Chat Server System Architecture

Section 5 – Friend Invitation and SocketIO Realtime Connection

# Introduction

Let's start with a quick model introduction.

The server will handle three types of connections:

1. HTTP request to expose REST API to clients,
2. Mongo DB connection to save application state,
3. TCP connection to realtime communications.

In this chapter, we'll delve into the invitation system. The invitation system's first goal is to accept or revoke client invitations and update the distributed system in real time.

The communications’ architecture relies on the [socket.io](https://socket.io/docs/v3) library. Socket.IO enables real-time, bidirectional and event-based communication between the browser and the server. It consists of:

1. A node.js server,
2. A Javascript client library fro the browser.

The Socket.IO implementation mainly exploits websockets, a technology now supported by all modern browsers. WebSocket is a communication protocol which provides a full-duplex and low-latency channel between the server and the browser. More information can be found [here](https://en.wikipedia.org/wiki/WebSocket). Although Socket.IO uses WebSocket as a transport when possible, it adds additional metadata to each packet. That is why a WebSocket client will not be able to successfully connect to a Socket.IO server, and a Socket.IO client will not be able to connect to a plain WebSocket server either. Here are the features provided by Socket.IO over plain WebSockets:

* reliability (fallback to HTTP long-polling in case the WebSocket connection cannot be established)
* automatic reconnection
* packet buffering
* acknowledgments
* broadcasting to all clients or to a subset of clients (what we call "Room")
* multiplexing (what we call "Namespace")

The bidirectional channel between the Socket.IO server (Node.js) and the Socket.IO client (browser, Node.js, or another programming language) is established with a WebSocket connection whenever possible, and will use HTTP long-polling as fallback. The Socket.IO codebase is split into two distinct layers:

1. the low-level plumbing: what we call Engine.IO, the engine inside Socket.IO;
2. the high-level API: Socket.IO itself.

Let’s see how we’ll implement this part of the system.

# Connecting Socket.IO to our Server

The Socket Server module is responsible for setting up and managing the socket.io server for real-time communication between the server and the client applications. The module includes the installation of the socket.io package, configuration of the socket server, and integration with the Express server.

### Installation of Socket.io Package

To begin with, the module requires the installation of the socket.io package using the following command:

npm install socket.io --save

### Configuration of Socket Server

After the successful installation of the socket.io package, the module creates a new file, ‘socket-server.js’, within the Discord backend folder to handle the socket server configuration. The ‘socket-server.js’ file contains the necessary code to set up the socket.io server and integrate it with the Express server.

### Integration with Express Server

The module includes a function, registerSocketServer, which is responsible for adding the socket.io server to the Express server. This function is exported and can be used in the server to trace module to integrate the socket server with the main server.

// socket-server.js  
const socket = require('socket.io');  
const registerSocketServer = (server) => {  
 const io = socket(server);  
 io.origins('\*:\*'); // Allow connections from all origins  
 io.on('connection', (socket) => {  
 console.log('User connected:', socket.id);  
 // Additional logic for handling user connections  
 });  
 // Export the function  
 module.exports = { registerSocketServer };  
};

### Integration with Main Server

In the server to trace module, the registerSocketServer function is imported and used to add the socket.io server to the main HTTP server before starting the server. This integration allows for real-time communication between the server and the connected client applications.

// server-to-trace.js  
const http = require('http');  
const express = require('express');  
const { registerSocketServer } = require('./socket-server');  
  
// Create HTTP server and Express application  
const app = express();  
const server = http.createServer(app);  
  
// Add socket.io server to the HTTP server  
registerSocketServer(server);  
  
// Start the server  
server.listen(3000, () => {  
 console.log('Server started successfully');  
});

### Testing and Verification

After setting up the socket server, the module includes instructions for testing the server to ensure that the socket.io server is working correctly. This involves starting the server and checking for any errors, followed by connecting the React application to the socket server to verify the connection.

Overall, the Socket Server module provides the necessary functionality to establish and manage real-time communication between the server and client applications using socket.io.

This detailed description outlines the key components and functionality of the Socket Server module, including the installation, configuration, integration, and testing of the socket.io server for real-time communication.

# Connecting to Socket.IO from Clients

The Real Time Communication module is responsible for establishing a connection between the React application and the socket.io server, enabling bidirectional communication and event handling.

### Installation and Setup

To begin, the module requires the installation of the socket.io-client package. This can be achieved by navigating to the front end of the application and executing the following command:

npm install socket.io-client

### Socket Connection Configuration

Once the package is installed, a new folder named real-time-communication should be created within the src folder of the application. Within this folder, a new file named socket-connection.js should be created.

The socket-connection.js file contains the necessary code to establish a connection with the socket.io server. The following code snippet demonstrates the import of the io object from the socket.io-client package and the creation of a function to connect to the server:

// Importing the io object from socket.io-client  
import io from 'socket.io-client';  
  
// Function to connect with the socket server  
const connectWithSocketServer = (userDetails) => {  
 // Establishing the socket connection  
 const socket = io('http://localhost:5002');  
  
 // Event to listen for successful connection  
 socket.on('connect', () => {  
 console.log('Successfully connected with Socket.io server');  
 console.log('Socket ID:', socket.id);  
 });  
  
 // Additional event handling can be added here  
};  
  
export default connectWithSocketServer;

### Integration with Application

The connectWithSocketServer function is then integrated into the application, specifically within the dashboard component. Upon verifying the existence of user details and retrieving the token from local storage, the function is executed to establish the connection with the socket.io server.

// Importing the connectWithSocketServer function  
import connectWithSocketServer from './real-time-communication/socket-connection';  
  
// Integration within the dashboard component  
useEffect(() => {  
 // Check for user details and retrieve token from local storage  
 if (userDetailsExist) {  
 // Set user details in store and connect with socket server  
 connectWithSocketServer(userDetails);  
 }  
}, [userDetailsExist]);

### Verification and Testing

Upon successful integration, the React application can be started, and the console should display a confirmation of the connection with the socket.io server, along with the associated socket ID. Additionally, the server terminal should reflect the same ID, confirming the bidirectional communication.

### Conclusion

The Real Time Communication module facilitates the seamless integration of socket.io for real-time event handling and bidirectional communication within the React application. It enables the application to emit and handle events, providing a robust foundation for real-time updates and interactions.

# Adding JWT Token to Events Emitted to Server

The module is responsible for validating JWT tokens at the server side in a React application. It ensures that only authenticated and authorized users are able to access protected events and routes. The module consists of two main components: the client-side implementation for attaching the JWT token to the initial connection with the socket IO server, and the server-side middleware for verifying the JWT token in the HTTP requests from the Axios.

### Client-Side Implementation

The client-side implementation involves attaching the JWT token to the initial connection with the socket IO server. This is achieved by passing the user details, including the username, email, and token, to the connectWithSocketServer function. The token is attached to the initial connection using a configuration object, ensuring that the authorization of the user is validated at the server side.

// Client-Side Implementation  
const connectWithSocketServer = (userDetails) => {  
 const config = {  
 authorization: {  
 token: userDetails.token  
 }  
 };  
 // Connect with socket server using the configuration object  
 // ...  
};

### Server-Side Middleware

The server-side middleware is responsible for verifying the JWT token in the socket IO events. It consists of a function verifyTokenSocket that receives the socket and the next object. The function decodes the token and retrieves the user details, ensuring that the token is valid and authorized. If the token is not valid, an error is returned to the client side, preventing unauthorized access to protected events and routes. The code is in *authSocket.js* file inside *middleware* folder.

// Server-Side Middleware  
const verifyTokenSocket = (socket, next) => {  
 // Verify token in socket IO events  
 // Decode token and retrieve user details  
 // Handle token validation and authorization  
 // ...  
};

### Integration

The module integrates the client-side implementation with the server-side middleware by using the ***verifyTokenSocket*** middleware before handling custom logic for the socket IO events. This ensures that the JWT token is validated before allowing the user to access protected events and routes.

// Integration of Client-Side and Server-Side  
// Use verifyTokenSocket middleware before handling custom logic  
// ...

All the middleware registration code is inside the ***socketServer.js*** file.

Overall, the module provides a robust and secure mechanism for validating JWT tokens at the server side, ensuring that only authenticated and authorized users are able to access protected events and routes in the React application.

# Server Store - Saving Information About Connected Users

The New Connection Handler module is responsible for handling new user connections to the server. It receives the socket object for each user and adds the user details to the server’s connected users map.

### Code Snippet:

// server.js  
const serverStore = require('serverStore');  
const socketHandlers = require('socketHandlers');  
  
// ... (other code)  
  
// New user connection event  
socket.on('connection', (socket) => {  
 socketHandlers.newConnectionHandler(socket, io);  
});

// socketHandlers.js  
const serverStore = require('serverStore');  
  
// New Connection Handler  
const newConnectionHandler = async (socket, io) => {  
 const userDetails = socket.user;  
 const data = {  
 socketID: socket.id,  
 userID: userDetails.userID,  
 userDetails: userDetails  
 };  
 serverStore.newConnectedUser(data);  
};  
  
module.exports = {  
 newConnectionHandler  
};

### Functionality:

1. When a new user connects to the server, the socketHandlers.newConnectionHandler function is called with the user’s socket object and the main server’s io object.
2. The newConnectionHandler function retrieves the user details from the socket object and creates a data object containing the user’s socket ID, user ID, and user details.
3. The newConnectionHandler then calls the serverStore.newConnectedUser function to add the user’s information to the connected users map in the server store.

### Usage:

The New Connection Handler module is used to manage new user connections to the server and update the connected users map with the details of each connected user.

This module is a crucial part of the server’s functionality as it ensures that user connections are properly handled and their details are stored for future reference.

The Testing Server Store module is responsible for storing information about connected users in the server store. This module ensures that the server is able to create real-time communication and handle user connections effectively.

### Functionality

The module allows the server to store the socket ID and user ID of connected users, enabling the server to send direct messages and add more logic to the application. The module also handles reconnecting to the socket IO server in case of connection refusal.

### Implementation

The module utilizes socket IO to automatically reconnect in case of connection refusal. Upon successful connection, the server stores the socket ID and user ID in a map, allowing for easy identification of connected users.

// Sample code for storing user information in the server store  
const connectedUsers = new Map();  
  
// When a user connects, store their socket ID and user ID  
socket.on('connect', () => {  
 connectedUsers.set(socket.id, userId);  
});  
  
// Later on, the server can use the stored information to send direct messages to the user

### Error Handling

In case of any errors related to the server store or connection issues, the module provides the option to review the attached code for the application. This allows for easy comparison and identification of any issues with the server store.

### Conclusion

The Testing Server Store module plays a crucial role in enabling real-time communication and user management within the application. By effectively storing user information and handling connection issues, the module ensures a seamless user experience.

# Creating Disconnect Handler

The Disconnect Handler module is responsible for managing the disconnection event of users from the server. It ensures that the server is able to track and handle the disconnection of users from the Socket Io server, allowing for real-time functionality and communication between users.

### Functionality

The Disconnect Handler module performs the following key functions: 1. Listens for the disconnection event of specific socket connections. 2. Handles the disconnection event by passing the details of the disconnected socket object to the Disconnect Handler. 3. Deletes the disconnected user from the list of connected users, allowing for accurate tracking of online users. 4. Provides the ability to track online users and update their status in real-time, enabling communication and interaction between users.

### Implementation

The Disconnect Handler module is implemented as follows:

// disconnectHandler.js  
  
const server = require('../server.js');  
  
const disconnectHandler = (socket) => {  
 server.removeConnectedUser(socket.id);  
 console.log(server.getConnectedUsers());  
};  
  
module.exports.disconnectHandler = disconnectHandler;

The disconnectHandler.js file contains the implementation of the disconnectHandler function, which takes the socket object as a parameter. Within the function, the server’s removeConnectedUser method is called to delete the disconnected user from the list of connected users. Additionally, the console.log statement is used to log the updated list of connected users.

