Teamcode: Collaborative Online Judge System

Author: Xinxin He

Github account: xinxhe

Github: https://github.com/XinxinHE/online-code-collaborative-system

Table of Content

Overview

Collaborative Online Judge System (COJ) is a full-stack system supporting code room, instant communication, and collaborative code editing, compiling, execution, and result judgment. This document covers the details of the implementation of code room and instant communication from an engineering perspective.

Major Use Cases

1. User can pick a problem and join its code room. They can join a specific code room, or start a new code room. The maximum number of users in a room is five.
2. When entering code room, the user is supposed to enter his or her nickname.
3. After entering a code room, each user will be assigned with a specific color for cursor identification.
4. User can chat with each other through chat box, the chat box allows user to send texts to the group.

Modules

Navbar
- Login/Signup

New Problem (only for admin)
- Problem name
- Problem description (text editor)
- Difficulty (dropdown list)

Problem List
For each problem:
- Problem difficulty
- Problem subject
- Room list with number of users in each room
- New room link

- Random room link

Problem Detail

- Problem subject and description
- Code editor
- Code room link
- Timer
- Chat box

UI Design

http://qdnr0b.axshare.com/#g=1&p=problems

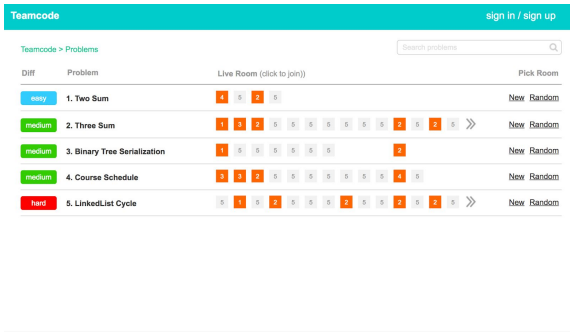

Fig.1. Home Page: problem list and room list

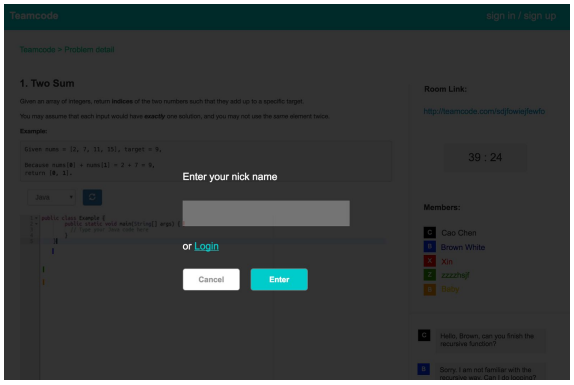

Fig.2. Enter nickname

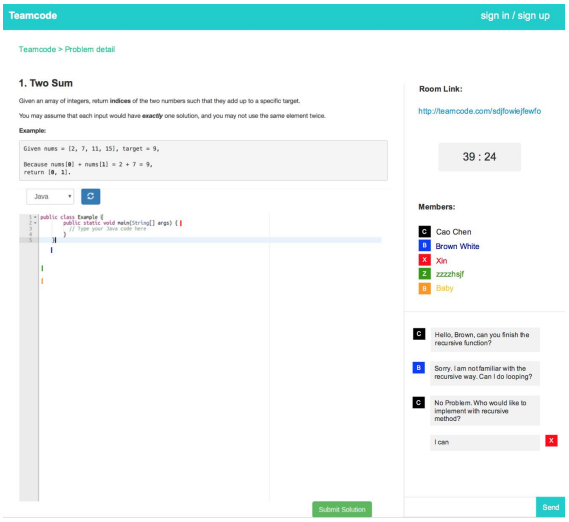

Fig.3.Coding room: chat box, timer, code editor

High-level Stack Program

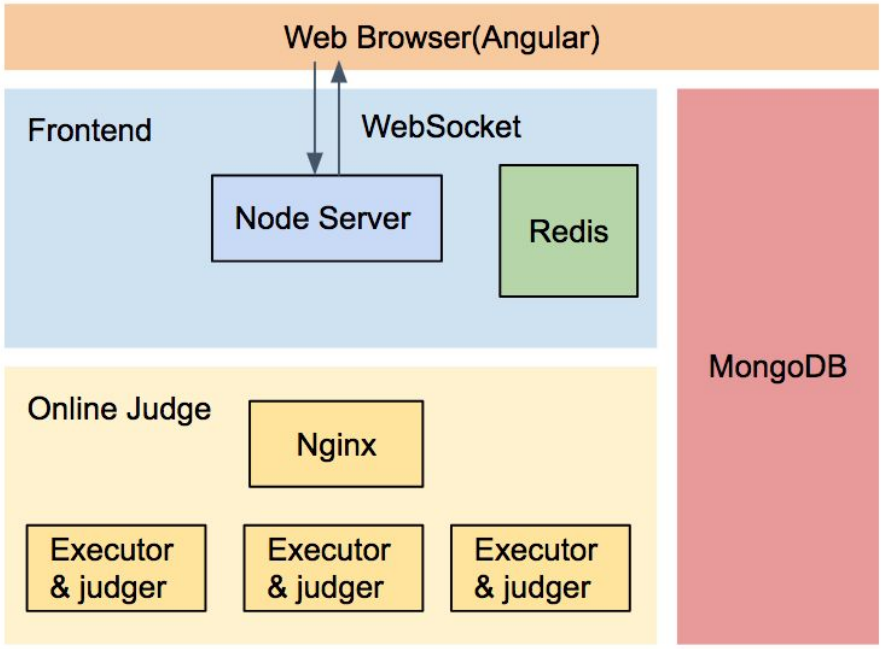

Fig.4. Web Application Architecture Diagram

Stack	Technologies
Frontend - client	**Angular4** : views, components, services **Socket.io** : maintain and response client-side socket events for coding room, code editor, message box, timers, room members, etc.
Frontend - server	**Node.js**: javascript engine **Express**: Node.js web framework **Socket.io**: maintain collaboration data structure and a list of socket events **Redis**: cache codes when all of the users leave room **MongoDB**: store problem list and details
Backend - executor	**Nginx**: load-balancer **Flask**: backend framework for code executing **Docker**: code executing environment

