Chat Room
Javascript Full Stack

Software Engineering Crash Course

Deadline

- Deadline:
  - Deadline: 7/11 Tue.
- Penalty:
  - Penalty for hard deadline: 1 coffee + 2 week leetcode study plan.
  - For each late day, additional penalty +1 coffee.
- Started on 7/5 Wed.
- Break On 7/12 Wed. & 7/13 Thur.
- Finished FSE Requirements, Demo, Code Review on 7/16 Sun.
- Further Extension on 8/9 Wed.
- Finished Private Messaging with Summary on 8/28 Mon.

Daily Timeline & Goals 7/5~7/7 Wed.~Fri.

- 7/5 Wed.:
  - Learn some few Javascript.
  - Played Zelda on Nintendo Switch For 3 Hours.
- 7/6 Thur.:
  - Learn some few ExpressJS socket.io
  - Explored Some Examples.
  - No Bug-Free Code Until Now.
- 7/8 Fri.:
  - Mentor Pair Programming.
  - Bug-Free Code For The First Time.
  - Github Initial Commit.

Daily Timeline & Goals 7/8 Sat.

- Fully Understand The Codes So Far, and Add Comments.
- Change some namings to meaningful namings.
- Why express.static(“/”) doesn’t work.
- express.stack(path), path = “.” “/” “public”
- Server RESTful API HTTP GET: All clients get the new chat message when a new message is posted by 1 client.
- Server Socket.io Broadcasting: The new client get all chat messages when a new client enters the room.
- First Github Pull Request.   Learned Some Git.

Daily Timeline & Goals 7/9 Sun.

- Implement timestamp.
- Refactor: Server generates UTC/Unix timestamp, then client transforms its own Time Zone.
- Refactor: Remove Client to Server Socket.io Event Emitting. Moved The Broadcasting To "/messages" POST Router.
- Frontend: Show All Chat Logs when client enters the room
- Frontend: Show Timestamp
- Change DB namings & understand express request & response fields, .

Daily Timeline & Goals 7/10 Mon.

- Implement register: Server RESTful API.
- Implement login: Server RESTful API.
- Implement logout: Server RESTful API.

Daily Timeline & Goals 7/11 Tue.

- Implement register: Client HTTP Request Frontend & HTML.
- Implement login: Client HTTP Request Frontend & HTML.
- Implement logout: Client HTTP Request Frontend & HTML.

Daily Timeline & Goals 7/14 Fri.

- Authentication with express-session
  - https://github.com/expressjs/express/blob/master/examples/auth/index.js
- Fix: Bypass Async Wait For DB I/O & session.regenerate() For Session Login
- Implement sender info. (database, server & frontend)
- Login / Logout Redirection, Homepage / Chat Room Redirection
- Restrict Client Access to HTML Static Files
- Restrict non-logged-in Users To Access "/messages" Router
- Window Alert For Wrong Username / Password / Register Username Conflict By Status Code.

Daily Timeline & Goals 7/15 Sat.

- Introduction to CI
- feat: add css & html to chat room page for the chat box style looks.
- refactor: use html div to replace ul li list to control each element's css styles(username, timestamp, chat text).
- refactor: use Javascript DOM createElement() setAttribute() textContent appendChild() getElementById() to add a new chat post to html page.
- feat: chat room page: trim timestamp string to exclude other time information.
- feat: chat room page: change <input type="text"> to <textarea> for the chat text input box.

Daily Timeline & Goals 7/16 Sun.

- feat: css fixed navigation bar.
- feat: css: send button navigation bar.
- feat: css login page.
- jQuery To Replace onclick()
  - Quote, “The problem with the DOM element properties method is that only one event handler can be bound to an element per event.”, unquote.
  - Attach Multiple Events, Examples:
  - https://www.w3schools.com/jquery/event_on.asp
- Finished FSE Requirements.
- Demo.
- Code Review.

Daily Timeline & Goals 7/31 Mon.

- Deployed to render.com

Daily Timeline & Goals 8/9~8/11 Wed~Fri.

- change SQL with "select … from … where …"
- use try await
- MVC.
- Router-level middleware.

Daily Timeline & Goals 8/12 Sat.

- Discussion: To solve the “this” binding problem when router mounts middleware, discussed Singleton.
- Discussion: To solve the module export / import order problem, discuessed Singleton instance getter.

Daily Timeline & Goals 8/13 Sun.

- refactor: To solve the “this” binding problem when router mounts middleware, used Singleton.
- refactor: To solve the module export / import order problem, used Singleton instance getter.
- refactor: Network Servers (express, socket.io, http) to a Module.
    - Then export them with Singleton instance getters.
- refactor: Root-Level Middleware.
- Design Doc: Private Messaging Design Doc
- refactor: MVC message naming
- feat: implemented rooms MVC model & SQL.

Daily Timeline & Goals 8/14 Mon.

- feat: RESTful API GET POST /rooms
- fix: module dependency cycle
  - for user_model & room_model

Daily Timeline & Goals 8/15 Tue.

- feat: RESTful API GET POST /messages by socket.io

Daily Timeline & Goals 8/24 Thur.

- doc: write design doc for a new private messaging & its socket.io design & way of implementation.

Daily Timeline & Goals 8/25 Fri.

- fix: remove remove old private messaging design
- chore: html textarea to replace input
- feat: each chat room has its own URL
- feat: room selection html css

Daily Timeline & Goals 8/26 Sat.

- feat: room selection frontend
- feat: after login, redirect to room selection page.

Daily Timeline & Goals 8/27 Sun.