### Integration

The Disconnect Handler module is integrated into the server by requiring the disconnectHandler module and calling the disconnectHandler function when a disconnection event occurs.

// server.js  
  
const { disconnectHandler } = require('./disconnectHandler.js');  
  
io.on('connection', (socket) => {  
 // Other event listeners and handlers  
   
 socket.on('disconnect', () => {  
 disconnectHandler(socket);  
 });  
});

In the server.js file, the disconnectHandler module is required, and the disconnectHandler function is called when a disconnection event is detected. This integration ensures that the Disconnect Handler module effectively manages disconnection events for the server.

### Testing

The functionality of the Disconnect Handler module can be tested by simulating user disconnections and verifying that the module accurately tracks and handles the disconnection events. This can be achieved by running the server and observing the console logs for the updated list of connected users.

Overall, the Disconnect Handler module plays a crucial role in managing user disconnections from the server, enabling real-time communication and interaction between users.

# Redux - Preparing Actions And Reducers for Friends Logic

The module “Redux - Preparing Actions And Reducers for Friends Logic” focuses on implementing the logic to invite older users to become friends within a web application. The module involves creating the necessary actions and reducers in Redux to handle friend invitations, manage the list of friends, and keep track of pending invitations and online users.

### Dialogue Creation and User Interaction

The module begins by creating a dialogue for inviting older users to become friends. This dialogue includes a button that, when pressed, triggers the sending of the friend invitation. Upon sending the invitation, the other user should be able to see the incoming invitations list.

### Implementation of Redux Logic

The module involves writing React logic and server logic to facilitate the functionality of sending and managing friend invitations. This includes creating a Redux producer to handle the state of friends, pending invitations, and online users.

### Friends Producer Implementation

The Friends producer is implemented to maintain the state of friends, pending invitations, and online users. The initial state of the Friends producer is defined, and a switch statement is used to handle different action types. The producer is then exported as the default.

// Friends Producer  
const initialState = {  
 friends: [],  
 pendingInvitations: [],  
 onlineUsers: []  
};  
  
const friendsProducer = (state = initialState, action) => {  
 switch (action.type) {  
 case 'SET\_PENDING\_INVITATIONS':  
 return {  
 ...state,  
 pendingInvitations: action.pendingInvitations  
 };  
 case 'SET\_FRIENDS':  
 return {  
 ...state,  
 friends: action.friends  
 };  
 case 'SET\_ONLINE\_USERS':  
 return {  
 ...state,  
 onlineUsers: action.onlineUsers  
 };  
 default:  
 return state;  
 }  
};  
  
export default friendsProducer;

### Friends Actions Implementation

The module also involves creating actions for managing friends, pending invitations, and online users. These actions are defined in a separate file and are imported into the Friends producer for dispatching.

// Friends Actions  
export const setPendingInvitations = (pendingInvitations) => {  
 return {  
 type: 'SET\_PENDING\_INVITATIONS',  
 pendingInvitations  
 };  
};  
  
export const setFriends = (friends) => {  
 return {  
 type: 'SET\_FRIENDS',  
 friends  
 };  
};  
  
export const setOnlineUsers = (onlineUsers) => {  
 return {  
 type: 'SET\_ONLINE\_USERS',  
 onlineUsers  
 };  
};

### Conclusion

The module concludes by preparing the friends actions to send an HTTP request to the server with the new invitation. This completes the implementation of the Redux actions and reducers for managing friend invitations and user relationships within the web application.

# Preparing Action To Send Friend Invitation

The module “Sending Friend Invitation” is responsible for creating an action to send a friend invitation in the application. This module utilizes Redux Thunk middleware to handle asynchronous API requests and dispatch actions accordingly.

### Functionality

The module consists of the following key functionalities: 1. **Action Creation**: The module creates an action named “sendFriendInvitation” which accepts the necessary data, such as email and handler, to send a friend invitation. 2. **API Request**: It makes an API request to send the friend invitation to the server, utilizing the API client and handling any potential errors or exceptions. 3. **Alert Messages**: It dispatches alert messages to notify the user about the status of the friend invitation, such as success or error.

### Code Implementation

The significant piece of code for the “Sending Friend Invitation” module is as follows:

// Action Creation  
export const sendFriendInvitation = (data, closeDialogueHandler) => {  
 return async (dispatch) => {  
 try {  
 const response = await API.sendFriendInvitation(data);  
 if (response.error) {  
 dispatch(openAlertMessage(response.exception));  
 } else {  
 dispatch(openAlertMessage('Invitation has been sent'));  
 closeDialogueHandler();  
 }  
 } catch (exception) {  
 if (response.code === 401) {  
 dispatch(logoutUser());  
 } else {  
 dispatch(openAlertMessage(exception));  
 return { error: true, exception };  
 }  
 }  
 };  
};  
  
// API Request  
const API = {  
 sendFriendInvitation: async (data) => {  
 try {  
 const response = await API.client.post('/friend/invitation', data);  
 return response.data;  
 } catch (exception) {  
 if (exception.response && exception.response.code === 401) {  
 logoutUser();  
 } else {  
 return { error: true, exception };  
 }  
 }  
 },  
 client: API.client, // API client for making requests  
};

### Dependencies

The module relies on the following dependencies: - Redux Thunk middleware for handling asynchronous actions - API client for making HTTP requests to the server

### Usage

The “Sending Friend Invitation” module can be utilized by importing the sendFriendInvitation action and dispatching it with the required data and close dialogue handler.

Overall, the module provides a robust and secure mechanism for sending friend invitations within the application, handling API requests, and alerting users about the status of the invitation.

# Connecting Send Friend Invitation Action To Dialog

The module is responsible for connecting the “Send Friend Invitation” action to the dialog in the application. It involves creating an HTTP request to the server and implementing the necessary actions and components to enable the functionality of sending friend invitations.

### HTTP Request Preparation

The module begins by preparing an HTTP request to the server, which will be used to send the friend invitation data.

### Action Creation

An action is created in the Actions folder to handle the sending of friend invitations. This action will be executed when the user initiates the invitation process.

### Dialog Configuration

The module configures the front dialog in the dashboard folder to enable the user to send friend invitations. It involves importing the necessary components and actions to the dialog to facilitate the invitation process.

### Action Mapping

The module maps the “Send Friend Invitation” action to the dialog props using the connect function from React Redux. This allows the action to be accessible within the dialog component.

// Example code for mapping actions to props  
import { connect } from 'react-redux';  
import { getActions } from 'store/actions/friends';  
  
const mapActionsToProps = (dispatch) => {  
 return {  
 ...getActions, // Return all actions from the Friends file  
 dispatch // Pass the dispatch function  
 };  
};  
  
export default connect(null, mapActionsToProps)(DialogComponent);

### Invitation Data Handling

The module handles the data for the friend invitation, such as the email of the user being invited. It ensures that the invitation data is properly passed to the server for processing.

### Testing and Error Handling

The module includes testing and error handling to verify the functionality of sending friend invitations. It also handles potential errors, such as invalid tokens or server-side issues, and provides feedback to the user.

### Future Development

The module outlines plans for future development, including creating server-side logic to accept and process friend invitations, and implementing a database model to store the invitation data.

Overall, the module plays a crucial role in enabling users to send friend invitations within the application, and sets the foundation for further development of the friend invitation feature.

# Creating Server Routes - Friends Invitations

The module is responsible for creating server routes for handling friend invitations. It involves the creation of routes, controllers, and validation schemas to facilitate the process of sending and handling friend invitations.

### Server Trace File

The server trace file is used to register the routes for logging, registration, and friend invitations. It involves creating a new file for friend invitation routes and defining the necessary logic for handling friend invitations.

### Friend Invitation Routes

The module creates a new file for friend invitation routes in the “Roots” folder. It defines the routes and uses Express, Router Joy, and validation middleware to handle the friend invitation process. The module also defines a validation schema for the post invitation, ensuring that the email address provided is valid.

Significant piece of code:

// Creating friend invitation routes  
const express = require('express');  
const router = express.Router();  
const validator = require('express-joi-validation').createValidator({ passError: true });  
const { postInvitationSchema } = require('./validationSchemas');  
  
// Define validation rules for email address  
const postInvitationValidation = validator.body(postInvitationSchema);  
  
// Define post request for friend invitation  
router.post('/invite', verifyToken, postInvitationValidation, friendInvitationController.postInvite);  
  
module.exports = router;

### Friend Invitation Controllers

The module creates a new file for friend invitation controllers, which includes the logic for handling friend invitations. It defines the post invite controller to handle the process of sending friend invitations and saving the information in the database.

Significant piece of code:

// Post invite controller  
exports.postInvite = (req, res) => {  
 const { targetEmail } = req.body;  
 // Logic to save invitation in the database  
 // Send response to the user  
 res.send('Friend invitation sent successfully');  
};

### Error Handling

The module also includes error handling to ensure that the process of sending friend invitations is smooth and any errors are appropriately handled.

### Conclusion

The module provides the necessary infrastructure for handling friend invitations, including routes, controllers, validation, and error handling. It ensures that the process of sending and handling friend invitations is efficient and reliable.

# Creating Friend Invitation Model

Let's create right now the logic of our control order, which is responsible to save our front invitation

in the database.

And the first thing which we will need to do it as we have already created one model of the name of

the user, we need to create a new model of the name of friend invitation that we'd like to just save

the information about the invitations to the database.

One thing which I would like to correct in the user model is here that the good practice is that we

just give a model name if the uppercase, not the small case.

So we should call like this that that would be the user, not the user of the small letter at the beginning.

And now, as we have that, you said, the trick is we can create a new file of the name of a friend.

Invitation.

Todd Jealous and Hugh, let's create our motel.

And at the beginning, we will need that money goals, which we have also right here, and we'll just

get the schema from concept schema from the mungo's schema.

And thus we have that.

Let's prepare that schema and we'll go for that.

Friend invitation.

Schema.

And that will be the new schema.

And we just need to configure that what fields we would like to have.

We would like to have an object ID of the database of the user.

We just sent this invitation.

So the type of this field will be that schema dot types object ID, so will be just an object ID.

This one, which widget the MongoDB is given for this.

And also, we would like to keep that reference.

We'll update a file, the reference and what does it mean if all of the reference that this ID is related with the user model?

Later on, thanks to this, ID will be able to get more details about this user with a simple function, which will allow us to connect and to get the user details.

Thanks to that object, IDF will pass this reference and also that will be the sender I.D. and also would like to get the receiver ID.

Because the sender would be the person which is sending this invitation and the receiver that will be the target which would like to invite and also will pass the same type that also that will be the schema.

Dot Type's Dot object I'd.

And also will refer this that is related with the user model, which we have created.

And at the at the end, we just need to do it one thing which will be model exports and we'll just go for the mungo's. Dot Motel.

And the name will be friend invitation, and we'll pass this schema, which we have created.

Invitation schema and our model is ready and we can use it in.

We can use that logic in our controller.

# Custom Friend Invitation Validation

Now we can use the model, which we have created, I have made a simple typo that I wrote Mongo Dot

Molder, but that should be the model of the same with the user.

And now as we have that.

Let's go to our post invite controller and start writing the logic of it.

And now this is the e-mail of the user, which we would like to invite.

And also, we need the idea of the user.

We just sent that invitation.

And of course, we can get that idea from their gravity token from their request dot user.

Because as far, remember, if the JVP token is valid in our media that were of torches will decode

the user details and will attach this to request DOT user that information of the user, which is the

user details.

And now we can use it.

And as we have that, we will getting that information.

First thing, which I would like to check, because we need to check some things, and the first thing

which I would like to check is that check if friend.

That.

We would like.

To invite is not us because we can send the invitation to the same user as our email address, so it

doesn't make sense that we would like to be friends with that yourself ourselves.

