Documentation

Visitors tracker backend

1. Introduction

This document provides the description of the app for Visitors management and contains next paragraphs:

- Purpose – description of how app could be used.
- Document conventions – data that is needed to understand a description of the app properly (list of definitions, some notes etc).
- References – some useful links.
- Product features – general description of application functionality.
- Operating environment – requirements to the hardware.
- Design and implementation constraints – general description of application architecture, list of used programming languages and technologies (frameworks, libraries etc).
- System features – detailed description of each software component.

1.1. Purpose:

Each company needs to save some data about people it works with, for example, about employees, customers, suppliers etc. The absence of this information could affect company`s success and could even lead to its closure.

The necessity of information about people has led to appearance of subsystems for employees and customers management in office software. Soon these subsystems were transformed into individual programs.

Today the development of office software isn`t stopped, and this software usually should contain subsystems for working with people. The software that would be described in this document is a good basis for such subsystems; it will provide you the information about how to change this app due to your requirements and how to use it in your project.

1.2. Document conventions:

This software was initially created to save data about company`s visitors, so people would be named as visitors. But you should remember that you can save data about any people using this app.

1.3. References:

Code of project: https://github.com/vladstudennikov/Visitors-Tracker.

2. Overall description:

2.1. Product features:

The program allows managing data about people: adding new data, modifying, and deleting records. To work with the application over the network, there`s a REST API available, allowing to manage data using HTTP protocol.

The program can be roughly divided into 3 parts: a database with information about people, a library for convenient work with database and a REST API for working with the program over the network. The database is a simple XML file; the database library includes a model that represents the database as a class, a library for validating data about people, which can be easily modified, and an API for higher-level operations with the model. The REST API enables working with the library over the network and allows you to integrate this program into other products as the component.

2.2. Operating environment:

Hardware requirements: RAM > 1Gb, permanent memory > 200 Mb.

OS – Windows 10, 11.

2.3. Design and implementation constraints:

Stack of technologies:

- JavaScript;
- Node JS (20.11.1);

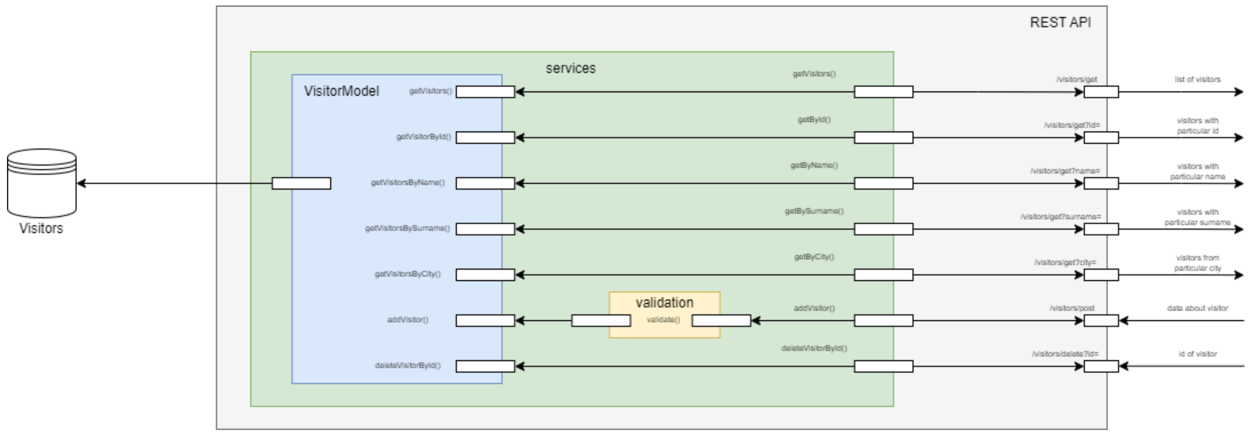
List of Node JS libraries:

- Express (4.19.1);
- Cors (2.8.5);
- Body-parser (1.20.2);

Description of software architecture:

The software could be represented as a set of small components, the main component is REST API, it uses the functionality of services component which contains basic functions to manage data in DB using VisitorModel - class to work with database. Also there`s used the validation module which automatically checks data before it`s adding.

Architecture of the app could be described using the diagram:



You can separate VisitorModel or services component and use them in other software. Also you can easily add your own database into project, but you should change the model. Your own Visitor model should be inherited from VisitorInterface class to make service layer work as it needed, if it would be inherited from VisitorInterface, then you have an ability not to change everything else in the project. Main class of services component is called VisitorApi, it contains all the functions of this software.

