Service Oriented Architecture
ToDo List Application

Prundel Răzvan Ştefan 258

The application presented below is a simple ToDo List in which the user can create, update and delete tasks. The frontend is created in React, while the backend is created in NodeJS. All the tasks are stored in a MongoDB database.

The architecture for the solution is not complicated at all:

For the database we use MongoDB. It stores JSON-like documents without a specific structure.

For the backend we are using NodeJS and Express as the server. We are running a Javascript server side application (REST API) that will be the middleware between the client and the database.

For the frontend, the client side, we are using NodeJS, ReactJS and Bootstrap 3.

Each of the three services its deployed in its own container.

Mongodb: will use the image “mongo” and will expose the container port 27017 in the host machine.

Backend: will use the image “node”. Also, will expose the port 6200 in the host machine.

Frontend: will use the image “react”. Also, will expose the port 3000 in the host machine.

The container creation process works like this: first the mongo image starts, then the backend image and last but not least the frontend image.

Running the program

To run the program execute the following commands in the root directory of the project:

- docker-compose build
- docker-compose up

After running the commands open a browser windows and go to the address: http://localhost:3000/. Wait for all the processes to start. This may take a while, but if you have opened the local host in the browser it will automatically refresh and show the interface after all the processes have started.

At the respective address the user will be presented with the frontend interface and can start adding tasks.

Bellow you can see four diagrams that are explaining the project architecture.

