BOSSCODER ACADEMY

TOP 10

Resume Worthy Projects

FOR SOFTWARE ENGINEERS

Disclaimer

EVERYONE LEARNS UNIQUELY.

What matters is developing the problem solving ability
to solve new problems.

This Doc will help you with the same.

Introduction

One of the most important factors in getting your resume shortlisted is the projects you showcase. They’re not just bullet points, they’re proof of your skills, creativity, and ability to solve real problems.

In this doc, you’ll find 10 project ideas that are practical, impactful, and recruiter-friendly. So, whether you’re just starting out or looking to level up, these projects can give your resume the edge it needs.

So, let’s, without wasting your time, let’s start with these Projects!

01 Notes App with AI Summarisation

Create a simple notes-taking application that allows users to:

- Create, edit, and delete notes.
- View a list of saved notes.
- Summarize long notes using an AI API.
- User authentication (Signup, Login).

HOW TO BUILD IT?

01 FRONTEND

a. Components:

- NoteList – Displays list of saved notes.
- NoteEditor – Add/edit individual notes.
- NoteCard – Renders preview of a single note.
- SummaryView – Shows AI-generated summary.
- Auth – Signup and Login forms.

b. State Management:

- Use Redux Toolkit (or Singleton) for managing notes and user sessions.

c. API Integration

- Fetch and update notes via backend API.
- Use a summarization API route (POST/api/notes/:id/summarize) to get AI-generated summaries.

02 BACKEND

a. API Endpoints:

- GET `/api/notes` – List all notes of a user.
- POST `/api/notes` – Create a new note.
- PUT `/api/notes/:id` – Update an existing note.
- DELETE `/api/notes/:id` – Delete a note.
- POST `/api/notes/:id/summarize` – Generate AI summary using OpenAI API (or mock summarization).
- POST `/api/auth/signup` – Create a new user.
- POST `/api/auth/login` – Authenticate user and return token.

03 DATABASE

a. Collections:

- users
- notes

b. Schema for Tasks:

```
{
 userId: ObjectId,          // Reference to User
 title: String,
```

```
  content: String,
  summary: String,          // Optional
  createdAt: Date,
  updatedAt: Date
}
```

02 A To-Do List with Smart Suggestion

A productivity app that allows users to:

- Create, edit, and delete to-do items
- Categorize tasks (e.g., Work, Personal)
- Mark tasks as complete or incomplete
- Get smart suggestions for tasks using AI (e.g., based on task history or input)
- User authentication (Signup, Login)

HOW TO BUILD IT?

01 FRONTEND

a. Components:

- TaskList – Displays all tasks
- TaskInput – Form to add or edit a task
- TaskItem – Individual task card with actions
- SuggestionBox – Displays AI-generated task suggestions
- Auth – Signup and Login forms

b. API Integration:

- Fetch tasks and suggestions from backend API

- Use a smart suggestion API route (POST /api/tasks/suggest) for AI-based task hints

02 API ENDPOINTS:

- GET `/api/tasks` – Get all tasks for a user
- POST `/api/tasks` – Add a new task
- PUT `/api/tasks/:id` – Edit a task
- DELETE `/api/tasks/:id` – Delete a task
- POST `/api/tasks/suggest` – Generate task suggestions using OpenAI API or rule-based logic
- POST `/api/auth/signup` – Create a new user
- POST `/api/auth/login` – Authenticate user and return token

03 DATABASE

a. Collections:

- users
- notes

b. Schema for Tasks:

```
{
  userId: ObjectId,          // Reference to User
  title: String,
  category: String,          // e.g., "Work",
"Personal"
  isCompleted: Boolean,
  createdAt: Date,
  updatedAt: Date
}
```

03 Resume Creator with AI Bullet Generator

A dynamic resume builder that allows users to:

- Create, edit, and save resume content via form inputs
- Generate impactful bullet points for experience using AI
- Download resumes as PDFs
- Save and manage multiple resume versions
- User authentication (Signup, Login)

HOW TO BUILD IT?

