

Cost Manager Project

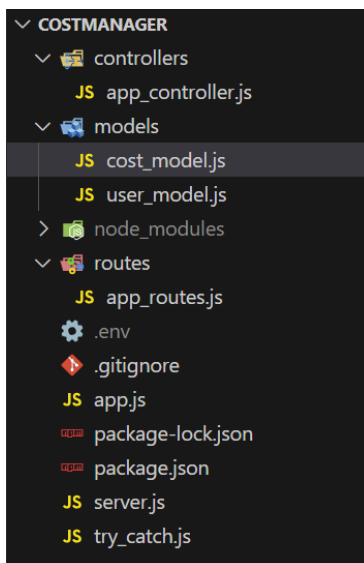
team manager: Omer Peled

team members: Omer Peled, 315110015, 0508334100, opeled6@gmail.com

Hila Itzhak, 209323955, 0504083499, hila87219@gmail.com

YouTube: [קישור ליוטיוב](#)

code



main file – app.js

```
JS app.js ×
JS app.js > ...
1  /* The file serves as the entry point of the program,
2  creating and initializing an instance of the AppServer class to start the server
3  and handle incoming requests asynchronously in a Node.js application. */
4
5  // Importing the AppServer class - our main file
6  const { AppServer } = require("../server");
7
8  // Defining the main function, which serves as the entry point of the program
9  const main = async () => {
10     // Creating a new instance of the AppServer class
11     const server = new AppServer();
12
13     // Initializing the server asynchronously by calling its init method
14     await server.init();
15 }
16
17 // Calling the main function to start the program execution
18 main();
```

server.js

```
JS server.js > X
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

JS cost_model.js X

models > JS cost_model.js > costSchema > id

```
1 // The file defines a Mongoose schema and model for representing Cost data, with specific fields.
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 costSchema using the Schema class
10 const costSchema = new Schema({
11   // Definition of fields for the cost document
12   user_id: {
13     type: String,
14     required: true // Indicates that this field is required
15   },
16   year: {
17     type: Number,
18     required: true
19   },
20   month: {
21     type: Number,
22     required: true
23   },
24   day: {
25     type: Number,
26     required: true
27   },
28   id: {
29     type: String,
30   },
31   description: {
32     type: String,
33     required: true
34   },
35   category: {
36     type: String,
37     required: true
38   },
39   sum: {
40     type: Number,
41     required: true
42   }
43 });
44
45 // Creating a model named "Cost" using the costSchema
46 const Cost = mongoose.model("Cost", costSchema);
47
48 // Exporting the Cost model to make it accessible from other files
49 module.exports = Cost;
50
```

user_model.js

JS user_model.js X

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

JS app_controller.js M X

controllers > JS app_controller.js > AppController > addCost

```
1  /* The file defines an AppController class with methods to handle adding new cost items,
2  generating detailed reports, and retrieving developer information through HTTP requests */
3
4  // Importing the Cost model
5  const Cost = require('../models/cost_model');
6  // Importing custom error types
7  const { InputError, DatabaseError, ReportGenerationError } = require('../try_catch');
8
9  // Definition of the AppController class
10 class AppController {
11   constructor() {}
12
13   // Method to add new cost items
14   async addCost(req, res) {
15     try {
16       // Destructuring request body to extract required fields
17       const { user_id, year, month, day, description, category, sum } = req.body;
18
19       // Creating a new Cost instance with extracted fields
20       const cost = new Cost({ user_id, year, month, day, description, category, sum });
21       // Saving the new Cost instance to the database and sending a response with the created cost's ID
22       await cost.save().then((newCost) => {res.status(201).json({ message: "success", id: newCost._id })});
23     } catch (error) {
24       // Handling different types of errors and sending appropriate responses
25       if (error instanceof DatabaseError) {
26         res.status(500).send(`Database Error: ${error.message}`);
27       } else if (error instanceof InputError) {
28         res.status(400).send(`Input Error: ${error.message}`);
29       } else {
30         res.status(500).send(`Internal Server Error: ${error.message}`);
31       }
32     }
33   };
34 }
```

```

35 // Method to get a detailed report per specific month and year - for a specific user
36 async report(req, res) {
37   // Destructuring query parameters
38   const { user_id, year, month } = req.query;
39   try {
40     // Finding costs matching the specified criteria
41     const report = await Cost.find({ user_id, year, month });
42
43     // Formatting the report data
44     const formattedReport = {};
45     report.forEach(cost => {
46       if (!formattedReport[cost.category]) {
47         formattedReport[cost.category] = [];
48       }
49       // add the data to the category
50       formattedReport[cost.category].push({
51         day: cost.day,
52         description: cost.description,
53         sum: cost.sum
54       });
55     });
56
57     // Predefined categories for the report
58     const categories = ['food', 'health', 'housing', 'sport', 'education', 'transportation', 'other'];
59     const finalReport = {};
60     categories.forEach(category => {
61       // Including each category in the final report, even if no costs exist for it
62       finalReport[category] = formattedReport[category] || [];
63     });
64
65     // Sending the final report as JSON response
66     res.json(finalReport);
67   } catch (error) {
68     // Throw a custom error of type ReportGenerationError
69     throw new ReportGenerationError(error.message);
70   }
71 };
72
73 // Method to get information about the developers
74 about(req, res) {
75   // Developer information
76   const developers = [
77     { firstname: 'Hila', lastname: 'Itzhak', id: 209323955, email: 'hila87219@gmail.com' },
78     { firstname: 'Omer', lastname: 'Peled', id: 315110015, email: 'opeled6@gmail.com' }
79   ];
80
81   // Sending developer information as JSON response
82   res.json(developers);
83 };
84 }
85
86 // Exporting the AppController class to make it accessible from other files
87 module.exports = { AppController };
88

```

Routes

app_routes.js

JS app_routes.js X

```
routes > JS app_routes.js > AppRouter > setRoutes
1  /* The file defines an AppRouter class that sets up routes and binds them to controller methods,
2  facilitating routing functionality within a Node.js application using Express.js. */
3
4  // Importing the Router module from Express.js
5  const { Router } = require('express');
6
7  // Importing the AppController class
8  const { AppController } = require('../controllers/app_controller');
9
10 // Definition of the AppRouter class
11 class AppRouter {
12
13     // Constructor that initializes the router
14     constructor() {
15         this.init();
16     }
17
18     // Initialization method to set up the router, controller, and routes
19     init() {
20         this.setRouter(); // Method to set up the router
21         this.setAppController(); // Method to create an instance of the AppController
22         this.setRoutes(); // Method to define the routes and bind them to controller methods
23     }
24
25     setRouter() {
26         this.router = Router();
27     }
28
29     setAppController() {
30         this.appController = new AppController();
31     }
32
33     setRoutes() {
34         // Define route handlers for specific HTTP methods and URL paths
35         this.router.post('/addcost', this.appController.addCost.bind(this.appController));
36         this.router.get('/report', this.appController.report.bind(this.appController));
37         this.router.get('/about', this.appController.about.bind(this.appController));
38     }
39
40     // Method to get the configured Express Router
41     getRouter() {
42         return this.router;
43     }
44 }
45
46 // Exporting the AppRouter class to make it accessible from other files
47 module.exports = { AppRouter };
```


try_catch.js

JS try_catch.js X

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 };
```