- feat: room creation html
- feat: room creation frontend
- feat: room selection page button in chat room
- fix: post rooms duplicate usernames
- feat: reload selection page when create rooms
- fix: check room_id input exist or not
- doc: understand session store
  - figure out the way for the server to get session data (except for req.session)
  - in branch doc/success-session-store-try

Daily Timeline & Goals 8/28 Mon.

- feat: mounting of express-session middleware moved to upper dependency server/network.js from lower dependency router/root_middleware.js
- feat: get session store data in socket.io connection event
- feat: socket.io join room_id from session store
- feat: socket.io emit to room_id
- feat: add room_id subtitle to chat room page.
- refactor: frontend util function
- refactor: execute some of the initialization functions inside the ajax script in html.
- fix: await checking room exist
- deploy: deployed to render.com

Daily Timeline & Goals Future

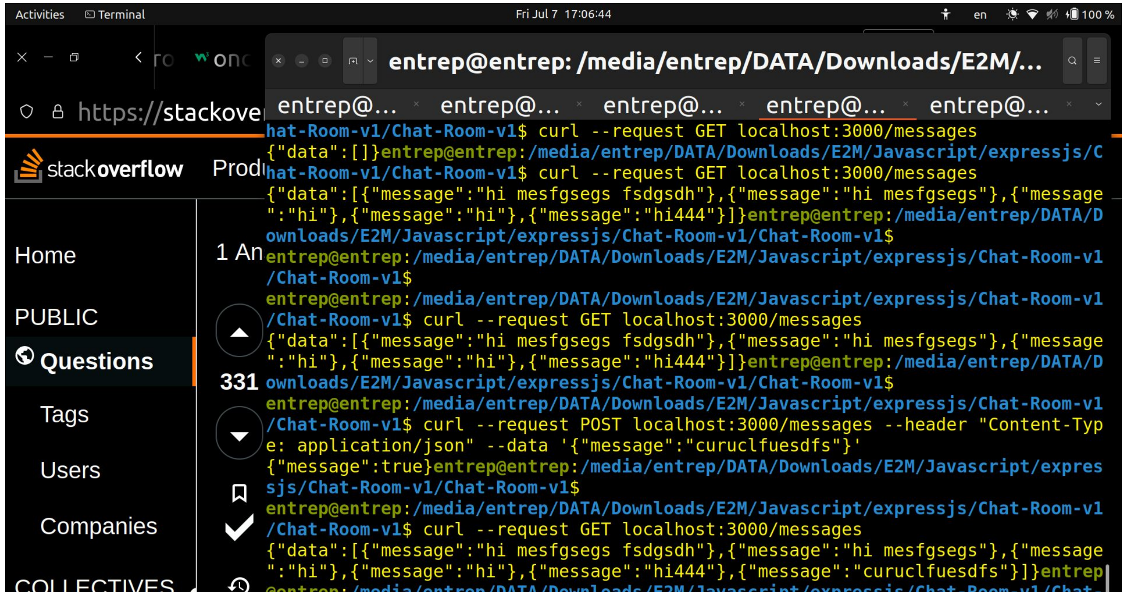
- Render Deployment: ok 7/31
- Read About RESTful API: ok 8/12
- Router-Level Middleware: ok 8/11
- Online Status. Notification. Show username & room id.
- NoSQL (detailed checkpoint TODO): mongoose boot camp ok.
- Data Access Object
    - MongoDB. PostgreSQL. Redis. Interface
- Testing
- JWT
- Online Status. Unread Message Status
- Database Rollback by socket.io Acknowledgement
- Scaling Up To Multiple Servers

Guide & Notes

- Node.js http Module Server class:
  - https://nodejs.org/api/http.html#class-httpserver
- Express Routing:
  - https://expressjs.com/en/guide/routing.html
  - https://expressjs.com/en/starter/basic-routing.html
- Express Middleware Intro:
  - https://expressjs.com/en/guide/writing-middleware.html
- Mounts middleware on Express router:
  - https://expressjs.com/en/api.html#app.use
  - https://stackoverflow.com/questions/10695629/what-is-the-parameter-next-used-for-in-express
- res req, http module:
  - https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction
- Express
  - https://reflectoring.io/express-middleware/

