1.1.3. Finalization phase

This web application is a goal-tracking tool. It gives opportunities to users to set and oversee their individual objectives. This consists of the client-side component created with ReactJS that renders the user interface (also here HTML, CSS) and the server-side component done using Node.js, on the basis of which the internal API functions; Express.js that manages data and CORS Middleware integrated into Express.js

Backend implementation

The server side of the application contains Node.js, which is used to create a server-side JavaScript runtime. In addition, here used Express.js framework, that makes it easier to create web applications with Node.js and is a popular tool for backend development.

Moreover, there is a node.js package CORS, which is a feature that enables frontend to make requests for resources on the server. The API responds to cross-origin requests securely via CORS.

The application utilizes the express.json() middleware for parsing receiving JSON data in order to process data, that is sent to the server. So, this is quite helpful, when managing data given in the request body, such in my case, while setting a new goal.

By HTTP requests, the frontend of application interacts with the server. In the backend there is initialization of an array ‘notes’, which is used as in-memory store for data. Therefore, in this array data on the user-created objectives is kept.

Besides, the application has several API endpoints:

endpoint for obtaining all goals, which uses the HTTP method GET;

endpoint for generating new goals, which uses the HTTP method POST;

endpoint for removing goals based on their array index, which uses the HTTP method DELETE.

Frontend implementation

The application is developed on the client side with React, a well-liked JavaScript user interface toolkit. The component-based design of React encourages declaratory and perfomance.

The root component of the application is ‘App’. It controls the user interface's state, especially the notes state variable, which indicates the list of goals. There are hooks:

‘useState’ for controlling state;

‘useEffect’ for fetching data from the server. This guarantees that the application starts with the most recent set of objectives. The ‘fetchNotes’ function oversees making a GET request

to the server to retrieve the list of goals. This obtained data modifies the ‘notes’ state, making it renderable.

The process of establishing and adding a new goal is handled by the ‘addSpecificGoal’ function. It makes a POST request to the server, accompanied with the JSON data for the new objective. When the server handles the request and produces the new goal, the state is updated to incorporate the objective.

The ‘deleteSpecificGoal’ function manages goal deletion. This method sends a DELETE request to the server when a user hits the "DONE!" button for a particular goal. The client gets the changed data and reflects the changes once the server refreshes the list of goals.

The components of the user interface are the ‘Header’, ‘Footer’, ‘Goal’, and ‘GoalsCreation’. There are typical for consistent structure and layout ‘Header’ and ‘Footer’ components of the application, and the ‘GoalsCreation’ component which allows users to establish new objectives. It displays a form with fields to enter for the goal's title and content, and after submission, it transmits the data to the server.

The ‘Goal’ component is in charge of generating individual objectives. Users can delete an objective by clicking the supplied button "DONE!". This procedure calls the ‘deleteSpecificGoal’ function and sends a request to the server to eliminate the goal from the array.