So we just need to check.

We'll go for the mail.

Two lower case.

To make sure we'll get that lower case, and we'll compare this with the target male to lower Karl case

because this one is our male.

And this is the male of the user which would like to invite to make sure we'll just change this for

only lower letters.

And if that will be the same, we just need to send the information to the user that we are not able

to invite ourself and how we can do it.

We'll just return.

Address thoughts of the response.

Object will go for the status of four oh nine because it's going for the conflict that's in the conflict

ocurre.

And the response will be sorry dot.

You cannot become friend.

We've.

Yourself like this or you can ask your own custom message.

And now that will be the information for the client that he would like to invite.

Ourself and now as we have that, we can check that later on if we'll be able to get that information.

And what we'd like to do it also, we would like to check if that you so which we would like to invite

exists in the database and we will go for concert target you, sir.

So that's the user which would like to invite will go for eight user.

So we'll just need to import this user model and we'll go for the find one.

And we would like to find find this user by his email and will pass mail would go for the tie target

email address to lower case like this.

Now if we will be able, if will not find this user will send the information to this user.

We just sent the invitation that the target the user which would like to invite has not been found.

So we'll go for response status of four 04 not found and it will just send a text message with friend

of we can use that matter which which he has passed so we can go for the batiks and here will pass this target mail address.

Like this has not been found.

Now, please check.

Mail address of the user, of course, which you would like to invite, so we have two cases right now.

If the user we would like to invite will be our self or with this user, which we would like to invite will not find this user.

And one more thing, we also need to check if.

Check Eve invitation has been already sent because they can be situation that we have already sent the invitation, but we didn't receive the response, so we would like not to save this invitation once again in the database, but just send the information to the user that.

Invitation has been already received by the other side, and we just are waiting for the response.

And we'll do that later on, because for now, as I have this logic, I would like to test that in the next video that if we will be not able to send the invitation to our self and also to send the invitation to the user, which does not exist in our database.

But what we need to do it.

We need to take this user and get that user model to let us to get the information from the database.

So we'll go for the contact user and we just need to require that model of the user, which we have created before.

And that will be the models, and here will have the user model like this until next week.

Let's check if that would be working.

# Testing Friends Invitations Custom Validations

My server is automatically restarted because of the note them on.

And now I can go to the my application.

Open the up front, Diallo can check the responses, which the server is sending to me.

First, I would like to invite my person so I would go for the test number two attachment.

Com because I have log in at this account, which I have created before, and I will send and that what I'm just getting, I'm getting the information.

Sorry, you cannot become friends with our self.

We for yourself should be our self, probably.

And now, as we have that, we know that it's working.

So let's try to invite the user which does not exist so we can go like I just a -- email dummy mail at Gmail dot com, and now we should get the information that this user does not exist.

Let's send it front of this e-mail has not been found.

Please check my email address.

So that's the reason why we are not closing that dialog because we would like to get that response.

And if something is wrong, we can just correct that e-mail address of the user. And now it's working that we can proceed to the next step.

# More Custom Validation Related With Friend Invitations

We need to check more things.

Now let's go and check if the invitation has been already sent, how we can do it.

Let's define a new variable of the name of concert invitation already received.

Awake invitation does not find one.

And what we'll have right here, if what how we can find this invitation, as you remember in our front

invitation motel.

We have the user.

The two fields the sender ID were the type of this is just the object idea given by the MongoDB and

also the receiver ID given by that idea and how we can do it in the friend post invite, we can find

if that invitation exists by checking if sender thought I'd is is target user DOD I.D. because.

Here we are a look in if that user exists and if that see if that you said this will God, it will get

all of the details of this user from the database in this variable with that constant value, which

is target user.

And we can right now use this target user and every target every user has and their field, which is

an idea which we are getting by the database, MongoDB and we can use it that will be looking for the

invitation within the sender is the user of this specific I.D. and the receiver ID is the user.

Descender, that's exactly not to be like this, because the receiver idea.

The sender ID is our ID, so the target user that ID should be the receiver because we are looking for

this user, which we have already invited and the sender ID should be there.

Our ID as we are sending that and we can get that I.D. from the JVP token and it's common in the field

as we have encoded that as the user I.D..

So that's the our idea and that's the idea of the receiver because we are looking for this user by his

mail and who he will be found.

We'll get that in that value target user and we can just use that idea, which we are seeing right here.

And now.

We can check that invitation.

He's already sent if invitation already received, we would like to return the information because if

will not find an invitation of this to specific file use.

Fields will be not able to.

And you will not save the new invitation in the database.

So we'll go if the invitation already received, we'll just return the response.

Which will be the status of four oh nine, which will go for the conflict of the name of invitation

has been already set, ladies.

And now what we need to check also, as we are checking the invitation has been already sent.

Also, we would like to check.

If you, sir.

Check if the user which we would like.

To invite is already our friend, and we should be checking for that because it's going to be a situation

that we would like to send the invitation of the user, but she's already our friend and we can go and.

We'll check if the users are already different, so we'll go for the users.

Already, friends and how we can check that it will go and go for the Typekit you target user and every

target user have a collection of.

We don't have added that, so we just need to go to our user right now on right now, and we will create

that every user will have a new field of the name of the friends where it have an array of the object.

So that will be just an array of the users where we'll go for the type of schema.

Dot Type's Dot Object ID.

And here we would like to keep the information.

We would like to keep that just the and that an array of the object IDs of the users, which are our

friends.

So we'll just do it like this.

And now we can take that field, which is the friends in our controller and we'll go for this user,

which we have found and we'll go forward their friends.

And now we'll check if any I.D..

Friend ID will go find.

Target user friends find, and that method will fight, if any, if that statement, which right now

passed will be true.

The information that you share will be returned and that will be that will have a value, so that will

have a value.

We would like to return the information that the user are already friends, but now we just need to

compare if that that will be fronted because that's the NRA of the Friends side.

And now if this idea?

Will be the same.

We'll go just to string if you compare the object it, this is good to use to shrink.

And now if you compare an I.D. of the friends for four of every friend, we just need to compare it.

Today, you said we just sent this invitation, so we'll go for the user ID to sink, the idea of this

user would just send that invitation.

And now if the users are already different.

If users are already friends, we would like to just send the information that.

Return response, dot status and would go for four, oh nine, and we'll just send rent already, added

police check.

Friend friends least like this.

So now, once again, I would explain that because I was also confused about this.

And now every user has and field of the name of the friends where this is.

This is an array of the users ideas.

So every user have the collection of his friends like this and now we can take that field of the friends.

Now we will have no friends in this array, so we don't have how to.

We don't have the possibility to check it up because we asked if we will accept the invitation will add the object idea to two users if they are friends together.

And now we can check if this you said it, which would like to invite will take this array of the friends and will find, if any idea of the friend.

Is this same of the I.D., which is this user, which from the user which is just sending in this invitation?

And if yes, we would like to send the information that they are already dear friends and now us, we have that.

After that, we can create the new invitation and save it in the database, create the new invitation in the database.

Because if?

One of our customer validation, if if not, our request, will pass our customer validation and the friends will not be different with yourself.

That may others will exist.

They are not friends or will be able to send to save that invitation in database and will do that in the next video.

# Saving Friend Invitation In Database

As we have our custom validation for different invitations, we just need to right now save that invitation

in the database if we are able to post all of that validation.

And first thing, which I would like just to change correct here we are using the front invitation to

find if that invitation exist.

And here we should use different invitation to find one like this and import this as a model of the

same as we have been doing with the user.

So we'll go for a concert friendly invitation and we'll will just require this model, which we have

created because this model of to create that request to the database.

So we'll have that just go to the models and here to friend invitations, friend invitation.

Let's save it.

Now we have information about the error that the schema is not defined where we have updated that information.

Our user model.

Now, as you see, we are going referencing to the schema.

But from where we can go, get the schema that will just const schema is just the Mongo start schema

like this.

Now let's save it, and everything is fine, let's say.

Right now, the invitation.

And as will pass all of the validation we create and your invitation in the database and how we can

do it, let's go until it's I go as a new invitation.

Won't go away, invitation got to create.

And now if you'll create the invitation, go for the sender ID will be just the user ID from their JV

token.

So the idea of the user will just send the invitation, send their I.D. and now receiver I.D. will be

the target.

A user dot.

I'd just.

So this user, which we have found in the database, as you see, create method on the Mongoose model,

is just creating the document and saving that in the database.

Now us, we have that we should.

Later on, if invitation has been successfully.

Sent.

Or created?

We would like to.

Update, France invitations.

If order you serve is already is online, but will do it that later on, because it's related with that

like communication, not evolution, a different invitation, other user will see that immediately on

his pending invitation list.

And if we have that, we can send the information of the status.

We'll go for response status of two, oh, one and two.

We'll just send information.

That invitation has been sent.

So thanks, dad.

Here we are using the invitation, we should go, of course, for the friendly invitation because that's the name of the model which we have imported at the top and French invitation to create it creates a new document in the database and say in the US, save it.

And as we have, that's new invitation later on will be able.

To save that in the.

We are saving a database of which are sending information back to the user, that invitation has been sent.

And we have saved that in the database.

So in the next video, we can check if we will provide an valid email address of the second user.

If we'll be able to get that information, that invitation has been sent, and let's check what they feel will have in our database if that invitation has been successfully saved in database.

# Testing Friend Invitations

Let's check right now if we are able to send the invitation, I have opened my mom up laughs when I

have my collection as I'm seeing if I have created a new model and my application has been restarted.

I have a new collection of the name of friend invitations, and now we can check if we'll send the invitation

will able to see that right here in the database.

I have the users right here which I have created before, and they have test number one, test number

to test number free and my personal mail address and now I'm logged in acid test number one and I will

try to invite this user, which I have at the second place, will be test number two that I would like

to be friends each other with this user.

So I would just refresh my dashboard.

I would go for the art front and I will paste this email address.

Let's send it and what I'm checking.

I'm checking the information.

I am getting the information.

That friend already added, Please check Friends' List, but this friend is not already added because

we don't have an invitation friends invitation right here.

So something is wrong.

And we need to check this at our server without prejudice once again.

Still, I'm getting this conflict in here.

We are just getting that four oh nine and we just need to check why it's happening.

And the reason why we are getting that information that the users are already two friends together is that's only in my case because I have created this separate project a new future to our Discord clone and I have been using the same database.

And that's as you see, I have already friends already in my application.

But as you see, I don't have any friends asymmetric.

My private mail, so we'll just invite right now my private email, because this user is already with the friend, with the test number two in my database, so I just need to go for the user, which I'm not different with someone.

Go for this email address and that will send that invitation.

And now I'm getting the information.

That invitation has been sent and I will go back here to my collection.

And now, if I would go to different invitations, I'm seeing one document that this user of this specific idea has sent this invitation.

The receiver is the idea of this user, which I would like to invite to see.

One is at the end so I can check if the same ideas of this user, which I would like to invite and my private email has the same idea as we would like to invite.

So everything is working correctly.

So that's good that I have already, dear friends added in my application.

But in the next step, I will remove them because we have checked that our validation that the users are already different is working correctly.

So even so, we have even check that.

Now what we should take her right now about that one create the real time.

Possibility that if the second user is logged in, he will get that invitation in the real time, and even if you will log in the first time to the dashboard, he will see which user has invited him.

# Preparing SocketIO Client Event Listeners for Realtime Friends Invitations

We are able right now to save invitation in database.

What we can do it right now is to take care about this, that every user will see their real invitations, which they are awaiting for decision and how we can do it.

Let's go back to our code and at the beginning will prepare our React application for the incoming data from the server.

We'll go to the file, which is the socket connection.

Yes.

And this file is in our dashboard, not the dashboard folder, but in our real application, in the

source folder.

In this code frontend.

And here we have to follow the written communication and the insight will be socket connection.

Now we are only waiting for the event.

