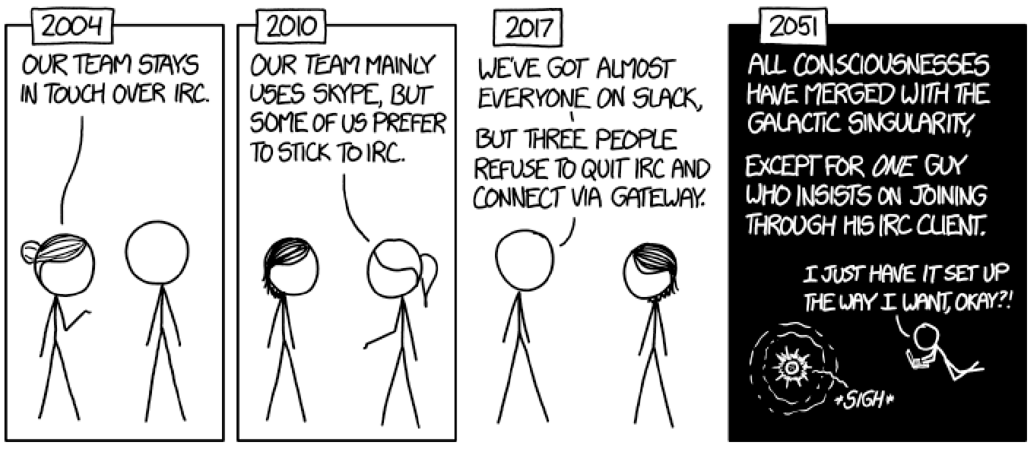
RTC_

< REAL TIME CHAT APPLICATION />

RTC

You need to create a Real Time Chat application with **NodeJS** + **ExpressJS** or **Rust**, and a client in **NextJS**.


2004
OUR TEAM STAYS IN TOUCH OVER IRC.
2010
OUR TEAM MAINLY USES SKYPE, BUT SOME OF US PREFER TO STICK TO IRC.
2017
WE'VE GOT ALMOST EVERYONE ON SLACK,
BUT THREE PEOPLE REFUSE TO QUIT IRC AND CONNECT VIA GATEWAY.
2051
ALL CONSCIOUSNESSES HAVE MERGED WITH THE GALACTIC SINGULARITY,
EXCEPT FOR ONE GUY WHO INSISTS ON JOINING THROUGH HIS IRC CLIENT.
I JUST HAVE IT SET UP THE WAY I WANT, OKAY?!
SIGH

Requirements

Before starting take care of this section:

- ✓ Your application accepts **multiple simultaneous connections** and implements *servers*, *channels*
- ✓ Each user must register before they can use the application
- ✓ The client and the server must use socket (message, status etc...) and a REST API (create, update etc...)
- ✓ You need to produce a document specification of your socket usage
- ✓ The server must enforce permissions and return appropriate error responses
- ✓ Your application should respect UI/UX standard
- ✓ **All** of your data should be persisted

Take care of dark patterns

To persist you should use at least one database
Think about which type is the best (SQL, NoSQL, etc...)

Features

On the client side, every user can do the following actions:

- ✓ Register and/or Login to an account
- ✓ Create server
- ✓ Join multiple server simultaneously with invitation code
- ✓ Leave a server

You will need to add 3 mandatory roles (feel free to add more if needed):

- ✓ `Owner`
- ✓ `Admin`
- ✓ `Member`

User creating server automatically become `Owner` of this server

No more than one owner on a server

Each `Member` should be able to do the following actions:

- ✓ Write message in a channel
- ✓ See who joined the server
- ✓ See who is connected (Consider connected someone actively online in a server)
- ✓ See who is typing inside a channel
- ✓ Delete his own message

Think about Socket.IO.

User should see previous message of a channel.

