration
- Sets up Jest and React Testing Library

`logo.svg`

- React logo (spinning logo in default app)
- Can be deleted if not used

3.5 .gitignore Explained

Location: `.gitignore` (root level)

Purpose: Tell Git which files to ignore (not track)

Contents:

```
# dependencies
/node_modules

# production
/build
```

```
# misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

Why Ignore These?

File/Folder	Reason to Ignore
node_modules/	250 MB, 1000+ folders, can be regenerated with `npm install`
build/	Generated files, can be rebuilt with `npm run build`
.DS_Store	Mac system file, no value for project
.env.local	Contains secrets (API keys, passwords), security risk
npm-debug.log	Error logs, temporary files

What Git DOES Track:

✅ **Source code** (src/)
✅ **Public files** (public/)
✅ **Configuration** (package.json, .gitignore)
✅ **Documentation** (README.md)

❌ **NOT tracked:** node_modules, build, logs, environment files

Benefits:

1. **Smaller repository** (500 KB instead of 250 MB)
2. **Faster clone** (seconds instead of minutes)
3. **Security** (no secret keys leaked)
4. **Cleaner history** (only meaningful changes tracked)

SECTION 4: BUILDING THE PORTFOLIO WEBSITE

4.1 Create Folder Structure

Goal: Organize code into logical folders

Structure to create:

```
src/
├── data/              # Store your personal information
│   └── profileData.js
├── components/        # Reusable UI components
│   ├── Header.js
│   ├── Header.css
│   ├── About.js
│   ├── About.css
│   ├── Skills.js
│   ├── Skills.css
│   ├── Experience.js
│   ├── Experience.css
│   ├── Education.js
│   ├── Education.css
│   ├── Projects.js
│   ├── Projects.css
│   ├── Footer.js
│   └── Footer.css
├── App.js             # Main component (combines all)
├── App.css            # Main styles
└── index.js           # Entry point (don't modify)
```

How to create:

Method 1: Using Terminal (Git Bash/Command Prompt)

```
# Create folders
mkdir src/data
mkdir src/components

# Create files
touch src/data/profileData.js
touch src/components/Header.js
touch src/components/Header.css
touch src/components/About.js
touch src/components/About.css
touch src/components/Skills.js
touch src/components/Skills.css
touch src/components/Experience.js
touch src/components/Experience.css
touch src/components/Education.js
```

```
touch src/components/Education.css
touch src/components/Projects.js
touch src/components/Projects.css
touch src/components/Footer.js
touch src/components/Footer.css
```

Method 2: Using VS Code GUI

1. **Create folders:**
   - Right-click `src` folder → **New Folder** → Type `data` → Enter
   - Right-click `src` folder → **New Folder** → Type `components` → Enter
2. **Create files:**
   - Right-click `data` folder → **New File** → Type `profileData.js` → Enter
   - Right-click `components` folder → **New File** → Type `Header.js` → Enter
   - Repeat for all component files

Why this structure:

- **Organized** - Easy to find files
- **Scalable** - Add new components easily
- **Professional** - Industry-standard structure
- **Maintainable** - Clear separation of concerns

4.2 Component Files Overview

All component files are available in the GitHub repository:
https://github.com/ShubhamCyberStack/portfolio-website

Navigate to: `src/components/` to see all component code

Components Created:

1. profileData.js

- **Location:** `src/data/profileData.js`
- **Purpose:** Stores all your personal information
- **Contains:** Name, title, skills, experience, education, projects, contact info
- **Type:** JavaScript object exported as default

- **Why separate:** Easy to update info without touching components

2. Header.js & Header.css

- **Location:** `src/components/`
- **Purpose:** Top section of website
- **Displays:** Profile photo, name, title, social links (LinkedIn, GitHub, Email)
- **Styling:** Purple gradient background, circular profile image, social link buttons
- **Responsive:** Adjusts for mobile screens

3. About.js & About.css

- **Location:** `src/components/`
- **Purpose:** About Me section
- **Displays:** Your description/summary from profileData
- **Styling:** Light gray background, centered text, decorative underline
- **Layout:** Max-width container for readability

4. Skills.js & Skills.css

- **Location:** `src/components/`
- **Purpose:** Display your technical skills
- **Displays:** Grid of colored skill cards
- **Styling:** Purple gradient cards, hover effects (lift up, scale)
- **Responsive:** Grid adjusts columns based on screen size
- **Dynamic:** Loops through skills array from profileData

5. Experience.js & Experience.css

- **Location:** `src/components/`
- **Purpose:** Work experience timeline
- **Displays:** Company, position, duration, responsibilities
- **Styling:** Vertical timeline with purple line, circular markers
- **Layout:** Cards with shadow, left-aligned timeline
- **Dynamic:** Loops through experience array

6. Education.js & Education.css

- **Location:** `src/components/`
- **Purpose:** Educational background

- **Displays:** Institution, degree, duration, location, grade
- **Styling:** Cards with left purple border, hover effect
- **Responsive:** Grid layout, stacks on mobile
- **Dynamic:** Loops through education array

7. Projects.js & Projects.css

- **Location:** `src/components/`
- **Purpose:** Showcase your projects
- **Displays:** Project name, description, technologies, GitHub/demo links
- **Styling:** Cards with technology tags, hover effects
- **Links:** GitHub and Live Demo buttons (if provided)
- **Dynamic:** Loops through projects array
- **Optional:** Only shows if projects exist

8. Footer.js & Footer.css

- **Location:** `src/components/`
- **Purpose:** Bottom section
- **Displays:** Copyright, tech stack, social links
- **Styling:** Dark background, centered text, link hover effects
- **Dynamic:** Copyright year auto-updates
- **Info:** Shows "Built with React & deployed on AWS EC2 with Nginx"

4.3 How Components Work Together

Component Hierarchy:

```
App.js (Main Container)
├── Header (name, photo, links)
├── About (description)
├── Skills (skill cards grid)
├── Experience (timeline)
├── Education (education cards)
├── Projects (project cards)
└── Footer (copyright, links)
```

Data Flow:

```
profileData.js (Data Source)
        ↓
    App.js (Receives data, passes to components)
        ↓
    Components (Receive data via props, display it)
```

Example of data flow:

1. **profileData.js** contains:

```
name: "Shubham Saini"
```

2. **App.js** imports profileData:

```
import profileData from './data/profileData';
```

3. **App.js** passes to Header:

```
<Header name={profileData.name} />
```

4. **Header.js** receives and displays:

```
function Header({ name }) {
  return <h1>{name}</h1>;
}
```

5. **Result:** "Shubham Saini" appears on website

4.4 Styling Approach

CSS Organization:

Each component has its own CSS file:

- `Header.js` uses `Header.css`
- `About.js` uses `About.css`
- etc.

Benefits:

- **Modular** - Styles don't conflict
- **Maintainable** - Easy to find and edit
- **Organized** - Each component is self-contained

Global Styles:

`App.css` contains:

- CSS reset (remove default margins/padding)
- Global font family
- Smooth scrolling
- Custom scrollbar (purple theme)

Color Theme:

Consistent purple gradient throughout:

- Header background
- Section title underlines
- Skill cards
- Timeline markers
- Scrollbar
- Hover effects

Colors used:

- Primary: `#667eea` (blue-purple)
- Secondary: `#764ba2` (dark purple)
- Background: `#f8f9fa` (light gray)
- Text: `#333` (dark gray)

Responsive Design:

Media queries in each CSS file:

```
@media (max-width: 768px) {
  /* Mobile styles */
}
```

Adjustments for mobile:

- Smaller font sizes

- Single column layouts
- Reduced padding
- Smaller profile image
- Stack elements vertically

4.5 File Locations Reference

All code files are in GitHub repository: https://github.com/ShubhamCyberStack/portfolio-website

To view code:

1. Go to repository
2. Click on folder/file you want to see
3. GitHub shows the code with syntax highlighting

Direct links to key files:

- **profileData.js:** `src/data/profileData.js`
- **App.js:** `src/App.js`
- **App.css:** `src/App.css`
- **All components:** `src/components/` folder

To get code locally:

```
# Clone repository
git clone https://github.com/ShubhamCyberStack/portfolio-website.git

# Navigate into folder
cd portfolio-website

# Open in VS Code
code .
```

To copy a specific file:

1. Open file on GitHub
2. Click "Raw" button (top right)
3. Copy all text
4. Paste into your local file

SECTION 5: TESTING LOCALLY

5.1 Start Development Server

Ensure you're in project root:

```
pwd
# Should show: /c/Users/shubh/Documents/Projects/shubham-portfolio
```

Start server:

```
npm start
```

What happens:

1. Webpack compiles your code
2. Dev server starts on port 3000
3. Browser opens automatically
4. Shows your portfolio website

Expected output:

```
Compiled successfully!

You can now view shubham-portfolio in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.5:3000
```

5.2 View in Browser

Automatic:

- Browser opens automatically
- Goes to `http://localhost:3000`

Manual:

- Open browser

- Type: `localhost:3000` or `127.0.0.1:3000`
- Press Enter

What you should see: ✅ Purple gradient header with your name
✅ Profile photo (or placeholder)
✅ About Me section
✅ Skills grid (colorful cards)
✅ Work Experience timeline
✅ Education section
✅ Projects section
✅ Footer with copyright

5.3 Test All Features

1. Navigation Links:

- Click **LinkedIn** button → Opens your LinkedIn profile in new tab
- Click **GitHub** button → Opens your GitHub profile in new tab
- Click **Email** button → Opens email client with your email

2. Scrolling:

- Scroll up and down
- Should be smooth (CSS scroll-behavior: smooth)
- All sections visible

3. Hover Effects:

- Hover over skill cards → They lift up and grow
- Hover over social links → Background changes to white
- Hover over project links → Color changes, underline appears

4. Content Display:

- All your information appears correctly
- No "undefined" or errors
- Images load (if you added them)

5.4 Test Responsiveness

Method 1: Resize Browser Window

1. **Desktop view** (wide window):
    - Skills: Multiple columns
    - Projects: 2-3 columns
    - Education: 2 columns
2. **Tablet view** (medium window):
    - Skills: 2-3 columns
    - Projects: 2 columns
    - Education: 1-2 columns
3. **Mobile view** (narrow window):
    - Skills: 2 columns
    - Projects: 1 column
    - Education: 1 column
    - Profile image: Smaller
    - Font sizes: Reduced

How to resize:

- Drag edge of browser window
- Make it narrow (like a phone screen)
- Watch how layout adapts

Method 2: Browser DevTools

1. Press **F12** (opens Developer Tools)
2. Press **Ctrl + Shift + M** (toggle device toolbar)
3. Or click phone/tablet icon (top-left of DevTools)
4. Select device from dropdown:
    - iPhone 12 Pro
    - iPad Air
    - Galaxy S20
    - Or set custom dimensions

What to check:

- Layout doesn't break

- Text is readable (not too small)
- Buttons are tappable (not too small)
- No horizontal scrolling
- Images fit screen

5.5 Check Browser Console for Errors

Open Console:

- Press **F12**
- Click "**Console**" tab

What to look for:

✅ **No errors** (no red text)

- Warnings (yellow) are usually okay
- Errors (red) need to be fixed

❌ **Common errors:**

- "Module not found" → Check import paths
- "Cannot read property of undefined" → Check data exists
- "Unexpected token" → Syntax error in code

If you see errors:

1. Read error message carefully
2. Click on error to see which file/line
3. Go to that file in VS Code
4. Fix the error
5. Save file
6. Browser auto-reloads with fix

5.6 Hot Reload Feature

What is Hot Reload:

- Dev server watches for file changes

- When you save a file, it automatically:
  - Recompiles code
  - Refreshes browser
  - Preserves component state (usually)

Test it:

1. Open `src/data/profileData.js`
2. Change your name: `name: "Shubham Saini Updated"`
3. Save: `Ctrl + S`
4. Watch browser → Automatically updates!
5. Change back and save again

Benefits:

- Instant feedback
- No manual refresh needed
- Faster development
- See changes immediately

5.7 Performance Check (Optional)

Use Lighthouse:

1. Open website in Chrome
2. Press **F12** (DevTools)
3. Click "**Lighthouse**" tab
   - If not visible, click `>>` icon, select Lighthouse
4. Select categories:
   - ✅ Performance
   - ✅ Accessibility
   - ✅ Best Practices
   - ✅ SEO
5. Click "**Generate report**"
6. Wait 30 seconds
7. Review scores (0-100)

Good scores:

- Performance: 90+
- Accessibility: 90+
- Best Practices: 90+
- SEO: 90+

If scores are low:

- Lighthouse shows suggestions
- Follow recommendations
- Re-run after changes

SECTION 6: GIT VERSION CONTROL SETUP

6.1 Verify Git is Initialized

Check if Git is already set up:

```
# Check Git status
git status
```

Expected output (Git is initialized):

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Or:

```
On branch main

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/components/
        src/data/

nothing added to commit but untracked files present
```

If you see this (Git NOT initialized):

```
fatal: not a git repository (or any of the parent directories): .git
```

Then initialize Git:

```
git init
```

Why create-react-app usually initializes Git:

- Convenience for developers
- Industry best practice
- Version control from day one
- Automatic .gitignore creation
- First commit: "Initialize project using Create React App"

6.2 Configure Git (First Time Only)

Set your identity:

```
# Set your name (appears in commit history)
git config --global user.name "Shubham Saini"

# Set your email (should match GitHub email)
git config --global user.email "your.email@example.com"

# Verify configuration
git config --list
```

Why needed:

- Every commit records author name and email
- Shows who made changes
- Required for pushing to GitHub
- Used in contribution graphs

Global vs Local:

- `--global` - Applies to all Git repositories on your computer
- Without `--global` - Only applies to current repository

6.3 Check What Files Will Be Tracked

```
# See current status
git status
```

You'll see:

Untracked files (red):

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/components/
        src/data/
        src/App.js
        src/App.css
```

Modified files (red):

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
        modified:   src/App.js
        modified:   package.json
```

Files Git will IGNORE (not shown):

- node_modules/ (listed in .gitignore)
- build/ (listed in .gitignore)
- .env files (listed in .gitignore)

6.4 Stage All Files

Add files to staging area:

```
# Add all changed/new files
git add .
```

The `.` (dot) means "all files in current directory and subdirectories"

Verify files are staged:

```
git status
```

Now files are green:

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   src/components/Header.js
        new file:   src/components/Header.css
        new file:   src/data/profileData.js
        modified:   src/App.js
```

What staging means:

- Files are ready to be committed
- Like packing items before shipping
- Can unstage if needed
- Next step: Commit (permanent snapshot)

6.5 Create First Commit

Commit staged files:

```
git commit -m "Initial commit: Complete portfolio website with React"
```

