**My Blog Site**

A web app of social networks – by Thong Le, 2019

**Reader target:** Developer

**Introduction**

1. **Problem**

Create a twitter kind of blogging website

1. **Requirements**

* Using Node.js as back-end and ReactJS as front-end
* User can sign up for an account
* User can login with a registered account
* User can add new tweet
* User can view other users’ tweets

1. **Considerations**

* Handling scalability and security in my own way
* Make appropriate and careful assumptions and state those

1. **Solution**

I created a web application that meets requirements with considerations

Assumptions:

* + Users access to website using only Google Chrome and Firefox
  + Users can’t delete their account or change their name, or either their account is removed, so foreign key to restrict ‘tweets’ to ‘users’ is not necessary
  + There is no critical real-time interaction between users and server, so prevention on multiple access from the same user’s device is not necessary
    - Example of a critical real-time interaction: Without user’s HTTP request, server sends a message that needs response from user.

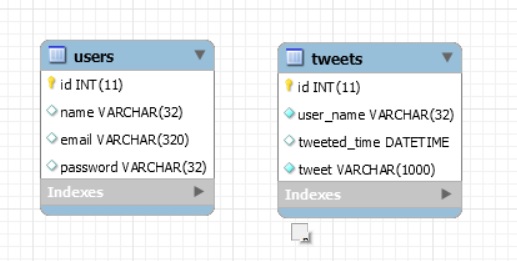
Extra feature:

* + Email verification: to prevent spams
  + Multiple access prevention: to prevent users from logging in from multiple devices, or to notify user a potential threat that account has been accessed by unauthorized person
  + Forgot password: a temporary feature to reset and change password if user considers his/her account has been accessed by unauthorized person

Future release:

* + Forgot password: after reset password, will kick session that another user established
  + User-user interactions: follow, private message
  + Tweet reactions: like, retweet
  + Improve performance of login persistence

**Database Design**

The requirements don’t ask to have any strict relationship. Therefore, I implemented two simple and independent tables in order to meet the basis of requirements.

**Architecture Design**

1. **Overview**

Client

Server

Database

1

0..\*

1

1

Fig. 1: Overview of web application communication and connection

* One or many clients communicate with a server
* A server connects to a database
* Arrow head: pointer to a communication end
* Double arrow-headed line: mutual communication

1. **Client**

App

Home

User

SignUp

Login

not authenticated

authenticated

Fig. 2: Overview of components in client side

* Single arrow-headed line: a component conditionally mounts another component
* Double arrow-headed line: mutual redirection
  + Redirection: a component has a link to redirect to another
  + App Component will mount any component if condition is met

SignUp

SignUpFirst

SignUpFinal

code verified

SignUpCode

inputs validated

Fig. 3: Detail of SignUp Component

* Dash line: pops up a modal component
* SignUp Component will mount SignUpFirst Component first. If passed verification in SignUpCode Component, SignUp Component will unmount SignUpFirst Component and mount SignUpFinal Component

clicked

Login

ChangePassword

ForgotPassword

Password authenticated and has length < 6

Fig. 4: Detail of Login Component

User

NewTweet

clicked

Fig. 5: Detail of User Component

1. **Server**

server\_session

connector

mailer

user

tweet

fs

(built-in module)

Fig. 6: Detail of server modules

* Line: points to a module that another module requires/uses
* Built-in module: provided by Node.js