We are only listening for the event of the same connect, but we can listen for more done on the Connect.

We can listen for our custom events and the first event, which our custom event, which I would like

to listen for, which will be so on.

Friends, invitations and who prepare an action at our set of variables would be irresponsible to omit

that event.

And if that event will be emitted by a server to at.

Because that will that soccer is related with every specific connection, every single, a React application.

So it the server will emit friends invitation to us will receive a data because we prepared this at

the server and we'll go and to dispatch.

Set spending can set spending France invitations will check you if you have created that action before

where we'll be getting the spending invitations.

From the data.

Which will be coming from the server.

And now let's check what we would like to have that said, for pending France invitations action.

We just need to take care about this in our story.

So if we have the France actions?

Here we have sent front invitation, but here I would like to create more.

Actions in this file.

And we can do it because in our here, in our producer, in the front producer, we have pending France invitations and as you see, we can create an action which will be responsible to this part of the action of the name of the type of this, which is the satanic French invitation.

And we should save the information in our story about users, which will be sending notifications.

So we will go right now to France actions and will prepare.

Simple action, which will be able to dispatch let's go and export concert set pending front invitations where we'll be accepting the pending France invitations.

And we'll just return the type of the action, which we would like to dispatch, and that will be the transactions set pending friends here, exactly this one, which we have defined before suspending friends invitations and also just the spending for our friend invitations, which we are getting right here to that action and now.

We can dispatch this action to change our Star State.

Let's go back to socket connection and head here where we would like to dispatch said friends invitations, we just need to import that from the correct file.

And that will be coming from.

That would be store will go for their actions, and that will be their friends actions.

Let's go back and we can right now take the spending invitations which will be coming in data from the server and here will just this dispatch.

That action where we will pass this data, so will change our story, state and defense of the Star State will be able to render the new friends invitations and that will be at our react.

Part of application in the next step will just create the set of a logic to send your real pending invitations to every single user, which is just log in to our application.

# Preparing at Server Pending Friend Invitations

Let's work right now at the server side to have a Real-Time Communication System to arbitrate the pending

invite friends invitations, and we'll go to the folder of the name of.

Socket handlers and in this folder will create a new folder of the name of updates.

And here we have the functions which they will be responsible to send to the users the information about

the new pending invitations or the friends list, or they're later on the new chat messages that will

create the functions which we will be able to reuse.

And if we'll pass the user ID of the receiver, he will be able to get this update and we can start

with the Friends Dot Trace file.

So here we'll have an update related with the friends and what we will need right here in this file.

Of course, we will need the user model.

So we just need to go to the top level of our server, and we'll just go for this user model, which

we have created before.

Also, we would like get the rent.

Invitation.

Let's go for this model.

What do we had to find different invitation model?

So that will be the models.

And let's go for different invitation.

Also, what we would like to have, we would like to get this server store where we are keeping the

information about the users, which they are right now connect because that will be necessary, that

if any user will be sent to the other user, their friend invitation, we would like only sent from

the server only to this directed user which received a pending invitation.

And if this user is connected, he's an online user who has received an update from the server.

So we'll just go for the server store like this.

Now, let's prepare the fangs first function will be update friend.

Friends spending in the nation's.

And that will be async task, because here we have a logic related with database and here will be accepting

to use our I.D. because now first we find in our server store if any user right now is online of the

specific user I.D. because as we said before, one user can be locked into multiple devices so he will

have a different socket I.D. But the user I.D. of the user will be saved.

So we just need to find all of the connections.

So all of the socket IDs, if this user is logged in at the multiple devices and emit directly to every

dot socket connection amid this event, to every other single connection.

And as we have, we just need to track that and try catch block that if any error or cure will just

log this error in our console to know what is going on and why we have an error.

What you would like to do it.

First, we need to find the pending invitations.

So this invitations for a single user that he's awaiting the invitation, so how we would like to do

it, who we would like to go for the await.

And the invitation will take our front invitation motel and we can use find method that will find a

document in our collection where the receiver I.D. will be that user I.D. So we would like to send the

update to the right to the user, which was invited.

So if that user was invited, we need to update his list of pending invitations.

So we are looking for the all pending invitations where the receiver of this.

Invitation is this user I.D..

And now us, we have that.

What we can do with with that we can go and use that populate method.

And as you remember, every.

Let's go back and check our database.

As you remember, every friend's invitation has an feel to us as sender ID and the Receiver I.D. And

the populate method is given this functionality that if you only use depopulate method and specified

the field, which we would like to populate and you must be connected with the object I.D..

Instead of instead of returning database, the object ID of this user, Mungo's will look our MongoDB

will look for the user of this specific object ID, like if they see one and this one.

And instead of just this idea of this field, he will return all of these fields, which are the male

username and password.

It's like, then there it's all about this population, this field.

And that's the reason why we have been creating these refs in the models because we can use simple populate.

As you see, we have been using in models that ref.

And also in the front invitation that if will use their populate, will be able.

Instead of getting the idea of this, user will be able to get all of that details of him, which we

have in the database.

Let's go back to the friends of Charles in that updates folder, and now if I will populate this field,

not the idea would be returned, but the object with that data of this user and we can do it one more

thing we can add comma and pass the second parameter to that populate function, and we would like to

specify we can specify the fields which we would like to get that I would love.

I would look I would not like to get a passport of this user, but only the ID, the user name and the

mail.

So now this function will let us to find all of the different invitations where the receiver is this

user I.D. and will be able to get the details of this user.

We just sent this.

Invitation.

Now.

Second thing which we would like to do it is to find if you, sir.

Of.

Specified user I.D..

Has active connections.

Why we would like to do it because.

We just need to check in our stores state.

Which we have created before where we are keeping sort of a restore, where we are keeping the connected users.

If the user of the specific user I.D. so like depends of the account if this user is locked in.

But as we said before, this user can be locked in at the deep end devices.

So we will be checking in our map then to get all online connections.

And if this user is online only in this situation, he will receive the update of friends pending invitations if he's not online.

We'll just keep that update process.

And for that, we'll need to write some extra logic to get the users that all of the socket connections of the specific user, if he's if he has active connection and he's on the right now to do that in the

next video.

# Emitting Events to Clients of Specific ID

Let's go right now to our socket server and prepared the function, which will be responsible, not

the socket server, but the server store and prepare the function, which will be responsible to find

the online users of the specific user I.D..

And we can named up concert, get active connections, or you can use get online users how depends how

you would like to name it.

And who just will be receiving here that user ID, which we would like to check if this user is online

and now just to find an empty array of active connections.

And now that will be just an empty array, and now we'll take our map, which we have right here.

And let's go and use that map connected users dot for each.

So we'll go for every element in this map.

What will receive a function with the key?

Right here, what we will see for every element, the key and the value, as you remember, the value

is an object with the data and keys that socket I.D. of this user.

And now for it forever.

You sir, if value dot user I.D., the user ID of the connected user is the same for this one, which

would like to get online users.

We would like to take our active connections right?

And just push the key because key is there.

As you remember, the key is the socket ID and to emit and event to the client, we need to his socket

I.D. And if will get all of the socket connections, the socket IDs, which we would like to know who

should receive the update will be able to omit from the server and event to him, and he will get that

update about the pending invitations.

And now, as we have that like this, what we can do it, we can just return active connections.

What you would like to do, it also, of course, added to this model.

Experts get active connections.

Let's check if everything is fine.

We are accepting the user I.D. We are looking for the users, which they are online of the specific.

It.

And if that's correct, we are pushing his socket I.D. to active connections, and we are returning

an array of socket I.D. So we will be able to send the updates to the specific users.

So let's go to the Friends actress.

Now would be able to use that here, that function, which we have created.

Find if you sort of specified, you said it has active connections, it should be renamed that we are

looking for.

We are looking for find all active connections.

Of specific user I.D. And now, as we have that, we can go and create that receiver list and will go

for the server store dot get active connections.

This function, which we have created and will pass the user ID, which of the user, which we would

like to receive our update and as we have that we can right now emit.

To all of this users, which are right now on this list of this, that specific socket I.D. the update

about different invitations and how we can do it to Emmett and event to other users, we need the socket

object of active connection or AoE object.

So our the server instance, because only we've got this Io object we are able to emit and even to the

users, and we would like to use it right here in our updates and how we can do it.

To make that easy, that will be able to use our aerial object easily across all our server we can create

in the server store.

New field of the name of Iowa.

So here we can keep the reference to our socket server, and if we'll have that reference, we'll be

able to create the function, which will allow us to get that to get that socket server.

Now, if that's OK, et cetera, et cetera, we'll be able to send the updates.

Of course, if we have that, I'll admit.

In our set of store, we just need to create two functions.

Set socket, server instance like this where we will be accepting back that Io instance.

And now if you'll accept that we should go and just assign to Io Io, Easton's like this.

And now we fully set this.

We can go and also create the getter for this.

So we'll go for the get socket server instance.

And that will be just an oral function where we'll be returning this.

I'll.

Now we are seeing an error.

We don't know where is the air or exactly we just need to save all files.

OK?

Because it should be let I know.

And the initial value will set us.

No.

And now it should be fine.

Let's go back right now to our friends.

And now when we are in this file, what we like to do is right now we would like after we get the receiver

released, so we'll get all online users of the specific user I.D. What we can do it.

Let's get right now this Aoki stance because we've this EOD stand server instance will be able to emit.

And even so, this users will go for the server storage.

Don't get.

Socket.

Server instance.

Like this?

And now we just need to go back to the set of her story.

To the of star, because here we have created set soccer server instance and get socket server instance.

But we should add this at the end to have an access to the functions in the other files.

Now, if we look at this, it's fine because in their file friends.

We are able to get that socket server instance.

But the one thing which we are missing that in our server store, we are not setting anywhere.

This instance and where we can do it in this socket server, that will be the simplest way to do it.

And now after will create our server instance, IO server instance, here we can go and you server store

dot set socket server instance, because if we only keep right here the reference to our Io server,

so even if any.

If the server will be restarted on anything like this, if I will get a new reference that will be just

safe in our Star State.

And now what?

We are missing, of course, we are missing the server store and we just need to require this to make

this work.

And so we'll go for console servers, store and will just require and we'll just go for server store

like this.

Now we are able to set this instance of our I object and now in the French doctrine, yes, we have

that function of the French.

Pending invitations and as we receive, we have at least we have that IOC stance object that would be

able to wish that Emmet A. Evans and now we can go and go for the receiver list and for every element

in this list, we'll get the receiver.

Receiver socket I.D..

And now.

A.O. Scott, I'll you us possibility to Emmett and even only to specific users that will be not meeting

to Evans events to everyone, but only to this one of the specific socket idea which is connected right

now, and we can just omit that event to the receiver socket, i.e..

And now after this user, which is connected to our application, will receive and even it will be admitted

to him.

And if you are listening for this, even you will be able to catch it.

Catch this and that will be friends in the.

Stations like this that will be the name of our event.

And here we are, the next parameter to that emit.

We just need to pass the data.

We would like to send the pending invitations and that should be if the pending invitations exist,

will go for the pending invitations.

And if not, we'll just set it as an empty array.

And now that's our logic to send and update friends pending invitations.

And we can right now use it across our server.

And if the past only the user ID, we should receive the update, you just get the updated friends invitations.

And now, of course, at the end, what we need to do it.

Let's go and add their model exports.

And here we just need to pass the update.

France pending invitations.

OK, like this and where we would like to use it.

Of course, we need to use it in the place where we are sending the invitation.

So in the controllers?

Front invitation.

Here we have front invitation controller.

And Hugh, we are seeing that we are using this post invite.

So let's go to this post invite controller.

And now if we successfully send safe the new pending invitation in the database, the next step send

update set send pending invitations update to specific user and who should receive that update.

