**Technical design for contact book app:**

**Frontend:**

**Tools:**

* Though it is a small application, we can use any one of the UX tools like **Sketch.** It helps us to design a proper UX sketch in an efficient way.
* We can use **postman** to check the endpoint values. For testing the UI functionality, debugging and making styling changes, we can make use of **Chrome dev tools.**

**Frameworks:**

* We can use **React** for frontend, as it is highly reusable and scalable.
* For Backend we can use **Express** one of the NodeJS framework. It is popularly used and it also reduces the time taken for generating endpoints with less code.
* For Database we can make use of **MongoDB.** It’s schema less way of storing gives us the ability to alter the details of every contact as per our choice. It also offers cloud storage.

**NPM Libraries:**

* We need the default node modules that comes along with the installation of express and react. In addition to that we need some of the following modules.
* We can use **axios** to make api calls. It avoids the burden of intercepting the requests.
* We need **node-sass,** to convert the scss files back to css.
* Since it a small application we can use **React-bootstrap.** It gives us some of the basic components readily available.
* For routing between pages, we can use **React-router-dom.**
* We can use **mocha** and **Enzyme** to mock the components and unit test them.
* Since it is a small-scale application, we are in no need of global data store like redux.

**Directory Structure:**

Client

* build
* node\_modules
* src
  + Components
    - Profile
    - Loader
    - Alert
    - Header
    - Footer
  + Pages
    - ProfileList
      * ProfileList.js
      * ProfileList.spec.js
      * ProfiList.scss
    - Profile
      * Profile.js
      * Profile.spec.js
      * Profile.scss
    - main.scss
* route.js
* package. json

**Benefits of Single page application:**

* Single page applications are generally faster while comparing with multiple page application.
* In a multi-page application, every time the page gets reload when the user making an interaction with the website or navigating to other pages. Whereas in single page application this loading time gets drastically reduced. Add on to that it won’t hit the server for every action. It loads the entire page and only changes the component.
* In short it makes things simpler, improve the performance, reduce the loading time. It also makes the debugging easier.
* Though get messier when our application size grows. It is always difficult to manage multiple components with router. For small to medium applications is the best choice.

**Comments**:

* One of the main advantages of React is, it’s reusability. So, we need to make the components as much as Reusable.
* For e.g. In our application we can create a component called Profile this will contain a rounded image and Customer details. We can call this component multiple times inside our profile list screen.
* We can create common components like header and footer this will be common across the application.
* We can also use some utility components like Alert and Loader. When the page gets loading, we can display the loader. Suppose if we want to display any error or warning message, we can make use of Alert.
* We can also make use of container and handler for pages. Handler for getting the input initialising it and container for rendering the components and performing the functionality.

**Backend:**

**API Description:**

For this application we only need one or two end points to get the details. But we can create collection of requests to read, write, update and delete operation. This helps us to enhance the application on a later point of time without any hurdle.

Having decided to use CRUD end points, we can create the endpoints in following manner.

**Endpoints**:

Request: GET <https://localhost:5000/api/profiles>

Description: This request helps us to get all the profiles.

Sample response:

[{

Id: 1

Name: ‘John doe’,

Contact: 444-444-5555

Address: Ireland

},

{

Id: 2

Name: ‘Chris doe’,

Contact: 444-444-5555

Address: GBR

}]

Request: GET <http://localhost:5000/api/profiles/1>

Description: This request helps us to get a single profile

Sample response:

[{

Name: ‘John doe’,

Contact: 444-444-5555

Address: Ireland

},

{

Name: ‘Chris doe’,

Contact: 444-444-5555

Address: GBR

}]

Similarly, we can also create request for Add, Update and Delete operations

Request: POST <http://localhost:5000/api/profiles>

We also need to pass the request data since it is a post request.

Request: PUT <http://localhost:5000/api/profiles/1>

This request helps us to update the details.

Request: DELETE <http://localhost:5000/profiles/1>

This will delete the profile with an ID of one.

**Framework**:

We can use **Express** as a framework. It is a one built on top of NodeJS. It makes most of the NodeJS functionality simpler. It also helps us to save time and lines of code.

We can use Mongoose along with express in order to connect with the database. It helps us to execute the mongodb queries, creates the modals and so on.

**Comments**

* Everyone creates backend API in their own way. There is no hard and fast rule. But still, we can follow a structure.
* We can have a file called app.js which is the central server. From there we can call routes. From routes we could call our controller.
* Controller in turn will connect with the database using mongoose. We can store the config details inside a .env file.
* In order to establish a database connection, we can create a file called db.js.
* For handling errors, we can create a error handling middleware. If there is any utility method required, we can add them inside utils.
* In addition to this for loading and deleting automatically we can create a seeder.js file.