Cost Manager - Final Project

Team Manager: Omer Morim

Team Members:

- Omer Morim | ID: 302867908 | Phone: [Add Here] | Email: omermorim29@gmail.com
- Roni Ben Shoshan | ID: 211441456 | Phone: [Add Here] | Email: [Add Here]

Video Link (YouTube): [Paste Link Here]

Additional Comments / Notes:

[Write any relevant comments or notes here...]

index.js

```
/**
 * @file index.js
 * @description Main application entry point. Sets up Express app and connects to MongoDB.
 */


const express = require("express"); // Import the Express framework
const mongoose = require("mongoose"); // Import Mongoose for MongoDB interactions
const bodyParser = require("body-parser"); // Import Body-Parser to handle JSON requests
require("dotenv").config(); // Load environment variables from a .env file

const app = express(); // Create an Express application

app.use(bodyParser.json()); // Enable JSON request body parsing

// Connect to MongoDB using the connection string from environment variables
mongoose.connect(process.env.MONGO_URI, {
    serverSelectionTimeoutMS: 5000, // Timeout for server selection set to 5 seconds
})
    .then(() => console.log("Connected to MongoDB")) // Log success message if connected
       .catch((err) => console.error("Failed to connect to MongoDB", err)); // Log error if
connection fails

// Import route handlers
const usersRoutes = require("./routes/users");
const costsRoutes = require("./routes/costs");
const aboutRoutes = require("./routes/about");

// Define API routes
app.use("/api/users", usersRoutes); // Routes for user-related operations
app.use("/api", costsRoutes); // Routes for cost-related operations
app.use("/api", aboutRoutes); // Route for team/about information

// Define the port, using the environment variable if available, otherwise default to 3000
const PORT = process.env.PORT || 3000;

// Start the server and listen for incoming requests
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

// Export the app instance for use in testing or external modules
module.exports = app;
```

routes/about.js

```
/**
 * @file routes/about.js
 * @description Provides the team member details (ID, name, birthday, marital status).
 */
const express = require('express');
const router = express.Router();

// Route: Get team member full details
router.get('/about', (req, res) => {
    res.json([
        {
            id: 211441456,
            first_name: 'Roni',
            last_name: 'Ben shoshan',
            birthday: '28-07-2000',
            marital_status: 'single'
        },
        {
            id: 302867908,
            first_name: 'Omer',
            last_name: 'Morim',
            birthday: '13-05-1990',
            marital_status: 'single'
        }
    ]);
});

module.exports = router;
```