The users which they are the receiver of this invitation and we have created our friends update, which

we require in the next step and will go for this function, which we have created, which is update

friends pending invitations and who were just need to pass the user ID of this user, which should receive

that.

Update and the user should receive the update that should be target user dot I.D. and just use to string

like this.

And now this user of this specific I.D. will receive their updates.

And if he's online, if he's online under multiple devices like on the phone and computer or at the

multiple web dumps, every of this connection will receive that update.

We just need to require that the Friends update.

So let's go at the top to make sure that would be working, and we'll go for a concert friend, friends

updates.

And that will be required.

And we just need to find this file and is at the top level of our server.

Here we have that socket handlers inside of this.

We have the updates and we just go for the France fire.

And now we have right a lot of logic, which is quite complicated.

And in the next video?

We are not able to find the friends updates, but we have required that friends updates, let's copy

that pasted.

OK, now it's fine that I had a single typo and now we can check on our server and not on our server, but at the next video.

If everything is working that if we send an invitation to other user, does he, if he will receive that event?

But last thing which we can check if we have our friends update.

Let's go to that file.

And the name of the event, which will be a meeting, is different invitations.

Let's go to the client application in the source folder.

She retired realtime communication friends invitation and let check if the name of the events is the same or go to socket connections and as you see will be, every client application will be listening for the event of the name of friends invitations and here will be receiving invitations data so we can even lock console lock.

Friends invitations event came, and we let us log this data console lock pending invitations like this.

And now in the next video and check if our life update of pending invitations is working.

# Testing Real Updates of Friends Invitations

Let's test our friend's invitation systems, a real time updates.

Now what are we doing at the beginning?

I will restart my server with the comment of npm start.

Everything is fine.

We have to know them on which we have started that before, but I would like to just restart this manually

and now everything is looking fine.

Now, if I would like to restart the my reactor application I'm seeing and problem that.

In the.

That the story is not being defined.

So right now, we just need to find where is that problem exactly?

So let's go in our file of the socket connection dot J.S. in the realtime communication where we have

our discord front and up here we are using this third object to change the Star State, as we have been

doing in our components.

But in the components we have mapping the Star State to props or actions to props.

But we can use the star object directly to change our Star State.

And the reason why our app is not working right now because we just need to import the star.

And we don't have that unnecessary import.

And let's go and at this that we would like to import star.

From the store folder and the store file, and we need to check if that's the default export or it's

not to make sure that will be working.

So in the store, the chairs at the bottom we have as a default export store, so that should be working

as should be.

And now if we go back to the relaxed communication now, we'll have an access to that store object and

we don't have any error right now.

Let's refresh our application.

And now how we can check if our pending invitations are working.

We need to log in to two different accounts.

But if we just need to go for the privat top of the browser because if we not go for the private tap

will be automatically logged in as this first user.

So let's make this like this.

And now this user.

Here is the user test number one.

And now she's our log in for the different account, I just need to refresh that.

And I will go for test number two at Gmail dot com and now will enter the password, which is just the

password, because that's the account which I have created.

You can just register more accounts.

All depends how you would like to test.

Would be great if you would perform more tests if everything is fine now.

I will open the console because here, if this user will invite this user, the test number two will

invite the test, no test, no one will invite the test number to this.

You said it should receive the information to update from the server that about the new invitations

array of new pending invitations.

And later on, if she if you will receive that, will be able to render the real list of the pending

invitations and add this system to accept or reject that invitation.

Now how we can do it, let's go and just go for that, Fred.

But now I have a removed in database that every user don't have any friends is just empty, right?

So that invitation would be created successfully in our database.

This invitation is related with the user of the test number two and test number one and my private email,

so I should be able to send the invitation to this user because the invitation has not been sent and

they are not the friends.

Let's go OUTFRONT.

This simple error, which will correct later on and will go for test number two at.

Gmail.com.

Now, before we sent that, let's press it, we are getting the information that.

Friend of Test, no.

OK, because I just need to use test number two.

OK, now the correct email address of the second you said an hour sent this invitation.

Sorry, you cannot become the friends with ourself.

So probably I just need to log in to the other account, which is this test no.

One at Gmail dot com.

Now I'll go on set password log in.

And now this one is the U.S. number two, this one is the you number one. Once again, let's go and at.

Test.

Number two at Gmail dot com, let's send this invitation, and we are just receiving the invitation has been sent and the second user should receive the information from the server as this user is the

test number two about the pending invitations, but we are not seeing any message in the console. So in the next video, we'll fix that where we have an error that our server is not sending to us that real update.

# Fixing Bugs and Retesting

We have exactly two problems, which are why our pending invitation updates are not working correctly,

the first problem is that it will go to server start chairs.

One will have a function which is that get active connections.

We are using for a loop to iterate for every element in the connected users map.

But for each on the app, map returns as a function.

What are we can receive the key and the value which we are setting right here as a key value pairs.

But for each giving us the first, it's giving us the value and the US a second parameter is giving

us key.

So that's the reason why we have been not able to find the Active Connections Array because that was

switched from glee.

Now it's fine.

And the second problem is if we are testing our application at two Chrome Incognito tabs.

If we're looking at two different accounts, there is like a conflict that they shared some local storage,

even if they are two private windows.

And to test it to make sure that everything is working just we need to use another browser.

I will make sure that it will work out right here.

I will create a local storage.

And as the second browser to test our invitation updates, I will use the Firefox went.

I will go and I will open the new privat window.

Now also, I will go to the same page is locals whole three thousand logging, and no one will be able

to test my invitations.

Also, we can go to France Invitations Collection and refresh our collection.

And here if you have any invitations, you can just delighted them if we don't have a decision to send

the invitations once again.

So now every user can invite and every other user.

And now let's log in in their Chrome US test no.

One at Gmail dot com.

And here our goal just past the password.

Now I'm logging an acid test.

No one.

And in another browser, I will just log in as there's no queue at Gmail dot com.

And also here I will provide the password to log in.

And now, if I have two different browsers, I'm absolutely sure that they will not use the same local

storage.

So I will not have the conflict with the data which we have in our app memory on the single browser.

Now, of course, to check if we'll would be able to receive that update, let's open the developer

tools in that second.

Browser, which will be receiver of the invitation success, no.

Two will be receiving the invitation, it's quite small.

I can make that larger.

We are seeing the information that we have successfully connected.

So let's go and sense right now that invitation to this second user, I will go for the front.

Let's go for the test number two at Gmail dot com.

Now this invitation does not exist in our database, and if we send that invitation, the second user should receive the information, the update of the pending invitations and let's click send.

We are getting the invitation has been sent information and we open the second browser and then getting the information that friends invitations event came so this you should successfully receive the Real-Time update about the pending invitations.

And here we have an array of these invitations where I'm seeing that this user have only one friend invitation of the and the I.D. is this one which we are getting right now.

And we are seeing the sender details because we have used the populate method.

And here we have that idea of this user, that mail of this user would just send the invitation to us and the username as we just prepare that data on the server.

And now when we are receiving that pending invitations, we can in the next part, just ring around and there the real pending invitations from the other users and that second user will be able to make the decision if you would like to accept or reject that invitation.

# Rendering Real Pending Invitations List

We were able to see successfully this message on our browser that if one the users send the invitation

to the second one, he received the Real-Time update thanks to Socket Io about the pending France invitations.

What we would like to do, which right now we would like to take that information from our story because

here we are dispatching the ActionScript pending France invitations and sheer effects to that the action

helper will go to.

Friends read you, sir.

And if this action is being dispatched, we are changing our Star State and depending friends, invitations

will be not right now to empty area, but just an array with at least an array of the objects.

We just are the real spending invitations and we can take right now that pending friends invitations

and render the real spending invitations list.

And in our Friends sidebar, in our dashboard folder in React application here we have pending invitations

list.

As you remember, we have prepared the dummy invitations and now our invitations has the same format.

We have the field center idea, which is populated.

What do we have?

The user name and email?

And also, every invitation has a specific idea.

So now this dummy invitations will be no longer necessary because we'll be rendering the real list.

Oh, we can do it.

Of course we can do it as we have been doing that before by just mapping Star State to prop.

So let's go and define concept map store state to props.

And here it will be receiving all of the friends, and I would like to use that spirit operator to all

of the fields from the Friends producer who will be available under the props like this.

Of course we need to use.

Import to connect, as we have been doing before from React Redux, to use our Star State in their component from the Props React Redux like this?

And now as we have that, let me use that connect function properties, mobster state.

Two props with that connect function, and here we just need to use the braces, and now we are not mapping the actions we don't need to pass and no, because the first parameter is being provided, we don't need to pass the second one because it's optional in the connect function.

And now.

What we can do it if we are mapping Star State for props now, this values which we have in the France producer, will be available under our props, which we are getting into a component.

So now in our pending invitations list, we can just render the real invitations, which is pending in France invitations.

So now will not have our damndest, but the real data and we can take this spending frenzy invitations, which we'll be getting from our store and just instead of rendering the dumb invitations would be rendering the real pending friends invitations.

But one thing which I would like to do it now, we are only able to get that data if any user will send to us invitation, but also if the user will establish a connection with Circuit Io.

So this user will be online at the beginning when she's first going online.

We should just also send him this list of pending friends invitations and also the list of the friends which would be doing later on.

And now we can in the next video will add the logic.

Just if the user will connect to socket aisle, he will receive the list of pending friends invitations and also he will be receiving the same event if any user will send to him the invitation.

So he will have the real time information.

What is going on with his friends invitations list?

# Fixing Bug With Pending Invitations Rendering

After we have made that changes to render the real pending France invitation list, our dashboard stopped

working and for a reason is that that we don't have an access to pending France invitations because

our friends producer is not being used.

So if we go to the store to trace file in our store folder in the main React application, here we have

that French producer.

And to have access to that feels that the values with which we have the state right here.

We just need to go to our store to trace and ask.

We have been doing with type of producer and other producer.

We just need to add this producer to the combine producers function and will go just for the import.

Friends review, sir.

From and that would be the same folder, producers, folder and friends read user.

Now if we'll take this, we'll just pass that France read you sort of like this and we can save it.

Let's go back to our application.

Let's refresh that and let's check if we'll be able to see any response.

Still, we are getting an error, probably, let's check what we have in the terminal.

Everything is fine right here, but maybe if we have at the ready to serve, we just need to restart

our application.

Which the npm start and controversy just to stop our application, and now let's check if that will

be working.

We just need to wait for a while when our application will be restarted, and still we have an error

that this one process is not defined, that is to react back.

And what we have right here cannot reach properties of undefined reading map.

So still, we don't have an access to that friend's producer because here we should go just for friends because we use the Friends producer.

But in the file of the pending invitation list, as you see, we are using this under the name of the friends.

So this name must be the same, which we are defining right here.

And we're just using that producer, French producer.

And now if we open our application, let's refresh that.

And we are see, we are able to see our dashboards once again and everything is working.

And now we are not able to sit and invitations because now invitations has been sent by the server.

So we are just rendering that empty array, which is right here as empty array as a default value.

And the next video will take care.

If the user will log in first time, she will just connect with the our socket IO server will just send

him at the beginning the list of the pending friends invitations, so he will be able to see who really invited him to different.

# Initial Update Of Pending Invitations

How we can get at the beginning of the first connection with socket Io, the real released of the pending

friends invitations, if we will go to our.

Socket connection server at the back end of our application of the server in order not to hear the socket

connection, but on our server will go for the socket server right here in the Discord backend folder

and hearing the Sockets server.

We are on connection.

We have that new connection hand letter, which is being executed if you server connect the first time.

And if we'll go right now to this new connection handler, we can use that update, which we have created

before that after the first connection was sent to this user.

His pending invitations?

And we'll use this new connection handler, and after we at the new connected to you, Margolis, this user, as he's online, we can just go and update.

Thank.

Invitation invitation list pending friend invitation.

Prince, invitations released like this.