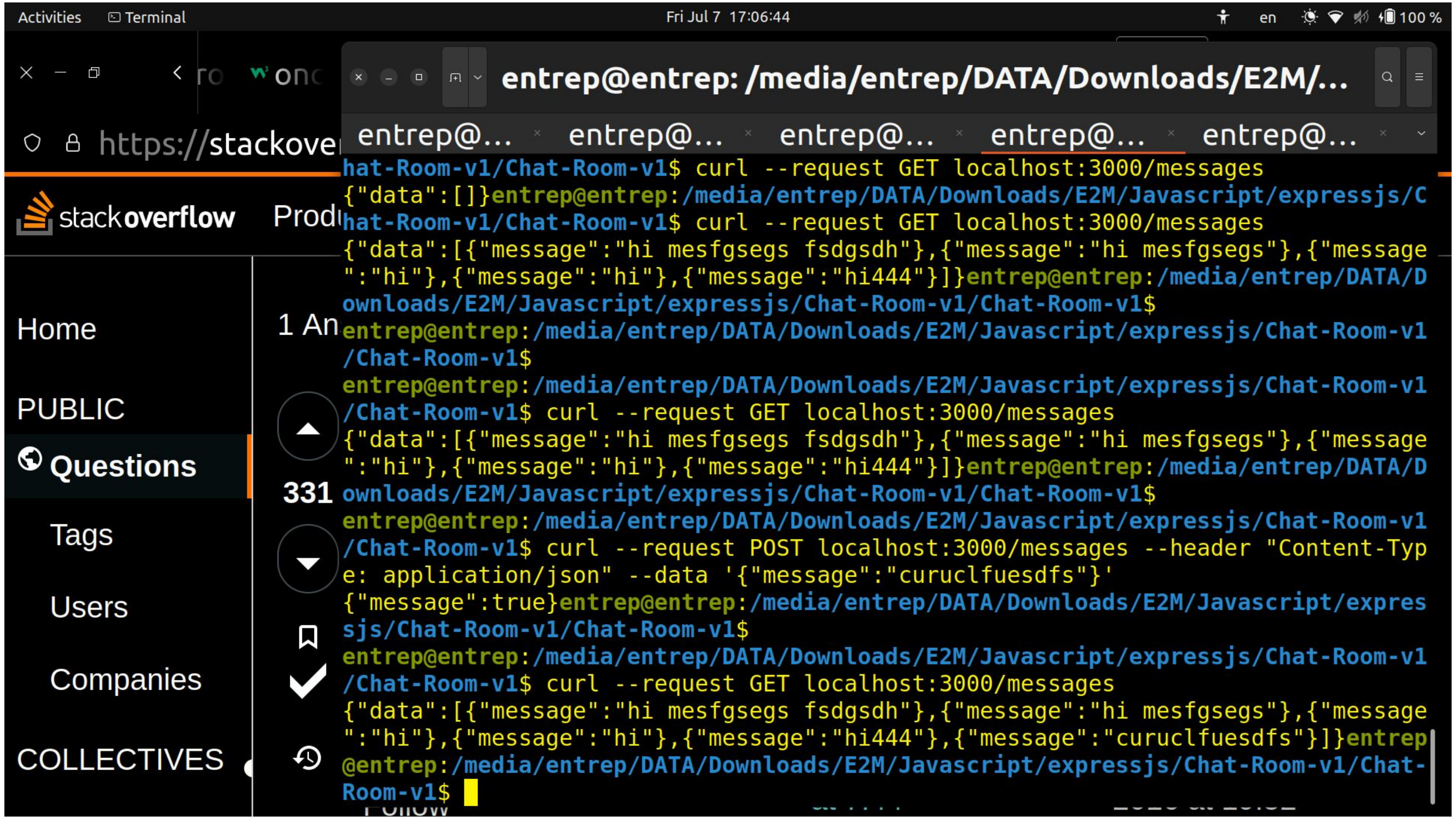
Guide & Notes

- Use curl to simulate HTTP requests to the server
  - Verify server API works well.
  - Need to specify Content-Type. Mixing of Single & Double Quotes.
- Use browser F12 to verify client side.



```
hat-Room-v1/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[]}entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/C
hat-Room-v1/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[{"message":"hi mesfgsegs fsdgsdh"},{"message":"hi mesfgsegs"},{"message
":"hi"},{"message":"hi"},{"message":"hi444"}]}entrep@entrep:/media/entrep/DATA/D
ownloads/E2M/Javascript/expressjs/Chat-Room-v1/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[{"message":"hi mesfgsegs fsdgsdh"},{"message":"hi mesfgsegs"},{"message
":"hi"},{"message":"hi"},{"message":"hi444"}]}entrep@entrep:/media/entrep/DATA/D
ownloads/E2M/Javascript/expressjs/Chat-Room-v1/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$ curl --request POST localhost:3000/messages --header "Content-Typ
e: application/json" --data '{"message":"curuclfuesdfs"}'
{"message":true}entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expres
sjs/Chat-Room-v1/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[{"message":"hi mesfgsegs fsdgsdh"},{"message":"hi mesfgsegs"},{"message
":"hi"},{"message":"hi"},{"message":"hi444"},{"message":"curuclfuesdfs"}]}entrep
@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1/Chat-
```

Guide & Notes

- 

```
hat-Room-v1/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[]}entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/C
hat-Room-v1/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[{"message":"hi mesfgsegs fsdgsdh"},{"message":"hi mesfgsegs"},{"message
":"hi"},{"message":"hi"},{"message":"hi444"}]}entrep@entrep:/media/entrep/DATA/D
ownloads/E2M/Javascript/expressjs/Chat-Room-v1/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[{"message":"hi mesfgsegs fsdgsdh"},{"message":"hi mesfgsegs"},{"message
":"hi"},{"message":"hi"},{"message":"hi444"}]}entrep@entrep:/media/entrep/DATA/D
ownloads/E2M/Javascript/expressjs/Chat-Room-v1/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$ curl --request POST localhost:3000/messages --header "Content-Typ
e: application/json" --data '{"message":"curuclfuesdfs"}'
{"message":true}entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expres
sjs/Chat-Room-v1/Chat-Room-v1$
entrep@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1
/Chat-Room-v1$ curl --request GET localhost:3000/messages
{"data":[{"message":"hi mesfgsegs fsdgsdh"},{"message":"hi mesfgsegs"},{"message
":"hi"},{"message":"hi"},{"message":"hi444"},{"message":"curuclfuesdfs"}]}entrep
@entrep:/media/entrep/DATA/Downloads/E2M/Javascript/expressjs/Chat-Room-v1/Chat-
Room-v1$
```

Guide & Notes

- Commit Message: https://www.conventionalcommits.org/en/v1.0.0/
- Pull request
  - git checkout -b feat/new-branch
  - New Feature Development on this branch, then git add . + git commit
  - git push origin feat/new-branch
  - Go to github page, change to the branch, and press the “compare and pull request” button
  - Delete the forked branch
  - Git pull
- Delete branch:
  - git checkout go-to-branch
  - git branch -D branch-to-delete
- Didn’t add / commit, but switch branch:
  - Work in progress
  - Git stash
  - Git stash list
  - Git stash pop
  - Or use git restore .
- Rename a local branch name: git branch -m old_name new_name
- git reset: undo commit (but not pushed)