Commit message guidelines:

- `-m` flag = message
- Brief but descriptive
- Present tense ("Add feature" not "Added feature")
- Explains WHAT changed, not HOW

Good commit messages:

```
"Initial commit: Complete portfolio website with React"
"Add Skills section with grid layout"
"Fix responsive design on mobile devices"
"Update about section with new internship info"
```

Bad commit messages:

```
"changes"
"fix"
"update"
"asdf"
```

After committing:

```
# View commit history
git log --oneline
```

Output:

```
abc1234 (HEAD -> main) Initial commit: Complete portfolio website with React
```

- `abc1234` - Commit hash (unique identifier)
- `(HEAD -> main)` - Current position
- Rest - Your commit message

6.6 Understanding Git Workflow

Complete Git workflow:

```
Working Directory → Staging Area → Repository
     (edit)            (add)         (commit)
```

Example:

1. **Working Directory** (you edit files):

```
# Edit src/App.js
# Status: Modified, not staged
```

2. **Staging Area** (mark files for commit):

```
git add src/App.js
# Status: Staged, ready to commit
```

3. **Repository** (permanent snapshot):

```
git commit -m "Update App.js"
# Status: Committed, in history
```

Visual:

```
[Edit files] → git add → [Staging] → git commit → [History]
```

SECTION 7: PUSHING TO GITHUB

7.1 Create GitHub Repository

Step-by-step:

1. **Go to GitHub.com**
   - Log in to your account
2. **Create new repository**
   - Click "+" icon (top-right)
   - Click "**New repository**"
3. **Fill in repository details:**
   - **Owner:** ShubhamCyberStack (your username)
   - **Repository name:** `portfolio-website`
   - **Description:** "My personal portfolio website built with React and deployed on AWS EC2"
   - **Visibility:**
     - ✅ **Public** (recommended - required for free hosting)
     - ❌ Private (if you want to keep code secret)
4. ⚠️ **CRITICAL: DO NOT CHECK ANY BOXES!**
   - ❌ **DO NOT** check "Add a README file"
   - ❌ **DO NOT** check "Add .gitignore"
   - ❌ **DO NOT** check "Choose a license"

Why? Your project already has these files. Checking these creates conflicts.

5. **Click "Create repository"**

What happens:

- Empty repository is created
- You see a page with setup instructions
- Keep this page open - you'll need the commands

7.2 Add Remote Repository

What is a "remote":

- A remote is a link to your GitHub repository
- Tells Git where to push/pull code
- Usually named "origin" (convention)

Add GitHub as remote:

```
# Add remote (replace with YOUR repository URL)
git remote add origin https://github.com/ShubhamCyberStack/portfolio-website.git
```

Get your repository URL:

- From the GitHub page after creating repository
- Format: `https://github.com/USERNAME/REPO-NAME.git`
- Copy entire URL

Verify remote was added:

```
git remote -v
```

Expected output:

```
origin  https://github.com/ShubhamCyberStack/portfolio-website.git (fetch)
origin  https://github.com/ShubhamCyberStack/portfolio-website.git (push)
```

- `origin` - Remote name
- `(fetch)` - Used for pulling
- `(push)` - Used for pushing

7.3 Rename Branch to Main

Why rename:

- Old default: `master`
- New default: `main`
- GitHub uses `main` as default
- Better to match GitHub's default

Check current branch:

```
git branch
```

If output is `* master`:

```
# Rename to main
git branch -M main
```

If output is `* main`:

- Already correct, no action needed

Verify:

```
git branch
```

Expected output:

```
* main
```

The `*` indicates current branch.

7.4 Push to GitHub

Push your code:

```
# Push to main branch on origin remote
```

```
git push -u origin main
```

Command breakdown:

- `git push` - Upload commits to remote
- `-u` - Set upstream (remember this branch for future pushes)
- `origin` - Remote name (GitHub)
- `main` - Branch name

After first push with `-u`:

- Future pushes: Just `git push` (no need for `-u origin main`)
- Git remembers where to push

7.5 Authentication

GitHub will ask for authentication:

Method 1: Browser Window Opens (Easiest)

If a browser window opens:

1. Log in to GitHub (if not already)
2. Click "**Authorize Git Credential Manager**"
3. Return to terminal
4. Push completes automatically

Method 2: Terminal Asks for Credentials

If terminal asks for username and password:

```
Username for 'https://github.com': ShubhamCyberStack
Password for 'https://ShubhamCyberStack@github.com':
```

⚠️ CRITICAL: DO NOT use your GitHub password!

Instead, use your **Personal Access Token**:

- The token you created earlier
- Looks like: `ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
- Paste it (nothing appears while pasting - that's normal)

- Press Enter

Why token instead of password:

- GitHub deprecated password authentication
- Tokens are more secure
- Tokens can be revoked
- Tokens have limited permissions

7.6 Verify on GitHub

After successful push:

```
Enumerating objects: 50, done.
Counting objects: 100% (50/50), done.
Delta compression using up to 8 threads
Compressing objects: 100% (45/45), done.
Writing objects: 100% (50/50), 15.00 KiB | 1.00 MiB/s, done.
Total 50 (delta 5), reused 0 (delta 0)
To https://github.com/ShubhamCyberStack/portfolio-website.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Visit your repository:

1. Go to: `https://github.com/ShubhamCyberStack/portfolio-website`
2. Refresh page if already open

You should see:

```
portfolio-website/
├── public/
├── src/
│   ├── components/
│   ├── data/
│   ├── App.js
│   ├── App.css
│   └── index.js
├── .gitignore
├── package.json
├── package-lock.json
└── README.md
```

Verify:

- ✅ All folders present
- ✅ All files uploaded
- ✅ Click through folders to check contents
- ✅ Code is readable with syntax highlighting
- ✅ Commit history shows your commits

7.7 Understanding Push/Pull

Git workflow with GitHub:

```
Your Computer (Local)  ↔  GitHub (Remote)
        Git                  Web Interface

   [Make changes]
        ↓
   [git add .]
        ↓
   [git commit]
        ↓
   [git push]    ————————→   [Uploaded]

   [git pull]    ←————————   [Download changes]
```

Common commands:

Command	Direction	Purpose
`git push`	Local → GitHub	Upload your commits
`git pull`	GitHub → Local	Download latest changes
`git fetch`	GitHub → Local	Check for changes (don't merge)
`git clone`	GitHub → Local	Copy entire repository

When to push:

- After completing a feature
- After fixing a bug
- Before ending work session

- Multiple times per day

When to pull:

- Before starting work
- When working with others
- After someone else pushes
- To get latest code on server

7.8 Update README.md (Optional but Recommended)

Make your repository professional:

Edit README.md:

1. Open `README.md` in VS Code
2. Replace content with:

```
# Portfolio Website

## 🌐 Live Demo
[View Live Website](http://65.2.181.108)

## 📝 Description
My personal portfolio website showcasing my skills and experience in Cloud Computing, DevOps, AWS

## 🛠️ Built With
- **Frontend:** React.js
- **Styling:** CSS3
- **Version Control:** Git & GitHub
- **Deployment:** AWS EC2 (Ubuntu)
- **Web Server:** Nginx

## 🚀 Features
- Responsive design (mobile-friendly)
- Modern UI with gradient colors
- Interactive components
- Fast loading (optimized build)
- SEO friendly

## 📁 Project Structure
```

portfolio-website/ ├── public/ # Static files ├── src/ │ ├── components/ # React components │ ├── data/ # Personal data │ ├── App.js # Main component │ └── index.js # Entry point └── package.json # Dependencies

```
## 🏃 Running Locally

### Prerequisites
- Node.js (v18+)
- npm (v9+)

### Installation
```bash
# Clone repository
git clone https://github.com/ShubhamCyberStack/portfolio-website.git

# Navigate to project
cd portfolio-website

# Install dependencies
npm install

# Start development server
npm start
```

Open http://localhost:3000 in your browser.

