LearnGrid

Complete Project & Learning Guide

Everything you need to understand your project —
React, JavaScript, JSX, Shell Scripting & how it all works together

09	**Shell Scripting**	How .bat files work internally on Windows
10	**Task Scheduler**	How Windows auto-starts your app
11	**The Full Flow**	Everything connected — boot to browser
12	**Next Steps**	What to learn next to grow as a developer

Built with React · Vite · JavaScript · GitHub API · Windows Batch Scripts

CHAPTER 01

What We Built

A personal daily learning tracker — like GitHub's contribution graph but for your own learning journey.

The Big Picture

LearnGrid is a web application that runs on your laptop. It tracks every day you learn something — whether you write a note, upload a PDF, push code to GitHub, or solve a LeetCode problem. All of this activity shows up on a heatmap exactly like the one on your GitHub profile.

Features we built:

- GitHub-style contribution heatmap
- Write and save daily notes
- Upload PDFs to GitHub storage
- Connect your GitHub account
- Connect your LeetCode account
- Color-coded activity by type
- Current & best streak tracking
- Active days counter
- Data persists after browser refresh
- Auto-starts when laptop boots

The Tech Stack

Here are all the technologies used and what each one does:

Technology	Role	Analogy
React	Builds the UI (buttons, heatmap, cards)	Like LEGO bricks for web pages
JavaScript	The programming language everything is written in	The brain of the app
JSX	HTML-like syntax inside JavaScript files	A shortcut to write UI faster
Vite	Runs the dev server, builds the final app	Like a factory that packages your app
localStorage	Saves data in your browser permanently	Like a notepad stored in your browser
GitHub API	Fetches your GitHub commit activity	Asking GitHub: what did this user do?
Batch Script	Starts the server automatically on Windows	A recipe of commands for Windows to run

Task Scheduler	Runs the batch script on every boot	An alarm clock for your computer
serve	A tiny web server that hosts your built app	A waiter that serves your app to the browser

CHAPTER 02

How the Web Works

Before React, understand what a browser actually does and what HTML, CSS, and JavaScript are.

The Three Languages of the Web

Every website — no matter how complex — is made of three things at its core. Think of building a house:

Language	Role in a house	Role in a website
HTML	The structure — walls, rooms, floors	The content — headings, buttons, text, images
CSS	The decoration — paint, furniture, style	The design — colors, fonts, spacing, layout
JavaScript	The electricity — lights, doors, appliances that move	The behaviour — clicks, animations, fetching data

What happens when you open localhost:5173?

When you type localhost:5173 in your browser, this is what happens step by step:

1. Your browser sends a request to the local server running on your laptop (the 'serve' program)
2. The server responds with index.html — a very simple file that just loads your app
3. index.html tells the browser to load main.jsx (your React app)
4. React runs in the browser and builds the entire UI you see
5. When you click, type, or interact — JavaScript handles everything in real time
6. No page reloads happen — React updates only what changed (this is called a SPA — Single Page Application)

Key insight: React never actually reloads the page. It just updates parts of the screen. This is why it feels so fast and smooth.

CHAPTER 03

JavaScript Basics

JavaScript is the language the entire app is written in. Here are the key concepts you need to know.

Variables

Variables store data. We use three keywords:

```
// const — value never changes (use this most of the time)
const name = 'Shivam';
const age = 20;

// let — value can change
let score = 0;
score = score + 1;   // now score is 1

// var — old way, avoid using it
var old = 'dont use this';
```

Functions

Functions are reusable blocks of code. There are two ways to write them:

```
// Traditional function
function greet(name) {
  return 'Hello ' + name;
}

// Arrow function (modern, used in our app)
const greet = (name) => {
  return 'Hello ' + name;
};

// Short arrow function (when return is one line)
const greet = (name) => 'Hello ' + name;

// Calling a function
const result = greet('Shivam');   // result = 'Hello Shivam'
```

Arrays

Arrays store lists of things. In our app we use them to store entries:

```
const entries = ['note1', 'note2', 'note3'];

// Access by index (starts at 0)
entries[0]   // 'note1'

// Common array methods used in our app:
entries.filter(e => e !== 'note1')  // removes note1
entries.map(e => e.toUpperCase())   // transforms each item
entries.slice(0, 2)                 // first 2 items only
[newEntry, ...entries]              // add item at start (spread operator)
```