3. System features:

3.1. Visitor model:

Visitor model is the model of database, the place of the model in filesystem: ./models/Visitor.js. The class of model is named as Visitor.

By default database with visitors is placed in “db” folder, here you can see a single file named “visitors.xml” – in this file all visitor`s data is saved.

Attributes of visitor:

The attributes that are saved about visitor in the database include:

- Id – unique identifier of the visitor;
- Name;
- Surname;
- Patronym;
- City;
- Address;
- Passport – an object that includes next fields: number (the identifier of passport), issued_by – establishment that issued a passport, issued_on – the date when a passport was issued.

Methods of the model:

Model includes some methods for working with database:

- getVisitors() – get the full list of visitors;
- getVisitorById(id) – get visitor by his unique id;
- getVisitorsByName(name) – get all visitors with particular name;
- getVisitorsBySurname(surname) - get all visitors with particular surname;
- getVisitorsByCity(city) - get all visitors from particular city;
- addVisitor(name, surname, patronym, city, addr, passport) – add visitor (here you should transfer data about name, surname, Patronym, city address and passport. Passport is an object that has to include next fields: number, issued_by, issued_on).

- deleteVisitorById(id) – delete visitor by his unique id in the database.

Visitor interface:

Visitor model is inherited from Visitor interface – it is the class that describes all basic features that should have a model.

Adding your own model into the project:

Interface was added for the reason that you could add your own database and own model into the project: to do this follow the next instruction:

- Create the database.
- Fill database with some info (optionally).
- Add the link to a database to the settings.js (change field dbLink).
- Create your own Model that would be inherited from VisitorInterface class.
- Add the link to your model to the settings.js (change the field VisitorModel).

Exceptions:

It should be mentioned that model throws an Exception when method could not be accomplished as needed, to simplify the work all methods in case of problems throw default Exception, message for the exception is taken from file “error_messages.js” (place in filesystem: ./db/error_messages.js).

Used patterns:

Visitor class is singleton – it means that it could be only 1 object of this class in program. It is needed to avoid problems in process of adding and deleting of visitors.

3.2. Services:

Services component gives the user more abstract methods to work with model. All classes from this component are placed in “services” folder (./services). This component includes simple validation layer, but it is described in separate document (see References).

API:

Services component includes the api that is described in class VisitorApi (./services/VisitorApi.js). To purpose of this class is to connect all logic from other classes into 1 class for convenience.

API contains next methods:

- getVisitors() – get the full list of visitors;
- getVisitorById(id) – get visitor by his unique id;
- getVisitorsByName(name) – get all visitors with particular name;
- getVisitorsBySurname(surname) - get all visitors with particular surname;
- getVisitorsByCity(city) - get all visitors from particular city;
- addVisitor(name, surname, patronym, city, addr, passport) – add visitor (here you should transfer data about name, surname, Patronym, city address and passport. Passport is an object that has to include next fields: number, issued_by, issued_on).
- deleteVisitorById(id) – delete visitor by his unique id in the database.

Description of classes:

There are several other classes in this component: VisitorAdder and VisitorDeleter. The logic of these classes is used in api.

VisitorAdder is class that contains the logic for adding new Visitors. The logic of adding is pretty simple: before adding we should validate fields and then use the method from model to add new Visitor.

VisitorDeleter is class that contains method for deleting users. These classes are inherited from class ValidAPIClass. If some class is inherited from

ValidAPIClass, then before creation of an object of this class would be checked whether the used model of Visitor is inherited from VisitorInterface. This check is needed to prevent errors in case you have created your custom model.

In this model also described a class of PassportData – if you create an object of this class and then call the method getAsDict(), you will get passport data in correct format.

Creation of your own API:

Of course, you can create your own API – to do this accomplish next steps:

- Create API class;
- Add the link to an api to settings.js (change the field ServiceAPI);

3.3. REST API:

REST API is used to work with services layer using HTTP protocol. REST API was created using Express framework and is represented by class controllers.js.

List of URLs:

- /get – used to get the data about visitors. If this url is not accompanied by the parameter, then you`ll get full list of visitors. The link could be completed with attribute by which you want to get visitors, for example, /get?id=2 – get the visitor with id = 2, /get?name=Olivia – get data about all visitors with name Olivia etc.
- /post – add the data about new visitor.
- /delete – deletion of the visitor. To this URL the parameter id has to be added to define which visitor is to be added.