Build for Production

```
npm run build
```

📧 Contact

- **Email:** your.email@example.com
- **LinkedIn:** Shubham Saini
- **GitHub:** @ShubhamCyberStack

📄 License

This project is open source and available under the MIT License.
```

🙏 Acknowledgments

- React team for amazing framework
- AWS for cloud hosting
- Nginx for web server

Commit and push update:

```bash
git add README.md
git commit -m "Update README with project details"
git push origin main
```

Benefits of good README:

- Professional appearance
- Easy for others to understand
- Shows how to run project
- Good for job applications
- GitHub displays it prominently

✅ WINDOWS DEVELOPMENT SETUP COMPLETE!

You now have:

- ✅ React project created
- ✅ Portfolio website built
- ✅ Code tested locally
- ✅ Git version control set up
- ✅ Code pushed to GitHub
- ✅ Professional README added

Next: Deploy to AWS EC2 (Part 3)

PART 3: AWS EC2 UBUNTU DEPLOYMENT

SECTION 1: PREREQUISITES ON AWS

1.1 AWS Account Setup

If you don't have an AWS account:

1. **Go to AWS website:**
   - Visit: https://aws.amazon.com/
   - Click "**Create an AWS Account**"
2. **Fill in account details:**
   - Email address
   - Password
   - AWS account name (e.g., "Shubham Personal")
3. **Choose account type:**
   - Select "**Personal**" account
4. **Enter payment information:**
   - Credit/Debit card required
   - ⚠️ **Won't be charged if you stay in Free Tier**
   - Used for verification only
   - $1-2 may be held temporarily (refunded)
5. **Verify identity:**
   - Enter phone number
   - Receive verification code (SMS or call)
   - Enter code to verify
6. **Choose support plan:**
   - Select "**Basic Support - Free**"
   - No need for paid plans initially
7. **Complete registration:**
   - Wait for account activation (up to 24 hours, usually instant)

- Check email for confirmation

AWS Free Tier includes:

- 750 hours/month EC2 t2.micro (12 months)
- 30 GB storage (12 months)
- 15 GB data transfer out (12 months)

Note: 750 hours = One t2.micro instance running 24/7 all month

1.2 Create EC2 Instance

Navigate to EC2:

1. Log in to AWS Console
2. Click search bar at top
3. Type "EC2"
4. Click **"EC2"** (Virtual servers in the cloud)

Launch Instance:

1. **Click "Launch Instance" button** (orange button)
2. **Name your instance:**
   - Name: `portfolio-website-server`
   - Purpose: Easy identification in instance list
3. **Choose AMI (Operating System):**
   - Section: "Application and OS Images (Amazon Machine Image)"
   - Select **"Ubuntu"**
   - Choose **"Ubuntu Server 22.04 LTS"**
   - Architecture: **64-bit (x86)**
   - **Important:** Look for "Free tier eligible" badge

Why Ubuntu:

- Popular and well-supported
- Great for web servers
- Easy package management (apt)

- Large community
- Stable and secure

4. **Choose Instance Type:**
    - Select **"t2.micro"**
    - **Free tier eligible**
    - Specs: 1 vCPU, 1 GiB RAM
    - Sufficient for portfolio website
5. **Key Pair (Login):**
    - Click **"Create new key pair"**
    - **Key pair name:** `shubham-portfolio-key`
    - **Key pair type:** RSA
    - **Private key file format:** `.pem` (for SSH on Linux/Mac/Git Bash)
    - Click **"Create key pair"**
    - ⚠️ **File downloads automatically to Downloads folder**
    - **CRITICAL:** Keep this file safe! You can't download it again
    - If lost, you can't access your instance

What is a key pair:

- Public key: Stored on EC2 instance
- Private key (.pem file): You keep this
- Like a key and lock
- Required to SSH into instance

6. **Network Settings:**

Click **"Edit"** button to expand settings:

Firewall (security groups):

- Select **"Create security group"**
- **Security group name:** `portfolio-web-server-sg`
- **Description:** "Allow SSH and HTTP traffic"

Inbound security group rules:

Rule 1 (SSH):

- ✅ **Allow SSH traffic from**
- Source type: "**My IP**" (recommended for security)
- Port: 22 (auto-filled)
- Purpose: Allows you to connect to instance

Rule 2 (HTTP):

- ✅ **Allow HTTP traffic from the internet**
- Source type: "**Anywhere**" (0.0.0.0/0)
- Port: 80 (auto-filled)
- Purpose: Allows anyone to visit your website

Optional: Add HTTPS rule (if you plan to add SSL certificate later)

- Add rule → HTTPS → Port 443 → Anywhere

Why security groups:

- Act as firewall
- Control inbound traffic (who can connect)
- Control outbound traffic (what instance can connect to)
- Can be modified later

7. **Configure Storage:**
    - **Size:** 8 GiB (default)
    - **Volume type:** gp3 (General Purpose SSD)
    - **Free tier eligible:** Up to 30 GB
    - 8 GB is sufficient for our project
    - Can be increased later if needed
8. **Advanced Details:**
    - Leave all as default
    - No changes needed for basic setup
9. **Summary:**
    - Review all settings on right panel
    - Number of instances: 1
    - Verify: Ubuntu, t2.micro, 8 GB, security groups correct

10. **Click "Launch instance"** (orange button)

What happens:

- AWS creates virtual machine
- Takes 30-60 seconds
- Instance initializes
- Operating system boots

11. **Success message:**

```
Successfully initiated launch of instance (i-xxxxxxxxxxxxx)
```

12. **Click "View all instances"**

1.3 Get Instance Details

In EC2 Dashboard:

1. **View instances:**
   - You see list of instances
   - Your instance: `portfolio-website-server`
2. **Check instance state:**
   - Wait until "**Instance state**" shows "**Running**" (green)
   - Status checks: Wait for "**2/2 checks passed**"
   - Takes 2-3 minutes
3. **Get Public IP Address:**
   - Click on your instance (checkbox or row)
   - Bottom panel shows instance details
   - Find "**Public IPv4 address**"
   - Example: `65.2.181.108`
   - **Copy this IP address** - you'll need it for:
     - SSH connection
     - Website access
     - Nginx configuration

4. **Note other details:**
    - **Instance ID:** `i-xxxxxxxxxxxxx` (unique identifier)
    - **Public IPv4 DNS:** `ec2-xx-xx-xx-xx.compute-1.amazonaws.com`
    - **Key pair name:** `shubham-portfolio-key`
    - **Security groups:** `portfolio-web-server-sg`

1.4 Manage Key File

Current location:

```
C:\Users\shubh\Downloads\shubham-portfolio-key.pem
```

Recommended location:

```
C:\Users\shubh\.ssh\shubham-portfolio-key.pem
```

Why move to .ssh folder:

- Standard location for SSH keys
- Organized (all keys in one place)
- Easy to find
- Follows Unix conventions

Move using Command Prompt:

```
# Create .ssh folder if doesn't exist
mkdir C:\Users\shubh\.ssh

# Move key file
move C:\Users\shubh\Downloads\shubham-portfolio-key.pem C:\Users\shubh\.ssh\
```

Move using File Explorer:

1. Open Downloads folder
2. Find `shubham-portfolio-key.pem`
3. Cut (Ctrl + X)
4. Navigate to `C:\Users\shubh\`

5. Create folder named `.ssh` (if doesn't exist)
6. Open `.ssh` folder
7. Paste (Ctrl + V)

Verify:

```
# List .ssh folder
ls ~/.ssh/
```

Should show: `shubham-portfolio-key.pem`

SECTION 2: CONNECTING TO EC2

2.1 Open Git Bash

Why Git Bash:

- Provides Unix-like terminal on Windows
- Has SSH built-in
- Supports Unix commands
- Better for server administration

Open Git Bash:

1. Press `Windows` key
2. Type "Git Bash"
3. Click "Git Bash" application
4. Terminal window opens

2.2 Navigate to Key Location

Go to .ssh folder:

```
# Navigate to .ssh folder
cd ~/.ssh

# List files
ls -la
```

Expected output:

```
-rw-r--r-- 1 shubh shubh 1674 Jan 30 10:00 shubham-portfolio-key.pem
```

If key is still in Downloads:

```
cd ~/Downloads
ls -la | grep .pem
```

2.3 Set Correct Permissions

Why change permissions:

- SSH requires private key to be read-only
- Security requirement
- Prevents unauthorized access
- If permissions too open, SSH refuses to use key

Set permissions:

```
# Make key read-only for owner
chmod 400 shubham-portfolio-key.pem

# Verify permissions changed
ls -la shubham-portfolio-key.pem
```

Expected output:

```
-r-------- 1 shubh shubh 1674 Jan 30 10:00 shubham-portfolio-key.pem
```

Permission explanation:

- `-r--------` means:
  - Owner: Read only (r)
  - Group: No permissions
  - Others: No permissions
- `400` in octal notation
- Most secure setting for private keys

2.4 Connect to EC2 Instance

SSH Command:

```
ssh -i "shubham-portfolio-key.pem" ubuntu@YOUR-EC2-PUBLIC-IP
```

Replace `YOUR-EC2-PUBLIC-IP` with actual IP!

Example:

```
ssh -i "shubham-portfolio-key.pem" ubuntu@65.2.181.108
```

Command breakdown:

- `ssh` - Secure Shell (remote connection protocol)
- `-i` - Identity file (specify private key)
- `"shubham-portfolio-key.pem"` - Your private key file
- `ubuntu` - Username (default for Ubuntu AMI)
- `@65.2.181.108` - Server IP address

First time connecting:

You'll see security warning:

```
The authenticity of host '65.2.181.108 (65.2.181.108)' can't be established.
ECDSA key fingerprint is SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Type: `yes` and press **Enter**

What this does:

- Adds server to known_hosts file
- Future connections won't ask again
- Security measure to prevent man-in-the-middle attacks

Successful connection:

```
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-1028-aws x86_64)
```

```
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Jan 30 10:30:00 UTC 2026

  System load:  0.0               Processes:             98
  Usage of /:   20.1% of 7.57GB   Users logged in:       0
  Memory usage: 20%               IPv4 address for eth0: 172.31.x.x
  Swap usage:   0%

Last login: Thu Jan 30 10:25:00 2026 from xxx.xxx.xxx.xxx
ubuntu@ip-172-31-x-x:~$
```

You're now connected!

Prompt changed to:

```
ubuntu@ip-172-31-x-x:~$
```

- `ubuntu` - Current user
- `ip-172-31-x-x` - Instance hostname (internal IP)
- `~` - Current directory (home directory)
- `$` - Regular user (not root)

2.5 Verify You're on EC2

Check system information:

```
# Display hostname
hostname
```

Output: `ip-172-31-x-x`

```
# Display OS information
cat /etc/os-release
```

Shows: Ubuntu version details

```
# Check public IP
curl ifconfig.me
```

Shows: Your EC2 public IP (65.2.181.108)

```
# Check available disk space
df -h
```

Shows: 8 GB storage

```
# Check CPU and RAM
free -h
```

Shows: ~1 GB RAM

2.6 Troubleshooting Connection Issues

Error: "Permission denied (publickey)"

Cause: Key file has wrong permissions or wrong key file

Solutions:

1. **Check key permissions:**

   ```
   ls -la ~/.ssh/shubham-portfolio-key.pem
   ```

   Should be: `-r--------` (read-only for owner)

2. **Fix permissions:**

   ```
   chmod 400 ~/.ssh/shubham-portfolio-key.pem
   ```

3. **Verify key path:**

   ```
   # Use full path
   ssh -i "/c/Users/shubh/.ssh/shubham-portfolio-key.pem" ubuntu@YOUR-IP
   ```

4. **Verify username:**
    - Ubuntu AMI uses `ubuntu` user
    - Amazon Linux uses `ec2-user`
    - Make sure you're using `ubuntu`

Error: "Connection timed out"

Cause: Security group doesn't allow SSH from your IP

Solutions:

1. **Check security group:**
    - AWS Console → EC2 → Security Groups
    - Select `portfolio-web-server-sg`
    - Check Inbound rules
    - SSH (Port 22) should allow your IP
2. **Update security group:**
    - Edit inbound rules
    - SSH rule → Source type → **My IP**
    - Save changes
    - Try connecting again
3. **Check your current IP:**

```
curl ifconfig.me
```

    Your IP might have changed (dynamic IP from ISP)

4. **Temporary fix:**
    - Change SSH source to **Anywhere** (0.0.0.0/0)
    - ⚠️ Less secure, but works
    - Change back to your IP after connecting

Error: "Host key verification failed"

Cause: Server key changed or IP reused

Solution:

```
# Remove old host key
ssh-keygen -R YOUR-EC2-IP

# Try connecting again
ssh -i "shubham-portfolio-key.pem" ubuntu@YOUR-EC2-IP
```

Error: "Connection refused"

Cause: Instance not running or SSH service not started

Solutions:

1. **Check instance state:**
   - AWS Console → EC2 → Instances
   - Verify state is "Running"
   - Check status checks (2/2 passed)
2. **Reboot instance:**
   - Select instance
   - Actions → Instance State → Reboot
3. **Wait a few minutes and try again**

2.7 Disconnect and Reconnect

To disconnect:

```
# Type exit or press Ctrl+D
exit
```

Prompt returns to your local machine.

To reconnect:

```
# Same SSH command
ssh -i "~/.ssh/shubham-portfolio-key.pem" ubuntu@YOUR-EC2-IP
```

Keep terminal open:

- Don't close terminal while working
- If disconnected accidentally, just reconnect
- Your work on server is saved

SECTION 3: INSTALLING DEPENDENCIES ON EC2

⚠️ **Important:** All commands in this section run on EC2 (Ubuntu), NOT on your Windows machine!

