Yes, you can absolutely implement this solution using Node.js instead of C#. The Azure Bot Framework fully supports Node.js, and you can achieve the same functionality with a few adjustments. Below, I'll guide you through the process step-by-step and provide a sample code snippet to help you get started.

---

### Key Differences When Using Node.js

- **Bot Logic**: You'll use the Bot Builder SDK for Node.js instead of C#.
- **State Management**: The Bot Builder SDK provides tools to manage conversation state in Node.js.
- **API Calls**: You can make HTTP requests to your API using libraries like Axios or the built-in `http` module.
- **Code Structure**: You'll work with JavaScript functions and modules instead of C# classes.
- **Deployment**: The deployment process is similar, but you'll need to set up a Node.js environment.

---

### Step-by-Step Implementation Using Node.js

#### Step 1: Set Up Azure Resources

- Create the necessary Azure resources: **Speech Resource**, **Azure Communication Services (ACS)**, and an **Azure Function** (for your mock API).
- When creating the **Azure Bot** in the Azure portal, select **Node.js** as the runtime stack instead of C#.

#### Step 2: Configure the Bot for Speech

- In your Azure Bot resource, enable **Direct Line Speech** and connect it to your Speech resource. This step remains the same regardless of the language.

#### Step 3: Implement the Bot Logic in Node.js

1. **Set Up a Node.js Project**:
   - Create a new folder for your project and initialize it:

```bash
npm init -y
```

- Install the required dependencies:

```bash
npm install botbuilder botbuilder-ai botbuilder-dialogs axios
```

2 **Write the Bot Logic**:

- Use the Bot Builder SDK to create a bot that collects user inputs (e.g., first name, last name, monthly income), calls the API, and responds based on the result.

3 **Sample Code Snippet**: Here's an example of how your bot logic might look in Node.js:

```javascript
const { ActivityHandler, MemoryStorage, ConversationState } = require('botbui
const axios = require('axios');

class QuotationBot extends ActivityHandler {
    constructor() {
        super();
        // Set up conversation state with in-memory storage
        const memoryStorage = new MemoryStorage();
        this.conversationState = new ConversationState(memoryStorage);
        this.dialogState = this.conversationState.createProperty('dialogState

        // Handle incoming messages
        this.onMessage(async (context, next) => {
            // Get or initialize the conversation state
            const state = await this.dialogState.get(context, { step: 0, data

            if (state.step === 0) {

                await context.sendActivity('Hello, thank you for calling. May
                state.step = 1;
            } else if (state.step === 1) {
                state.data.firstName = context.activity.text;
```

```
                await context.sendActivity(`Thank you, ${state.data.firstName
                state.step = 2;
            } else if (state.step === 2) {
                state.data.lastName = context.activity.text;
                await context.sendActivity(`Great, ${state.data.firstName} ${
                state.step = 3;
            } else if (state.step === 3) {
                state.data.income = context.activity.text;
                // Prepare data for the API call
                const jsonRequest = {
                    firstName: state.data.firstName,
                    lastName: state.data.lastName,
                    income: state.data.income
                };
                // Call the API and handle the response
                const response = await this.callQuotationAPI(jsonRequest);
                const status = response.status;
                const message = status === 'accepted' ? 'Your quotation has b
                                status === 'referred' ? 'Your quotation has b
                                'Unfortunately, your quotation has been decli
                await context.sendActivity(message);
                state.step = 0; // Reset for the next conversation
            }

            // Save the updated state
            await this.conversationState.saveChanges(context);
            await next();
        });
    }

    // Function to call the mock API
    async callQuotationAPI(jsonRequest) {
        try {
            const response = await axios.post('https://your-mock-api.azureweb
            return response.data;
        } catch (error) {
            console.error('Error calling API:', error);
            return { status: 'error' };
        }
    }
}

module.exports = QuotationBot;
```

```
module.exports = { quotationBot };
```

4 **Explanation**:

- **State Management**: The `ConversationState` with `MemoryStorage` tracks the conversation step and user data.
- **Conversation Flow**: The bot asks for the first name, last name, and income in sequence, then sends the data to the API.
- **API Call**: The `callQuotationAPI` function uses Axios to make a POST request to your Azure Function.
- **Response**: The bot replies based on the API's response (`accepted`, `referred`, or `declined`).

**Step 4: Set Up the Bot Server**

1. **Create an Entry Point (`index.js`)**:

- Add the following code to set up the bot server:

```
```

```javascript
const { BotFrameworkAdapter } = require('botbuilder');
const restify = require('restify');
const QuotationBot = require('./QuotationBot');

// Create the server
const server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, () => {
    console.log(`Server is listening at ${server.url}`);
});

// Create the adapter
const adapter = new BotFrameworkAdapter({
    appId: process.env.MicrosoftAppId,
    appPassword: process.env.MicrosoftAppPassword
});

const bot = new QuotationBot();

// Listen for incoming messages
server.post('/api/messages', (req, res) => {
    adapter.processActivity(req, res, async (context) => {
        await bot.run(context);
    });
});
```

- Install Restify:

```bash
npm install restify
```

2 **Environment Variables**:

- Set `MicrosoftAppId` and `MicrosoftAppPassword` in a `.env` file or in your Azure Bot configuration after deployment.

**Step 5: Deploy the Bot**

- Deploy your Node.js bot to Azure using the Azure CLI or Visual Studio Code.
- In the Azure portal, under your bot's **Configuration**, set the environment variables (`MicrosoftAppId` and `MicrosoftAppPassword`).

**Step 6: Connect to ACS**

- Link your ACS phone number to the bot, as you would with C#, to route calls to it.

**Step 7: Test the Bot**

- Call the ACS phone number, provide the requested inputs (first name, last name, income), and verify the bot's response based on the API result.

---

## Additional Tips

- **Speech Recognition**: Ensure clear speech for accurate input recognition.
- **Voice Output**: Use a neural voice in your Speech resource for a natural-sounding bot.
- **Error Handling**: Add checks for invalid inputs or API errors to improve reliability.

With this approach, you can build and deploy your bot entirely in Node.js. Let me know if you need help with any specific part!