

Team Manager:

- Omri Shmuel

Team Members:

Full Name	Id	Phone	Email
1. Omri Shmuel	318688652	050-2578783	Omri164@outlook.com
2. Astewuol Tsagaow	321072670	053-2776230	omerzadi2014@gmail.com
3. Omer Tzaadi	207450628	054-8815358	astul1996@gmail.com

[video](#)

App.js

```
1  const createError = require('http-errors');
2  const mongoose = require('mongoose');
3  mongoose.Promise = global.Promise;
4  const express = require('express');
5  const path = require('path');
6  const cookieParser = require('cookie-parser');
7  const logger = require('morgan');
8
9  const indexRouter = require('./routes/index');
10 const usersRouter = require('./routes/users');
11 const apiRouter = require('./routes/api');
12
13 const connectDB = require('./models/database');
14
15 const app = express();
16
17 // connect to mongodb
18 connectDB()
19 //mongoose.Promise = global.Promise;
20
21 // view engine setup
22 app.set('views', path.join(__dirname, 'views'));
23 app.set('view engine', 'pug');
24
25 app.use(logger('dev'));
26 app.use(express.json());
27 app.use(express.urlencoded({ extended: false }));
28 app.use(cookieParser());
29 app.use(express.static(path.join(__dirname, 'public')));
30
31 app.use('/', indexRouter);
32 app.use('/users', usersRouter);
33 app.use('/api', apiRouter);
34
35 // catch 404 and forward to error handler
36 app.use(function(req, res, next) {
37   next(createError(404));
38 });
39
40 // error handler
41 app.use(function(err, req, res, next) {
42   // set locals, only providing error in development
43   res.locals.message = err.message;
44   res.locals.error = req.app.get('env') === 'development' ? err : {};
45
46   // render the error page
47   res.status(err.status || 500);
48   res.render('error');
49 });
50
51 module.exports = app;
```

api.js

```
const express = require('express');
const router = express.Router();
const Cost = require('../models/costs');
const User = require('../models/users');

/**
 * @route POST /api/add
 * @desc Adds a new cost entry for a user
 * @access Public
 */
router.post('/add', async (req, res) => {
  try {
    const { userid, category, description, sum, date } = req.body;

    // Check if all required fields are provided
    if (!userid || !category || !description || !sum) {
      return res.status(400).json({ error: 'Missing required fields: userid, category, description, sum' });
    }

    // If date is not provided, use the current date and time
    const costDate = date ? new Date(date) : new Date();

    // Create a new cost item
    const cost = new Cost({
      userid,
      category,
      description,
      sum,
      date: costDate
    });

    // Save the new cost item to the database
    const savedCost = await cost.save();

    // Respond with the newly added cost item
    res.status(201).json(savedCost);
  } catch (error) {
    // Catch any errors and return them
    res.status(500).json({ error: error.message });
  }
});

/**
 * @route GET /api/report
 * @desc Fetches monthly report for a specific user
 * @access Public
 */
router.get('/report', async (req, res) => {
  const { id: userid, year, month } = req.query;

  // Validate input parameters
  if (!userid || !year || !month) {
    return res.status(400).json({ error: 'Missing required query parameters: id, year, month' });
  }
});
```

```
try {
  const startOfMonth = new Date(year, month - 1, 1); // Start of month
  const endOfMonth = new Date(year, month, 0); // End of month

  // Fetch the costs within the month and year range
  const costs = await Cost.find({
    userid: userid,
    date: { $gte: startOfMonth, $lte: endOfMonth }
  });

  if (costs.length === 0) {
    return res.status(404).json({ error: 'No costs found for this user in the specified month and year'
  })

  // Prepare the response structure
  const categories = {
    food: [],
    education: [],
    health: [],
    housing: []
  };

  // Group costs by category
  costs.forEach(cost => {
    const { category, sum, description, date } = cost;
    const day = new Date(date).getDate(); // Extract the day

    if (categories[category]) {
      categories[category].push({ sum, description, day });
    }
  });

  // Format the final report
  const report = {
    userid: userid,
    year: parseInt(year),
    month: parseInt(month),
    costs: Object.keys(categories).map(category => ({ [category]: categories[category] })))
  };

  res.json(report);
} catch (error) {
  res.status(500).json({ error: error.message });
}
});

/**
 * @route GET /api/users/:id
 * @desc Retrieves user details and their total spending
 * @access Public
 */
router.get('/users/:id', async (req, res) => {
  const userId = req.params.id;

  try {
    const user = await User.findOne({ id: userId });
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
  }
});
```

```
116 // Calculate total costs
117
118 const totalCosts = await Cost.aggregate([
119   { $match: { userid: userId } },
120   { $group: { _id: null, total: { $sum: '$sum' } } },
121   ]);
122
123 res.json({
124   first_name: user.first_name,
125   last_name: user.last_name,
126   id: user.id,
127   marital_status: user.marital_status,
128   total: totalCosts[0]?.total || 0,
129 });
130 } catch (error) {
131   res.status(500).json({ error: error.message });
132 }
133 });
134
135 /**
136  * @route GET /api/about
137  * @desc Provides information about the development team
138  * @access Public
139  */
140 router.get('/about', async (req, res) => {
141   const team = [
142     { first_name: 'Omri', last_name: 'Shmuel' },
143     { first_name: 'Omer', last_name: 'Tzaadi' },
144     { first_name: 'Astewuol', last_name: 'Tsagaow' },
145   ];
146
147   res.json(team);
148 });
149
150 module.exports = router;
```