Verify you're on EC2:

- Prompt shows: `ubuntu@ip-xxx-xxx-xxx-xxx:~$`
- Not: `shubh@DESKTOP-XXX` or similar

3.1 Update System Packages

Why update first:

- Gets latest package information
- Fixes security vulnerabilities
- Ensures compatibility
- Best practice before installing anything

Update package list:

```
sudo apt update
```

Command breakdown:

- `sudo` - Run as administrator (super user do)
- `apt` - Package manager for Ubuntu
- `update` - Refresh package database

Output:

```
Hit:1 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
...
Fetched 15.0 MB in 3s (5000 kB/s)
Reading package lists... Done
```

```
Building dependency tree... Done
All packages are up to date.
```

Optional - Upgrade packages:

```
# Upgrade all installed packages (optional)
sudo apt upgrade -y
```

- `-y` flag auto-answers "yes" to all prompts
- Takes 5-10 minutes
- Downloads and installs updates
- Not required but recommended

3.2 Install Node.js and npm

Why needed:

- Build your React project
- Install dependencies (npm install)
- Create production build (npm run build)

Install:

```
# Install Node.js and npm
sudo apt install -y nodejs npm
```

- `-y` flag skips confirmation prompts
- Installs both Node.js and npm
- Takes 2-3 minutes

Output:

```
Reading package lists... Done
Building dependency tree... Done
...
Setting up nodejs (12.22.9~dfsg-1ubuntu3) ...
Setting up npm (8.5.1~ds-1) ...
Processing triggers for man-db (2.10.2-1) ...
```

Verify installation:

```
# Check Node.js version
node -v
```

Expected: `v12.22.9` (or similar)

```
# Check npm version
npm -v
```

Expected: `8.5.1` (or similar)

Note: Ubuntu's default Node.js is older than on Windows. That's okay - it works fine for building React projects.

If you need newer version (optional):

```
# Add Node.js 20.x repository
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -

# Install Node.js 20
sudo apt-get install -y nodejs

# Verify version
node -v
```

Should show: `v20.x.x`

3.3 Install Nginx Web Server

Why Nginx:

- Serves static files (HTML, CSS, JS)
- Fast and efficient
- Industry standard
- Perfect for React apps

Install Nginx:

```
# Install Nginx
```

```
sudo apt install -y nginx
```

Takes 1-2 minutes.

Output:

```
Reading package lists... Done
Building dependency tree... Done
...
Setting up nginx (1.18.0-6ubuntu14) ...
Created symlink /etc/systemd/system/multi-user.target.wants/nginx.service
```

Start Nginx service:

```
# Start Nginx
sudo systemctl start nginx

# Enable Nginx to start automatically on boot
sudo systemctl enable nginx
```

Check Nginx status:

```
# Check if Nginx is running
sudo systemctl status nginx
```

Expected output:

```
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled)
     Active: active (running) since Thu 2026-01-30 10:30:00 UTC; 5s ago
       Docs: man:nginx(8)
    Process: 1234 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on;
    Process: 1235 ExecStart=/usr/sbin/nginx -g daemon on;
   Main PID: 1236 (nginx)
      Tasks: 2 (limit: 1141)
     Memory: 5.0M
        CPU: 10ms
     CGroup: /system.slice/nginx.service
             ├─1236 nginx: master process /usr/sbin/nginx -g daemon on;
             └─1237 nginx: worker process

Jan 30 10:30:00 systemd[1]: Starting A high performance web server...
Jan 30 10:30:00 systemd[1]: Started A high performance web server.
```

Look for:

- ✅ `Active: active (running)` in **green**
- ✅ `enabled` (starts on boot)

Press `q` to exit status view

3.4 Install Git

Why needed:

- Clone your code from GitHub
- Pull updates later
- Version control on server

Install Git:

```
# Install Git
sudo apt install -y git
```

Takes 1 minute.

Verify installation:

```
# Check Git version
git --version
```

Expected: `git version 2.34.1` (or similar)

3.5 Test Nginx is Working

Get your public IP:

```
# Display public IP
curl ifconfig.me
```

Shows: `65.2.181.108` (your EC2 IP)

Test Nginx from server itself:

```
# Request default Nginx page
curl localhost
```

Should output HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Test from browser:

1. **Open browser on your Windows machine**
2. **Go to:** `http://YOUR-EC2-PUBLIC-IP`
   - Example: `http://65.2.181.108`
3. **You should see:** "Welcome to nginx!" page

If you see this page: ✅ Nginx is working correctly
✅ Port 80 is open in security group
✅ Ready to deploy your website

If you don't see the page:

- Check security group (HTTP port 80 open?)
- Check Nginx is running: `sudo systemctl status nginx`
- Try: `sudo systemctl restart nginx`

SECTION 4: CLONING FROM GITHUB

4.1 Navigate to Home Directory

Verify current location:

```
# Show current directory
pwd
```

Expected output: `/home/ubuntu`

If not in home directory:

```
# Go to home directory
cd ~

# Verify
pwd
```

Why home directory:

- Standard location for user files
- Full permissions (no sudo needed)
- Easy to find
- Organized structure

4.2 Clone Your Repository

Clone command:

```
# Clone from GitHub
git clone https://github.com/ShubhamCyberStack/portfolio-website.git
```

Replace with YOUR repository URL!

What happens:

```
Cloning into 'portfolio-website'...
remote: Enumerating objects: 50, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (45/45), done.
remote: Total 50 (delta 5), reused 50 (delta 5), pack-reused 0
Receiving objects: 100% (50/50), 15.00 KiB | 5.00 MiB/s, done.
Resolving deltas: 100% (5/5), done.
```

Time taken: 5-10 seconds

What was cloned:

- All source files (src/)
- Configuration (package.json)
- Documentation (README.md)

- Git history (all commits)
- **NOT cloned:** node_modules (will install separately)

4.3 Navigate into Project

```
# Enter project folder
cd portfolio-website

# Verify location
pwd
```

Expected: `/home/ubuntu/portfolio-website`

List files:

```
# List all files
ls -la
```

You should see:

```
.git/                   # Git repository
public/                 # Public files
src/                    # Source code
.gitignore              # Git ignore rules
package.json            # Dependencies
package-lock.json       # Locked versions
README.md               # Documentation
```

Verify folders:

```
# Check src folder
ls src/

# Check components
ls src/components/

# Check data
ls src/data/
```

All your files should be present!

4.4 Verify Repository Contents

Check Git status:

```
git status
```

Output:

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Check Git log:

```
git log --oneline
```

Shows your commit history:

```
abc1234 (HEAD -> main, origin/main) Initial commit: Complete portfolio website with React
```

Verify package.json:

```
cat package.json
```

Shows your project configuration.

SECTION 5: BUILDING FOR PRODUCTION

5.1 Install Dependencies

Why install dependencies:

- node_modules folder not in Git
- Need to download React, Webpack, Babel, etc.
- Required before building

Install command:

```
# Make sure you're in project folder
pwd
# Should show: /home/ubuntu/portfolio-website

# Install all dependencies
npm install
```

What happens:

```
npm WARN deprecated stable@0.1.8: Modern JS already guarantees Array#sort()...
npm WARN deprecated rollup-plugin-terser@7.0.2: This package has been deprecated...

added 1476 packages from 739 contributors in 156s

50 packages are looking for funding
  run `npm fund` for details
```

Time taken: 2-5 minutes

What's installed:

- React and React-DOM
- Webpack (bundler)
- Babel (transpiler)
- ESLint (linting)
- 1476 packages total (~250 MB)

Warnings are normal:

- Yellow warnings: Usually safe to ignore
- Deprecated packages: Old versions, still work
- Red errors: These need fixing

Verify installation:

```
# Check node_modules folder was created
ls -la

# Check size
du -sh node_modules/
```

Shows: ~250M (node_modules)

5.2 Build Production Version

What build does:

- Compiles JSX to JavaScript
- Bundles all files
- Minifies code
- Optimizes images
- Creates build/ folder

Build command:

```
npm run build
```

What happens:

```
> portfolio-website@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  52.47 KB  build/static/js/main.a12b34c5.js
  1.78 KB   build/static/css/main.d67e89f0.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:

  https://cra.link/deployment
```

Time taken: 1-3 minutes

Build output explanation:

File sizes:

- `main.a12b34c5.js` - All your JavaScript (React + your code)
- `main.d67e89f0.css` - All your CSS combined
- Sizes shown are AFTER gzip compression
- Actual files slightly larger (~150 KB JS, ~5 KB CSS)

File hashes:

- `a12b34c5` and `d67e89f0` are random hashes
- Change every build
- Purpose: Cache busting (browsers download new version)

5.3 Verify Build Folder

Check build folder was created:

```
# List all files
ls -la

# Should see build/ folder
```

Explore build contents:

```
# List build folder contents
ls -la build/
```

You should see:

```
asset-manifest.json    # List of all built files
favicon.ico            # Browser icon
index.html             # Main HTML (with script tags added)
logo192.png            # PWA icons
logo512.png
manifest.json          # PWA config
robots.txt             # SEO file
static/                # All CSS and JS files
```

Check static folder:

```
# List static folder
ls -la build/static/
```

You should see:

```
css/                    # Compiled CSS
js/                     # Bundled JavaScript
media/                  # Images, fonts (if any)
```

Check JavaScript files:

```
ls -la build/static/js/
```

You should see:

```
main.a12b34c5.js        # Your code + React
main.a12b34c5.js.map    # Source map (for debugging)
runtime-main.xxx.js     # Webpack runtime
```

Verify index.html has script tags:

```
cat build/index.html
```

Should contain:

```
<script src="/static/js/main.a12b34c5.js"></script>
```

5.4 Understanding Build Size

Compare sizes:

Original (development):

```
du -sh node_modules/    # ~250 MB
du -sh src/             # ~500 KB
```

```
# Total: ~250 MB
```

Built (production):

```
du -sh build/           # ~500 KB
```

Reduction: 250 MB → 500 KB = 99.8% reduction!

Why so small:

- Only necessary code included
- Minified (whitespace removed)
- Compressed
- Tree-shaking (unused code removed)
- Webpack optimization

This is why we build:

- Fast downloads
- Quick loading
- Better performance
- Smaller bandwidth usage

SECTION 6: NGINX CONFIGURATION

6.1 Understand Nginx Directory Structure

Default Nginx web root:

```
/var/www/html/
```

This is where Nginx looks for files to serve.

When someone visits `http://65.2.181.108`:

1. Nginx looks in `/var/www/html/`
2. Finds `index.html`
3. Sends it to browser

Currently contains:

- Default Nginx welcome page
- We'll replace with our React build files

6.2 Clear Default Nginx Files

Remove default files:

```
# Remove everything in web root
sudo rm -rf /var/www/html/*
```

Command breakdown:

- `sudo` - Need admin rights
- `rm` - Remove
- `-rf` - Recursive (folders too), force (no confirmation)
- `/var/www/html/*` - Everything in that folder

Verify it's empty:

```
# List contents
ls -la /var/www/html/
```

Expected output:

```
total 8
drwxr-xr-x 2 root root 4096 Jan 30 11:00 .
drwxr-xr-x 3 root root 4096 Jan 30 10:30 ..
```

Only `.` and `..` (empty folder indicators)

6.3 Copy Build Files to Nginx

Copy all build files:

```
# Copy build contents to Nginx web root
sudo cp -r build/* /var/www/html/
```

Command breakdown:

- `sudo` - Admin rights required
- `cp` - Copy
- `-r` - Recursive (copy folders and contents)
- `build/*` - Everything in build folder
- `/var/www/html/` - Destination

Verify files were copied:

```
# List Nginx web root
ls -la /var/www/html/
```

You should see:

```
asset-manifest.json
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
static/
```

All your build files are now in Nginx directory!

6.4 Set Proper Permissions

Why set permissions:

- Nginx runs as `www-data` user
- Needs permission to read files
- Security best practice

Set owner:

```
# Change owner to www-data
sudo chown -R www-data:www-data /var/www/html
```

- `chown` - Change owner

- `-R` - Recursive (all files and folders)
- `www-data:www-data` - User and group

Set permissions:

```
# Set permissions: owner=rwx, group=rx, others=rx
sudo chmod -R 755 /var/www/html
```

- `chmod` - Change mode (permissions)
- `-R` - Recursive
- `755` - Owner can read/write/execute, others can read/execute

Verify permissions:

```
ls -la /var/www/html/
```

Expected output:

```
drwxr-xr-x  3 www-data www-data  4096 Jan 30 11:00 .
-rw-r--r--  1 www-data www-data  1234 Jan 30 11:00 index.html
drwxr-xr-x  4 www-data www-data  4096 Jan 30 11:00 static
...
```

Permission explanation:

- `drwxr-xr-x` for folders
  - Owner (www-data): read, write, execute
  - Group: read, execute
  - Others: read, execute
- `-rw-r--r--` for files
  - Owner: read, write
  - Group: read
  - Others: read

6.5 Configure Nginx for React

Why special configuration needed:

React is a Single Page Application (SPA):

- Only one HTML file (`index.html`)
- React handles all routing with JavaScript
- But Nginx doesn't know this!

The problem:

- User visits: `http://yoursite.com/about`
- Nginx looks for: `/var/www/html/about` file
- File doesn't exist → 404 Error

The solution:

- Configure Nginx to always serve `index.html`
- React's JavaScript handles the routing
- User sees the correct page

Edit Nginx configuration:

```
# Open default Nginx config file
sudo nano /etc/nginx/sites-available/default
```

Nano editor opens (text editor in terminal)

Delete all existing content:

- Press `Ctrl + K` repeatedly to delete each line
- Or press `Ctrl + K` and hold until file is empty

Paste this configuration:

```
server {
    listen 80;
    listen [::]:80;
    server_name _;

    root /var/www/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    error_page 404 /index.html;
```

```
    # Cache static assets
    location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }
}
```

Save and exit:

1. Press `Ctrl + X` (exit)
2. Press `Y` (yes, save)
3. Press `Enter` (confirm filename)

6.6 Configuration Explained

Line-by-line explanation:

```
listen 80;
listen [::]:80;
```

- Listen on port 80 (HTTP)
- First line: IPv4
- Second line: IPv6

```
server_name _;
```

- Underscore (_) is wildcard
- Accept any domain name or IP
- Matches all requests

```
root /var/www/html;
```

- Document root (where files are)
- Nginx looks here for files

```
index index.html;
```

- Default file to serve

- If request is for `/`, serve `index.html`

```
location / {
    try_files $uri $uri/ /index.html;
}
```

Most important part for React!