Guide & Notes

- User login example with express-session
  - https://expressjs.com/en/resources/middleware/session.html
- Callback or I/O won’t wait
  - sqlite3 db.all() won’t wait.
  - express-session session.regenerate() won’t wait.
  - https://stackoverflow.com/questions/5010288/how-to-make-a-function-wait-until-a-callback-has-been-called-using-node-js
- CSS: For CSS, use a CSS editor website for styling instead of running the whole program.
  - https://jsfiddle.net/azetjL8g/
- …

Guide & Notes

- MVC:
  - https://progressivecoder.com/how-to-create-a-nodejs-express-mvc-application/
- JWT:
  - https://progressivecoder.com/nodejs-express-login-authentication-with-jwt-and-mysql/
  - (not checked link)
- Sqlite3 doesn't support Promise. Can't await db.all("select…")
  - https://stackoverflow.com/questions/62456867/cannot-await-for-sqlite3-database-get-function-completion-in-node-js
  - https://stackoverflow.com/questions/64372255/how-to-use-async-await-in-sqlite3-db-get-and-db-all
  - https://www.npmjs.com/package/sqlite#examples
  - Maybe use sqlite.
  - User wrap a Promise function, as in stackoverflow.

Guide & Notes

- express Request-Response Cycle != Return From Middleware Function
  - Use freecodecamp boilerplate-express for testing.
  - res.send() res.json() res.end(): don’t return from middleware function.
  - So we have to explicitly add “return;”
- router.get(“/users”, userController.func) → “this” keyword undefined
  - https://stackoverflow.com/questions/45643005/why-is-this-undefined-in-this-class-method
  - what gets passed to your router is just a reference to the .list method. The userController instance gets lost.
  - This is not unique to routers - this is a generic property of how things are passed in Javascript.
  - Use: router.get('/users', userController.func.bind(userController))
  - Also, it seems func in another file can use the “active_username_set” global variable in router’s file.

Guide & Notes

- MVC & express.js example
  - https://progressivecoder.com/how-to-create-a-nodejs-express-mvc-application/
  - https://github.com/dashsaurabh/node-express-mvc-demo/tree/master
- Session Cookie vs Token Authentication
  - express session vs json web token
  - stores in server vs client side
  - great picture: https://www.geeksforgeeks.org/session-vs-token-based-authentication/
  - great picture: https://hackernoon.com/using-session-cookies-vs-jwt-for-authentication-sd2v3vci

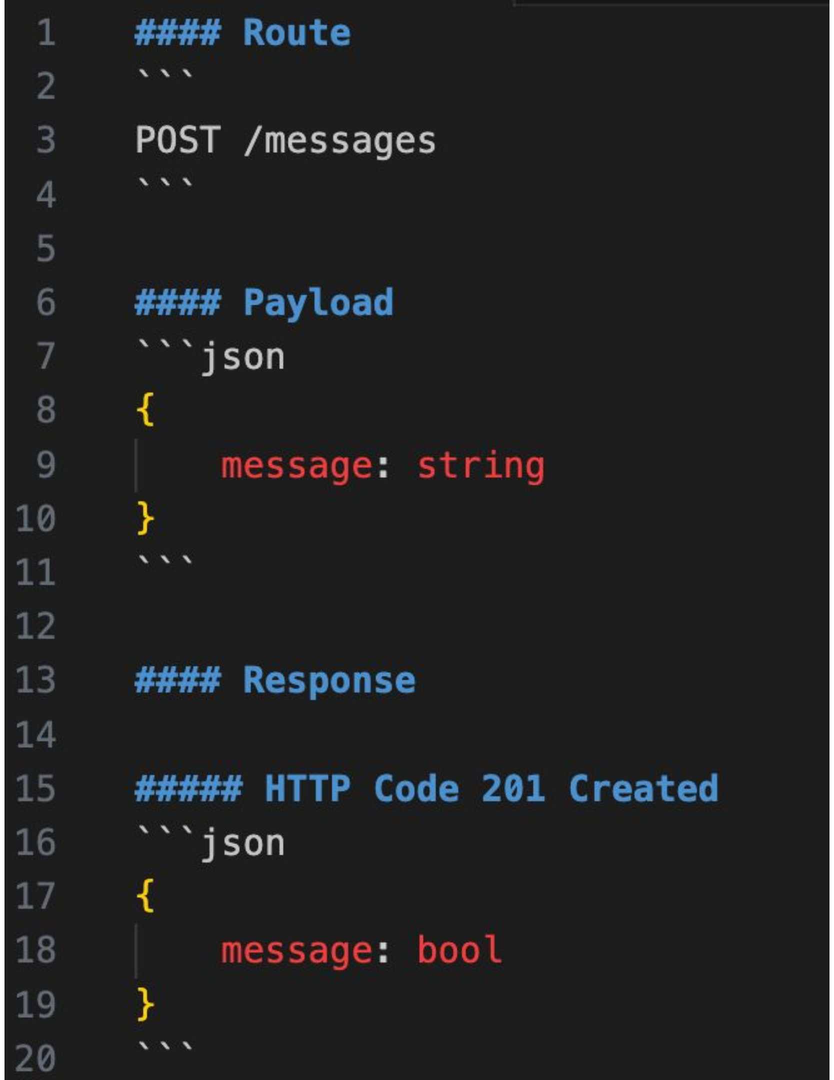
Guide & Notes

- server gets session data / express-session with session store
  - we can use req.session in router to get session data,
  - but what about other places like socket.io on event listener ?
  - https://stackoverflow.com/questions/19889552/how-to-access-express-session-memorystore-via-socket-io-objects
  - https://stackoverflow.com/questions/24887175/unable-to-get-session-from-session-store
  - store.all() store.get
  - server on event: socket.request.headers.cookie
  - https://www.section.io/engineering-education/session-management-in-nodejs-using-expressjs-and-express-session/
- express-session.MemoryStore
  - Warning The default server-side session storage, MemoryStore, is purposely not designed for a production environment. It will leak memory under most conditions, does not scale past a single process, and is meant for debugging and developing.
  - https://expressjs.com/en/resources/middleware/session.html
  - it seems we need to manually call store.destroy() for this MemoryStore
- socket.io with express.js: https://www.danielbaulig.de/socket-ioexpress/