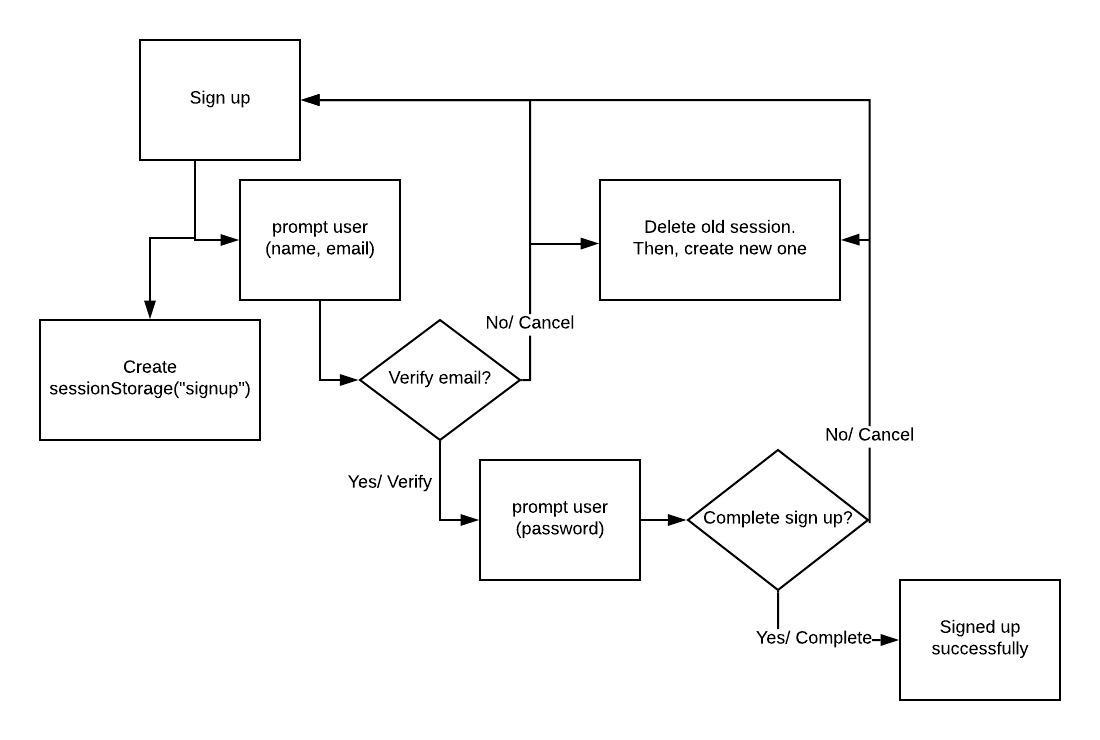
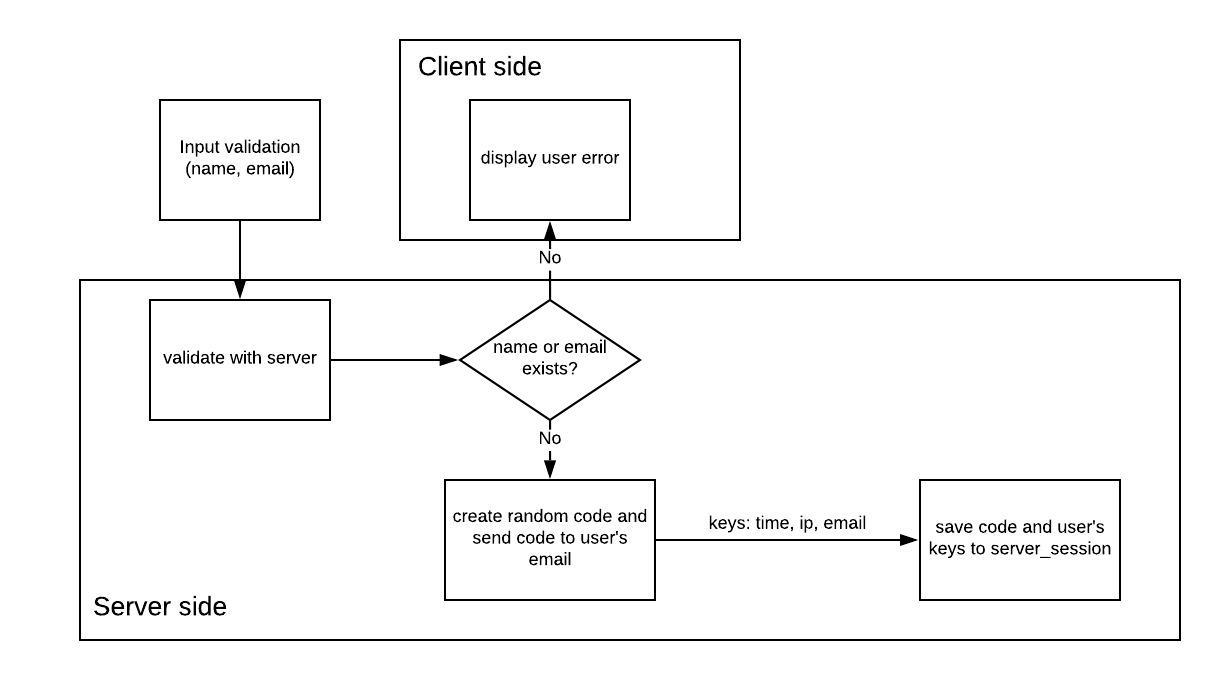
**Flow Diagram**