REST API Process

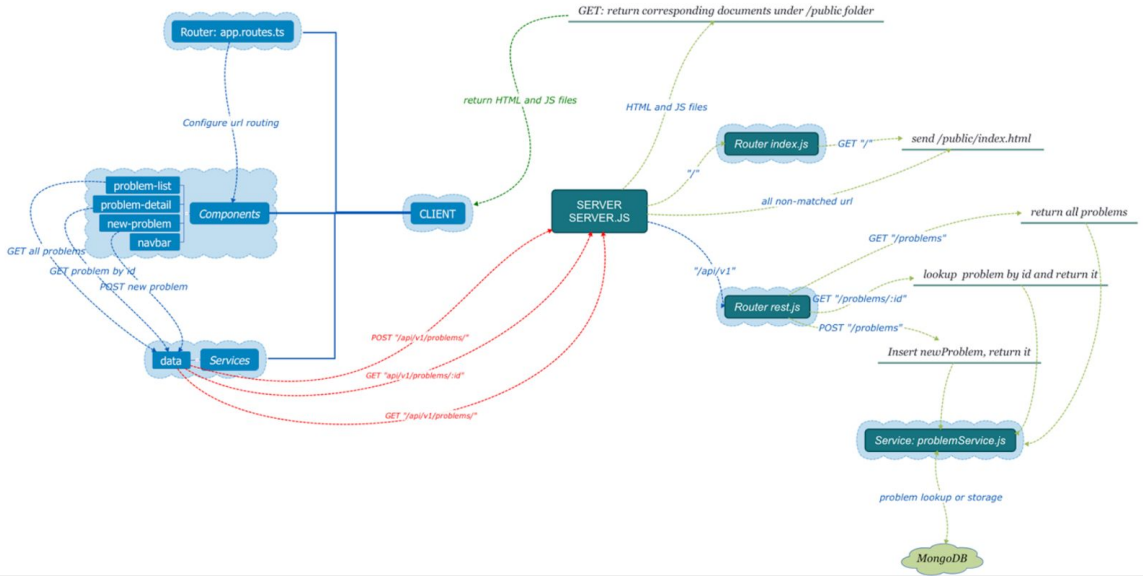

Fig.5. GET/POST problems with REST API

Socket IO Communication

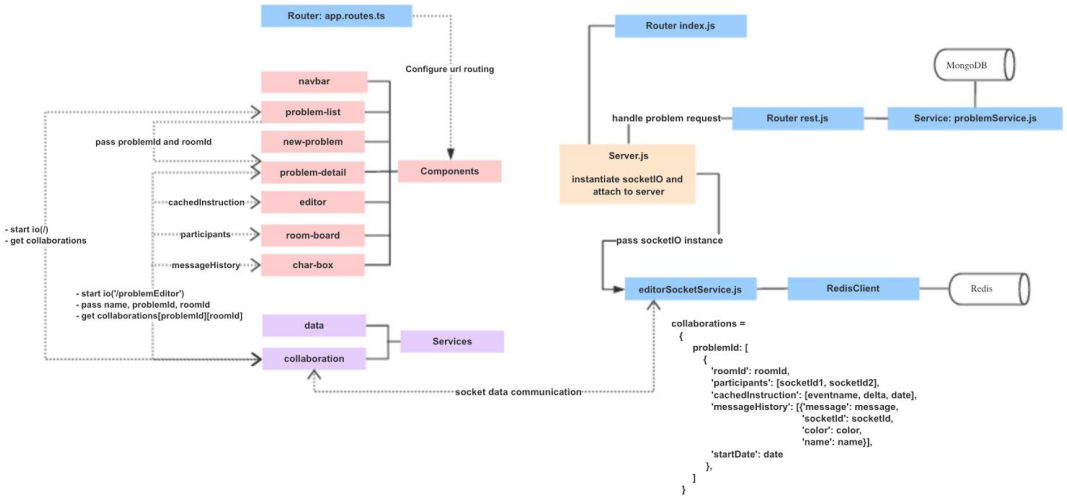

Fig.6. Bidirectional, real-time communication through socket io

Angular Components/Views

Problem List

Component: problem-list

- Get problem list through REST API
- Initialize root socket IO
- Get active room list and participant count through socket
- When entering a coding room, pop up a window to get nickname from user
- Pass problemId, roomId, and nickname through url parameters

Problem Detail Page

Component: problem-detail

- Get problemId, roomId, and nickname from url parameters
- Initialize sub socket IO by namespace '/problemEditor'
- Initialize Room board, Editor, and Chat Box components/views

Room Information Board

Component: room-board
- Get and list room participants with nickname, color
- Get room start date and initialize timer

Code Editor

Component: code-editor
- Initialize code editor with content and cursors
- Emit and respond editor content change through socket IO
- Emit and respond editor cursor change through socket IO

Chat box

Component: chat-box
- Send messages and update message history with socket IO

Socket Services at client side

collaboration.service.ts	Description
initSocket()	Initialize root socket IO through problem-list component.
getProblemsAndRooms()	Get problems and rooms through socket event and rxjs to problem-list component.
initCollaborationSocket()	Initialize a sub socket IO with namespace '/problemEditor' for a specific room, and pass problemId, roomId, nickname through socket query parameters. Called by problem-detail component.

