Assignment

Rättt Spår

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 following documentation. <link to v4 documentation, architecture page>, 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.

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

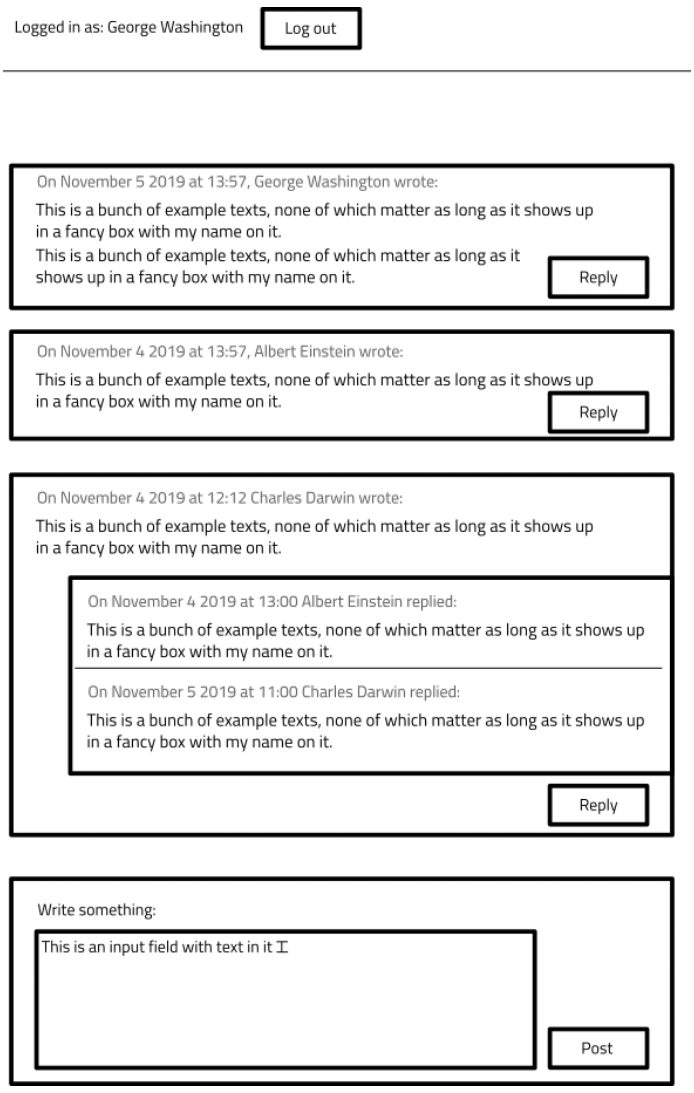


Figure 1: Example mockup

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 posts.

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.

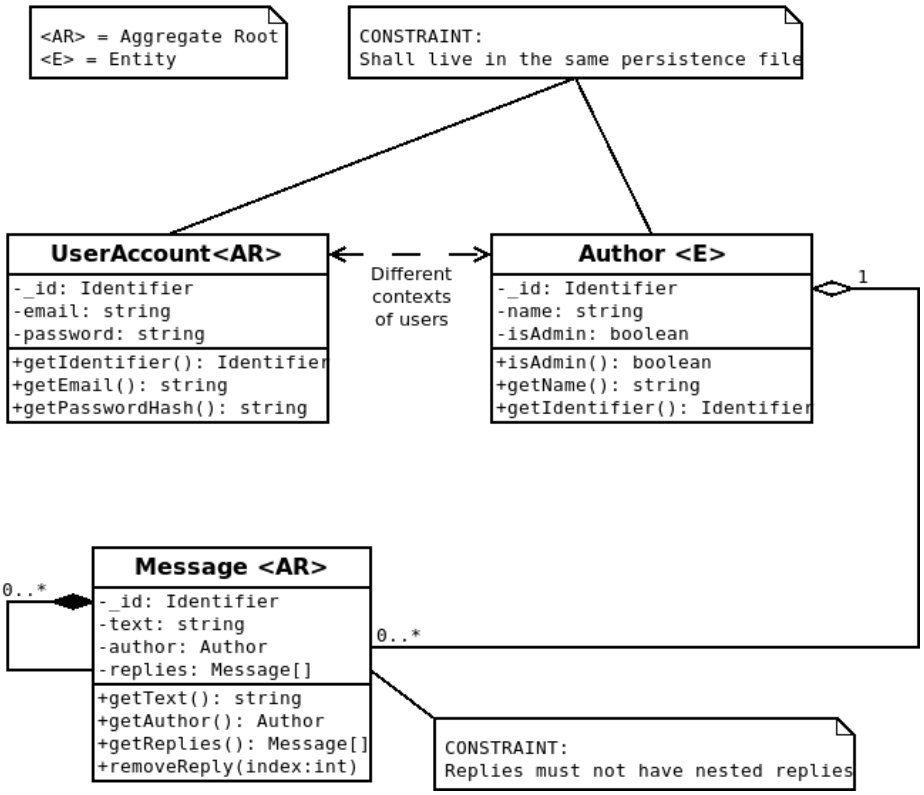
As a user *I want* to be able to delete my own messages *so that* I can fix mistakes or remove posts that I don't like.

As a user *I want* to read posts ordered from newest to oldest, and replies from oldest to newest *so that* I can easily see new posts 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

- 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`
- Authentication for routes that need it *must* use JWT with header `Authorization: Bearer <token>`.

  - Authorization middleware is already provided for you, but you must generate your own tokens on login.

- 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.