And now we'll just go for the Friends update, which will need to require to have an access to that functions and will go for the update.

Friends pending invitations.

Because that's the name update for pending invitations, which we have in different outrageous file in the update folder, as you see, and it will be using that function.

And what we need to do it if that user would, this specific so-called connection would like to receive that update of the pending invitations.

We just need to specify the user I.D. of this user because that's the function, how we have prepared that it's accepting the user I.D. It's looking for all of the online users with this specific user I.D.

The client instances the open applications and now is just sending to every single that you said that the new pending invitations, which are being added from database and well from where we can get that user ID, of course, from the user details, because user was able to get that handler to get his request was able to go to that even to new connection handler if he has a valid gravity token.

And we know that in the decoded JVP token under the socket user, which we are using in the middle of the socket, we are able to get the user ID from the user details the user I.D. like this.

And now we've discussed it will connect first time to our React application.

He should receive the pending invitations and that's tried this in the next video that will be working.

# Testing Initial Invitations

Before will test our initial update of pending invitations.

Of course, we are missing one thing.

We just need to require that friends updates because without this, that will be not working in the

new connection handler as we have been working before.

In the previous video, we'll just go for the Friends update, and we will just require this from the

update folder, from the socket handlers update.

And here we have that friends.

Now, if the safeties this them on will be restarted and we can test if that will be working.

I will open my application, and to now, as I remember, I have sent the invitation from test number

one to test no to account and now test number two at Gmail dot com.

Should have an invitation from the test.

No one.

Let's log in.

Now we have the budget request, something from that number to add invalid credentials are addressed

to right once again, the passport.

Logging and now everything is fine, and as you see.

At the first connection, we just got that initial update or friends invitations.

And here we have got all the array from the server of the pending invitations with the sender details

who sent this invitation.

And now it was automatically rendered on the list of the invitations.

And I'm seeing that when I am logged in.

As at this number two, the invitation came from test number one.

And here right now, I have this decision bottoms what I would like to do with that invitation.

What we can do it.

Also, let's open the new private account into another browser.

Let's go and open.

Our applications localhost three thousand and have no exact three, which are composite soldiers right now logging.

As a mark at Gmail dot com as my deferred account, which I have registered, and if I will log in right now, I will try to add this test number two and we'll see if the test number two is also seeing the invitation from this account.

So let's go and go for the upfront, and I will just write test number two at Gmail dot com.

Let's say send this invitation.

And as you see, we have that real time date which we have done before that we are able if the another user will send the invitation, we have that don't in the real time.

So it's working perfectly, if you will just hover with our mobile.

We are seeing the email of this user.

We just invited us and here we have that decision batons.

So in the next step, as we have that.

Real time invitations that we are able to see them right now after every user.

It's time to get the user possibility to make the decision if you would like to accept the invitation

and not if not if all of this functionality of the accepting and rejecting the invitations will just go and do it the same, like the initial update of the friends when we first will log in.

And if any user will accept or reject the invitation, if that you served, for example, accept that invitation automatically, she should see this user just number one.

That's number one of the friends list and the other user.

If he's online, he should get that same information.

So also, we'll have that realtime functionality and it will be doing that in the next videos.

But we will start with the decision buttons to allow the user to make the decision if you would like, accept or reject the invitation.

# Preparing Client Side To Accept Or Reject Friend Invitation

Let's add this functionality of make the decision, if you would like to accept or reject the invitation,

and as you see, we have that two button right here, so let's find them in the code.

The name of this file was there, the invitation decision buttons.

And that's exactly in the dashboard folder in our React application friends sidebar pending invitations

lists.

And here we have that file of the name invitation decision buttons.

Here we have two buttons where we are accepting USB props, accept invitation handler and reject invitation

handle, and it's coming from the parent component.

Which is the pending invitations list item, and here we have got two handlers, which we are passing

right here to this button and we are just executing that handlers.

I will press that button here what we are seeing that if we'll just accept the invitation will executing

the accept front invitation function, which is right now to empty our function.

And if you said we press ahead with the invitation, here we are executing same as a reject friend invitation

to.

So right now, we just need to create that actions and make it HDP.

Call to our server that if we would like to accept or reject that invitation and we can do it, that

as we have been doing that before.

So let's import Connect from React Redox, and we don't have created that actions yet, but we'll create

them under this name in the Friends Actions file and after we'll have that.

Connect, we also need to go and import get actions from let's go find this to the top level of our

application.

React will go for this story.

And here will be going for the actions folder.

And as you see, we have different actions and now we are able to connect these actions to this component.

So we will create these actions will have an access to them in this component.

Here, as we have been doing before, mop up actions to perhaps that will be the name of the functions

which we would like to use.

And here, let's just return.

Of course, this function is given us the dispatch, which allow us to change our Star State and we

will just reduce the spread operator to get all of the actions and we will just pass this dispatch object.

Let's offer up our component.

First, we need to pass.

And the second we are just need to pass this map.

ActionScript props because the first one is responsible for mapping Star State to props, which we have

in our reducers, but we are not interested to hear about this.

And now, as we have closed this now you seem to be working correctly, but we have not created that

actions, which is to accept French invitation and reject French invitation.

And we are mapping this, so let's create and do with it right now.

What is the name I will check this once again map actions for props.

And here we should not export default map access to props, but here we should use this connect function,

which we have imported from React Redux.

So that's the reason why I was looking like this and something was wrong for me.

Let's go to the transactions.

And are these actions right here because here we have Typekit actions, which we are using.

So now what?

We just need to do it.

The name of it was accept.

Friend invitation.

And here we are accepting the data, which is coming in right here to this invitation.

To that function out of function, and here we're just using the dispatch object, and we would like

to dispatch accept friend invitation and he will be just passing these data.

Also, we would like to go for the reject friend invitation and to pass the data and here will dispatch

or reject the friend.

Invitation also will pass the data like this.

Let's create that two options right now, accept and reject the invitation.

So let's go and create accept.

A friendly invitation letter will be accepting data and in the similar pattern would be just doing the

same.

Let's keep it up here.

We'll have to reject.

And now what we'd like to do it, of course, that will be the async task, so we just need to return

to our old function.

Thanks to Redux Funk, which is giving us possibility to make the API calls and async task in our actions,

which would like to dispatch.

And now we just need to wait for that response.

Await API Dot sent not able to go for the accept front invitation.

Of course, we have not created that function.

IPA file accept the invitation to it in the next step and of course, as we have been doing before.

Let us copy that.

Pasted right here.

If we will get an error, we'll just open up a message with a specific message which you just got from

the server and the accused.

If not, nothing will happen.

And here we'll just open the alert message.

The invitation has been sent.

We'll just go for the invitation accepted.

And just not execute across the aisle country because we are not accepting here to this function.

Close dialogue, candor, because it's not related with that.

Now let's copy all of that written statement.

Let's paste it and now we'll go right here.

And who would like to reject that invitation?

And if any error or cure would be not able?

You just got that message and here invitation rejected.

Or we can go to declined it.

It just depends on you how you would like to show it.

So we have to accept an invitation reject for an invitation.

We just need to create that API calls.

App functions in API and here everything is fine.

So let's go to API Dot Jayate.

And now, as we have sent front invitations, dance our secure routes where we are checking the response

quote, and now we are to their new export.

Konst, accept friend.

Invitation.

And now that will be an async task where we'll be accepting the data and we'll try to return await.

API client, as we have been doing before, and that will be the post which will created on our server and will go for the friend invitation and here will go for the accept like this and will just pass this data and this data is that invitation ID.

And we would like to catch and error if an error will occur.

So we're just the exception.

If an exception will be thrown so far, for example, will get 400 status and will go to check the response code.

And if the response calls would be four or one or four free will be automatically logout because it will mean that our gravity token is not valid anymore and just throw error through.

And we'll just attach this exception to this response.

We'll be able to show that later on that dialog message.

And now what I'm seeing ever finished looking fine, I can copy this right now and create similar action for reject front invitation here will be not accept the invitation, but reject here friend invitation, because here we have this same path with which we have created before friend invitation, invite friends, invitation, accept, friend invitation, reject like this.

And now our API calls are ready so we can just if we'll go right here directly, we can test this in our application.

Let's open our up where we have the pending invitation and if we'll press and then button accept, we are seeing that this request has been performed to the server, but this path of their friendly invitation accept has been not found.

We have just the error, which is being sent by the judge said by the browser.

We, of course, that will be our custom error will send that from our server.

Of course, the buttons are blocked after the one decision we have made, but this is given us the information that we are able to reach this API HDP request.

So we just go in through their friends actions because we have we are using them in our component where we are making the decision and here it's going to the API to trace fire and we are just making that request.

And the next part, let's start working on our server to just add this feature to add, to accept or reject the invitations.

# Preparing Decision Controllers

Let's go right now to our set socket set of error, which we have created not to the socket server,

but the server that chips in our dish called back in the folder, as you remember when we have been

creating a different invitation.

Request also, we prepared our friend invitation routes, and that would have been the root folder here

in the front invitation route.

As we have that slash invite, we can add two new routes and that will be just brutal post where we'll

go for the path of accept.

And now, if the user will accept the invitation, of course we need to check the JV.

If the JVP token is valid with our middleware custom, which we have created and will go and we validate

their body, which is coming in this request, so that will be in the invite decision schema, which

will create.

And at the end, if everything will be fine and will pass all of that middleware, the last one will

be just to handle our response, our request, which is coming from the client side and will just go

to different invitations.

Controllers will go for their controllers as before.

Same pattern right here and we'll just go for post accept.

Now what we are missing, of course, we are missing invite decision schema because we would like to

validate the data which is coming in this request and that will be Troy object.

What we will have that object in sight of the data, then who will be validating only the invitation.

It will be coming as a body of this request.

So we'll go just for the idea, as said Joy.

Don't string.

And that will be required that the idea of the invitation must be provided in the body so that check

this once again, we have to accept proof of middleware.

To check that out can validate the body of the request.

The body which is coming in the request with the invite decision schema, then the F11 is fine.

We'll pass it to the Post Accept controller, which we have not created yet.

And let's do it the same and we'll go here just for the post reject.

And what we don't have also right here is we just need to send the information right here with the reject

like this.

So we have accept, reject everything is looking fine.

Now we just need to go to the controllers for their two front invitation.

And here we have that friend invitation controllers.

And now we would like to hear we have the post invite and we need to create the post, accept, both

reject, so we'll go for the post, accept and post reject.

Let's require them up the him to not have the program as we had to before post accept all the required

to file, which will create of the name of post accept.

And also, let's go for that post reject and will go require and we'll just go for post reject like

this.

And now in the front invitation folder, let's create a new file of the name of Post Accept Dot Trace

and also the new file of the name of post reject.

Dot Trace.

And now as we have that.

Let's write just an simple response to check if we'll be able to reach this route.

So we just need to copy this post, accept we'll be looking like this.

Post Accept.

And of course, let's add the export, not export.

We just need to go the same pattern which we have in the post in right.

We just export the model which will create and will go for model, export, post, accept.

To use this file in the older files, this post accept function.

Our handler and now we can copy that.

Go to the post, reject, pasted and change.

It is only for post reject.

And here let's go for the post reject.

And now let's just make a simple response, and we'll just go for that return response object, which

we are getting by the Express to send their responses to the client and reject Handler will just send

in information that we were able to reach their reject handler and here will just return in the post,

accept their response, which will be return response sent and the message will be Accept handler like

this?

Let's save it, and now I'm seeing the information that my server crashed, but I need to find what's

the reason for that?

Cannot find the model post accept.

Here we have the post accept.

And the name of the file is.

This same friend, invitation controllers, post, accept, model exports, post, accept.

More posts reject.

Now we just need to find where the problem is, exactly.

The reason for that why it's not working is very simple error that in the front invitation controls,

we are just missing dot because here we are specifying the relative path.

And now if you'll just go with the DOT, let's save it and everything is working.