initParticipantList()	Listen to ‘getParticipants’ socket event and get participants for room-board component with corresponding nickname, then pass to the component via rxjs.
initTimer()	Listen to ‘getTime’ socket event and get room start date to room-board component via rxjs.
initEditor()	Listen to cursor changes and editor content change through socket IO event listener, and update these changes to editor instance passed in.
initChatbox()	Listen to ‘sendMessage’ socket event and get newly-sent message and update them to chat-box component/view via rxjs.
getTime()	Emit ‘getTime’ socket event.
sendMessage()	Emit ‘sendMessage’ socket event.
change()	Emit ‘change’ socket event.
cursorMove()	Emit ‘cursorMove’ socket event.
restoreBuffer()	Emit ‘restoreBuffer’ socket event to get cached coding content from redis if there is any.
getParticipants()	Emit ‘getParticipants’ socket event.

Socket Services at server side

editorSocketService.js	**Description**
collaborations	Object to maintain a real-time problem list, room list, participant list, content changes, start time, etc.

	
socketIdToRoomInfo	Get a specific participant information through socketId. 
io.on('connection', ...)	Root socket IO, emit back 'getProblemsAndRooms' socket event.
io.of('/problemEditor') .on('connection', ...)	Sub socket IO, register new socketid, update collaborations and socketIdToRoomInfo objects, listen to socket events from client and response back the updated data.