Assignment

Rätt Spår

Expected completion time: two workdays

November 2019

1 Introduction

As a step in the hiring process for Rätt Spår, you will be tasked with creating a small application using the framework created by Rätt Spår. The assignment includes both a back-end and a front-end part, with the back-end using Node.js and Typescript and the front-end using Vue.js and typescript, as well as Vuex and Vue-Router. To complete this task you will need to be familiar with all these technologies, as well as Domain Driven Design and the basics of the Rätt Spår framework, as described in the Rätt Spår development wiki[1], as well as understanding how to read user stories and break down larger tasks when necessary. You will also need to know how to do unit testing in Mocha and Chai, as well as in Jest.

2 Simple message board

The assignment is to create a simple message board, where users can read and post messages. There will be need of both a back-end and a front end for this application, and they both need to be unit tested. To accomplish this, you will need to implement the domain model provided in this assignment, a repository to manage this domain object and an API controller to implement the interface between back-end and front-end. You will also need to create a simple Vue-Router based front-end to communicate with the server.

3 User Stories

Following are a series of user stories, written in the form used at Rätt Spår får issue and task tracking. Some of them are going to be vague, or be large enough to contain more than one task. You are expected to find the task breakdown in these cases for this example.

[1] https://manager.rattspar.se/wiki/doku.php?id=en:architecture:start

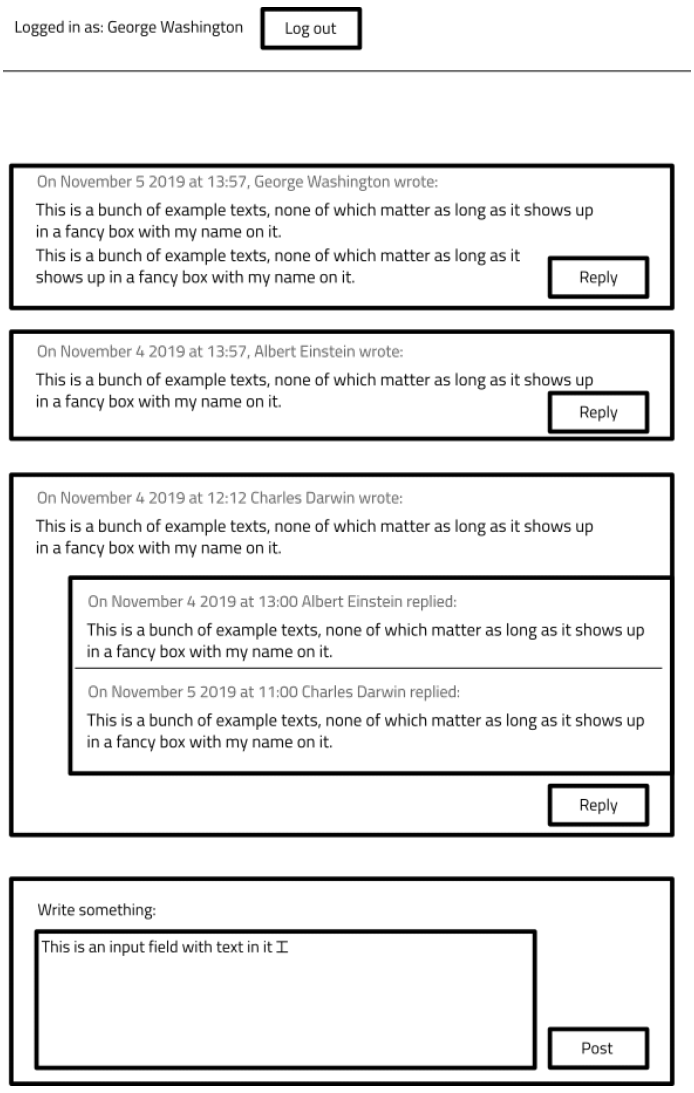


Figure 1: Example mockup

As a user *I want* to be able to register an account *so that* I can participate in the discussion

As a user *I want* to be able to login with an email and password *so that* I can access the message board securely

As a user *I want* to enter my name when I register *so that* it can be displayed together with my messages.

As a message controller *I want* to limit access to my routes to logged in users only *so that* the security of my messages are maintained.

As a user *I want* to write messages and reply to other peoples' messages *so that* I can have a discussion with other users.

As a user *I want* to be able to see who wrote a message and when *so that* I can keep track of the flow of messages.

