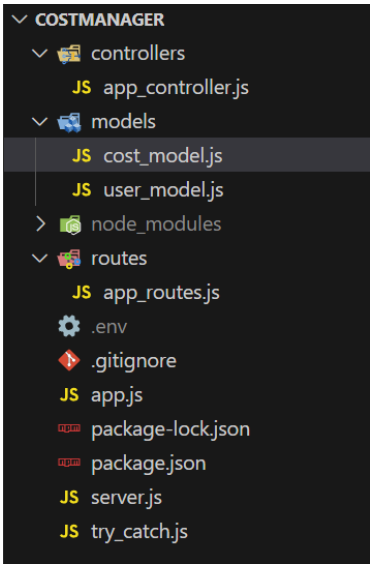
Cost Manager Project

team manager: Omer Peled
team members: Omer Peled, 315110015, 0508334100, opeled6@gmail.com
Hila Itzhak, 209323955, 0504083499, hila87219@gmail.com
YouTube: קישור ליוטיוב

code

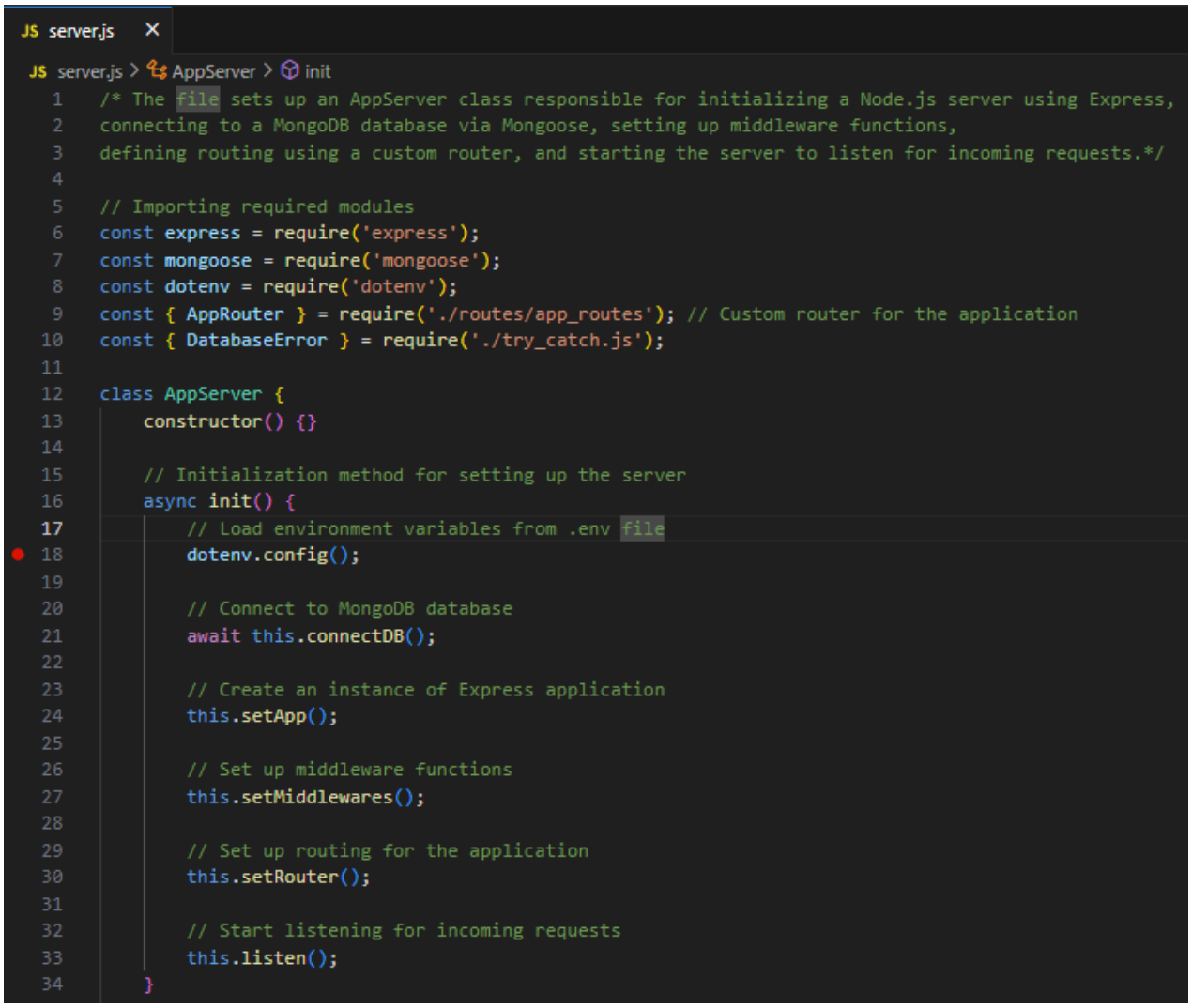


main file – app.js

app.js

```
/* The file serves as the entry point of the program,
creating and initializing an instance of the AppServer class to start the server
 and handle incoming requests asynchronously in a Node.js application. */

// Importing the AppServer class - our main file
const { AppServer } = require("./server");

// Defining the main function, which serves as the entry point of the program
const main = async () => {
    // Creating a new instance of the AppServer class
    const server = new AppServer();

    // Initializing the server asynchronously by calling its init method
    await server.init();
}

// Calling the main function to start the program execution
main();
```

server.js

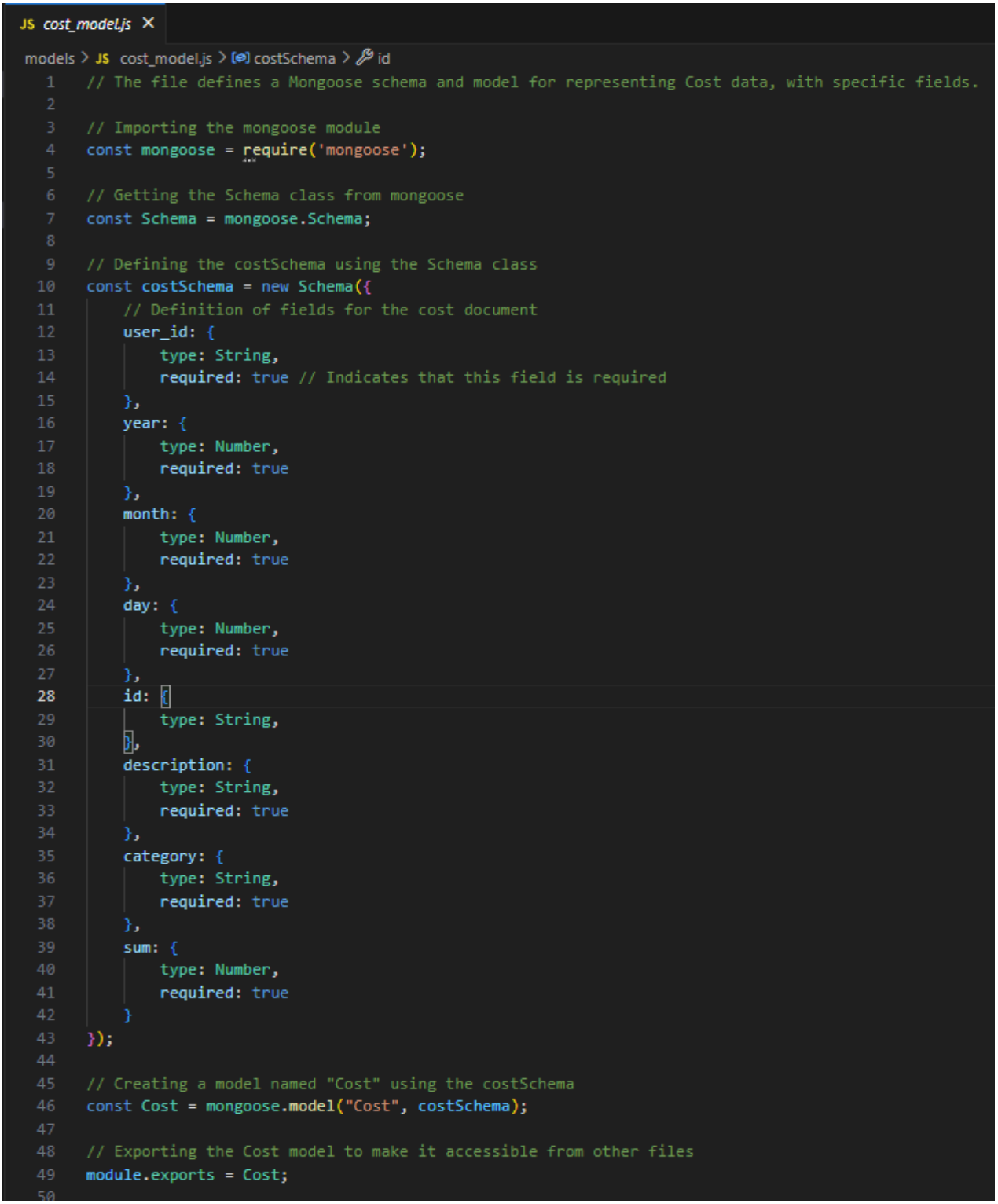


```
JS server.js
JS server.js > AppServer > init
1   /* The file sets up an AppServer class responsible for initializing a Node.js server using Express,
2   connecting to a MongoDB database via Mongoose, setting up middleware functions,
3   defining routing using a custom router, and starting the server to listen for incoming requests.*/
4
5   // Importing required modules
6   const express = require('express');
7   const mongoose = require('mongoose');
8   const dotenv = require('dotenv');
9   const { AppRouter } = require('./routes/app_routes'); // Custom router for the application
10  const { DatabaseError } = require('./try_catch.js');
11
12  class AppServer {
13      constructor() {}
14
15      // Initialization method for setting up the server
16      async init() {
17          // Load environment variables from .env file
18          dotenv.config();
19
20          // Connect to MongoDB database
21          await this.connectDB();
22
23          // Create an instance of Express application
24          this.setApp();
25
26          // Set up middleware functions
27          this.setMiddlewares();
28
29          // Set up routing for the application
30          this.setRouter();
31
32          // Start listening for incoming requests
33          this.listen();
34      }
```

