

React.js Exam – Budget Planner

Use the web server from the resources, install all its dependencies and run it via node. It will respond on port 5000.

Problem 1. Create a React application

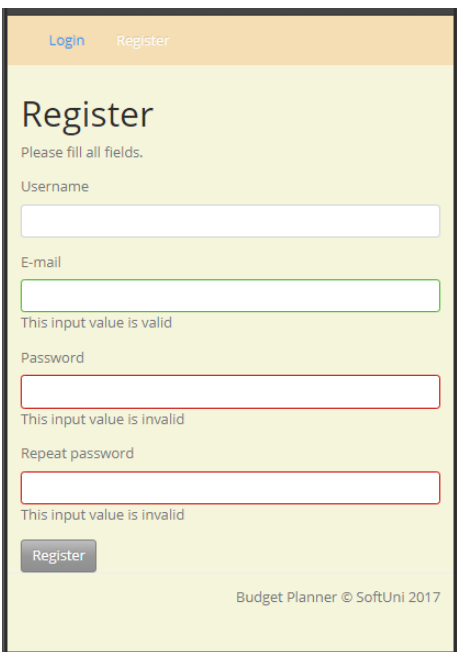
Create a React application and prepare the initial project structure. You will be creating a **personal budget planner** that only works for registered users. The user can **view** and **modify monthly budget**, **add** and **delete expenses**, and see a **yearly** summary. You may use the provided **HTML mock**, or write your own HTML, if you think it will save time – application appearance will **not** be evaluated.

Problem 2. Add Authentication

Make sure to **validate** all form values on the front-end and provide appropriate **error messages** to the user. The **name** and **e-mail** fields must not be empty and the **password** must be at least 4 characters long.

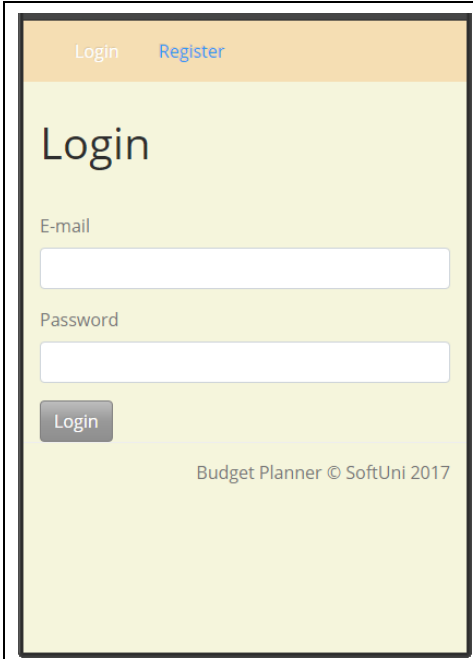
User Registration (Sign Up)

To register a user, you need to send a **POST** request to the server on **"/auth/signup"** with **"name"**, **"email"** and **"password"** data (sent as JSON):

	POST http://localhost:5000/auth/signup	
	Headers	Content-Type: application/json
	Request body	<pre>{ "name": "new_userd", "email": "pass1d23@abv.bg", "password": "New User" }</pre>
	Success	<pre>{ "success": true, "message": "You have successfully signed up! Now you should be able to log in." }</pre>
	Error response	<pre>{ "success": false, "message": "E-mail already exists!" }</pre>

User Login

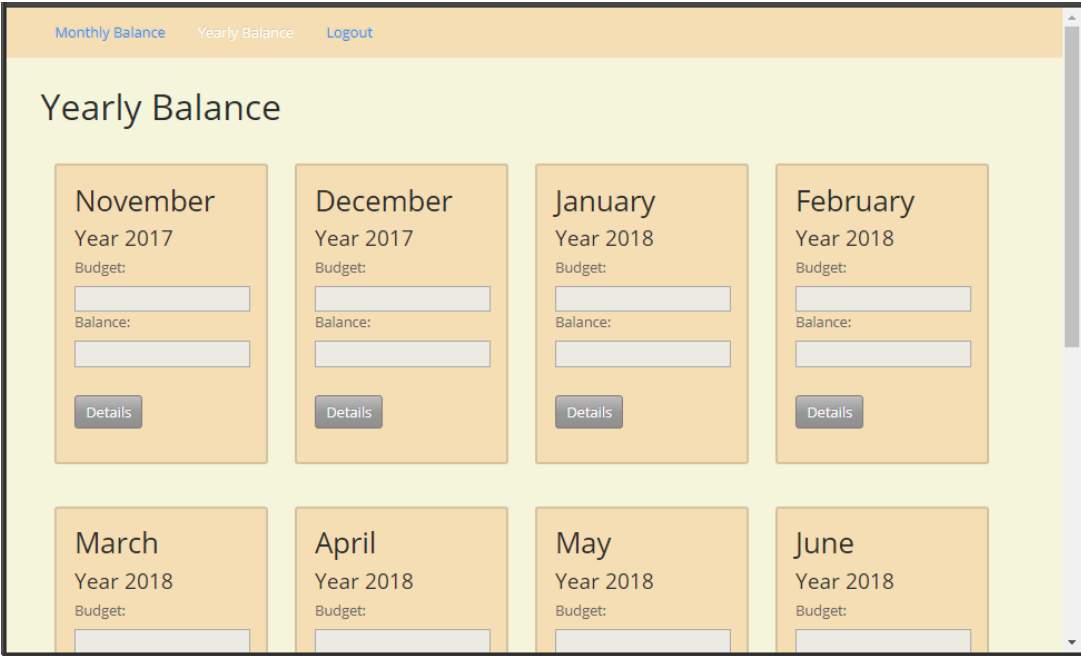
To login a user, you need to send a **POST** request to the server on `"/auth/login"` with **"email"** and **"password"** data (sent as JSON). You need to **save the user token** in your application state. Make sure you validate everything on the client application.