Fig. 7: Flow diagram of sign up process

* Assume user inputs are validated and all valid
* To sign up, user clicks Sign Up link
* Verify email: a modal will pop up when user’s name and email are valid
* sessionStorage(“signup”): {time, ip, name, email, openCodeDialog, codeEntered}
  + time: time to create session, in millisecond
  + ip: user’s IP
  + name: user’s name input
  + email: user’s email input
  + openCodeDialog: flag to toggle visibility of verification modal (SignUpCode)
  + codeEntered: flag to determine whether user verifies code/email or not

****Fig. 8: Flow diagram of input validation (name, email)

* User can’t have either name or email duplicate with another
* Code is sent to user’s email using mailer (module, fig.5)
* Random code: contains 6 random numbers, not starting at 0
* User’s keys are sent with a fetch from client side

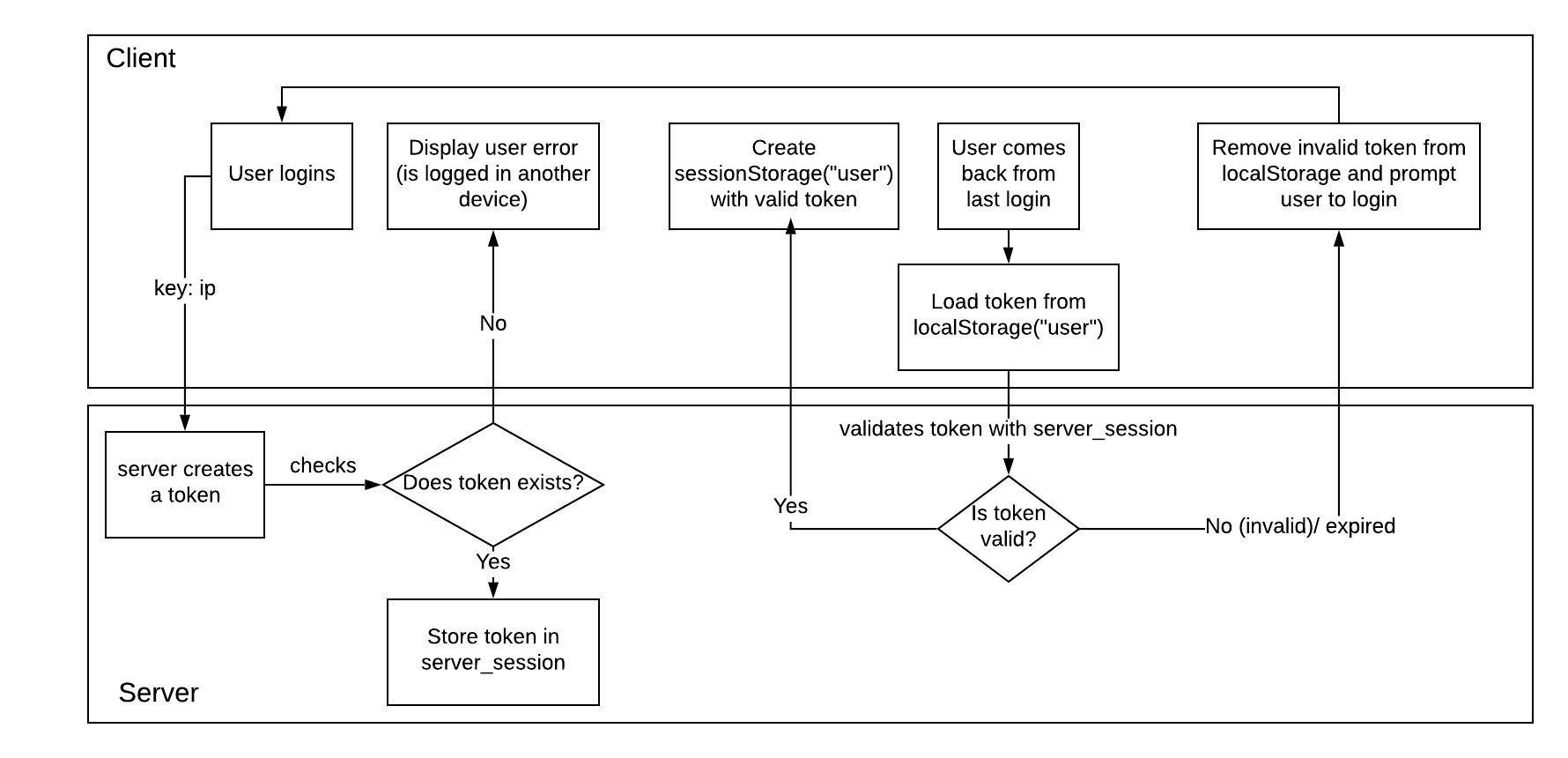
****

Fig. 9: Flow diagram of login persistence

* localStorage(“user”) and sessionStorage(“user”):
  + id: generated by server using ‘uuid5’
  + name: user’s name
* token in server:
  + id
  + uid: generated by server using ‘uuid5’
  + expiration: a time to validate if token is expired, in millisecond