01 FRONTEND

a. Components:

- ResumeForm – Multi-section form for entering resume data
- ResumePreview – Live preview of the generated resume
- BulletGenerator – UI to auto-generate job description bullets using AI
- SavedResumes – List of saved resume versions
- Auth – Signup and Login forms

b. Libraries/Tools:

- React-pdf or html2pdf.js – For generating downloadable PDFs

- Formik + Yup – For advanced form handling and validation
- Framer Motion – For smooth transitions

c. API Integration:

- Fetch/save resume data from backend API
- Use AI API (e.g., OpenAI) to generate bullet points based on role and experience input

02 API ENDPOINTS:

- POST `/api/auth/signup` – Register new user
- POST `/api/auth/login` – Authenticate and return token
- GET `/api/resumes` – Get all resumes for a user
- POST `/api/resumes` – Save a new resume
- PUT `/api/resumes/:id` – Update an existing resume
- DELETE `/api/resumes/:id` – Delete a resume
- POST `/api/resumes/:id/generate-bullets` – Generate AI-powered bullet points

03 DATABASE (MONGODB)

a. Collections:

- users
- resumes

b. Schema for Tasks:

```
{
  userId: ObjectId,       // Reference to User
  title: String,
```

```
  personalInfo: {...},
   education: [{...}],
   experience: [{...}],
   skills: [String],
   createdAt: Date,
   updatedAt: Date
}
```

04 Build your own Git or Reddis

Build a simplified version-control system that mimics core Git features. This project helps understand how Git works under the hood and includes:

- File tracking and change detection
- Saving snapshots (commits) of file states
- Viewing commit history
- Own mini command-line interface (CLI)
- Local file-based storage (no database required)

HOW TO BUILD IT?

01 FRONTEND (OPTIONAL / TERMINAL UI)

- This is a backend-heavy or CLI-based project.
- Optionally, build a terminal-based UI using Node.js or a lightweight frontend with minimal interaction for educational visualization.

02 CORE CLI FEATURES (NODE.JS OR PYTHON RECOMMENDED)

a. Commands to Implement:

- init – Initialize a new repository (.mygit/ folder with metadata)
- add <filename> – Track changes to a file

- commit -m "message" – Save a snapshot with a commit message
- log – Show commit history
- status – Show modified/untracked files
- checkout <commit-id> – Restore files to a previous commit state

b. Bonus CLI Features:

- diff – Compare file differences
- branch – Create/manage branches

c. Use Node.js + File System Module (fs) or Python + os/pathlib to manage file system operations.

03 BACKEND (LOCAL FILE SYSTEM AS DB)

- No separate backend server needed. Store everything inside a hidden folder like .mygit.
- Folder Structure Example:

```
.mygit/
├── commits/
│   └── <commit-id>.json
├── index.json          # Staging area
├── log.json            # Commit history
└── config.json         # Author info, etc.
```

- Sample Commit File:

```
{
  "id": "abc123",
  "message": "Initial commit",
```

```
  "timestamp": "2025-07-24T12:00:00Z",
    "files": {
      "README.md": "hash123...",
      "app.js": "hash456..."
    }
  }
```

04 DATABASE

- No external database is used.
- Use hashing (SHA1/MD5) to detect file changes.
- Store all data as local JSON or plain text files inside .mygit.

05 DEPLOYMENT

- This project is CLI-based. No deployment needed on Vercel or Render.
- You can publish it as a Node.js CLI tool on npm or a Python package on PyPI.

05 AI Mentor Chatbot

An intelligent chatbot that acts like a personal mentor, helping users with:

- Career guidance or project ideas
- Asking doubts, tech, or academic questions
- AI-powered Personalized responses based on user history

HOW TO BUILD IT?

