Detailed Description of Backend Code

Backend Code:

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const bodyParser = require("body-parser");
const app = express();
const PORT = process.env.PORT || 4000;
app.use(cors());
app.use(bodyParser.json());
mongoose.connect("mongodb://localhost:27017/Stock_Market", {
   useNewUrlParser: true,
   useUnifiedTopology: true,
});
const stockSchema = new mongoose.Schema({
   company: String,
   description: String,
   initial_price: Number,
   price_2002: Number,
   price_2007: Number,
   symbol: String,
});
const Stock = mongoose.model("Stock", stockSchema);
app.get("/api/stocks", async (req, res) => {
   try {
      const stocks = await Stock.find();
      res.json(stocks);
   } catch (error) {
      console.error(error);
```

```
        res.status(500).json({ error: "Internal Server Error" });
    }});
app.post("/api/watchlist", async (req, res) => {
    try {
        const {
            company,
            description,
            initial_price,
            price_2002,
            price_2007,
            symbol,
        } = req.body;
        const stock = new Stock({
            company,
            description,
            initial_price,
            price_2002,
            price_2007,
            symbol,
        });
        await stock.save();
        res.json({ message: "Stock added to watchlist successfully" });
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: "Internal Server Error" });
    }
});
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

1. Dependencies and Setup

- **express**:
    - A web application framework for Node.js that simplifies the creation of web servers and APIs. It provides a robust set of features for web and mobile applications.
- **mongoose**:
    - An Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a schema-based solution to model application data, manage database interactions, and validate data.
- **cors**:
    - Middleware that enables Cross-Origin Resource Sharing. It allows your backend server to respond to requests from different origins, which is useful for front-end applications making requests to the server from a different domain.
- **body-parser**:
    - Middleware to parse incoming request bodies before your handlers. It parses JSON payloads and makes them available as JavaScript objects in your route handlers.

2. Middleware Configuration

- **app.use(cors())**:
    - Configures the server to allow requests from different origins. This is essential for front-end applications that need to interact with the backend server from a different domain.
- **app.use(bodyParser.json())**:
    - Configures the server to parse JSON request bodies. This allows the server to handle JSON data sent in POST requests and convert it into JavaScript objects accessible via req.body.

3. Database Connection

- **mongoose.connect("mongodb://localhost:27017/Stock_Market", {...})**:
    - Establishes a connection to a MongoDB database hosted locally at mongodb://localhost:27017/Stock_Market.
    - **useNewUrlParser: true**: Uses the new URL parser engine to handle MongoDB connection strings.
    - **useUnifiedTopology: true**: Uses the new Server Discover and Monitoring engine, providing a more robust and stable connection experience.

4. Stock Schema and Model

- **const stockSchema = new mongoose.Schema({...})**:
    - Defines the schema for the Stock collection in MongoDB. The schema specifies the structure of the documents, including the data types for each field:
        - **company**: A String representing the name of the company.
        - **description**: A String providing a description of the stock.
        - **initial_price**: A Number for the stock's initial price.
        - **price_2002**: A Number representing the stock price in 2002.
        - **price_2007**: A Number representing the stock price in 2007.
        - **symbol**: A String for the stock's ticker symbol.
- **const Stock = mongoose.model("Stock", stockSchema)**:
    - Creates a model named Stock using the schema defined. This model allows interaction with the Stock collection in MongoDB, enabling CRUD operations such as creating, reading, updating, and deleting stock documents.

5. API Endpoints

- **GET /api/stocks**:
    - **Purpose**: Fetches all stock documents from the database.
    - **Implementation**:
        - Uses Stock.find() to retrieve all documents from the Stock collection.
        - Sends the retrieved data as a JSON response to the client.
        - Handles errors by logging them and sending a 500 Internal Server Error status code with an error message if an error occurs.
- **POST /api/watchlist**:
    - **Purpose**: Adds a new stock document to the Stock collection.
    - **Implementation**:
        - Extracts stock details from the request body.
        - Creates a new Stock instance using the provided data.
        - Saves the new stock document to the database using stock.save().
        - Responds with a success message if the operation is successful.
        - Handles errors by logging them and sending a 500 Internal Server Error status code with an error message if an error occurs.

6. Server Setup

- **app.listen(PORT, () => {...})**:
    - Starts the server and listens on the specified port, defaulting to 4000.
    - Logs a message indicating that the server is running and listening on the specified port.

Overall Architecture

The backend code is structured to handle API requests and interact with a MongoDB database. It defines a schema for the Stock collection, provides endpoints for fetching and adding stocks, and uses middleware for request handling. The integration with MongoDB through Mongoose ensures efficient data management and interaction, while middleware components handle cross-origin requests and JSON parsing.