Objects

Objects store related data together. Every entry in our app is an object:

```
// An entry object in our app looks like this:
const entry = {
  id:      1708123456,
  date:    '2026-02-19',
  title:   'Learned React today',
  content: 'useState is for managing state...',
  type:    'notes',
  pdf:     null,
};

// Access object properties
entry.title    // 'Learned React today'
entry['date']  // '2026-02-19'
```

Async / Await — Fetching Data

When we fetch data from GitHub or LeetCode, we have to wait for the response. JavaScript uses async/await for this:

```
// async means this function will do something that takes time
async function fetchGitHubActivity(username) {

  // await means: pause here and wait for the response
  const response = await fetch('https://api.github.com/users/' + username + '/events')
;
```

```
  // convert response to JSON (JavaScript object)
  const data = await response.json();

  return data;   // return the result
}

// Calling an async function
const activity = await fetchGitHubActivity('shivam');
```

CHAPTER 04

React from Scratch

React is a JavaScript library for building user interfaces. Think of it as building with smart LEGO bricks.

What is a Component?

Everything in React is a component — a reusable piece of UI. A component is just a JavaScript function that returns what should be shown on screen:

```
// A simple React component
function Button() {
  return <button>Click me</button>;
}

// Using it inside another component
function App() {
  return (
    <div>
      <Button />
      <Button />
    </div>
  );
}
```

Notice: HTML-like code inside JavaScript. That's JSX — explained next.

What is JSX?

JSX stands for JavaScript XML. It lets you write HTML-like code directly inside JavaScript. It's not actually HTML — the browser never sees it. Vite converts it to regular JavaScript before the browser gets it:

```
// What you write (JSX):
const element = <h1 style={{ color: 'green' }}>Hello Shivam</h1>;

// What Vite converts it to (real JavaScript):
```

```
const element = React.createElement('h1', { style: { color: 'green' } }, 'Hello Shivam
');

// JSX Rules:
// 1. Use className instead of class
// 2. All tags must be closed: <br /> not <br>
// 3. Return ONE parent element (or use <> </> fragment)
// 4. JavaScript goes inside { } curly braces
```

Props — Passing Data to Components

Props are how you pass data from a parent component to a child:

```
// Child component receives props
function EntryCard({ title, date, type }) {
  return (
    <div>
      <span>{type}</span>
      <h3>{title}</h3>
      <p>{date}</p>
    </div>
  );
}

// Parent passes props like HTML attributes
function App() {
  return <EntryCard title='React Notes' date='Feb 19' type='NOTE' />;
}
```

CHAPTER 05

Our App — File by File

Every file in your learngrid project and exactly what it does.

Project Structure

```
learngrid/
  src/
    App.jsx        ← The entire app (all components, logic, styles)
    main.jsx       ← Entry point — mounts React into index.html
  public/
    vite.svg       ← Default icon (can replace with your own)
  index.html       ← The one HTML file. React injects itself here.
  vite.config.js   ← Vite configuration
  package.json     ← Project info + list of all dependencies
  node_modules/    ← All installed packages (never edit this)
  dist/            ← Built app (created by npm run build)
```

index.html — The Shell

This is the only HTML file. It's almost empty — just a container for React:

```
<!DOCTYPE html>
<html>
  <head>
    <title>LearnGrid</title>
  </head>
  <body>
    <div id='root'></div>   <!-- React mounts here -->
    <script src='/src/main.jsx'></script>
  </body>
</html>
```

main.jsx — The Entry Point

This file starts React and attaches it to the HTML:

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'

// Find the <div id='root'> and put our App inside it
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>
)
```

App.jsx — The Brain

This is where everything lives. It's one big file containing all the components, all the logic, all the styles. Here's how it's organized:

```
// 1. IMPORTS — bring in React hooks
import { useState, useEffect, useRef } from 'react';

// 2. HELPER FUNCTIONS — pure utility functions
const today = () => new Date().toISOString().slice(0, 10);
function buildGrid(activityMap) { ... }
function calcStats(activityMap) { ... }

// 3. localStorage HELPERS — save and load data
function load(key, fallback) { ... }
function save(key, val) { ... }

// 4. API FUNCTIONS — fetch GitHub and LeetCode data
async function fetchGitHubActivity(username) { ... }
async function fetchLeetCodeActivity(username) { ... }
async function uploadPDFToGitHub(...) { ... }

// 5. MAIN COMPONENT — the App itself
export default function App() {
  // State declarations...
  // useEffect hooks...
  // Handler functions...
  // Return JSX (the UI)...
}