01 FRONTEND

a. Components:

- ChatWindow – Displays conversation history
- MessageInput – Input box to send messages
- ChatBubble – Renders each message (user or bot)
- Sidebar – Optional; shows user progress or saved chats
- Auth – Signup/Login forms

b. UI Features:

- Typing animation for bot responses
- Scroll to bottom on new message
- Message timestamps

02 BACKEND

a. API Endpoints:

- POST `/api/auth/signup` – Create a new user
- POST `/api/auth/login` – Authenticate and return JWT
- POST `/api/chat` – Accept user input, forward to OpenAI API, return response
- GET `/api/chat/history` – Retrieve past chat messages for a user
- POST `/api/chat/feedback` – (Optional) Save user feedback on responses

b. OpenAI Integration:

- Use GPT-4 or GPT-3.5 to generate chatbot replies
- Prompt engineering for structured roadmaps, advice, or code help

03 DATABASE

a. Collections:

- users
- messages
- sessions (optional for storing long-term progress)

b. Schema for Messages:

```
{
  userId: ObjectId,
  role: "user" | "bot",
  message: String,
  timestamp: Date,
  sessionId: String // To group by conversations
}
```

c. Schema for Sessions (Optional):

```
{
  userId: ObjectId,
  title: String,
  createdAt: Date,
  updatedAt: Date
}
```

06 Static Portfolio Website

A personal portfolio website that showcases:

- Introduction/About Me section
- Projects with descriptions and links
- Contact form (email-based or mock)
- Responsive design for mobile and desktop

HOW TO BUILD IT?

01 FRONTEND

a. Components:

- Navbar – Navigation bar for sections
- HeroSection – Introductory banner (name, title, social links)
- About – Short bio and skills
- Projects – List of projects with images and links
- ProjectCard – Displays individual project info
- ContactForm – Form to submit messages

b. State Management:

- No complex state management needed (basic local state with useState or useRef)

c. Optional Enhancements:

- Use animation libraries like Framer Motion or AOS for transitions and Add scroll-based section highlighting.
- (Only if using a contact form):Consider adding a Google Form or linking to a Google Sheet that collects data using Google Apps Script.Alternatively, use Formspree or EmailJS to handle form submissions directly.
- Host your site online using github pages or Netify.

07 Real-Time Collaborative Markdown Editor

Build a simplified Notion/Google Docs clone that allows users to:

- Create and edit Markdown documents
- Collaborate with multiple users in real-time
- Sync changes live using WebSockets
- Handle edit conflicts with Operational Transforms (OT) or Y.js
- Track version history (optionally with Redis)
- User authentication (Signup, Login)

HOW TO BUILD IT?

01 FRONTEND

a. Components:

- Editor – Main markdown editor with real-time sync
- DocumentList – Lists all user-created documents
- CollaboratorsBar – Shows active users in the document
- VersionHistory – (Optional) View and restore from previous versions
- Auth – Signup/Login forms

b. Libraries & Tools:

- Use react-markdown or markdown-it for rendering

- Use Y.js (with y-websocket) or ShareDB for real-time collaboration
- Optional: Use CodeMirror or Monaco Editor for rich editing experience

02 BACKEND

a. Components:

- POST `/api/auth/signup` – Create new user
- POST `/api/auth/login` – Authenticate and return token
- GET `/api/docs` – Fetch list of user's documents
- POST `/api/docs` – Create a new document
- GET `/api/docs/:id` – Fetch a document
- DELETE `/api/docs/:id` – Delete a document

b. WebSocket/Real-Time Features:

- Use Y-WebSocket Server or custom WebSocket server for syncing
- Enable broadcasting updates between connected clients
- Use Y.js or ShareDB to manage conflict resolution and merging

c. Version Control (Optional):

- Use Redis or a MongoDB-based log to store version snapshots
- Optionally implement rollback/restore functionality

03 DATABASE

a. Collections:

- users
- documents

b. Schema for Documents:

```
{
  userId: ObjectId,
  title: String,
  content: String,              // Raw Markdown content
  collaborators: [ObjectId], // Other users with
access
  createdAt: Date,
  updatedAt: Date
}
```

08 Productivity Tracker with Gamification

A powerful productivity app that transforms routine tasks into a fun, engaging challenge by:

- Turning daily tasks into point-based social challenges
- Tracking progress using dashboards and analytics
- Adding gamification elements: XP, streaks, levels, and badges
- Using OpenAI to generate weekly reports and personalized productivity tips
- Supporting user authentication and session tracking

HOW TO BUILD IT?