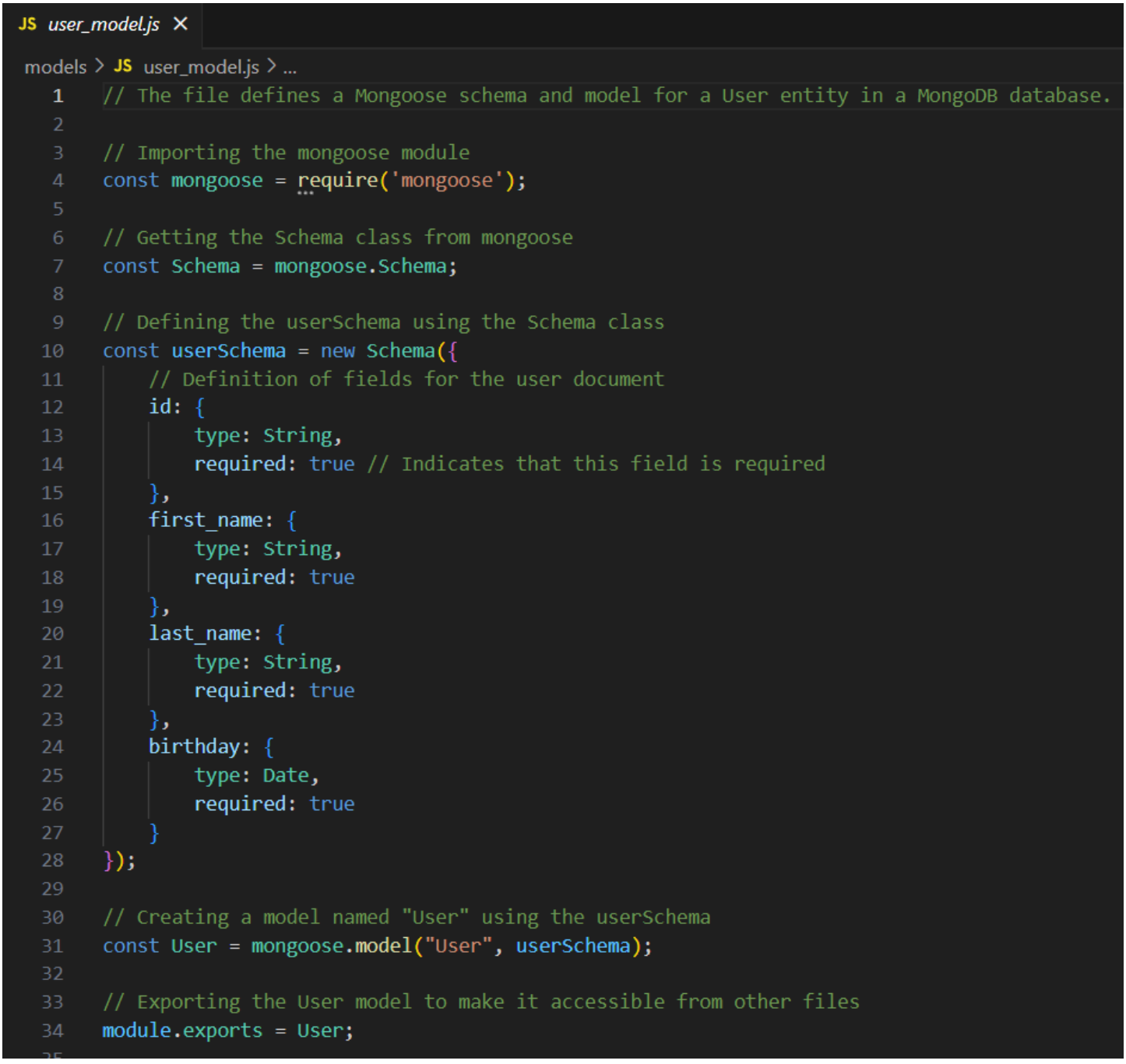
```
36      setApp() {
37          this.app = express();
38      }
39
40      async connectDB() {
41          try {
42              // Attempt to establish connection to MongoDB using Mongoose
43              if (!process.env.MONGO_URI) {
44                  throw new Error('Configuration Error: MONGO_URI was not provided');
45              }
46              const connectionDB = await mongoose.connect(process.env.MONGO_URI);
47              console.log(`MongoDB Connected: ${connectionDB.connection.host}`);
48          } catch (error) {
49              throw new DatabaseError(`Error connecting to database: ${error.message}`);
50          }
51      };
52
53      setMiddlewares() {
54          // Parse incoming requests with JSON payloads
55          this.app.use(express.json());
56      }
57
58      setRouter() {
59          // Create an instance of the custom router
60          const appRouter = new AppRouter();
61
62          // Use the router for the root path ('/')
63          this.app.use('/', appRouter.getRouter());
64      }
65
66      listen() {
67          // Get port from environment variable or default to 3000
68          const port = process.env.PORT || 3000;
69          this.app.listen(port, () => {
70              console.log(`Server is running on port ${port}`);
71          });
72      }
73  }
74
75  // Export the AppServer class to make it accessible from other files
76  module.exports = { AppServer };
77
```