	POST <code>http://localhost:5000/auth/login</code>	
	Headers	<code>Content-Type: application/json</code>
	Request body	<pre>{ "email": "pass1d23@abv.bg", "password": "New User" }</pre>
	Success	<pre>{ "success": true, "message": "You have successfully logged in!", "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWQ", "user": { "name": "new_userd" } }</pre>
	Error response	<pre>{ "success": false, "message": "Incorrect email or password" }</pre>

Successful login returns a **"token"** which is later used to authenticate the CRUD operations.

Problem 3. List Yearly Balance (15 points)

Add a page where all details **for a full year** are shown. You need to make a **GET** request to `"/plan/{year}"` to retrieve the information. Link each month to its details page. This route is only for authenticated users so you need to send a header with **"Authorization"** name and value **"bearer {token}"** in order to pass the authentication checks. Make sure your React application **redirects to the login page**, if the user tries to open the hotel details page and he's not logged in.



GET http://localhost:5000/plan/:year	
Request headers	Authorization: bearer <authToken>
Success	<pre>{ "1": { "budget": 0, "balance": 0 }, "2": { "budget": 0, "balance": 0 }, // The rest of months }</pre>

Problem 4. Monthly Balance View (25 points)

Add a page where all details for the current month are shown. Make a **GET** request to `"/plan/{year}/{month}"` to retrieve the information about a specific month. Both **year** and **month** are numbers (**month** between 1 and 12). This route is only for authenticated users so you need to send a header with **"Authorization"** name and value **"bearer {token}"** in order to pass the authentication checks. Make sure your React application **redirects to the login page**, if the user tries to open the month details page and he's not logged in.

List Income and Budget (10 points)

Show the income and budget values, returned by the server. They should be populated in the form.

List Monthly Expenses (15 points)

Add a listing of all expenses for that month, taken from the array returned by the server.

[Monthly Balance](#)
[Yearly Balance](#)
[Logout](#)

Welcome to Budget Planner

Planner

Income:

Budget:

Save

November 2017

Expenses

Add expenses

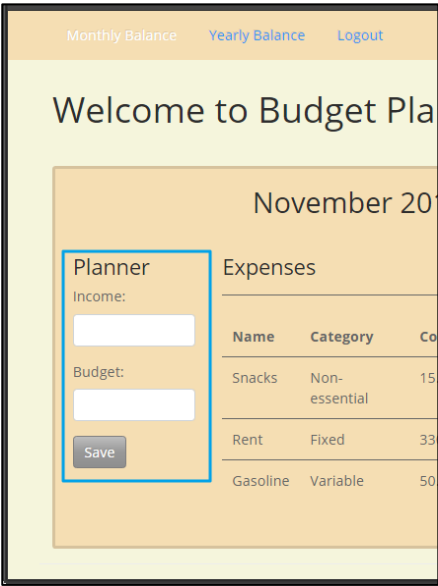
Name	Category	Cost	Payment Date	
Snacks	Non-essential	15.00	30-11-2017	Delete
Rent	Fixed	330.00	15-11-2017	Delete
Gasoline	Variable	50.00	-	Delete

Budget Planner © SoftUni 2017

GET http://localhost:5000/plan/:year/:month	
Request headers	Authorization: bearer <authToken>
Success	<pre>{ "income": 1300, "budget": 1100, "expenses": [{ "id": "3a5d2c4d-6b3e-dd81-493f-e1522c6c06d3", "year": 2017, "month": 11, "date": 11, "creationTime": 1511003119966, "name": "Sushi", "amount": 90, "category": "Food" }, // The rest of the expenses] }</pre>