routes/costs.js

```
/**
 * @file routes/costs.js
 * @description API for handling cost-related operations (add cost, generate monthly report).
 */

const express = require('express'); // Import the Express framework
const router = express.Router(); // Create an Express router instance
const Cost = require('../models/cost'); // Import the Cost model

/**
 * @route POST /api/add
 * @group Costs - Add a new cost item
 * @param {string} userid.body.required - ID of the user
 * @param {string} category.body.required - Category of the cost
 * @param {number} sum.body.required - Sum of the cost
 * @param {string} description.body.required - Description of the cost
 * @returns {object} 200 - The created cost item
 * @returns {object} 400 - Missing or invalid fields
 * @returns {object} 500 - Internal server error
 */
router.post('/add', async (req, res) => {
    try {
        const { userid, sum, category, description } = req.body;

        if (!userid) {
            return res.status(400).json({
                error: 'Bad Request',
                message: "Missing 'userid' field."
            });
        }

        if (!sum || !category || !description) {
            return res.status(400).json({
                error: 'Bad Request',
                message: "Missing required fields: 'sum', 'category', or 'description'."
            });
        }

        const cost = new Cost({ userid, sum, category, description });
        const savedCost = await cost.save();

        return res.status(200).json(savedCost);

    } catch (err) {
        console.error('Error saving cost:', err);
        res.status(500).json({
            error: 'Internal Server Error',
            message: `An error occurred while saving the cost: ${err.message}`
        });
    }
});
```

```
/**
 * @route GET /api/report
 * @group Costs - Get cost report for a user
 * @param {string} id.query.required - ID of the user
 * @param {number} year.query.required - Year of the report
 * @param {number} month.query.required - Month of the report
 * @returns {object} 200 - JSON report grouped by categories
 * @returns {object} 400 - Missing query parameters
 * @returns {object} 500 - Internal server error
 */
router.get('/report', async (req, res) => {
    try {
        const { id, year, month } = req.query;

        if (!id) {
            return res.status(400).json({
                error: 'Bad Request',
                message: "Missing 'id' query parameter."
            });
        }

        if (!year) {
            return res.status(400).json({
                error: 'Bad Request',
                message: "Missing 'year' query parameter."
            });
        }

        if (!month) {
            return res.status(400).json({
                error: 'Bad Request',
                message: "Missing 'month' query parameter."
            });
        }

        const startDate = new Date(year, month - 1, 1);
        const endDate = new Date(year, month, 0, 23, 59, 59, 999);

        const costs = await Cost.aggregate([
            {
                $match: {
                    userid: id,
                    date: {
                        $gte: startDate,
                        $lt: endDate,
                    },
                },
            },
            {
                $group: {
                    _id: { category: '$category' },
                             items: { $push: { sum: '$sum', description: '$description', day: {
$dayOfMonth: '$date' } } }
                },
            },
            {
                $sort: { '_id.category': 1 },
            }
        ]);

        const categories = ["food", "health", "housing", "sport", "education"];
        let report = categories.map(category => ({ [category]: [] }));

        costs.forEach(cost => {
            const category = report.find(r => Object.keys(r)[0] === cost._id.category);
            if (category) {
                category[cost._id.category] = cost.items.map(item => ({
                    sum: item.sum,
                    description: item.description,
                    day: item.day
                }));
            }
        });

        report.sort((a, b) => {
            const aValues = Object.values(a)[0].length;
            const bValues = Object.values(b)[0].length;
            return aValues === 0 ? 1 : bValues === 0 ? -1 : 0;
        });

        res.status(200).json({
            userid: parseInt(id),
            year: parseInt(year),
            month: parseInt(month),
            costs: report
        });

    } catch (error) {
        console.error('Error fetching report:', error);
        res.status(500).json({
            error: 'Internal Server Error',
            message: `An error occurred while fetching the report: ${error.message}`
        });
    }
});

module.exports = router;
```

routes/users.js

```
/**
 * @file routes/users.js
 * @description API for handling user-related operations (fetch user info, total costs).
 * @route GET /api/users/:id
 * @group Users - Operations related to users
 * @param {string} id.path.required - ID of the user
 * @returns {object} 200 - User details and total cost
 * @returns {object} 404 - User not found
 * @returns {object} 500 - Internal server error
 */



const express = require('express'); // Import the Express framework
const router = express.Router(); // Create an Express router instance
const User = require('../models/user'); // Import the User model
const Cost = require('../models/cost'); // Import the Cost model

// Route: Get user details by ID
router.get('/:id', async (req, res) => {
    try {
        const userId = req.params.id.trim(); // Remove leading and trailing spaces from the ID

        // Find the user by ID
        const user = await User.findOne({ id: userId });
        if (!user) {
            return res.status(404).json({ error: 'User not found' });
        }

        // Calculate the total cost for the user
        const totalCosts = await Cost.aggregate([
            {
                $match: { userid: userId } // Filter costs by user ID
            },
            {
                $group: {
                    _id: null, // Group all results together
                    total: { $sum: '$sum' } // Sum the 'sum' field across all matched documents
                }
            }
        ]);

        // Extract total cost from aggregation result (default to 0 if no costs exist)
        const total = totalCosts.length > 0 ? totalCosts[0].total : 0;

        // Return the user details along with total cost
        res.json({
            first_name: user.first_name,
            last_name: user.last_name,
            id: user.id,
            total: total
        });
    } catch (err) {
        console.error('Error fetching user details:', err);
        res.status(500).json({ error: 'An error occurred while fetching the user details' });
    }
});

// Export the router to make it available for use in other parts of the application
module.exports = router;
```