Models

cost_model.js



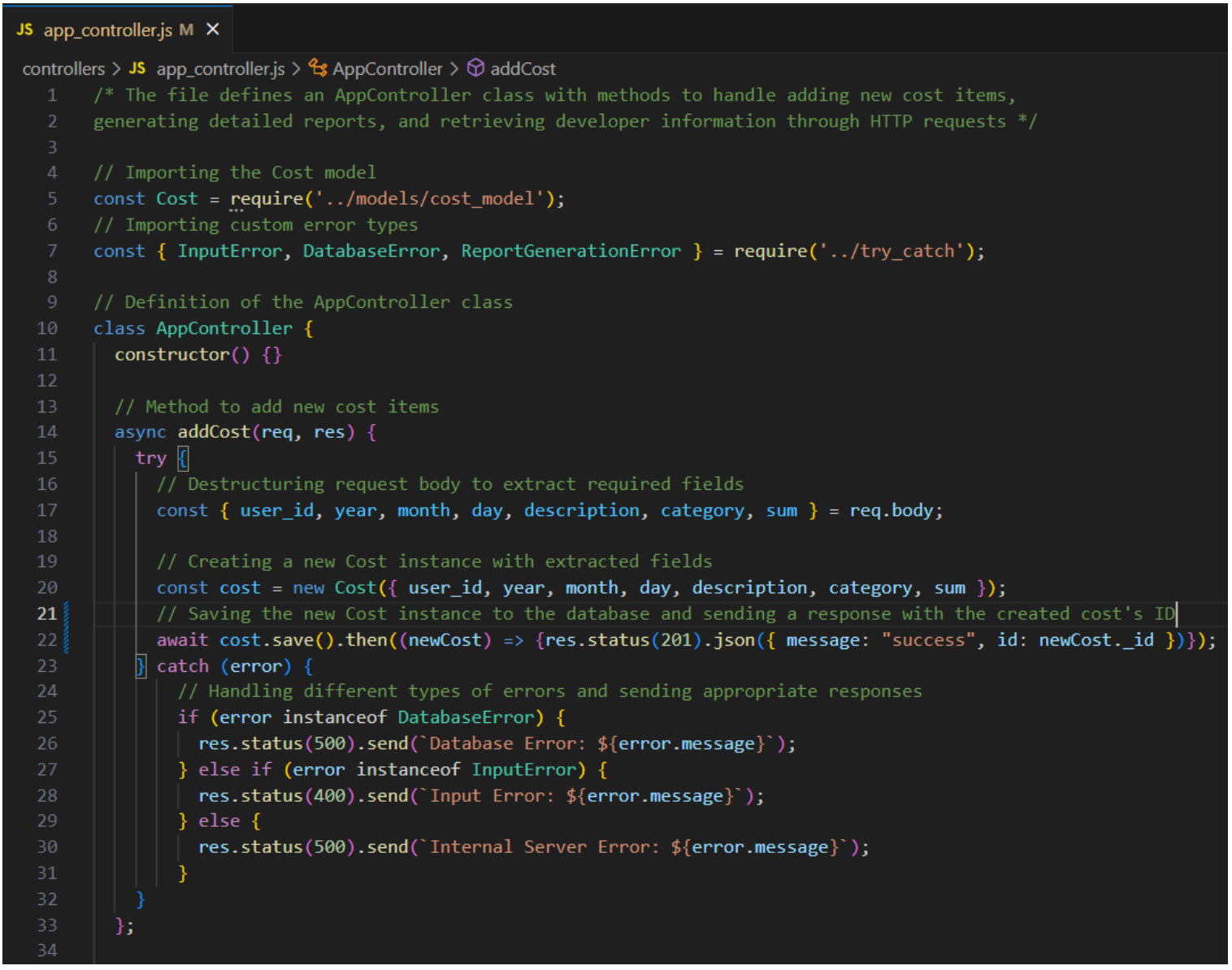
user_model.js



```
JS user_model.js ×

models > JS user_model.js > ...
1   // The file defines a Mongoose schema and model for a User entity in a MongoDB database.
2
3   // Importing the mongoose module
4   const mongoose = require('mongoose');
5
6   // Getting the Schema class from mongoose
7   const Schema = mongoose.Schema;
8
9   // Defining the userSchema using the Schema class
10  const userSchema = new Schema({
11      // Definition of fields for the user document
12      id: {
13          type: String,
14          required: true // Indicates that this field is required
15      },
16      first_name: {
17          type: String,
18          required: true
19      },
20      last_name: {
21          type: String,
22          required: true
23      },
24      birthday: {
25          type: Date,
26          required: true
27      }
28  });
29
30  // Creating a model named "User" using the userSchema
31  const User = mongoose.model("User", userSchema);
32
33  // Exporting the User model to make it accessible from other files
34  module.exports = User;
```