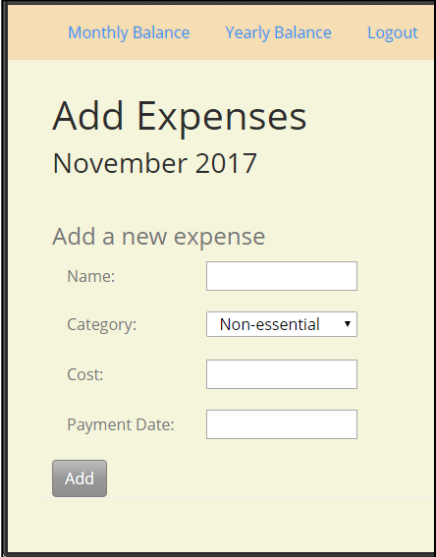
Problem 5. Update Monthly Income and Budget (15 points)

You should be able to update the **budget** and **income** values for each month through the form provided in monthly details page. The existing values **are replaced** by the newly provided values for the selected month. Make a **POST** request to **"/plan/{year}/{month}"** with the income (number) and budget (number) as part of the request body. This route is only for authenticated users so you need to send a header with **"Authorization"** name and value **"bearer {token}"** in order to pass the authentication checks. Make sure your React application **redirects to the login page**, if the user tries to open the hotel details page and he's not logged in.

	POST http://localhost:5000/plan/:year/:month	
	Headers	Content-Type: application/json Authorization: bearer <authToken>
	Request body	<pre>{ "income": 1400, "budget": 1100 }</pre>
	Success	<pre>{ "success": true, "message": "Plan saved successfully.", "plan": { "income": 1300, "budget": 1100 } }</pre>
	Error response	<pre>{ "success": false, "message": "Check the form for errors", "errors": { "year": "Year must be a number" } }</pre>

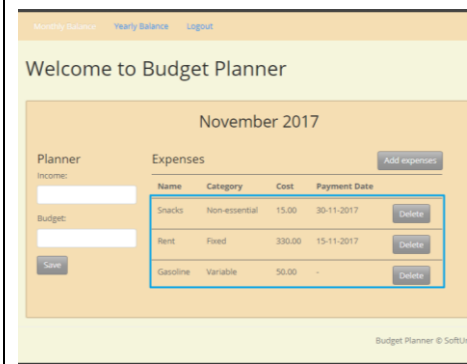
Problem 6. Add a new expense (20 points)

You should be able to add new expense through the **Add Expenses** page. You need to make a **POST** request to `"/plan/{year}/{month}/expense"` with the date (number), name (string), category (number) and amount (number). The category can be any string, but you may use the categories provided in the HTML skeleton. This route is only for authenticated users so you need to send a header with **"Authorization"** name and value **"bearer {token}"** in order to pass the authentication checks. Make sure your React application **redirects to the login page**, if the user tries to open the hotel details page and he's not logged in.

	POST <code>http://localhost:5000/plan/:year/:month/expense</code>	
	Headers	<code>Content-Type: application/json</code> <code>Authorization: bearer <authToken></code>
	Request body	<pre>{ "date": 11, "name": "Sushi", "category": "Food", "amount": 90.00 }</pre>
	Success	<pre>{ "success": true, "message": "Expense saved successfully.", "expense": { "id": "3a5d2c4d-6b3e-.", "year": 2017, "month": 11, "date": 11, "creationTime": 1511003119966, "name": "Sushi", "amount": 90, "category": "Food" } }</pre>
	Error response	<pre>{ "success": false, "message": "Check the form for errors", "errors": { "amount": "Amount must be a number" } }</pre>

Problem 7. Delete an existing expense (10 points)

You should be able to delete an existing expense through the monthly details page. You need to make a **DELETE** request to **"plan/expense/{expenseId}"**. This route is only for authenticated users so you need to send a header with **"Authorization"** name and value **"bearer {token}"** in order to pass the authentication checks. Make sure your React application **redirects to the login page**, if the user tries to open the hotel details page and he's not logged in.

	DELETE <code>http://localhost:5000/plan/expense/:expenseId</code>	
	Headers	Content-Type: application/json Authorization: bearer <authToken>
	Success	<pre>{ "success": true, "message": "Expense deleted successfully.", "expense": "3a5d2c4d..." }</pre>
	Error response	<pre>{ "success": false, "message": "Expense ID not found: 3a5d2c...", "errors": { "id": "Expense ID not found: 3a5d2c..." } }</pre>

Problem 8. Add Notifications (15 points)

Notify the user via popup or another method of successful actions or error.

Update Income and Budget (5 points)

Notify the user if the update was successful or if any errors have occurred.

Create Expense (5 points)

Notify the user if the creation was successful or if any errors have occurred.

Delete Expense (5 points)

Notify the user if the deletion was successful or if any errors have occurred.

Problem 9. (Bonus) Use a State Management Library (10 points)

You may optionally store your data and state, using a state management library, such as Flux, Redux, MobX or similar. For this task to count as completed, you must store and retrieve the **balance** and **expense** information through the library. Partial points will be awarded, but chose wisely.