Design Doc: API Send Messages

- Write one RESTful API /messages, take chat message and print it out at server log using console.log (this small step is to verify your code work)
    - https://stackoverflow.com/questions/7172784/how-do-i-post-json-data-with-curl
- Test that API using curl via command line
- Enhance this API by store the data into sqlite3 (READ THE TUTORIAL)
    - https://www.npmjs.com/package/sqlite3?activeTab=readme
- API Spec: next page
- Reference: Express Basic routing
- Reference: `cat backup.sql | sqlite3 hello.db` [Ref]
- Res req are what we learn http messages in computer network courses.
- Javascript has non-blocking I/O on async operations, including db.
    - Async await. promise.
- …

Spec

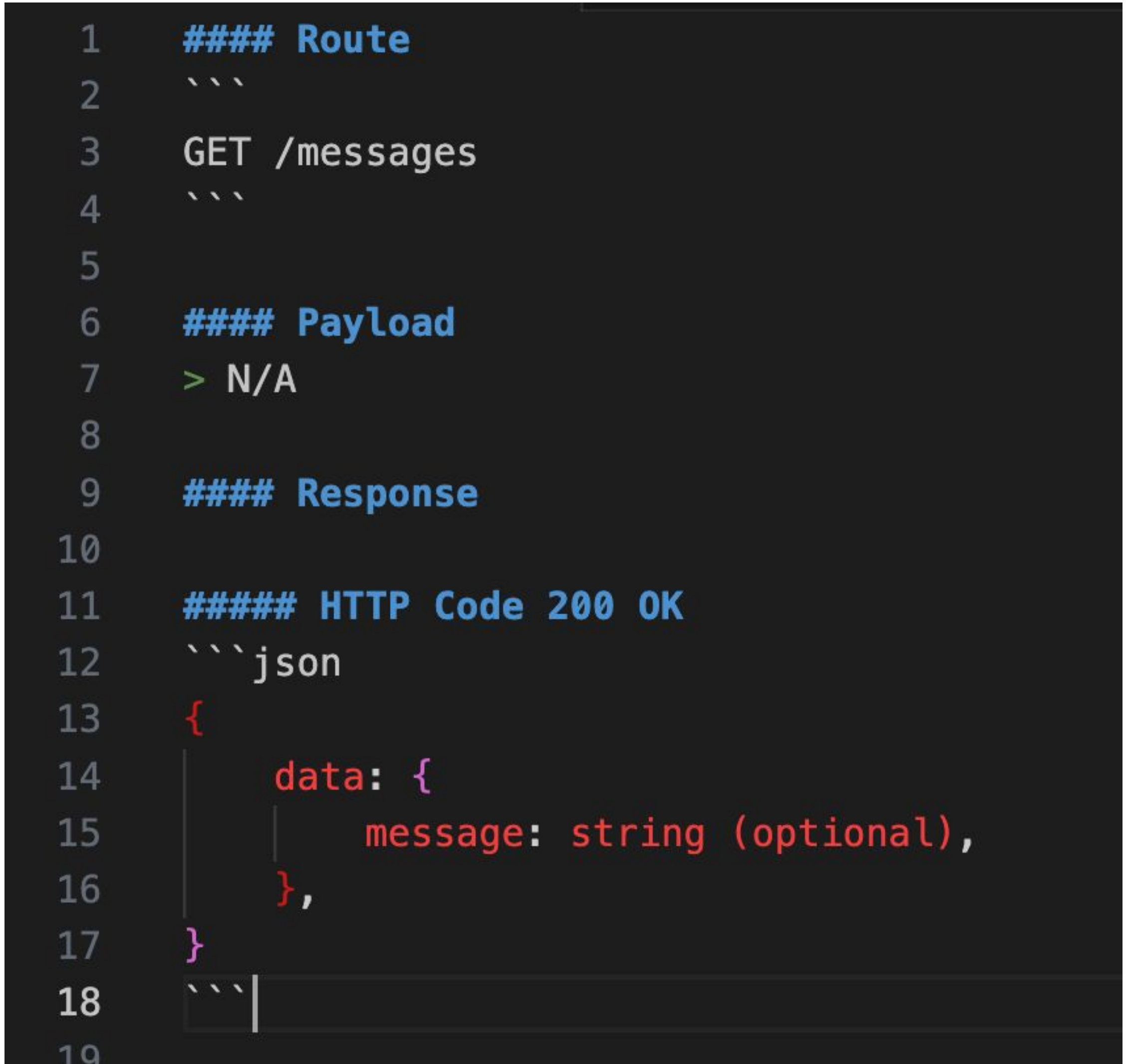


~~~markdown
#### Route
```
POST /messages
```


#### Payload
```json
{
    message: string
}
```


#### Response

##### HTTP Code 201 Created
```json
{
    message: bool
}
```
~~~

Design Doc: API Get Messages

- Write one RESTful API /messages, get all chat messages from DB
- Test that API using curl via command line
- API Spec: next page
- My NOTE: onclick vs form eventListener & enter keypress event.