Controllers

app_controller.js



```
  // Method to get a detailed report per specific month and year - for a specific user
  async report(req, res) {
    // Destructuring query parameters
    const { user_id, year, month } = req.query;
    try {
      // Finding costs matching the specified criteria
      const report = await Cost.find({ user_id, year, month });

      // Formatting the report data
      const formattedReport = {};
      report.forEach(cost => {
        if (!formattedReport[cost.category]) {
          formattedReport[cost.category] = [];
        }
        // add the data to the category
        formattedReport[cost.category].push({
          day: cost.day,
          description: cost.description,
          sum: cost.sum
        });
      });

      // Predefined categories for the report
      const categories = ['food', 'health', 'housing', 'sport', 'education', 'transportation', 'other'];
      const finalReport = {};
      categories.forEach(category => {
        // Including each category in the final report, even if no costs exist for it
        finalReport[category] = formattedReport[category] || [];
      });

      // Sending the final report as JSON response
      res.json(finalReport);
    } catch (error) {
        // Throw a custom error of type ReportGenerationError
        throw new ReportGenerationError(error.message);
    }
  };

  // Method to get information about the developers
  about(req, res) {
    // Developer information
    const developers = [
      { firstname: 'Hila', lastname: 'Itzhak', id: 209323955, email: 'hila87219@gmail.com' },
      { firstname: 'Omer', lastname: 'Peled', id: 315110015, email: 'opeled6@gmail.com' }
    ];

    // Sending developer information as JSON response
    res.json(developers);
  };
}

// Exporting the AppController class to make it accessible from other files
module.exports = { AppController };
```