// 6. STYLES — CSS-in-JS object
const styles = { root: {...}, sidebar: {...}, ... }
const css = `@import url(...) ...`
```

CHAPTER 06

React Concepts Used

The three most important React hooks — useState, useEffect, useRef — explained with examples from our app.

useState — Remembering Things

useState lets a component remember a value. When the value changes, React automatically re-renders the UI to show the new value. Without useState, the UI would never update:

```
// Syntax
const [value, setValue] = useState(initialValue);
//     ↑             ↑               ↑
//  current value   function to       what it starts as
//                  change it

// Examples from our app:
const [entries,  setEntries]  = useState([]);       // list of notes
const [view,     setView]     = useState('home');   // which page is shown
const [noteText, setNoteText] = useState('');       // text in the textarea
const [toast,    setToast]    = useState(null);     // notification message

// How it works in practice:
// User clicks 'Add Entry' button
// → setView('note') is called
// → React sees view changed from 'home' to 'note'
// → React re-renders App and shows the note form
```

useEffect — Doing Things When Something Changes

useEffect runs code in response to changes. It's how we trigger side effects like rebuilding the heatmap when entries change:

```
// Syntax
useEffect(() => {
  // code to run
}, [dependencies]);
```

```
// dependencies = list of values to watch

// Example from our app — rebuild heatmap when data changes:
useEffect(() => {
  const map = {};
  for (const e of entries)  map[e.date] = (map[e.date] || 0) + 1;
  for (const [d, c] of Object.entries(githubMap)) map[d] += c;
  setActivityMap(map);
}, [entries, githubMap, leetcodeMap]);
//  ↑ runs again whenever entries, githubMap, or leetcodeMap changes

// [] = empty array means run ONCE when component first loads
useEffect(() => {
  const saved = load('lt:entries', []);
  setEntries(saved);
}, []);
```

useRef — Accessing DOM Elements Directly

useRef lets you grab a real HTML element and interact with it directly. We use it for the hidden file input:

```
const fileRef = useRef();

// Attach ref to an element
<input ref={fileRef} type='file' style={{ display: 'none' }} />

// Trigger it from anywhere — like when user clicks the drop zone
<div onClick={() => fileRef.current.click()}>
  Drop PDF here
</div>
// fileRef.current = the actual <input> HTML element
// .click() = programmatically click it to open file dialog
```

CHAPTER 07

localStorage

How data survives after you close the browser and reopen it.

What is localStorage?

localStorage is a key-value storage system built into every browser. Think of it like a small notepad that lives inside your browser and never gets erased even when you close it. Each website gets its own separate storage:

```
// Save data
localStorage.setItem('key', 'value');
localStorage.setItem('lt:entries', JSON.stringify(entries));

// Read data
const raw = localStorage.getItem('lt:entries');
const entries = JSON.parse(raw);

// Delete data
localStorage.removeItem('lt:entries');

// Clear everything
localStorage.clear();
```

Why JSON.stringify and JSON.parse?

localStorage can only store strings — not arrays or objects. JSON.stringify converts objects to strings. JSON.parse converts them back:

```
const entries = [{ id: 1, title: 'React Notes', date: '2026-02-19' }];

// Object → String (to save)
JSON.stringify(entries);
// Result: '[{"id":1,"title":"React Notes","date":"2026-02-19"}]'

// String → Object (to load)
JSON.parse('[{"id":1,"title":"React Notes","date":"2026-02-19"}]');
// Result: [{ id: 1, title: 'React Notes', date: '2026-02-19' }]
```

Our save/load helper functions:

```
// Load from localStorage safely
function load(key, fallback = null) {
  try {
    const raw = localStorage.getItem(key);
    return raw ? JSON.parse(raw) : fallback;
  } catch {
    return fallback;  // if anything goes wrong, return fallback
  }
}

// Save to localStorage safely
function save(key, val) {
  try {
    localStorage.setItem(key, JSON.stringify(val));
  } catch {}  // silently ignore if storage is full
}

// Keys we use in our app:
// 'lt:entries'     → all notes and PDFs
// 'lt:github'      → github activity map
// 'lt:leetcode'    → leetcode activity map
// 'lt:connections' → usernames and github token
```

CHAPTER 08

GitHub API

How we talk to GitHub servers to fetch your activity and upload PDFs.

What is an API?

API stands for Application Programming Interface. Think of it like a restaurant menu — you (the app) place an order, the kitchen (GitHub's servers) prepares it, and the waiter (HTTP) brings back the response:

```
// Basic fetch request structure
const response = await fetch(URL, {
  method: 'GET',       // or POST, PUT, DELETE
  headers: {           // extra info we send with the request
    'Authorization': 'Bearer TOKEN',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),  // data we send (for POST/PUT only)
});
const result = await response.json();  // read the response
```

Fetching GitHub Activity

```
async function fetchGitHubActivity(username) {
  // Hit the GitHub Events API
  const r = await fetch(
    'https://api.github.com/users/' + username + '/events/public?per_page=100'
  );
  if (!r.ok) throw new Error('User not found');
  const events = await r.json();
  // events is an array of: { type, created_at, ... }
  // Build a map: { '2026-02-19': 3, '2026-02-18': 1 }
  const map = {};
  for (const e of events) {
    const date = e.created_at.slice(0, 10);  // '2026-02-19T...' → '2026-02-19'
```

```
    map[date] = (map[date] || 0) + 1;
  }
  return map;
}
```

Uploading a PDF to GitHub

```
async function uploadPDFToGitHub(file, token, owner, repo) {
  // Convert file to base64 (GitHub API requires this format)
  const base64 = await new Promise((resolve) => {
    const reader = new FileReader();
    reader.onload = () => resolve(reader.result.split(',')[1]);
    reader.readAsDataURL(file);
  });

  // Create unique filename with timestamp
  const fileName = 'pdfs/' + Date.now() + '_' + file.name;

  // PUT request to GitHub Contents API
  const res = await fetch(
    'https://api.github.com/repos/' + owner + '/' + repo + '/contents/' + fileName,
    {
      method: 'PUT',
      headers: { Authorization: 'Bearer ' + token },
      body: JSON.stringify({
        message: 'Upload ' + file.name,
        content: base64,
      }),
    }
  );

  const data = await res.json();
  return data.content.download_url;  // permanent GitHub URL
}
```

CHAPTER 09

Shell Scripting

What shell scripting is, how .bat files work, and what happens internally when Windows runs them.

What is a Shell?

A shell is a program that lets you talk to your operating system using text commands. On Windows, the shell is CMD (Command Prompt) or PowerShell. When you type commands like 'npm run dev', the shell reads them and tells Windows what to do.

What is a .bat file?

A .bat (batch) file is just a text file containing a list of shell commands. Windows reads them one by one and executes them exactly as if you typed them manually. It's automation — a script that does the typing for you:

```
// Without bat file — you'd have to type all this every time:
cd C:\Users\shiva\project\learngrid
serve dist -p 5173

// With bat file — Windows does it automatically on startup:
@echo off
serve C:\Users\shiva\project\learngrid\dist -p 5173
```

Anatomy of our .bat files

learngrid-server.bat — starts the web server:

```
@echo off
// ↑ Tells CMD: don't print each command before running it
// Without this, you'd see every command echoed to the terminal

serve C:\Users\shiva\project\learngrid\dist -p 5173
// ↑ Runs the 'serve' program
//   'serve' is a global npm package we installed
//   It starts a tiny HTTP server
//   'dist' = serve files from this folder
```

```
//   '-p 5173' = use port 5173
```

learngrid.bat — opens the app in browser:

```
@echo off

start '' 'C:\Users\shiva\learngrid-server.bat'
// ↑ 'start' = open something in a new window/process
//   '' = no title for the window
//   Launches the server bat file in background

timeout /t 2 /nobreak >nul
// ↑ Wait 2 seconds for server to start
//   /nobreak = don't let user skip the wait
//   >nul = hide the countdown output