Spec



~~~
#### Route
```
GET /messages
```


#### Payload
> N/A

#### Response

##### HTTP Code 200 OK
```json
{
    data: {
        message: string (optional),
    },
}
```
~~~

Design Doc: Front-end JS call RESTful API

- Use ajax to call RESTful API send messages when user click send button
- Use ajax to call RESTful API get messages when user enter chat room page
- …

Design Doc: Implement sender

- Enhance existing API by adding sender info to backend API & client-side call
- QQ:
  - Where should we get user info?
  - Where should we store sender info?

Design Doc: Implement Register API

- Do API Design
  - What’s the input data? -> Username & password
  - What should be the return value?
  - What should the route be? What is the HTTP method for creation? [Using HTTP Methods for RESTful Services]
  - What kind of error should we return if username already exists? I.e. username conflict [Client error responses]
  - Can we store plaintext password? -> we can skip this for now
- Implement it :”)
- POST for creation, status 201(Created) / 400(username conflict). Router path at “/register”

Design Doc: Implement Login API

- Do API Design
  - What’s the input data? -> Username & password
  - What should be the return value?
  - What should the route be? What is the HTTP method for login? [Using HTTP Methods for RESTful Services]
  - What kind of error should we return if username not exists? [Client error responses]
  - What kind of error should we return if password not match? [Client error responses]
  - How to verify password if we store encrypted password? -> we can skip this for now
- Implement it :”)
- POST method. Router path at “/login”. 200 ok for success. 401 Unauthorized for incorrect username. 403 Forbidden for incorrect password. 400 Bad Request.
- Case for being already logged in?

Design Doc: Implement Logout API

- Do API Design
    - What’s the input data?
    - What should be the return value?
    - What should the route be? What is the HTTP method for logout? [Using HTTP Methods for RESTful Services]
    - What’s the user behavior after he/ she logout?
- Implement it :”)
- POST method. Router path at “/logout”. 200 ok for success. 400 bad request
- Logout shall redirect the user to the home page, not the chat room. So response status code may be different.

Design Doc: Frontend for Register & Login

- It’s up to you to design Frontend HTML/ CSS/ JS
- Reference:
    - https://codesandbox.io/s/eqg36
- …

Design Doc: Private Messaging Architecture

- just wrap the codes with only public chat.
- Database Schema
- RESTful API
- Frontend
- Chat room html page
- Room selection frontend & html page

- homepage: (1) login page, if not logged in. (2) room selection page, otherwise
- room selection page:(1) select existing. (2) create a new room by specifying usernames
- the public room: all joins it automatically

Design Doc: Database Schema

- Table messages:
  - message : string
  - timestamp_utc : string
  - username : string
  - room_id : string
- Table rooms:
  - room_id : string
  - no array in sqlite3
- room_id: “public_room” “member0#member1#member2#room”
- Extensibility: not only 1-to-1 room, but rooms with members of any size.

Design Doc: Database Schema (cont.)

- Table users2rooms:
  - many-to-many relationships, to solve sqlite3’s no array problem.
  - username: string
  - room_id: string
  - select username … where room_id = … → not needed, socket.io join() does the job.
  - select room_id … where username = … → API: GET /rooms
- Process when a user logs in and gets all chat messages:
  - select room_id from users2rooms where username = …
  - for room_id in all selected chat rooms: select * from messages where room_id = room_id
  - socket.io join()

Design Doc: RESTful API

- URI: GET /rooms
  - a room_id list is sent from the server to the frontend every time frontend needs it
  - username specified in request body
- URI: POST /rooms
  - create a new chat room
  - member usernames specified in request body
- naming:
  - MVC: with model / controller suffix
  - message and room have their own /routes /controllers /models
  - routers, controllers, models, sql table have consistent names.
- URI: POST /users
  - when register, add the username to the public room in database (not socket.io)
- only 1 chat_room_controller instance & chat_room model instance for all rooms.
- UI Design: don’t need to authenticate for GET POST /rooms

Design Doc: RESTful API (cont.)

- URI: GET /messages/:room_id
  - get all messages
  - if room not exist in database, insert 1.
  - req.params.room_id
  - this is when the user selects a particular room on the room selection html page.
  - socket.io join()
- URI: POST /messages/:room_id
  - post new messages
  - if room not exist in database, insert 1.
  - call socket.io io.to(room_id).emit()
- Steps: MVC: models → controllers → routers
- Steps: tests with curl: create new rooms → logins → messages

Design Doc

- Server: after login, redirects to the room selection page room_selection.html, then:
    - (1) join the socket to the room_id
    - (2) redirect to the host:3000/messages/:room_id url.
- Client Proposal 1:
    - when calling enter_room_get_all_chat_logs() in client side, emit a join-room event to the server. (or the server doesn't know client's socket)
    - problem: duplicate joining the room if reloads
- Client Proposal 2:
    - when page reloads or (after login & closing old page) opening a new page, socket.io server get a new "connection" event
- To change room:
    - when I close the page or redirects to another page, the client side's global variable socket is destroyed. → the lifetime of a html global variable

Design Doc

- Client send room_id data to server:
    - ‘cuz client’s socket global variable gets destroyed when reloads,
    - ‘cuz (after login & closing old page) opening a new page, socket.io server get a new “connection” event
    - client: socket.on(“connect”, …) → emit a set-room_id event to the server.
- Use chore/socket-io-try branch for proof-of-concept:
    - when reloads / open a new tab / new login on a private tab of the same browser,
        - socket.id changes
    - socket.id the same for server side & client side,
        - socket.id the same at 5 places in client side
        - (1) when first initialized by io(). (2) before / after emit(“ci socket”) & calling enter_room_get_all_chat_logs(). (3) in XMLHttpRequest onload

Wrap up

- Watch writeup & checklist
- Record the demo video

Design Pattern: Singleton for Instance Getters

- Singleton, module export import order problem.

