Project 2: Online Code Editor

To build an online code editor, you'll need to create a frontend editor using ReactJS and a backend API using Node, Express, and a hosting service. Here's a high-level overview of the steps:

Set up a new React project using create-react-app.

Create a Node API with two endpoints:

POST /execute: takes code and language as input, executes the code on the server, and returns the output.

In this project, you will be required to make an API to build short URLs. The functionality will be similar to bitly. Using Node, Express, and MongoDB you can make your own URL Shortener service. However, you can use any backend language also depending on your expertise. The project is not specific to any particular backend language. Skills Required – Node, MongoDB,JavaScript Link Shortener

Task 2 Online Code Editor Creating an online code editor running in browsers offers complete IDE functionalities, and it's a valuable skill-building project. It can potentially lead to a startup as a free interviewing platform. This project involves two main components: Backend API: This server-side component takes code and language as input and provides the output after executing the code on the server. Frontend Editor: On the frontend, you can choose a language, edit and modify the code. After editing, you make a post request to the backend API and display the output on the website. Skills Required – HTML, CSS, ReactJS, Hosting Services

It seems like you have two project ideas: a URL shortener service and an online code editor. I'd be happy to help you with both!

Project 1: URL Shortener Service

To build a URL shortener service using Node, Express, and MongoDB, here's a high-level overview of the steps:

1. Set up a new Node project and install the required dependencies: express, mongodb, and mongoose (a MongoDB ORM for Node).
2. Create a MongoDB database to store the short URLs and their corresponding original URLs.
3. Define a schema for the short URL documents using Mongoose.
4. Create an Express API with two endpoints:
   * POST /shorten: takes a URL as input and generates a short URL, storing it in the database.
   * GET /:shortUrl: redirects the user to the original URL corresponding to the short URL.
5. Implement the logic to generate a unique short URL for each input URL.
6. Test the API using a tool like Postman or cURL.

Here's some sample code to get you started:

javascript

Edit

Run

1// server.js

2const express = require('express');

3const mongoose = require('mongoose');

4const app = express();

5

6mongoose.connect('mongodb://localhost/url-shortener', { useNewUrlParser: true, useUnifiedTopology: true });

7

8const ShortUrl = mongoose.model('ShortUrl', {

9 originalUrl: String,

10 shortUrl: String

11});

12

13app.post('/shorten', (req, res) => {

14 const originalUrl = req.body.url;

15 const shortUrl = generateShortUrl(); // implement this function to generate a unique short URL

16 const shortUrlDoc = new ShortUrl({ originalUrl, shortUrl });

17 shortUrlDoc.save((err, doc) => {

18 if (err) {

19 res.status(500).send({ message: 'Error creating short URL' });

20 } else {

21 res.send({ shortUrl: doc.shortUrl });

22 }

23 });

24});

25

26app.get('/:shortUrl', (req, res) => {

27 const shortUrl = req.params.shortUrl;

28 ShortUrl.findOne({ shortUrl }, (err, doc) => {

29 if (err ||!doc) {

30 res.status(404).send({ message: 'Short URL not found' });

31 } else {

32 res.redirect(doc.originalUrl);

33 }

34 });

35});

36

37app.listen(3000, () => {

38 console.log('Server listening on port 3000');

39});

Project 2: Online Code Editor

To build an online code editor, you'll need to create a frontend editor using ReactJS and a backend API using Node, Express, and a hosting service. Here's a high-level overview of the steps:

1. Set up a new React project using create-react-app.
2. Create a Node API with two endpoints:
   * POST /execute: takes code and language as input, executes the code on the server, and returns the output.
   * GET /languages: returns a list of supported programming languages.
3. Implement the logic to execute the code on the server using a library like child\_process or vm.
4. Create a React component for the code editor, using a library like monaco-editor or codemirror.
5. Implement the functionality to send the code to the backend API for execution and display the output.

Here's some sample code to get you started:

// backend/api.js

const express = require('express');

const app = express();

app.post('/execute', (req, res) => {

const code = req.body.code;

const language = req.body.language;

// implement the logic to execute the code on the server

const output = executeCode(code, language);

res.send({ output });

});

app.get('/languages', (req, res) => {

const languages = ['javascript', 'python', 'java', 'c++']; // list of supported languages

res.send({ languages });

});

app.listen(3000, () => {

console.log('Server listening on port 3000');

});

// frontend/CodeEditor.js

import React, { useState } from 'eact';

import MonacoEditor from 'onaco-editor';

const CodeEditor = () => {

const [code, setCode] = useState('');

const [language, setLanguage] = useState('javascript');

const [output, setOutput] = useState('');

const handleExecute = () => {

fetch('/api/execute', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ code, language })

})

.then(response => response.json())

.then(data => setOutput(data.output));

};

return (

<div>

<MonacoEditor

value={code}

language={language}

onChange={value => setCode(value)}

/>