Each `Admin` should be able to do anything a `Member` can do plus the following actions:

- ✓ Create channel inside the server
- ✓ Delete a channel
- ✓ Update a channel
- ✓ Delete a message from another `Member`
- ✓ Create invitation to join the server

Each `Owner` should be able to do anything an `Admin` can do plus the following actions:

- ✓ Manage user roles on his server
- ✓ Transfer ownership of the server

An `Owner` must not be able to leave his own server

Architecture

Before writing **ANY** code for your chat application, answer these:

Where will business logic live?

- ✓ In your route handlers?
- ✓ In separate functions?
- ✓ In classes?
- ✓ Somewhere else?

How will you handle database access?

- ✓ SQL queries directy in routes?
- ✓ Wrapped in functions?
- ✓ A separate layer?

How will you test your code?

- ✓ Can you test without starting a server?
- ✓ Can you test without a database?
- ✓ What if you need to switch databases later?

What happens when your team grows?

- ✓ Where does Alice add user features?
- ✓ Where does Bob add server features?
- ✓ How do you avoid conflicts?

What if requirements change?

- ✓ Switch from PostgreSQL to MongoDB?
- ✓ Add GraphQL alongside REST?
- ✓ Add real-time features?

Warning signs of bad architecture

If you're experiencing these, your architecture needs work:

✓ ”I can't test this without the database”
✓ ”Changing one thing breaks three other things”
✓ ”I don't know where to put this new feature”
✓ ”My route handlers are 200+ lines long”
✓ ”Everything is in one giant file”
✓ ”I'm copying the same code everywhere”

Signs of good architecture

You'll know you're on the right track when:

✓ You can test business logic without HTTP/database
✓ Each file/class has ONE clear responsibility
✓ You can switch technical details (DB, frameworks) easily
✓ New team members understand where code belongs
✓ Addingg features doesn't require rewriting everything

Final thought

> ”Weeks of coding can save you hours of planning.”
> **— Every developer who learned this lesson the hard way —**

Don't be that developer. **Think before you code**.

Good luck with your research !

Endpoints

Your chat application should **at least** these endpoints. This list will help you think about your architecture:

Authentication

- ✓ `POST /auth/signup` - Create a new user account
- ✓ `POST /auth/login` - Authenticate and get tokens
- ✓ `POST /auth/logout` - Invalidate tokens
- ✓ `GET /me` - Get current user information

Servers (Communities/Guilds)

- ✓ `POST /servers` - Create a new server
- ✓ `GET /servers` - List user's servers
- ✓ `GET /server/{id}` - Get server details
- ✓ `PUT /servers/{id}` - Update server
- ✓ `DELETE /servers/{id}` - Delete server
- ✓ `POST /servers/{id}/join` - Join a server
- ✓ `DELETE /servers/{id}/leave` - Leave a server
- ✓ `GET /servers/{id}/members` - List server members
- ✓ `PUT /servers/{id}/members/:userId` - Update member role

Channels

- ✓ `POST /servers/{serverId}/channels` - Create a channel
- ✓ `GET /servers/{serverId}/channels` - List server channels
- ✓ `GET /channels/{id}` - Get channel details
- ✓ `PUT /channels/{id}` - Update channel
- ✓ `DELETE /channels/{id}` - Delete channel

Messages

- ✓ `POST /channels/{id}/messages` - Send a message
- ✓ `GET /channels/{id}/messages` - Get channel message history
- ✓ `DELETE /messages/{id}` - Delete message

Real-time (WebSocket/SSE)

✓ `WS /ws` - WebSocket connection for real-time updates

Finished everything? Read the next page!

STOP ! Only read this if you have:
- Implemented **ALL** required enpoints above
- Written tests for your core features
- Your app is working stable

If you're not done with the basics, go back and finish them first!

Bonus

Still here? Congratulation ! You're ready for advanced features.

If you've completed all the required features and want to go further, here are some challenging additions:

- ✓ Kick a member
- ✓ Ban a member temporary
- ✓ Ban a member definitively
- ✓ Edit own message
- ✓ Enhance user profile
- ✓ Add possibility to use emojis/Unicode
- ✓ Add other status (away, invisible, etc...)
- ✓ Add mentions
- ✓ Enhance authentication systems (2FA, Captcha, etc...)

v 3.5.3