NodeJS and VueJs(v2) Assessment

Note: Please read the whole assessment carefully until you start coding. There are several notes in the end of the file I would suggest reading.

NodeJS:

1. Create a public endpoint **POST: /api/auth/sign-up** for registering users.
   1.1. The request must have the following fields in the body payload: **firstName, lastName, birthday, email** and **password**.
      1.1.1. Minimum value for the birthday must be 1920-01-01
      1.1.2. Maximum value for the birthday must be the **current date** - 18 years.
      1.1.3. Email value must be finished with **@newage.io**
      1.1.4. Password field must be encrypted with **Bcrypt.** Plain passwords must not be stored.
      1.1.5. Firstname and Lastname must only include Latin characters [a-zA-Z]
      1.1.6. Will be appreciated if you return status code **400** with proper error message in case of inappropriate request.

   1.2. The user can be stored in a **JSON** file. Storing the user(s) inside **MongoDB** will be appreciated
   1.3. Response must be the user object fields including the unique **uuid** for the record. E.g you can use **uuidv4.**
   1.4. Will be **appreciated** if in case of success the response status code is **201**

2. Create a public endpoint **POST**: **/api/auth/sign-in** for Logging in the user
   2.1. The request body must have **email** and **password** fields.
   2.2. If the email exists and the password comparison is successful you should generate a **JWT** token for this user.
   2.3. Token must have the user data except the password field.
   2.4. Token must expire after 60 minutes.
   2.5. **Note:** you can use NodeJs Passport or for authentication.

3. Create a private endpoint **GET: /api/auth/me** for getting the user data using **jwt token** from headers
   3.1. The response of the endpoint must be the decoded data stored inside the JWT token.

4. Create a private endpoint **PUT: /api/users/:userId** to update the user data by id.
   4.1. The request must have only **firstName** and **lastName** in the payload.
   4.2. You must check if the user which you are updating is your user and not somebody else's user.
   4.3. In case of success the response must return updated data for the particular user.
   4.4. Customized error messages will be appreciated

5. Create a private endpoint **DELETE: /api/users/:userId** to delete the user data by id.
    5.1. You must check if the user which you are deleting is your user and not somebody else's user.
    5.2. In case of successful delete, the user token must become useless and the user must be logged out of the system.

6. **Note:** All private endpoints must have **Authorization: Bearer JWT_TOKEN** in http requests, so please use *middleware* for private endpoints and if the *jwt token* is **expired** return an Unauthenticated error with status code: **401**

VueJs:

7. Create a user sign-up page with **firstName, lastName, birthday, email** and **password** fields.
    7.1. After clicking **Sign Up** button the request must go to **/api/auth/sign-up** endpoint in your NodeJs application
    7.2. After successful sign up you should be redirected to the **Sign In** page.

8. On the **Sign In** page there should be **email** and **password** fields.
    8.1. Validations will be appreciated.
    8.2. After filling the fields and clicking **Sign In** button
    8.3. You should send the request to **/api/auth/sign-in** endpoint. When you receive a response you should store the **Token** and **User Data** in **Vuex Store** and **LocalStorage** in case you refresh the page you should take **Token** from **LocalStorage** and set it back to **Vuex Store**
9. After successful login you should be redirected to the **/profile** page.
    9.1. On **/profile** page you should request **/api/auth/me** endpoint to fetch the current user data and display the fields from the data.
10. Create **/profile/:userId/edit** page.
    10.1. Other users must not be able to visit the page under your **userId.**
    10.2. The page must have input fields from the user profile and only **firstName** and **lastName** must be enabled.
    10.3. After clicking on **Update Profile** the request must go to **PUT: /api/users/:userId** with the appropriate payload.
    10.4. If the update is successful user data inside the Vuex must be updated as well.
11. Create **/profile/:userId/delete** page.
    11.1. Other users must not be able to visit the page under your **userId.**
    11.2. The page must have a text asking if the user really wants to delete the account.
    11.3. The page must have a red button with following text: **DELETE**
    11.4. After clicking the delete button the request must go to the **DELETE: /api/users/:userId** endpoint.
    11.5. After successful deletion the token and user data must be deleted from the local storage and the user must be logged out.

Note:

- *As you see the assessment consists of two parts, FrontEnd and BackEnd. It is great if you can handle the whole assessment but if not you can submit at least what you've done.*

- *Please have both tasks in one **git** repository.*

- *If you plan to use **bcrypt** and **passport** packages please read them carefully*
- *Please separate the code changes into small **commits** and do not push the whole code into one commit.*
- *The code must be written on Vue 2.*
- *You can use NuxtJs framework instead of VueJs.*
- *Please create separate module files with namespaces for Vuex Store*
- *If possible please move the appropriate logic from component to Store's mutations, actions and getters to use the whole potential of Vuex Store.*
- *To work with data from Vuex Store please use Vuex Helper functions.*
- *The design of the website is not important so you can craft it by yourself. SCSS styling will be appreciated.*