US should be.

Let's go back to our application.

Refresh it that.

You, sir, would have just invitation, let's press one of the buttons accept, and now let's check

the invitation, validate the error validating body.

And now we just got the information from the server that the idea has not been provided, but we are

validating our schema for this.

So our request right now failed at the level of this validation that it is required.

So we just need to provide this idea in this request where we accept or reject that friend invitation.

Let's go to our decision buttons, invite decision bottoms in our React application in the pending invitation

list.

Here we have that decision bottoms.

Let's go to the pending invitation list item.

What do we have that handlers?

And now, as you see, we are providing an IED in the pending invitation list item asset data and why

it's not working.

Let's chuck and pending invitation list.

And here we are marking the I.D. as an invitation.

Got I.D..

So before we press the button?

Go to that pending invitation list item.

Let's check that idea, which we are sending.

So let's just simple, if anyone press this button, let's look at this I.D. Let's copy that pasted

and see if this I.D. is being attached to the request, but also we can do this in the network top.

Let's refresh our application.

I will pass the baton here, I'm seeing the accept.

And I'm seeing that I have sent the idea in their request.

And now let's go back to our validation, why it's not working.

Where we have our custom routes.

So that's exactly the friend invitation routes and why our customer validation is not working.

The reason for that that we are getting that information from the server is that probably the ID should

be as strong as you see.

We are attaching this to the request, but the required should be used as a function like this.

And now, if without this braces that we like to execute that now, everything should be fine and we'll

be able to pass this validation, which is common with joy validation.

Now let's go back to our browser.

Let's refresh the application now.

Let's make the decision.

Server has been automatically started because we have other 10 of them on.

And now, as you see, we are getting the information invitation accepted.

These buttons are being blocked after we have made the decision because later on, that will disappear and this user will be added as a friend or not.

If we press the reject button and here if will press invitation accepted once again, let's refresh.

Let's press to reject an invitation rejected.

So we are able to reach our controls, and that's perfect because we can write our logic right now to make that accept or reject.

# Creating Logic Of Rejecting Invitation

Let's work right now on the Post Reject Control Letter, which we have in the dish called backhand controllers

folder friend invitation, and here we have the post reject.

If our request is able to reach our Post Reject controller, we would like to add the try catch block

this one we would like to remove.

And now if we'll try to do it something, let's catch an error.

And if an error occured, let's just console log this error soul able to see in the server logs what's

going on and who just returned the response of return response status of 500.

So if something happened, well, that will be our server error.

And let's go up to something went wrong.

Please try again like this.

Now, if we have our try here here, we would like to write our logic, we would like to get this idea,

which is being sent from the React application.

And this will be coming from the request body so we can use the object destruction to get this idea.

Also, we are able to get that user I.D. We have just sent this request and of course, we will get

that from the decoded gravity token.

Now what we like to do, it would like to remove that invitation from any friend invitation collections

collection like this.

So how we can do it will go for concert invitation exists now.

First, we will check if that invitation exists.

So will go just for the invitation exists and that will be a weight friend invitation.

Dot exists because on the front invitation model, which allows us to communicate with the DBE, we

have the function of the name exist, which is returning the true or false depends if this document

exists in our database.

Of course, we need to use the icons friend.

Invitation Motel require this motel and to just require this from the folder of their.

Models and here we have that friend invitation.

Now, as we have that, we can check if this user exists and we can find this invitation.

Not the user, but if that invitation exists, that by specifying the fields, we would like to find

this document.

And as we have the idea of the invitation, we can find this by the idea.

But as you remember, the idea of the documents in the MongoDB database are specified in this pattern.

Now, if we have that, we call and it will check if this invitation exists.

We would like to await it when this invitation, it would be removed.

Fined by I.D..

And delete like this.

And now, if this invitation will be delete because it was rejected, so we just want to remove that

invitation from the database.

What we'd like to do it.

We would like just to update the list of pending invitations at the user.

Which just making that decision.

So we'll use our custom logic, which we have created a date pending invitations.

And if there one or more, if more than one reject application, it will be locked into the same account.

All of the online users will get that information that the invitation has been rejected.

So we'll go for the friends updates, which we have created and will go for the update.

Friends pending invitations and here will pass this user ID.

We just make this decision.

If we accept that, if we'll accept the invitation, of course we need to update the list of the friends,

so we'll need to create the update.

Friends update friends up the handler, which will be doing in the next video that to have that real

time friends list.

And as we have that.

Now we just need to require this Friends update.

But after that will be successful, we don't we will not get any error.

We would like to just use our.

We would like to return the response with the response status of 200, if that will be successful.

And let just send the information invitation successfully rejected like this.

And now, as we have that, of course, we need to require this friends updates file to use it solely

to call and use it as this conference friends updates and to just need to require that this from there,

not that's not the correct name of the path.

But we just go to the top level of our application where we have our socket server handlers.

Here we have the updates and here we have the friends file responsible for the France updates.

And now this will pass the user ID of the user, which would receive that update and everything is looking fine.

Now we can in the next video, we can take test this logic.

If we reject the invitation, that invitation will disappear from our database and from this user, which is making this decision screen.

# Testing Rejection Of Friend Invitation

Before we test the decision to reject of the invitation, we have one typo here we are going and awaiting

invitation, fined by and delayed and of course, that should be different invitation because that's the name of our model.

Now let's open our database where we have the invitations and now invite React application.

I have only one invitation because the second one which I had, I just removed, and now I'm seeing

that the user of the name of this number to have one invitation from my private email.

And I'm also seeing that information, the database.

Let's press right now to reject the baton, and let's check what will happen after I have pressed his button.

I'm getting the information that the invitation rejected.

The invitation disappeared and once again we can invite each other and also.

That invitation has been removed from the database because we don't have any documents in different invitations collection.

So our custom logic to reject the invitation is working correctly.

In the next step, let's create the Post Accept controller where it will be just have functionality

to accept the invitation.

# Accepting Friend Invitation Logic

Let's add in our post accept at the beginning, try catch block rats, remove that response, which

we had before.

And what we would like to do with who would like to try to execute some logic and if any error with

or cure.

We would like to send that.

Just log this error at our server side.

So let's go and just add console.log error and we'll return the risk.

The response we have the status of 500, which will mean for the internal server error and we'll just

go for this will send that something went wrong.

Please try again later.

Or just please try again, like this will be fine.

And now look at this.

The Logic in our tribe blog.

And what we have right here.

We just need to get the idea of the invitation request body.

And also, we'll start with getting this front invitation from a white friend invitation signed by IDA.

This model also mungo's model given us possibility to execute the function of the named fine by I.D.

where will be not providing the idea like this that they're under?

Line I.D. I.D., because who you will execute, find my I.D., we are just passing the ID of the document

I would like to find.

Of course, what we need to do it.

We need to require this motel.

So let's go as a concert friend invitation require.

And let's go directly to motels folder.

And here we have that front invitation.

What we like to do it, if we have that, if that invitation does not exist, we would like to return

the response to the.

That.

With the status of four 01, which is which mean not found and will go for the error or cured.

Please try again to make the decision if you would like to accept or reject the invitation.

Now, if we have successfully found this invitation friendly invitation, as you remember in the front

invitation, we have Sender I.D. and the receiver ID I.D. of the user, we just sent this invitation

and the ID of this user, which should receive that invitation.

So in our post, accept we can get that two fields from that invitation document, which we have found

from the database and will go like this, that we will get the sender ID and the receiver I.D. and that

will be just invitation.

We'll be getting there from this invitation.

Now, if we have successfully found this invitation and that was accepted by the user, this users should

be their friends, each other, how we can do it, let's write the comment that we should add friends

to both users.

And now, as we have ID of the user, we just send this invitation and the user just received the invitation,

we can find this user so we can go in this simple way, ask the user one will be quite user.

Fined by I.D. and will pass the I.D. of the user, we just sent this invitation, of course, we need

to require that the model which will be user require.

And let's find this model and that will be at the top level of our server and use user file user model.

Of course, we right now need to go and take the User1 Friends era, because as you remember in the

user model, we have an ardent friends array, which is just an empty array, and we can even add the

validation that we would like to use the assign the new friends right to you, to this user friends

where we'll take all of their prévues friends, which she had.

So that would be user one friends and here will pass.

Here we have been looking this user for by the sender ID.

So here we should pass the receiver ID that this user should be front with this user, which received

the invitation.

So this one is the.

You said we just send the invitation so we can even name that better to make that easier.

Maybe like a sender.

Sender user and receiver user, so sender user user, we just send the invitation should at the front,

which is the receiver of that invitation, but as you remember, the user friends array is storing the

ideas of the users and later on we can get we have the information about the ideas of this user, which

we are friends.

So if we have that right here, we can do it one more thing and we'll go for the user receiver user.

So that will be the user.

We just received the invitation and we'll go further away to user find by ID and it will go for the

receiver ID.

And now, if this receiver user has been found, we'll go for that receiver user, dot friends and will

just the spread operator.

And here we should go for the sender user because we have changed the name and here will use the receiver

user previous friends and will add the sender I.D. to the array of the friends.

And now, if we are just changing the field of this, you said we just we got from the database.

You need to perform safe method on this models to safeguard in our database.

Let's go and use that await user one dot safe like this.

And the second user, not the user one, but the sender user safe and also a wide receiver user thought

safe.

So the new list of their friends will be saved in their database.

Now.

After this, if you at this friends each other, we would like to delete the.

We would like just to delete this invitation, so let's go and just write a comment to delete.

Invitation.

After we have added this, user of the US, the France, each others who will go for a wait because

it has the async task and will go friend invitation delete by.

Fined by it.

That's the method which is given by Mungo's to work on the database model and to go find my I.D. and

delete.

And here we just need to pass the idea of the invitation, which we would like to delete.

So this is the idea which is coming from the at the request board.

The last thing which we don't have, we should update a list of their friends if their users are allowed

to update the list of their friends, if the users are online.

So the same which we have been doing with the pending invitations, but will of this logic.

So we should do it at the both sites.

But we need to do it one more thing, this user, one user, which was making the decision, he should

update their list, update a list of pending invitations.

Update list of friend friends pending invitations, because here that invitation has been removed,

so he should not see anymore this invitation because he made the decision and how we can do it, we

can go for the Friends update.

Dot update, friends pending invitation.

And that should be the user.

We trust the user ID, which just sent this invitation.

So he was the receiver of the invitation so we can take the user I.D. from the token or go just directly

and pass the receiver.

Be here because the receiver of this invitation should update a list of the pending invitation so the

receiver I.D. to string like this.

And now we have the logic to update France pending invitations and after that process will be successful,

return the response, return response with the status of 200 and just send the information that Trent.

Successfully.

Added, Like this now we in the next video, we need to create that real time.

Updates related with the Friends list that every user, every user will receive the.

Actual list of the friends, which we have in the database for later on, he will be able to exchange

with this user, the messages.

But as you see, our custom logic is created.

So let's test in the next video if we will be able to accept different invitation or if any error will

be thrown.

As you see here, we have that send the user and the receiver user friends array.

But one account in my users has been created before we have added the friends.

All right.

So my private email is missing the fill, the friends.

So that would be good to delete this user without the field of the name of friends or even clear the

database with all of the users registered the new two accounts and see if everything is working correctly.

We can check that in the next video.

# Testing Friend Invitation Acceptation

Let's test.

Accepting the invitation from the older user, one thing which we are missing in our quote in the post

accepted JS file is this their friends update because we are using them, but they are not required

at the top of our control or control error.

So we just need to go and to require different updates and we can find them at the top level of our

server in the socket handlers.

And here we have that update folder with dear friends and now everything should be fine.

How we can test it before I will test it, I will go to my database and I will just drop the collection

of different invitations, friend invitations.

Because before I will be testing that functionality with the friends, I would like to clear my database.