- `location /` - Applies to all URLs
- `try_files` - Try these files in order:
  i. `$uri` - Exact file (e.g., `style.css`)
  ii. `$uri/` - Directory with index
  iii. `/index.html` - Fallback (React routing)

Example:

- Request: `/about`
- Try: `/var/www/html/about` (file doesn't exist)
- Try: `/var/www/html/about/` (folder doesn't exist)
- Serve: `/var/www/html/index.html` (found!)
- React sees URL is `/about`
- React shows About component

```
error_page 404 /index.html;
```

- Even on 404 errors, serve index.html
- React shows "Page Not Found" component
- Better UX than default Nginx 404 page

```
location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
}
```

Performance optimization:

- `~*` - Case-insensitive regex match
- Applies to: images, CSS, JavaScript files
- `expires 1y` - Tell browser to cache for 1 year

- `Cache-Control "public, immutable"` - File never changes (due to hashes)
- Result: Faster loading for returning visitors

6.7 Test Configuration

Check syntax:

```
# Test Nginx configuration
sudo nginx -t
```

Expected output:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

✅ Both lines should say "ok" and "successful"

If you see errors:

- Read error message carefully
- Usually points to line number with problem
- Common issues:
  - Missing semicolon (`;`)
  - Missing bracket (`{` or `}`)
  - Typo in directive name
- Fix error, save, test again

Check for common mistakes:

```
# Count opening braces
grep -c '{' /etc/nginx/sites-available/default

# Count closing braces
grep -c '}' /etc/nginx/sites-available/default

# Should be equal!
```

6.8 Restart Nginx

Apply new configuration:

```
# Restart Nginx service
sudo systemctl restart nginx
```

No output = success

Check Nginx is running:

```
# Check status
sudo systemctl status nginx
```

Expected output:

```
● nginx.service - A high performance web server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled)
     Active: active (running) since Thu 2026-01-30 11:05:00 UTC; 5s ago
    Process: 5678 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on;
    Process: 5679 ExecStart=/usr/sbin/nginx -g daemon on;
   Main PID: 5680 (nginx)
      Tasks: 2
     Memory: 5.2M
        CPU: 12ms
     CGroup: /system.slice/nginx.service
             ├─5680 nginx: master process /usr/sbin/nginx -g daemon on;
             └─5681 nginx: worker process

Jan 30 11:05:00 systemd[1]: Starting A high performance web server...
Jan 30 11:05:00 systemd[1]: Started A high performance web server.
```

Look for:

- ✅ `Active: active (running)` in **green**
- ✅ Recent start time
- ✅ No error messages

Press `q` to exit

If Nginx failed to start:

```
# Check error logs
sudo tail -f /var/log/nginx/error.log
```

- Read last few lines
- Fix the issue mentioned
- Restart again

Common issues:

- Port 80 already in use
- Configuration syntax error
- Permission issues

SECTION 7: DEPLOYMENT COMPLETE!

7.1 Get Your Public IP

On EC2:

```
# Display public IP address
curl ifconfig.me
```

Output:

```
65.2.181.108
```

Copy this IP address!

7.2 Access Your Live Website

Open browser (on your Windows machine):

1. **Open any web browser** (Chrome, Firefox, Edge)
2. **Type in address bar:**

   ```
   http://YOUR-EC2-PUBLIC-IP
   ```

3. **Example:**

   ```
   http://65.2.181.108
   ```

4. **Press Enter**

7.3 What You Should See

Your complete portfolio website:

✅ **Header Section**

- Purple gradient background
- Your name in large text
- Your title/headline
- Profile photo (circular)
- Social links (LinkedIn, GitHub, Email)

✅ **About Me Section**

- Light gray background
- Your description/summary
- Decorative purple underline on title

✅ **Skills Section**

- Grid of colorful cards
- Each skill in purple gradient
- Hover effect (lift and grow)

✅ **Work Experience Section**

- Timeline layout with vertical line
- QualityNZ Education internship
- Circular markers on timeline
- Responsibilities in bullet points

✅ **Education Section**

- Card layout
- PIET institution info
- Left purple border accent

✅ **Projects Section**

- Project cards
- Technology tags
- GitHub links

✅ **Footer Section**

- Dark background
- Copyright notice
- "Built with React & deployed on AWS EC2 with Nginx"
- Social links

7.4 Test Website Functionality

Test all features:

1. Click Social Links

- **LinkedIn** → Should open your LinkedIn profile in new tab
- **GitHub** → Should open your GitHub profile in new tab
- **Email** → Should open email client with your email address

2. Test Hover Effects

- Hover over skill cards → Should lift up and grow
- Hover over social links → Background changes to white
- Hover over project links → Color changes

3. Test Scrolling

- Scroll up and down
- Should be smooth
- All sections visible and styled correctly

4. Test on Mobile

- Open website on phone
- Or use browser DevTools (F12, toggle device toolbar)
- Layout should adapt:
  - Single column on mobile
  - Smaller text sizes

    - Stacked elements
    - Smaller profile image

7.5 Verify Deployment is Successful

Checklist:

- [ ] Website loads without errors
- [ ] All sections visible
- [ ] Your information displays correctly
- [ ] Links work
- [ ] Images load (if you added any)
- [ ] Styling looks correct
- [ ] Responsive on mobile
- [ ] Fast loading (under 2 seconds)

If anything is wrong:

- Check browser Console for errors (F12 → Console)
- Check Nginx is running: `sudo systemctl status nginx`
- Check files exist: `ls -la /var/www/html/`
- Check Nginx error log: `sudo tail /var/log/nginx/error.log`

7.6 Troubleshooting

Issue: Can't Access Website

Check 1: Security Group

1. AWS Console → EC2 → Security Groups
2. Select `portfolio-web-server-sg`
3. Inbound rules should have:
    - HTTP (80) → 0.0.0.0/0
4. If missing, add the rule and save

Check 2: Nginx Running

```
sudo systemctl status nginx
```

Should be "active (running)"

Check 3: Files Exist

```
ls -la /var/www/html/
```

Should have index.html and static/ folder

Check 4: Test from Server

```
curl localhost
```

Should output HTML

Issue: Website Shows Blank Page

Check browser Console:

1. Press F12
2. Console tab
3. Look for errors

Common causes:

- JavaScript files not loading (404 errors)
- File paths incorrect
- Permissions issue

Fix:

```
# Re-copy build files
sudo cp -r ~/portfolio-website/build/* /var/www/html/
sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 755 /var/www/html
sudo systemctl restart nginx
```

Issue: 404 Error on Subpages

Cause: Nginx configuration incorrect

Fix:

- Review Section 6.5 (Nginx configuration)
- Make sure `try_files $uri $uri/ /index.html;` is present
- Test config: `sudo nginx -t`
- Restart: `sudo systemctl restart nginx`

SECTION 8: ACCESSING YOUR LIVE WEBSITE

8.1 Your Website URL

Your website is accessible at:

```
http://YOUR-EC2-PUBLIC-IP
```

Example:

```
http://65.2.181.108
```

This URL is:

- ✅ Accessible from anywhere in the world
- ✅ Available 24/7 (as long as EC2 is running)
- ✅ Fast (served from AWS data center)
- ✅ Professional (real production deployment)

8.2 Monitor Your Website

Check if Website is Up

From any computer:

```
# Ping EC2 instance
ping YOUR-EC2-IP

# Check HTTP response
curl -I http://YOUR-EC2-IP
```

Should return:

```
HTTP/1.1 200 OK
Server: nginx/1.18.0
...
```

Check Access Logs

On EC2:

```
# View recent visitors
sudo tail -f /var/log/nginx/access.log
```

Shows:

- IP addresses of visitors
- Pages accessed
- Timestamps
- User agents (browsers)

Example log entry:

```
192.168.1.100 - - [30/Jan/2026:11:30:00 +0000] "GET / HTTP/1.1" 200 1234 "-" "Mozilla/5.0..."
```

Check Error Logs

```
# View errors (if any)
sudo tail -f /
```