Bonus: OOP (C++)

- Read This How to solve it; Example(s)
- Composition over inheritance
- Explain the difference between “has a” and “is a”
- Design: Overloading vs. overriding vs. template
    - In your Shape implementation, which one do you use?
- OOP:
    - https://en.wikipedia.org/wiki/SOLID
    - https://teddy-chen-tw.blogspot.com/2014/04/solid.html
    - Google test.

Bonus: OOP (C++) (Object-Oriented Design) (cont.)

- Step1: Draw the UML Class Diagram for your shape.cpp implementation
    - Interface - shape
    - Implementation - circle, rectangle, …
- Step2: TBA

What You’ve Learned So Far?

- Simple full-stack app using JS
- Elementary level RESTful API design
- Map some concepts w/ practical implementation
  - HTTP Method
  - HTTP Status Code
- Basic concept of socket in web app
- Elementary level of coding SE practice
  - Modularity -> what does this mean?
  - Branch Naming Convention
  - Git
  - ~~Break down feature into tasks~~ implement given tasks & compose tasks back to working feature (and make sure it works)

Try to Answer The Following Questions (RESTful API)

- Why POST /messages instead of POST /send-messages?
- Do you name the register API POST /register? If so, does this follow RESTful practice?
- Read Best Practices for Designing a Pragmatic RESTful API
- Let’s say we have a new feature with the following spec, write the RESTful API design
    - Search history chat message by a single keyword
    - Search chat message by username
    - Search history chat message by time period (start timestamp & end timestamp)

One Step Further - Auth (1 day work)

- Protect chatroom by jsonwebtoken
    - Only login user should have access to chatroom page
    - Server issues the JWT token to client-side [Hint: which API should we update?]
        - Let JWT token expire in one day
    - Store JWT token at user browser [Ref]
    - Clear token at user's browser when user logout [Hint: client-side JS work]
- Alt. for token: Survey and implement express-session
- Why token over session? What's RESTful best practice?
    - Keywork: RESTful API is stateless

One Step Further - Middleware (0.5 ~ 1 day work)

- Study Router-level middleware
  - Implement parameter validator middleware with the following rule
    - POST /messages API should return 4xx error when receive empty payload
    - Login API receive empty username and/ or password should return 4xx error
    - Register API receive empty username and/ or password should return 4xx error
    - Register API receive the the following reserved username should return 400 bad request
      - admin
      - bothemrun
      - kobe
      - shangyi
- Helpful Reference: Joi
- Bonus: Can you make the design follow the open-closed principle?

One Step Further - Middleware (cont.)

- Implement authenticate middleware to verify JWT token for /messages API
- Optional Reading: Chain of Responsibility (Design Patterns)

~~One Step Further - CI (continuous integration) & Linter (<0.5 day)~~

- Survey and apply prettier (optional: ESLint) (Ref)
  - npm install prettier
  - Try out prettier command
  - Update package.json, add a “format” script to format your JS file
- Survey Github Action
- Use Github Action to run prettier before every PR
- Github Action not support private & ubuntu only has node 12 but prettier needs at least 14

One Step Further - Unit Testing (< 0.5 Days work)

- Study Jest Getting Start

One Step Further - Integration Testing (1~2 Days work)

- You’ll create a test-db using sqlite3 command line
    - Create table
    - Insert some random messages record
- You’ll write first test case for GET /messages API
    - Call GET API
    - Verify return value is not None (your sqlite has sth inside b/c of first step)
- You’ll write a test case for POST /messages API
    - Call POST API
    - Call GET API
    - Verify Given GET API contains the data that you post
- Question: If the test case always create the same data, how can we make sure the code work?
    - My test case for POST is written with the following sequence
        - POST /messages - payload -> “hello world”
        - GET /messages -> return value contain “hello world”
    - When I run this test case first time, there is no “hello world”, so the test case can verify POST API work
    - When I run this test case second time, there is already a “hello world” in DB, so my assertion of GET will pass, but I can’t verify whether the “hello world” comes from my second POST API or first POST API
    - In other word, I might break the POST /messages API, but the DB still have the stale data, so the test case is still pass.

One Step Further - Integration Testing (cont.)

- You'll Study Setup and Teardown to automate the first step in previous page\
  - Why do we need setup & teardown?
- You'll add error handling for messages APIs
  - When user post a message but payload contains nothing, should return 400 bad request
  - Implement this test, let it fail
  - Fix the failed test case by modifying your POST /messages API
- You'll implement test case for register, login, and logout
  - You'll implement 2 error handling for these three APIs
    - username not found -> what return code should be?
    - username conflict -> what return code should be?
  - Let the test cases fail
  - Fix the failed test cases by modifying your APIs

One Step Further - Integration Testing (cont.)

- What you’re doing in previous page is called TDD (Test-Driven Development)
    - Add test case
    - Let it fail
    - Fix the test case by implementing/ modifying code
    - Let test case pass
- Why TDD?
- ~~Integrate your testing process into Github Action, run at every PR~~

One Step Further - CI (continuous integration) (< 0.5 day)

- Survey CircleCI
- Use CircleCI to run unit tests/ integration tests at each PR

One Step Further - Architecture (1~2 days work)

- Refactor your code by following MVC structure
- Keyword: **Fat models, Skinny controllers**

One Step Further - Architecture (cont.)

- Why fat models, skinny controllers?

N Step Further - Deployment~~/ CD (continuous delivery)~~ (<1 day work)

- Survey Render
- Manually deploy your application to Render (sqlite3 version)
- Access your application using your mobile phone & play around
- What’s the difference between HTTP (localhost) & HTTPS (Render)?
- ~~Refine Github Action and run deployment when PR merge to main branch~~

DAO Design Pattern

- DAO: Data Access Object
    - https://www.digitalocean.com/community/tutorials/dao-design-pattern
- Why? No Array in sqlite3, but some other databases do. So not to have strong dependency on the choice of database.