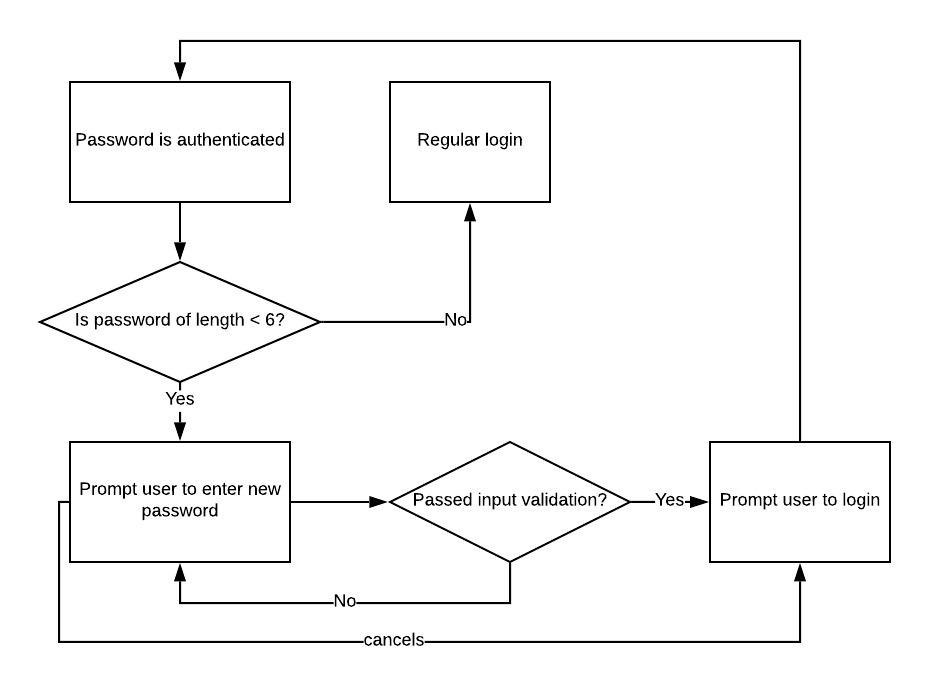


Fig. 10: Flow diagram of update password process

* Input validation: see Miscellaneous

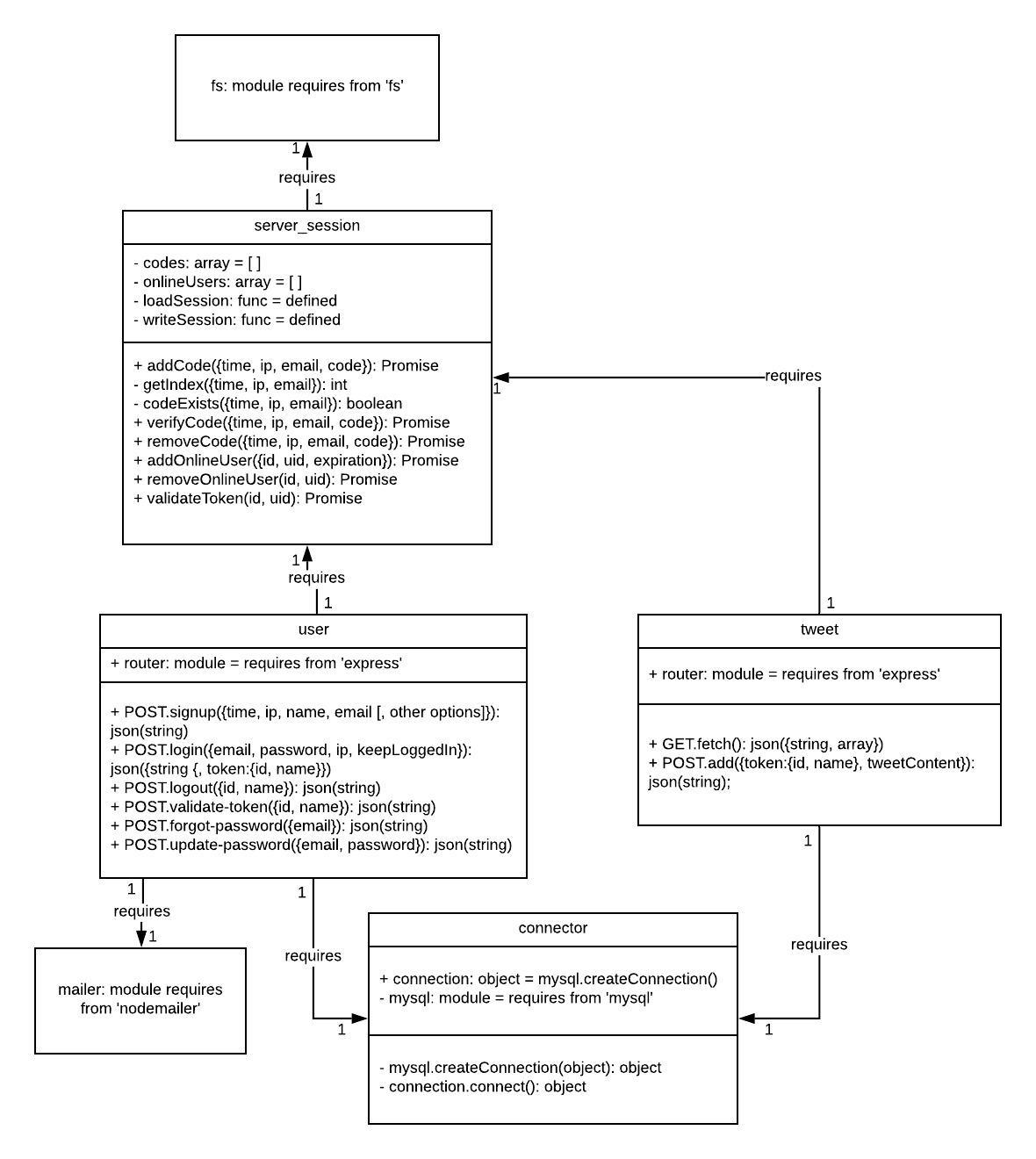
**Class Diagram**

Fig. 8: Class diagram of server modules

* ‘fs’, ‘express’,…: package name
* POST/GET.url: HTTP request method
* {property1: any, property2: any,…}: JSON

**Miscellaneous**

1. **Input validation**

Inputs are valid iff:

* Name: is not either empty or taken (case-insensitive)
* Email: is not either empty or used (case-insensitive), or follows correct format (Regex)

Regex: /^(([^<>()[\][\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/](file:///\\.,;:\s@%22%5d+(\.%5b%5e%3c%3e()%5b\%5d\\.,;:\s@%22%5d+)*)|(%22.+%22))@((\%5b%5b0-9%5d%7b1,3%7d\.%5b0-9%5d%7b1,3%7d\.%5b0-9%5d%7b1,3%7d\.%5b0-9%5d%7b1,3%7d\%5d)|((%5ba-zA-Z\-0-9%5d+\.)+%5ba-zA-Z%5d%7b2,%7d))$/)

* Password: is not either less than 6 characters or too simple (if score is 0)

Password strength is scored with ‘zxcvbn’ package. zxcvbn.score() returns:

# Integer from 0-4 (useful for implementing a strength bar)

0 # too guessable: risky password. (guesses < 10^3)

1 # very guessable: protection from throttled online attacks. (guesses < 10^6)

2 # somewhat guessable: protection from unthrottled online attacks. (guesses < 10^8)

3 # safely unguessable: moderate protection from offline slow-hash scenario. (guesses < 10^10)

4 # very unguessable: strong protection from offline slow-hash scenario. (guesses >= 10^10)

1. **Input max-length**

* Name: 32 characters
* Email: 320 characters
* Password: 32 characters
* TweetContent: 1000 characters

1. **User’s token lifetime**

Token is stored in server and it has expiration field

* If user chooses keepLoggedIn, expiration is 30 minutes (modifiable)
* If not, expiration is 5 minutes (modifiable)