01 FRONTEND

a. Components:

- TaskList – Displays tasks with checkboxes, categories, and deadlines
- TaskForm – Add/edit new habits or productivity tasks
- Dashboard – Visual charts (XP, task completion rate, streaks)
- ChallengeRoom – (Optional) Social challenge feature for team progress
- Achievements – Earned badges and milestones
- WeeklyAIReport – Shows OpenAI-generated summary and tips
- Auth – Signup and login forms

b. UI Features:

- Dynamic progress bars and level-up animations
- Charts using Recharts or Chart.js
- Badge/unlock animations using Framer Motion

02 BACKEND

a. API Endpoints:

- POST `/api/auth/signup` – Register user
- POST `/api/auth/login` – Login and return JWT
- GET `/api/tasks` – Get user's tasks
- POST `/api/tasks` – Add new task
- PUT `/api/tasks/:id` – Update or complete task
- GET `/api/dashboard` – Fetch XP, streaks, progress
- GET `/api/achievements` – Earned badges list
- POST `/api/report` – Generate weekly OpenAI productivity report
- POST `/api/challenges/join` – Join or create social challenges

b. Gamification Logic:

- XP for each completed task
- Streak increment if daily tasks are completed
- Badge unlocks based on activity milestones
- Challenge groups update shared progress

c. OpenAI Integration:

- Send task logs to GPT-4 or GPT-3.5

- Example prompt:

"Analyze this user's completed tasks and generate a 3-sentence weekly summary with 2 motivational tips to boost engagement."

03 DATABASE (MONGODB)

a. Collections:

- users
- tasks
- achievements
- weekly_reports
- challenges (optional)

b. Schema for Task:

```
{
  userId: ObjectId,
  title: String,
  category: String,
  completed: Boolean,
  date: Date,
  xp: Number
}
```

c. Schema for Weekly Reports:

```
{
  userId: ObjectId,
  summary: String,
  tips: [String],
  weekStart: Date,
  weekEnd: Date
}
```

09 SVR (Speech-to-Voice Recogniser)

Build an interactive gesture-to-speech communication system using IoT and AI. This real-world assistive tech project helps users:

- Convert hand gestures into real-time spoken speech
- Use sensors on a glove to detect motions/flexes
- Map gestures to text using a trained AI model
- Vocalize the output with text-to-speech (TTS) APIs
- Display live feedback on a screen or dashboard

HOW TO BUILD IT?

01 HARDWARE (IOT DEVICE)

a. Tools:

- Flex Sensors – Attach to glove fingers to detect bending
- Motion Sensor (IMU) – For hand orientation and movement
- Microcontroller – Arduino Uno or Raspberry Pi (preferably Pi for ML)
- Wires, breadboard, glove, optional screen (OLED or LCD)

b. Workflow:

- Gather analog/digital readings from sensors

- Send data via serial/USB or Wi-Fi/Bluetooth to a server or onboard ML system
- Classify gestures using a trained model
- Send gesture-to-text result to TTS engine

02 MACHINE LEARNING AND MODEL TRAINING

- Use Teachable Machine, TensorFlow, or Edge Impulse to train a gesture recognition model
- Input: Flex + IMU sensor values
- Output: Corresponding gesture text (e.g., “Hello”, “Thank you”)

03 FRONTEND (REACT.JS OR TKINTER DASHBOARD)

- LiveTextOutput – Show the translated gesture text
- SpeechPlayback – Play audio response
- SensorMonitor – Visualize live sensor values
- HistoryLog – List of past gestures with timestamps

04 BACKEND (PYTHON OR NODE.JS)

a. Core Tasks:

- Collect data from Arduino/Pi (via Serial, Bluetooth, or MQTT)
- Pass sensor data into gesture classification model
- Use Text-to-Speech (TTS) API like:
    1. Google Cloud TTS
    2. Amazon Polly
    3. pyttsx3 (offline)

b. Optional Backend APIs:

- POST `/api/gesture` – Receive data and return gesture label
- GET `/api/history` – View past recognized gestures

c. Optional Features:

- Add WebSocket for real-time updates
- Use Electron if packaging into a desktop app

05 DATABASE (OPTIONAL: MONGODB / SQLITE)

a. Collections:

- users (if multi-user setup)
- gestures (log recognized gestures with timestamp)

b. Gesture Schema:

```
{
  gestureLabel: String,
  sensorData: Object,
  recognizedAt: Date
}
```

06 DEPLOYMENT

- Hardware: Glove + Raspberry Pi / Arduino
- Backend Model Server: Python/Node on local device or cloud
- Frontend Dashboard: Deployed on local device or Vercel (optional)
- Database: Local JSON/SQLite or MongoDB Atlas if needed

10 Car Pollution Checker

A real-world automation project that reduces manual work in pollution compliance checks. The system:

- Uses a Raspberry Pi + camera module to capture images of passing vehicles
- Extracts license plate numbers using OCR
- Cross-checks the extracted plate against a backend pollution compliance database
- Displays or flags results (Compliant / Non-Compliant) in real-time

HOW TO BUILD IT?

01 HARDWARE (IOT CAMERA SETUP)

a. Tools:

- Raspberry Pi 4/3B+
- Pi Camera Module v2 or USB Webcam
- Optional: IR module for night capture

b. Workflow:

- Mount the camera to capture car images (roadside or at checkpoints)
- Use Python script on Pi to detect motion or capture at intervals
- Process image using OCR to extract the license plate text
- Query backend for pollution status based on extracted plate

02 COMPUTER VISION & AI

a. OCR Tools:

- Tesseract OCR (open-source)
- Or use Google Cloud Vision API / AWS Rekognition for better accuracy

b. Image Preprocessing (important for accuracy):

- Grayscale conversion
- Noise reduction (Gaussian Blur)
- Edge detection (Canny)
- Cropping/Detecting license plate region (optional: OpenCV object detection)

03 FRONTEND

a. Components:

- LiveCaptureFeed – Show current or recent Pi captures
- ScanLogTable – List recent checks with compliance status
- VehicleStatusCard – Display pollution details for a selected vehicle
- AdminUpload – (Optional) Interface to update vehicle data

b. Enhancements:

- Show red/green alerts for compliance
- Upload images from Pi to backend via REST or MQTT

04 BACKEND

a. API Endpoints:

- POST `/api/check-plate` – Accepts license plate string, returns compliance status
- GET `/api/vehicle/:plateNumber` – Fetch pollution data for a vehicle
- POST `/api/vehicle/update` – (Admin only) Update pollution certificate data
- GET `/api/logs` – View past scanned entries

b. Logic:

- Query MongoDB for the vehicle's compliance expiry
- If expired or not found, mark as non-compliant
- Optionally log all scanned entries with image, time, and status

05 DATABASE

a. Collections:

- vehicles – Vehicle data with pollution certificate info
- scan_logs – History of captured entries

b. Schema for Vehicles:

```
{
  plateNumber: String,
  ownerName: String,
  vehicleType: String,
  pollutionCertificateExpiry: Date,
  isCompliant: Boolean
}
```

c. Schema for Logs:

```
{
  plateNumber: String,
  imageUrl: String,
  capturedAt: Date,
  isCompliant: Boolean
}
```

04 DEPLOYMENT

- Frontend: Vercel (optional dashboard)
- Backend: Render / Railway / Pi-hosted server
- Raspberry Pi: Runs local script + OCR + upload logic
- Database: MongoDB Atlas or local MongoDB on Pi

BOSSCODER ACADEMY

WHY BOSSCODER?

01 STRUCTURED INDUSTRY-VETTED CURRICULUM

Our curriculum covers everything you need to get become a skilled software engineer & get placed.

02 1:1 MENTORSHIP SESSIONS

You are assigned a personal mentor currently working in Top product based companies.

03 2200+ ALUMNI PLACEMENT

2200+ Alumni placed at Top Product-based companies.

04 24 LPA AVERAGE PACKAGE

Our Average Placement Package is **24 LPA** and highest is **98 LPA**

Niranjan Bagade
Software Engineer,
British Petroleum

10 Years
Experience

NICE Software Eng. Specialist	Hike **83%**	**British Petroleum** Software Engineer

Dheeraj Barik
Software Engineer 2,
Amazon

2 Years
Experience

Infosys Software Engineer	Hike **550%**	**Amazon** SDE 2

EXPLORE MORE