N Step Further - NoSQL/ Refactor w/ test cases (1-2 days)

- Use MongoDB instead of SQLite
  - Use Docker to serve your MongoDB [docker pull mongo:5.0.12]
  - Study Mongoose or MongoDB Node Driver (official)
  - You’ll update POST & GET /messages APIs first
  - You’ll run test cases to verify the refactor doesn’t break anything
  - You’ll manually act as user to verify the feature doesn’t break
  - You’ll update register, login, and logout APIs

N Step Further - NoSQL/ Refactor w/ test cases (cont.)

- Does this application still work w/ Render? If not, why?

N Step Further - NoSQL/ Refactor w/ test cases (cont.)

- Does your refactoring follow the **Open-Closed Principle**? That is, do you modify the code of existing functions, or you can extend the functionality by adding code only? If not, how can you improve the design?
  - [Keyword: Data Access Object] (~1 day work)

N Step Further - Object-Oriented Analysis (< 1 day)

- Draw UML Sequence Diagrams for register/ login/ logout
- Draw UML Sequence Diagrams for chat

N Step Further - Your own feature (1 week work)

- Write a Use Case
- Create a Mockup [Ref] [Use a 10 minutes email to create an account]
- Implement the Feature
- The feature should contain all HTTP method of RESTful APIs
    - GET, POST, PUT, DELETE
- Implement the unit tests & integration tests
- Bonus: Can you implement a design patterns?
- Reference: Proposal & Report

Wrap Up

- Write your resume & attach the demo video

What You’ve Learned So Far?

- Basic knowledge of modern software development
    - Authentication
    - Middleware
    - Docker
    - SQL vs NoSQL
- Experience software development lifecycle
    - Feature Implementation
    - Testing
    - Refactoring
    - Use Case Proposal
- Basic knowledge of software architecture
    - MVC

Try to Answer The Following Questions (System Design)

- What's the difference between SQL & NoSQL?
    - [**Designing Data-Intensive Applications Chapter 2**] **Highly Recommended**
- In chat room app, do you prefer SQL or NoSQL? Why?
- What's the benefit of using MVC? What's the drawback?

Try to Answer The Following Questions (Refactoring)

- What is refactoring?
- How’s your feeling when refactoring [SQL -> NoSQL] w/ test cases? What will happen if you refactor the code w/o test cases?
- How do you ensure the refactoring doesn’t break the code?
- Write a general algorithm of refactoring
    - I.e. Step 1: xxx, Step 2: ooo, …

Try to Answer The Following Questions (UX/ PM)

- As a developer, can you understand the use case you wrote?
- To communicate, how do you feel when given only text-based writeup v.s. Clickable mockup?

Try to Answer The Following Questions (OOD)

- What is OOP?
- What is design patterns?
- Is MVC a kind of pattern?
- What’s the relationship between DP & OOP?

Try to Answer The Following Questions (Security)

- Can you conduct SQL injection for your application (sqlite3 version)? How to prevent this?
- What’s the risk of using JWT token? What if the client leaks there token?

Optional Topics

- Encryption. JWT. My Own Feature.
- Announcement. Admin.

- Security Issues: SQL Injection.   JWT.   Session Fixation.
- Design Pattern: OOP
- Unit Testing
- Integration Testing

- Use jQuery
- Template Engine
- Socket.io Example: Private Messaging To Scaling up
    - https://socket.io/get-started/private-messaging-part-4/

References For Beginners

- Project Example:
  - https://www.youtube.com/watch?v=SVnpp_OY4_E
- Better express generator intro:
  - https://www.section.io/engineering-education/nodejs-app-express-generator/
- Javascript + HTML
  - https://www.w3schools.com/jsref/
  - https://www.w3schools.com/js/
- Express.js
  - https://expressjs.com/en/starter/installing.html
- Socket.io
  - https://socket.io/get-started/chat
- Express.js template / view engine
  - https://www.digitalocean.com/community/tutorials/nodejs-express-template-engines#what-template-engines-should-i-use
  - https://www.educative.io/answers/what-is-a-view-engine-in-expressjs
- …

References For Beginners

- Pug:
  - https://pugjs.org/api/getting-started.html
- Express.js + socket.io chat room
  - https://www.freecodecamp.org/news/simple-chat-application-in-node-js-using-express-mongoose-and-socket-io-ee62d94f5804/
- jQuery:
  - https://www.w3schools.com/jquery/jquery_syntax.asp
- Client page:
  - addEventListener
  - jQuery
- Mobile browser by meta viewport:
  - https://www.youtube.com/watch?v=duKr29QU5ZI
- …

References For Beginners

- Sqlite3:
    - https://www.digitalocean.com/community/tutorials/how-to-use-sqlite-with-node-js-on-ubuntu-22-04
- …

Git Alias

```
[alias]

    str = !git remote -v update

    st =  status

    co = checkout

    br = branch

    ci = commit -a

    pu = pull

    ls = log --follow

    mg = merge --no-ff

    cibr = !export cur_br=$1 && echo $1 && (git push origin --delete ${cur_br} || echo "no remote branch ${cur_br}") && (git branch -D ${cur_br} || echo "no local branch ${cur_br}") && git branch ${cur_br} && git push origin ${cur_br} && echo "done"

    rv = reset HEAD^

    cm = !git co master && git pull origin master

    cmfse = !git co main && git pull origin main

    dd = !export cur_br=$1 && echo $1 && git push origin --delete ${cur_br} && git branch -d ${cur_br}

[pull]

    rebase = false


[user]
```