database.js

```
1  const mongoose = require('mongoose');
2
3  /**
4   * @description MongoDB connection utility
5   * @module database/connection
6   */
7
8  const uri = "mongodb+srv://OmriShmuel:Omr!3147@cluster0.mrr1c.mongodb.net/CostManager?retryWrites=true&w=majori
9
10 /**
11  * @func connectDB
12  * @desc Attempts to connect to MongoDB using Mongoose
13  */
14  const connectDB = async () => {
15    try {
16      await mongoose.connect(uri, {
17        useNewUrlParser: true,
18        useUnifiedTopology: true,
19      });
20      console.log('MongoDB connected');
21    } catch (error) {
22      console.error('Database connection error:', error);
23      process.exit(1); // Exit process with failure
24    }
25  };
26
27  module.exports = connectDB;
```

costSchema

```
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3
4  /**
5   * @description Defines the Cost schema for MongoDB
6   * @module models/costs
7   */
8
9  const costSchema = new Schema({
10    description: { type: String, required: true }, // Short description of the cost
11    sum: { type: Number, required: true }, // The amount spent
12    category: { type: String }, // Category of expense (e.g., food, education, etc.)
13    userid: { type: String, required: true }, // ID of the user associated with the cost
14    date: { type: Date, default: Date.now } // Date of the expense (defaults to now)
15  });
16
17  /**
18   * @description Mongoose model for the Costs collection
19   */
20  const Costs = mongoose.model('costs', costSchema);
21
22  module.exports = Costs;
```

PDF document made with [CodePrint.org](https://codeprint.org)

userSchema

```
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3
4  /**
5   * @description Defines the User schema for MongoDB
6   * @module models/users
7   */
8
9  const userSchema = new Schema({
10    id: { type: String, required: true }, // Unique user id
11    first_name: { type: String, required: true }, // User's first name
12    last_name: { type: String, required: true }, // User's last name
13    birthday: { type: Date, required: true }, // Date of birth
14    marital_status: { type: String, required: true } // Marital status (single, married, etc.)
15  });
16
17  /**
18   * @description Mongoose model for the Users collection
19   */
20  const Users = mongoose.model('users', userSchema);
21
22  module.exports = Users;
```


Snapshots of Jira, and Slack Alternatives (Discord)



