Integrating a Frontend Application with a Backend Server

Integrating a frontend application with a backend server is a fundamental aspect of modern web development. This process typically involves using RESTful APIs to communicate between the frontend and backend. This report will cover the essential concepts of RESTful APIs, how to make API calls from the frontend, and provide examples to illustrate the integration process.

Understanding RESTful APIs

Concept

REST (Representational State Transfer) is an architectural style for designing networked applications. RESTful APIs use HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on resources.

Key Characteristics

1. **Stateless**: Each API call is independent, with no stored context on the server between requests.
2. **Resource-Based**: Each piece of data (resource) is identified by a URL.
3. **methods**: Common HTTP methods include:
- **GET**: Retrieve a resource.
- **POST**: Create a new resource.
- **PUT**: Update an existing resource.
- **DELETE**: Remove a resource.

Example of RESTful API Endpoints

Assuming we have a RESTful API for managing a list of tasks, the endpoints might look like this:

- GET /tasks : Retrieve all tasks.
- GET /tasks/{id} : Retrieve a specific task by ID.
- POST /tasks : Create a new

task.
- PUT /tasks/{id} : Update an existing task by ID.
- DELETE /tasks/{id} : Delete a task by ID.

Making API Calls from the Frontend

To integrate a frontend application with a backend server, you need to make HTTP requests to the API endpoints. This is typically done using JavaScript and libraries like Axios or the Fetch API.

Using Fetch API

The Fetch API is a built-in JavaScript function for making HTTP requests. It returns a promise that resolves to the response object.

Example: Fetching Data

```javascript
```

```
fetch('https://api.example.com/tasks')
 then(response => response.json())
 then(data => console.log(data))
 catch(error => console.error('Error:',
error));
```

Example: Creating Data

```javascript
fetch('https://api.example.com/tasks',
{
method: 'POST',
headers: {
'Content-Type': 'application/json'
    },
body: JSON.stringify({  itle: 'New Task',
completed: false  } )
} )
 then(response => response.json())
 then(data => console.log(data))
 catch(error => console.error('Error:',
error));
```

Using Axios

Axios is a popular JavaScript library for making HTTP requests. It provides an easier and more concise syntax compared to the Fetch API.

Installation

You can install Axios using npm or yarn:

```bash
npm install axios
# or
yarn add axios
```

Example: Fetching Data

```javascript
import axios from 'axios';

axios.get('https://api.example.com/tasks')
 .then(response =>
console.log(response.data))
 .catch(error => console.error('Error:',
```

```
error));
```

Example: Creating Data

```javascript
import axios from 'axios';

axios.post('https://api.example.com/tasks', {
title: 'New Task',
completed: false
} )
 then(response =>
console.log(response.data))
 catch(error => console.error('Error:',
error));
```

Example Application: Integrating Frontend with Backend

Let's build a simple React application that integrates with a backend server to manage a list of tasks.

Backend Server

We'll use a basic Express server for our backend.

server.js

```javascript
const express = require('express');
const app = express();
const port = 3001;

app.use(express.json());

let tasks = [
{ id: 1, title: 'Task 1', completed: fals
e } ,
{ id: 2, title: 'Task 2', completed: tru
e } ,
];

app.get('/tasks', (req, res) => {
res.json(tasks);
} );
```

```
app.post('/tasks', (req, res) => {
const newTask = { id: tasks.length +
1, req.body } ;
tasks.push(newTask);
res.status(201).json(newTask);
} );

app.listen(port, () => {
console.log( Server running at
http://localhost: {port} );
} );
```

Frontend Application

We'll create a React application that interacts with our Express server.

App.js

```javascript
import React, { useState, useEffec
+ }from 'react';
import axios from 'axios';
import './App.css';
```

```
function App() {
const [tasks, setTasks] =
useState([]);
const [newTask, setNewTask] =
useState('');

useEffect(() => {
axios.get('http://localhost:3001/tasks')
 then(response =>
setTasks(response.data))
 catch(error => console.error('Error:',
error));
    }, []);

const addTask = () => {
axios.post('http://localhost:3001/task
s'{  itle: newTask, completed: fals
e   }  )
 then(response => setTasks([...tasks,
response.data]))
 catch(error => console.error('Error:',
error));
setNewTask('');
    }  ;
```

```
return (

  Task List

  type="text"
  value={newTask}
  onChange={(e) =>
  setNewTask(e.target.value)}
  placeholder="Add a new task"
  >    /

  Add Task

    {tasks.map(task => (

    {task.title}    {task.completed    ?
    '(Completed)'    ":}

              ))}
```

```
    );
}

export default App;
```

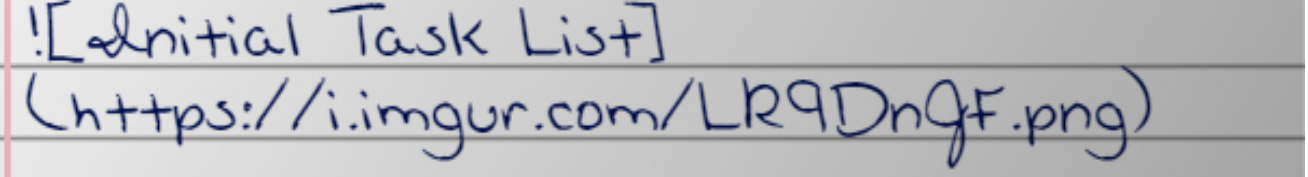
Running the Application

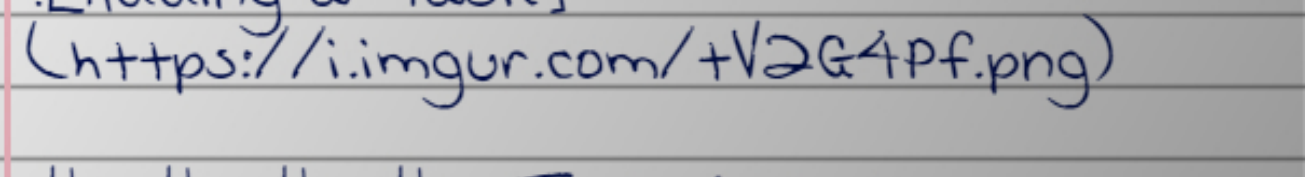
1. **Start the backend server**:
```bash
node server.js
```

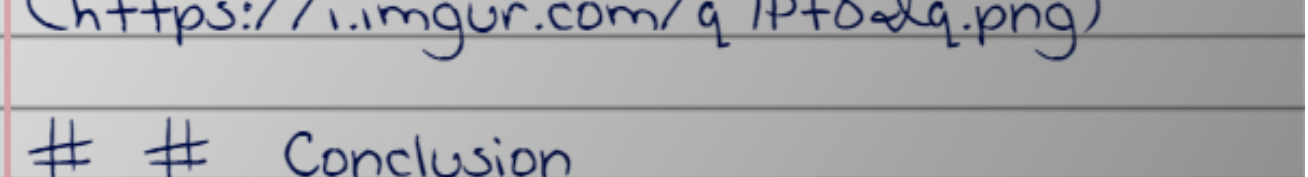
2. **Start the React application**:
```bash
npm start
```

3. **Open the application in a browser**: Navigate to `http://localhost:3000`.

Screenshots

Initial Task List


Adding a Task


Task Added


Conclusion

Integrating a frontend application with a backend server involves understanding RESTful APIs and making HTTP requests from the frontend to the backend. By using libraries like Axios or the Fetch API, developers can easily perform CRUD operations and manage data flow between the client and server. This report demonstrated a basic example of such integration using