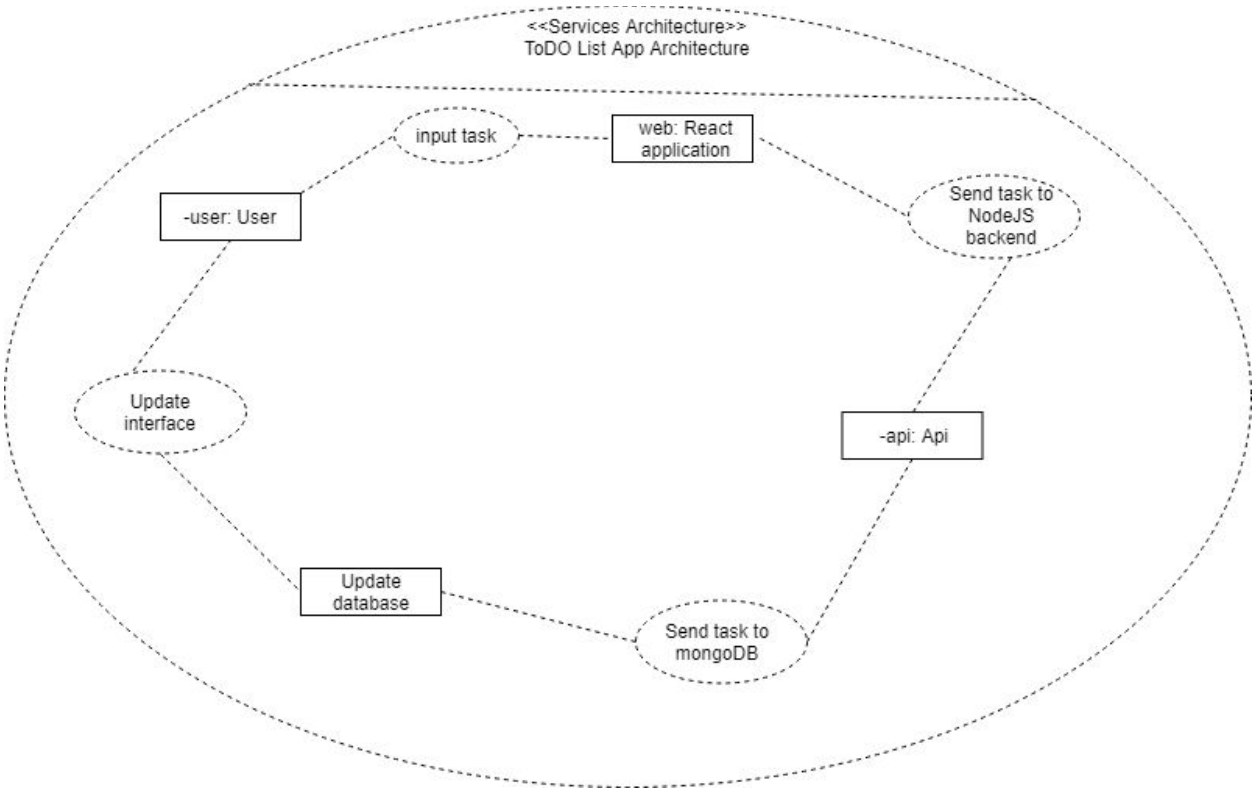

Diagram 1 - SOAML

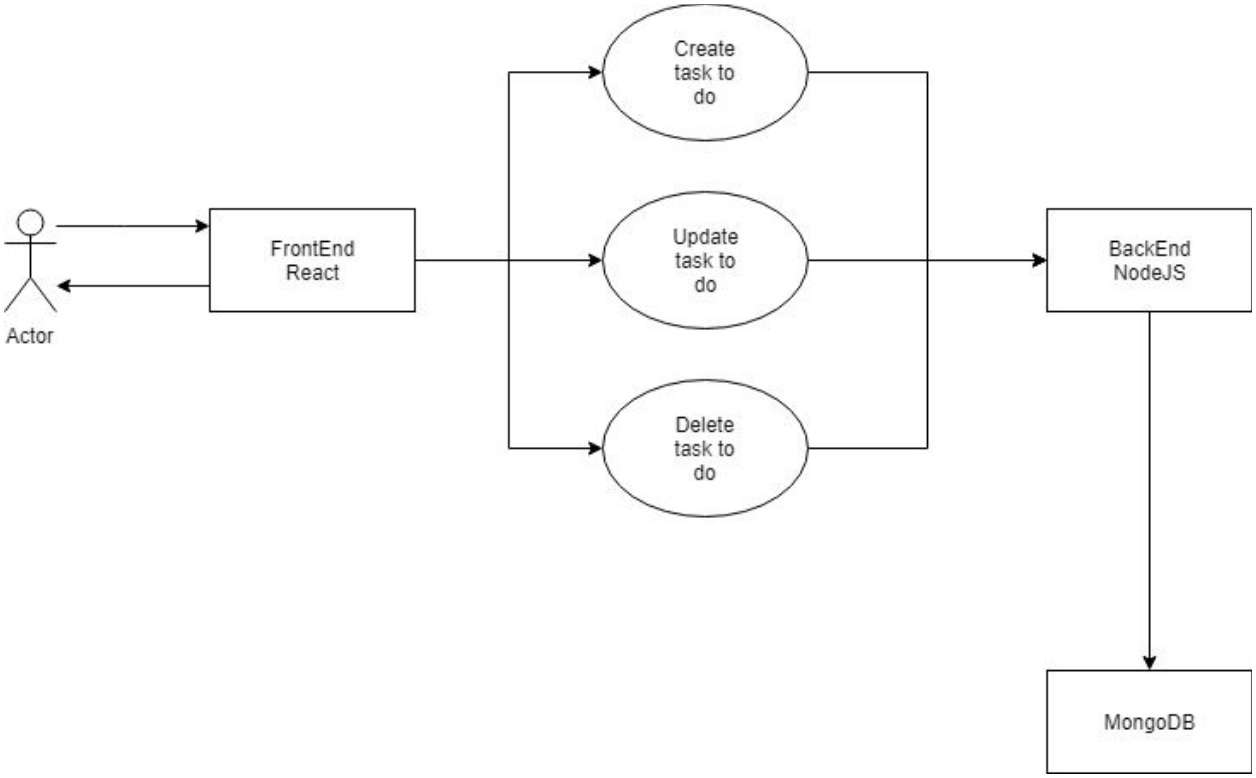

Diagram 2 - UML

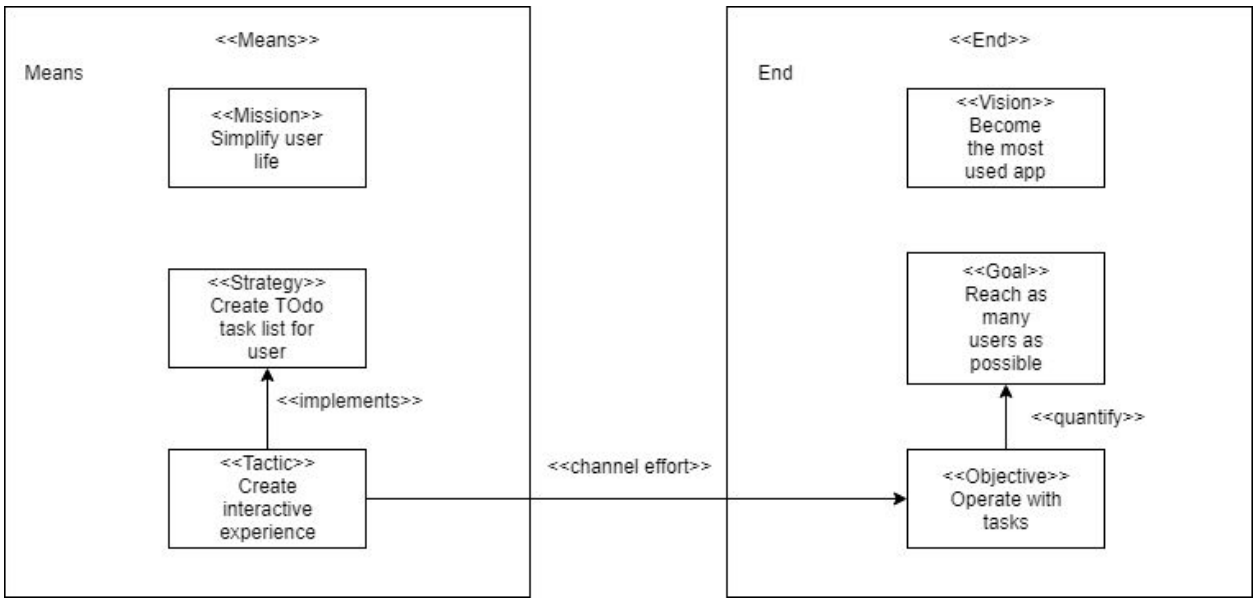

Diagram 3 - BMM

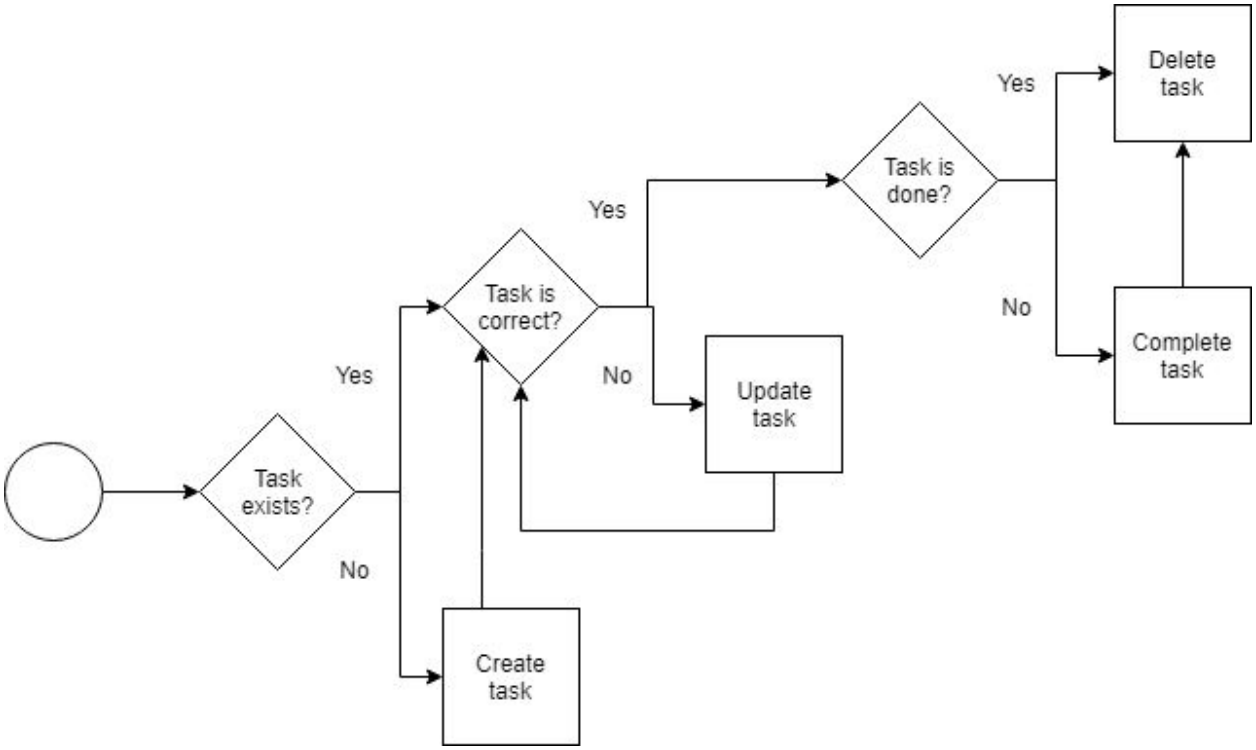

Diagram 4 - BPMN

The application uses a service facade that sits between the service and the contract. It eliminates the tight coupling between the service and its contract. This is intended to minimize changes to the service if there are changes to the contract. Inside the application we can see integration between processes and services. The component exposes the service is a way that is independent of its implementation.

In the classical client/server architectural pattern we have a client that requests data from the server over a protocol. Here we have a service requester/consumer that binds binds the service provider in order to invoke one of its web services.