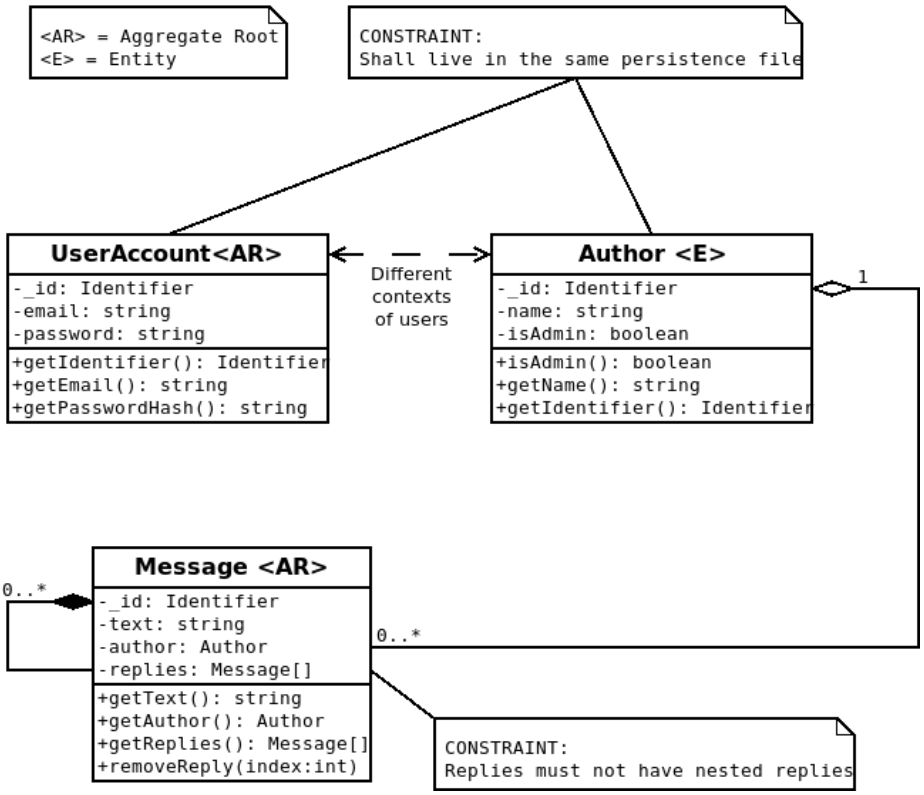
As a user *I want* to be able to delete my own messages *so that* I can fix mistakes or remove messages that I don't like.

As a user *I want* to show messages ordered from newest to oldest, and replies from oldest to newest *so that* I can easily see new messages as well as follow threads more easily.

As a user *I want* to be able to logout *so that* no one else can post as me when I'm not using the message board.

As a administrator *I want* to be able to remove all messages and replies *so that* I can moderate my message board.

4 Domain Model



5 Additional information

Provided items

To give you a starting-point, the assignment project contains:

Back-end:

- One fully implemented authentication API controller
- Domain models for User and Author objects.
- Repository interface for Users and Authors
- Emulated database adapter (described below).

Front-end

- Axios interceptors to handle the content type header and authorization header.

- Axios interceptor for automatic renegotiation of tokens. An Axios call will return ‘401 unauthorized‘ if a token could not be refreshed.
- Vuex store with state and mutations to automatically maintain tokens and token payload. `$store.state.tokenPayload` will contain the decoded contents of the JWT if available.

On the back-end, you will need to implement the repositories whose interface you've been given, as well as create the domain model for messages and the interface and implementation for the message repository. You will also need to create an API controller to handle message related requests.

On the front-end, you will need to use Vue, Vuex and Vue-Router to create enough views to handle registering and logging in users, showing messages and replies as well as writing new messages and replies. The number of views and the look of the views is not as important as your use of Vue. Typescript's class syntax for Vue components is to be used.

Database adapter

When implementing your repository, you will need to persist data somewhere. Rätt Spår provides a simple to-disk JSON collection database that you can use. The API to program against the db adapter is as follows:

Db object

`open(collection: string): DbContext`
: This opens a specific collection by name `collection`. Using this method, you will get a `DbContext` object that you can use to perform database operations.

DbContext

`find(criteria: {[key: string]: any}): {[key: string]: any}[`
: The `criteria` parameter is a union set of `key:value` pairs, each of which must be true for an object to be considered matched and be returned. If the empty object, `{}`, is passed to the function, all objects in the collection are returned.

`save(data: {[key: string]: any}): any`
: The `data` is the object to be saved to the collection. If the object does not contain an `_id` field, one is added to it. `_id` is of type `number`.

`delete(criteria: {[key: string]: any}): void`
: Using `criteria` the same way as in `find`, all matching entries in the database are deleted.

`commit(): void`
: This saves the database back to disk. This is *not* done automatically, and should always be called after a save or delete command.

All methods are asynchronous, returning `Promise` versions of the return types above. Use `Async/Await` to get the results.

Tips

- Use the DDD primitives from the framework when implementing the domain model
- Use the repository pattern for access to persistence.
- The persistence adapter provided reads from JSON file and exposes a simple `open`, `read`, `write` interface. You should mock this adapter for your tests.
- Use the dependency injection decorators to provide dependencies in controllers or services.
- The server *only* allows requests made with header `Content-Type: application/json`
  - Axios interceptor will set the `Content-Type: application/json` automatically.
- Authentication for routes that need it *must* use JWT with header `Authorization: Bearer <token>`.
  - Authorization is already handled for you, as long as you use Axios.
- You may use any feature of the framework, including domain or application services, as long as you can explain why you chose to do so.