models/user.js

```
/**
 * @file models/user.js
 * @description Mongoose schema and model for users.
 */

const mongoose = require('mongoose');

/**
 * Mongoose schema for a user.
 * @typedef User
 * @property {string} id - Unique user ID
 * @property {string} first_name - First name of the user
 * @property {string} last_name - Last name of the user
 * @property {Date} birthday - Birth date of the user
 * @property {string} marital_status - Marital status (e.g., single, married)
 */
const userSchema = new mongoose.Schema({
    id: { type: String, required: true, unique: true },
    first_name: { type: String, required: true },
    last_name: { type: String, required: true },
    birthday: { type: Date, required: true },
    marital_status: { type: String, required: true }
});

module.exports = mongoose.model('User', userSchema);
```

models/cost.js

```
/**
 * @file models/cost.js
 * @description Mongoose schema and model for cost items.
 */

const mongoose = require('mongoose');

/**
 * Mongoose schema for a cost item.
 * @typedef Cost
 * @property {string} userid - ID of the user to whom the cost belongs
 * @property {string} description - Description of the cost item
 * @property {string} category - One of: food, health, housing, sport, education
 * @property {number} sum - Amount of the cost
 * @property {Date} [date] - Date of the cost (defaults to current date if not provided)
 */
const costSchema = new mongoose.Schema({
    userid: { type: String, required: true },
    description: { type: String, required: true },
    category: {
        type: String,
        required: true,
        enum: ['food', 'health', 'housing', 'sport', 'education']
    },
    sum: { type: Number, required: true },
    date: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Cost', costSchema);
```

tests/about.test.js

```
/**
 * @file tests/about.test.js
 * @description Unit tests for the /api/about endpoint using Jest and Supertest.
 * Verifies that team member details are returned correctly and completely.
 */

const request = require('supertest');
const app = require('../index');
const mongoose = require('mongoose');

/**
 * @group About API
 * @description Tests the /api/about endpoint for full team member details.
 */
describe('About API Endpoint', () => {

    /**
     * Test case: Should return a list of team members with full details.
     *
     * @returns {void}
     */
    it('should return team members with full details', async () => {
        const response = await request(app).get('/api/about');

        expect(response.status).toBe(200);
        expect(Array.isArray(response.body)).toBe(true);

        response.body.forEach(member => {
            expect(member).toHaveProperty('id');
            expect(member).toHaveProperty('first_name');
            expect(member).toHaveProperty('last_name');
            expect(member).toHaveProperty('birthday');
            expect(member).toHaveProperty('marital_status');

            // ?????? ????
            expect(typeof member.id).toBe('number');
            expect(typeof member.first_name).toBe('string');
            expect(typeof member.last_name).toBe('string');
            expect(typeof member.birthday).toBe('string');
            expect(typeof member.marital_status).toBe('string');
        });
    });
});

// Close DB connection after tests
afterAll(async () => {
    await mongoose.connection.close();
});
```

tests/cost.test.js

```
/**
 * @file tests/cost.test.js
 * @description Unit tests for cost-related API endpoints using Jest and Supertest.
 * Tests cover creating a new cost entry and validating error handling for missing fields.
 */

const request = require('supertest'); // Import Supertest for API testing
const app = require('../index'); // Import the main Express application

/**
 * Group of tests related to cost API endpoints.
 * @group Cost API
 */
describe('Cost API Endpoints', () => {

    /**
     * Test case: Create a new cost entry with valid data.
     * Sends a POST request to /api/add with a valid cost object.
     * Expects a 200 OK response and verifies the returned fields.
     *
     * @function
     * @name it - should create a new cost entry
     * @returns {void}
     */
    it('should create a new cost entry', async () => {
        const newCost = {
            description: 'clean',     // Cost description
            category: 'housing',      // Cost category
            userid: '1',              // User ID
            sum: 200                  // Cost amount
        };

        const response = await request(app)
            .post('/api/add')
            .send(newCost)
            .set('Content-Type', 'application/json');

        // Validations
        expect(response.status).toBe(200); // Expect status 200
        expect(response.body).toHaveProperty('_id'); // Expect returned object to have an _id
            expect(response.body.description).toBe(newCost.description); // Expect description to
match
        expect(response.body.category).toBe(newCost.category); // Expect category to match
    });

    /**
     * Test case: Fail to create a cost entry when a required field is missing.
     * Sends a POST request without the required `userid` field.
     * Expects a 400 Bad Request response with an appropriate error message.
     *
     * @function
     * @name it - should return 400 if a required field is missing
     * @returns {void}
     */
    it('should return 400 if a required field is missing', async () => {
        const incompleteCost = {
            description: 'Dinner', // Cost description
            category: 'food',      // Cost category
            sum: 100               // Cost amount (userid is missing)
        };

        const response = await request(app)
            .post('/api/add')
            .send(incompleteCost)
            .set('Content-Type', 'application/json');

        // Validations
        expect(response.status).toBe(400); // Expect status 400
        expect(response.body.error).toBe('Bad Request'); // Expect error key with message
    });
    const mongoose = require('mongoose');

    afterAll(async () => {
        await mongoose.connection.close();
    });
});
```