start http://localhost:5173
// ↑ Opens this URL in your default browser
//   Windows knows 'start' + URL = open browser
```

What happens internally when Windows runs a .bat file

1. You type 'learngrid' in CMD
2. CMD searches for 'learngrid' in PATH directories (including C:\Windows\System32)
3. It finds learngrid.bat in System32
4. CMD opens a new process and starts reading the file line by line
5. Line 1: @echo off — sets a flag to suppress command echo
6. Line 2: start '' '...\learngrid-server.bat' — Windows creates a CHILD PROCESS
7. The child process runs serve.exe with your dist folder as argument
8. serve.exe opens a TCP socket on port 5173 and listens for connections
9. Line 3: timeout waits 2000ms
10. Line 4: start http://localhost:5173 — Windows opens your default browser
11. Browser sends HTTP GET request to 127.0.0.1:5173
12. serve.exe responds with index.html from your dist folder
13. Browser loads index.html, which loads your React app
14. React runs and renders the LearnGrid UI

Key concept: serve.exe keeps running in the background as a process. It stays alive listening on port 5173 until you restart your computer or kill the process.

CHAPTER 10

Task Scheduler

How Windows Task Scheduler auto-starts your server every time you boot your laptop.

What is Task Scheduler?

Windows Task Scheduler is a built-in service that runs programs automatically based on triggers — like system startup, a specific time, or user login. It's how Windows runs background services invisibly.

What we configured:

Setting	Value	What it means
Task Name	LearnGrid Server	Identifier for the task
Trigger	When the computer starts	Runs on every Windows boot
Action	Start a program	Execute our .bat file
Program	learngrid-server.bat	The file to run
Run with privileges	Highest	Ensures it can access all folders

What happens internally on boot:

1. You press power button — BIOS/UEFI initializes hardware
2. Windows bootloader starts — loads the Windows kernel
3. Windows starts its services — Task Scheduler service starts early
4. Task Scheduler reads its task database (stored in C:\Windows\System32\Tasks\)
5. It finds 'LearnGrid Server' task with trigger 'At startup'
6. It executes learngrid-server.bat as a background process
7. serve.exe starts and opens port 5173
8. You log in, open CMD, type 'learngrid'
9. Browser opens — app is already running because server started at step 7

CHAPTER 11

The Full Flow

Everything connected — from laptop boot to saving a note. The complete picture.

Boot → App Open

```
LAPTOP BOOTS
    ↓
Windows starts Task Scheduler
    ↓
Task Scheduler runs: learngrid-server.bat
    ↓
serve.exe starts → listening on localhost:5173
    ↓
You open CMD → type: learngrid
    ↓
learngrid.bat runs:
  - Waits 2 seconds (server already running so instant)
  - Opens browser to http://localhost:5173
    ↓
Browser requests index.html from serve
    ↓
Browser loads App.jsx → React runs
    ↓
React reads localStorage → loads your saved entries
    ↓
LearnGrid UI appears with all your data ■
```

Saving a Note

```
You type a title and content in the Add Entry form
    ↓
You click 'Save Entry'
    ↓
```

```
addEntry() function runs in App.jsx
    ↓
Creates an entry object: { id, date, title, content, type }
    ↓
Calls setEntries([newEntry, ...oldEntries])
    ↓
React detects state changed → re-renders the UI
    ↓
Calls save('lt:entries', updatedEntries)
    ↓
localStorage.setItem stores the JSON string
    ↓
useEffect detects entries changed → rebuilds activityMap
    ↓
Heatmap re-renders with today's cell now GREEN ■
```

Uploading a PDF

```
You drag a PDF onto the drop zone
    ↓
onDrop event fires → handlePDF(file) runs
    ↓
FileReader converts PDF to base64 string
    ↓
fetch() sends PUT request to:
  https://api.github.com/repos/owner/repo/contents/pdfs/filename.pdf
  with base64 content in the request body
    ↓
GitHub receives it → stores the file → returns download_url
    ↓
setPdfFile({ name, url: download_url }) saves it to state
    ↓
When you save the entry, the url is stored with it in localStorage
    ↓
You can click the link anytime to view/download from GitHub ■
```

CHAPTER 12

Next Steps

Now that you understand the project, here's exactly what to learn and in what order.

Recommended Learning Path

Week 1-2	**JavaScript Fundamentals** Variables, functions, arrays, objects · Array methods: map, filter, reduce · Promises and async/await · ES6+ features: destructuring, spread operator
Week 3-4	**React Basics** Components and JSX · Props and state (useState) · useEffect and lifecycle · Event handling (onClick, onChange)
Week 5-6	**React Intermediate** Custom hooks · Context API (sharing state globally) · React Router (multiple pages) · Forms and validation
Week 7-8	**Tools & Ecosystem** Git and GitHub properly · npm and package management · Vite and build tools · Tailwind CSS for styling
Week 9-10	**Backend Basics** Node.js fundamentals · Express.js (build an API) · Databases: SQLite or PostgreSQL · REST API concepts
Week 11-12	**Full Stack** Connect React to your own API · Authentication (login/signup) · Deploy full stack app · Build something from scratch

Free Resources to Use

javascript.info	Best JavaScript tutorial on the internet — free, complete
react.dev	Official React docs — excellent interactive examples
roadmap.sh/react	Visual learning roadmap for React

theodinproject.com	Free full-stack curriculum from scratch
freecodecamp.org	Free interactive courses with certificates
github.com/topics/react	Real open source React projects to read

---

You built a real app. That's the best way to learn.

Keep building. Keep learning. The heatmap will fill up. ■