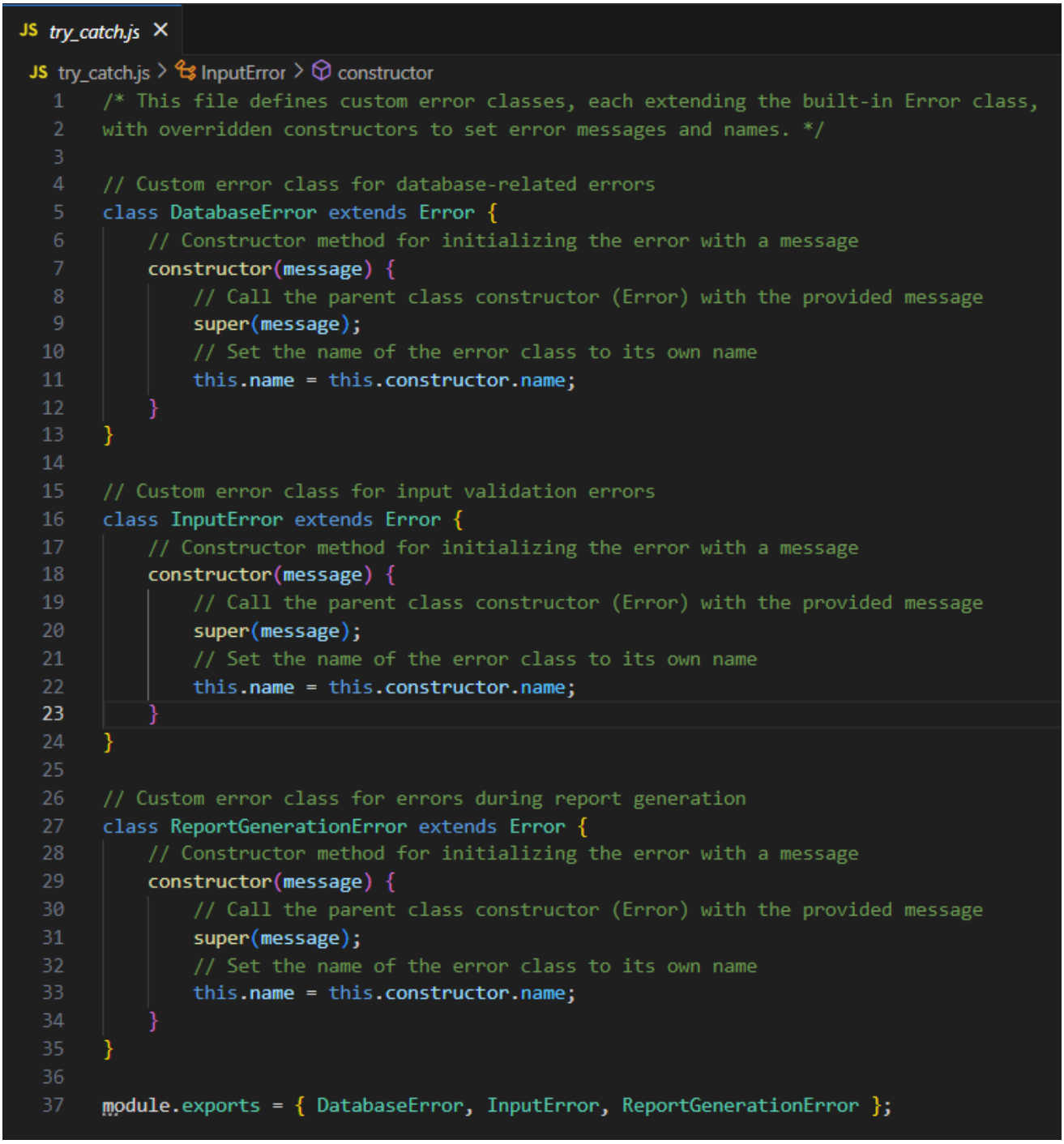
Routes

app_routes.js

routes > JS app_routes.js > AppRouter > setRoutes

```
/* The file defines an AppRouter class that sets up routes and binds them to controller methods,
facilitating routing functionality within a Node.js application using Express.js. */

// Importing the Router module from Express.js
const { Router } = require('express');

// Importing the AppController class
const { AppController } = require('../controllers/app_controller');

// Definition of the AppRouter class
class AppRouter {

    // Constructor that initializes the router
    constructor() {
        this.init();
    }

    // Initialization method to set up the router, controller, and routes
    init() {
        this.setRouter(); // Method to set up the router
        this.setAppController(); // Method to create an instance of the AppController
        this.setRoutes(); // Method to define the routes and bind them to controller methods
    }

    setRouter() {
        this.router = Router();
    }

    setAppController() {
        this.appController = new AppController();
    }

    setRoutes() {
        // Define route handlers for specific HTTP methods and URL paths
        this.router.post('/addcost', this.appController.addCost.bind(this.appController));
        this.router.get('/report', this.appController.report.bind(this.appController));
        this.router.get('/about', this.appController.about.bind(this.appController));
    }

    // Method to get the configured Express Router
    getRouter() {
        return this.router;
    }
}

// Exporting the AppRouter class to make it accessible from other files
module.exports = { AppRouter };
```

try_catch.js



```
JS try_catch.js > InputError > constructor
1   /* This file defines custom error classes, each extending the built-in Error class,
2   with overridden constructors to set error messages and names. */
3
4   // Custom error class for database-related errors
5   class DatabaseError extends Error {
6       // Constructor method for initializing the error with a message
7       constructor(message) {
8           // Call the parent class constructor (Error) with the provided message
9           super(message);
10          // Set the name of the error class to its own name
11          this.name = this.constructor.name;
12      }
13  }
14
15  // Custom error class for input validation errors
16  class InputError extends Error {
17      // Constructor method for initializing the error with a message
18      constructor(message) {
19          // Call the parent class constructor (Error) with the provided message
20          super(message);
21          // Set the name of the error class to its own name
22          this.name = this.constructor.name;
23      }
24  }
25
26  // Custom error class for errors during report generation
27  class ReportGenerationError extends Error {
28      // Constructor method for initializing the error with a message
29      constructor(message) {
30          // Call the parent class constructor (Error) with the provided message
31          super(message);
32          // Set the name of the error class to its own name
33          this.name = this.constructor.name;
34      }
35  }
36
37  module.exports = { DatabaseError, InputError, ReportGenerationError };
```