tests/report.test.js

```
/**
 * @file tests/report.test.js
 * @description Unit tests for the /api/report endpoint using Jest and Supertest.
 * Tests include successful report retrieval, empty reports, and validation errors.
 */

const request = require('supertest');
const app = require('../index');
const mongoose = require('mongoose');

/**
 * @group Report API
 * @description Tests for generating monthly cost reports via /api/report
 */
describe('Report API Endpoints', () => {

    /**
     * Test case: Successfully retrieves a monthly report for a known user.
     * Assumes user '123123' exists and has cost data for May 2025.
     *
     * @returns {void}
     */
    it('should return a monthly report for a user', async () => {
        const response = await request(app)
            .get('/api/report')
            .query({ id: '123123', year: '2025', month: '5' }); // May 2025

        expect(response.status).toBe(200);
        expect(response.body).toHaveProperty('costs');
        expect(Array.isArray(response.body.costs)).toBe(true);
    });

    /**
     * Test case: Returns empty categories if no data exists for the given month.
     *
     * @returns {void}
     */
    it('should return an empty costs array if no data is found', async () => {
        const response = await request(app)
            .get('/api/report')
            .query({ id: '123123', year: '1999', month: '1' }); // No data expected

        expect(response.status).toBe(200);
        expect(response.body).toHaveProperty('costs');
        expect(response.body.costs.length).toBe(5); // 5 categories always returned
        response.body.costs.forEach(category => {
            const values = Object.values(category)[0];
            expect(Array.isArray(values)).toBe(true);
        });
    });

    /**
```

```
     * Test case: Missing required query parameters should result in 400.
     *
     * @returns {void}
     */
    it('should return 400 if parameters are missing', async () => {
        const response = await request(app)
            .get('/api/report'); // No query parameters

        expect(response.status).toBe(400);
        expect(response.body.error).toBe('Bad Request');
    });
});

// Close MongoDB connection after tests
afterAll(async () => {
    await mongoose.connection.close();
});
```

tests/user.test.js

```
/**
 * @file tests/user.test.js
 * @description Unit tests for user-related endpoints using Jest and Supertest.
 * This file tests retrieving user details by ID and handling of non-existent users.
 */

const request = require('supertest'); // Simulate HTTP requests
const app = require('../index'); // Main Express application

/**
 * @group User API
 * @description Tests for the /api/users/:id endpoint to retrieve user information.
 */
describe('User API Endpoints', () => {

    /**
     * Test case: Successfully retrieves user details by ID.
     *
     * Sends a GET request to `/api/users/1` and expects:
     * - status 200
     * - `first_name` and `total` in the response
     *
     * @returns {void}
     */
    it('should return user details by id', async () => {
        const response = await request(app)
            .get('/api/users/123123'); // ?????? ????? ????

        expect(response.status).toBe(200); // ?????? ?????? ?????
        expect(response.body).toHaveProperty('first_name');
        expect(response.body.first_name).toBe('mosh'); // ???? ????
        expect(response.body).toHaveProperty('total'); // ???? ????? ?? ???? total
    });
    const mongoose = require('mongoose');

    afterAll(async () => {
        await mongoose.connection.close();
    });

    /**
     * Test case: Fails to find a user that does not exist.
     *
     * Sends a GET request to `/api/users/99999` (non-existent ID).
     * Expects:
     * - status 404
     * - error message: "User not found"
     *
     * @returns {void}
     */
    it('should return 404 if user not found', async () => {
        const response = await request(app)
            .get('/api/users/99999');
        expect(response.status).toBe(404); // Should return Not Found
           expect(response.body.error).toBe('User not found'); // Must include appropriate error
message
    });

});
```