Also, I will drop the users and I will register the new accounts.

So after that has been done, I will go to localhost three thousand logging and I will go forward to

create a new account.

Let's go off the test.

No one at the Gmail dot com.

The username will be test number one and the password will be just that password and that will register

as this user.

This one is the list of the friends will exchange soon.

Now I will open the new private window where I will go also for localhost three thousand log in and

how to register a new account, which will be the test number two.

At Gmail, dot com username will be test number two, and we will just use the password after we have

been using in the previous account.

So now I have two accounts registered and in the database, if I will refer the collections.

Now I'm seeing the users and every user has friends array, which is empty now in this browser.

I had the user number one, so I will go and I will add the friend, which is test number two at Gmail

dot com.

That will send this invitation.

I'm getting the information that the invitation has been sent.

And the second user is able to see that invitation.

And now let's accept the invitation.

Now we are getting the information invitation accepted, this invitation disappeared because the life

update was performed by our friends updates and we are not able to see anymore dear friend pending invitations.

Also at the second account, we are also not seeing that.

So now here we should refresh our database once again.

And here we have that users users collection.

And here in sight of this, we have to document and now we are seeing that this user of this specific ID is is the friend with the second user, so they are the friends each other because this idea is being contained in the friends array, so their friends invitation system is working correctly.

So what we can do with right now create that on the initial update, like the same with the pending invitations will get the real list of their friends on the left and later on after will.

At this will just take care about the online indicator that will inform which users are online right now.

# Preapring Client Side For Friends Updates

Let us prepare right now our React application for the live update of the friends.

How we can do it.

We'll go to the socket connection file in our reactor application in the source as a Discord front end

source folder.

And here we have our Real-Time Connect communication folder, where we have that socket connection file.

Here we can remove that on already because that future is working correctly.

So remove that two console locks and what we are seeing right here that we have.

We are listening for the even friends invitations.

And also let's listen for the event of the name of.

Friends list, and that will be very similar to this one, which we have been doing with the friends

invitations and what we'd like to do it right here.

We would like to get their friends, concert friends and they should came in the data.

And now if we get that friends from the data, we'll go and we'll just use the store dispatch set.

Friends and will pass this friends, which we are getting from the server.

Of course, we have not created that set of transactions, and this set of transactions will be coming

from the Friends Action Folder file.

So we'll just use it as this set of friends, and let's go to the friends auctions and create that helper

function to dispatch the.

Store dispatch, we just need to dispatch this action.

Which is set friends and will do it right now in the star friends actions and here where we have set

pending friends invitations here will add the export concert, said friends.

And here will be accepting the friends.

And of course, we'll just return the type of their function, type of the action which will like to

dispatch and the type will be set France, which we have created before.

So that will be different actions.

Dot set friends and who will just pass this friends.

Let's check.

Our producer is everyone is find that if this action will be dispatched, said Friends.

Let's go to Friends producer, and now we have an action of the name, front action set friends and

our stories state will be changed for this friends, which they are coming from the server.

So the one demographic which we can do it.

Let's go right to now exactly where we would like to go is the friends list.

So in the discord front and in the source for their dashboard, friends list friends list right here.

We have got dummy friends, but here we will be right now using their real friends, which they are

coming from the server.

So let's go and use just the friends right here, which will be getting from the props so we can use

the object object structure structuring right here that will directly get that from the props.

And of course, we need to map our Star State to have an access to our store, our components, so we

just need to use connect from.

React.

Products like this, we just need to remove that dummy friends right now because there will be no necessary

now as we have thought.

What we'd like to do it right now.

We would like also to just write the function, which we have been doing before map stores state to

props.

We know that we can access our store state from the Friends producer by mapping this to the props here.

We can use the friend's name because in the store dot trace file, we are typing that under the name

of friends will have an access to their fields, which they are in the French producer.

So let's go back to our fire, which is the friends list.

And here what we would like to do it.

We would like to just return that friends with the spread operator and the French producer, we have

the France field.

So the name will be the same.

So we'll get this friends.

And now we'll be able to map through them and render the list of the real friends.

Of course, we need to connect this component to this function, which we have created.

Connect.

Map story state two props.

And here we have that frenzy, which like this.

Now, if we have created that, let's check if our app will be working, let's refresh that right now.

And now I'm not seeing any friends, but it's looking fine that everyone is like, we have some error

because we are not able to press anything at check.

What we have in the console front is not defined in the socket connection.

As you see, we have an error in the socket connection file.

So let's go to socket connection here where we are listening for the event, and we would like to set

friends with the friends value which they are coming from the data.

It's in the front, in the React app, in the realtime communication folder, in the socket connection

dot J.S..

Now let's refresh our up and now everything is working.

This one will clear later on and now everyone is working.

US should be.

We are able to get the empty array of the friends from the star because warping mapping for the our list of the friends, which is the empty array.

And now we're at our logic like the same with the pending invitations that on the initial connection and any invitation will be accepted.

Our list of our friends will be accepted.

So later on, we will be able to choose one of our friends and exchange with him the messages.

# Creating Server Logic For Real Friends Updates

Let's go right now to our server, to the updates folder on our server side.

And it's exactly in the so-called handler's updates, and here we have that friends thought jet.

And what we would like to do it right here.

We would like to create the function, which will be not responsible to update their friends pending

invitations, but to update their friends, so we'll go just and write their concert, update friends

and that will be an async task where we'll be accepting the user ID.

Now.

We ought to try catch block that will try to execute our logic.

And if we catch an error, we'll just log this error to us.

All developers will know what is going on now.

As we have that, would you like to do it?

Let's.

At the beginning, I find this user which should receive the update from the database, also just as

the user await user dot find by I.D..

And now, if this user, we just need to pass the ID of this, you said we would like to find.

And as the next parameter, when we are using defined by ID, we can specify the fields which we only

would like to get from this document.

And I will said that I would like to get on the ID and different array.

Now we can use that populate method, as we have been using before.

And now what we would like to populate will take and will populate their friends, and we know that

we have in the friends area, we have the object ideas of other users, so we will populate this array

will not get the ideas, but we will get a full object of this users.

But I don't want to get like a password of this users so we can specify that the array of the friends

should include the name of this user, the ID and also the email.

So we'll go for the ID, username and email because that's the fields which we have in our database,

in the user document.

Now, of course, we need to get this user from the models, and it's already done with their friends

pending invitations.

Now, as we have that, we can check if this user has been found and if the user has been found.

What we will do it right now.

We will just prepare and a friends list and how we can do it will just map, let's go and prepare the

friends list where we'll just map through the.

Iterate through all of the elements in the user friends.

All right, well, we'll get every friend and we'll just prepare the object, which we'll have in our

friends list and this object forever, friend for every friend in the user friends.

We would like to return the idea of this front.

So will be the idea.

Would it be just a friend?

I.D. The Mail will be front dot mail and the username will be.

Friend dot username.

We are iterating using the F.

So we just need to use this F if one or to get the values from the user friends for every friend.

And now as we have that, we'll have our friends list because we are mapping through the all of the

friends of this user.

And now what we would like to do it find.

Active connections.

Of specific I.D. So it's going with the online users as we have been doing this and with the pending

invitations.

Now if we are doing that, we can go and specify the receiver list where we'll go to the server store,

get active connections and who pass this user I.D..

Now what we can change right here now.

Everything is fine.

And let's go.

And get the IOC stance, too.

So we'll be able to emit and event to this online users, as we have been doing right here.

So we just need to get Io Io server in stocks.

And now if we have that Io server instance from our storage, so let's go Io server store dot get socket

server instance as before now.

We should go for the rest of our list for each.

And here we'll pass that receiver.

And now for every receiver.

So that exactly will be the receiver I because that will be socket ideas of this user as the same like

here.

So we can even use the same, which is receiver socket I.D. And to every user which is locked in, we

will go to every of the every user of the specific ID.

So we'll go to the Will Emmitt and event to their receiver socket I.D. And the name of the event will

be Friends list.

And you'll just pass that data, and now it will be as simple as friends and even the friends list exist,

which we just got from our database and prepared with the map function will send their friends list

and if not, we'll send an empty array so that I can find extra validation right here just before we

send that event to the client.

One thing which we can change.

He's like, here we are checking if any of this user is online and only this user user should receive

if he is online.

So what?

I think that should be at the top level because we, if we don't have any active connections of this

specific user ID, is not necessary to execute all of the rest of the logic, like checking if the user

exists in the database and preparing data because there are several list will be empty because this

user is not online.

So I think it will be better to take this aggressive list of the top.

So just before where we are checking if the user exist will look at one more if.

And now.

Just placed it at the top.

So here we are, getting the receiver list.

And now, if the are severe, least doubt length will be larger than zero will execute all of that logic,

which we have right here.

Now, let's save it and should be fine.

So now we can do it the same also right here with the updated friends spending invitation.

But I left that for you if you would like to do it, just to not execute logic for looking for the invitations

if if we have not online users, because not necessarily so if we have the update friends, first we

are checking the receiver list.

If the.

If the length of this list is larger than zero, so we have more than one online user of the specific

idea because as we said before, he can open to applications and log in at the same account.

We'll just emit and even to him with the new friends list at the initial connection and also the second

user also will receive that.

And now it's absolutely fine.

And we can check it if the of civil list, if there will be no active, go to the server store.

And if we have not get active connections, as you see if there would be no online user of the specific

I.D., that will be the empty.

Right.

So that's fine if we are checking here this for the length.

OK.

It's looking absolutely fine.

So here as we have update friends pending invitations, let's pass and export.

Also this update friends function, which we have created.

It's looking fine.

And now what?

What, what?

We would like to do it.

Let's go to the new connection handler.

And now here we are updating of the first connection with the socket IO server.

We are updating the friends pending invitations.

So here we can go for the update.

Friends released as the also initial update and will go for the Friends update.

Update, friends, and we'll just pass the user details.

Dot user I.D. of the user, which should receive this update now it's looking fine and will test our logic if every user will receive their friends list.

So we'll be just testing that in the next video.

# Testing Friends Updates

Let's test right now if we are able to see the real friends least, I will open the application and

I will refresh that right now.

And now the initial connection right now has been executed with the socket server and what I am seeing

that the test no.

One, I am log in right here in the Test No.

One, and I'm seeing that I am a friend with the test number two.

So that's the logic which we have made that before.

We have added this to users as the friends each other.

And the second user, which is logged in as a test number two, he's able to see a friend of the name

of test number one.

So it's perfect.

We are only missing one functionality that our list of the friends will be not updated if will accept

the invitation from the user or their user.

So let's go to the post.

Accept that J.S..

In their friends friend invitation controllers in our server, because here you will accept the invitation.

We have prepared right here to update the list of the friends of the users are online.

So here we just need to update the update.

Friends right here, friends.

Update updates like here and.

We just need update to friends and past their.

Sender ID, so the user, we just send the invitation, the idea of this user, because if we will accept

the invitation, we would like to update this friends list at the both sides of this user, which send

the invitation and this user, we just reject also that it was the receiver of the invitation.

So we have friends updates, update friends and here will pass the receiver I.D..

And we can use to trying to make sure that everything will be working fine.

Now, as we are doing that, what we have, we are passing the sender I.D. and receiver idea, which

we are getting from the invitation.

Now as we have two accounts already.

Here I will stay as a test, no one and I will all log out of the test user number two, I'll just refresh

that to make sure that we catch all of the changes, but we have been doing the server site, so everything

should be fine.

And I will create a new account of the name of test number three at Gmail dot com.

That's at the username test.

No free and we'll just pass the password.

Let's go for the register.

And now let's try to add different, which is the test number one test tester number on at Gmail dot com.

Let's send the invitation.

Now we are have that real time update thanks to Socket Io.

And if we need to accept the invitation, this two users should see each other with check if that would be working.

Now I just accepted the invitation and as you see, this front has been added.

So our real time update is working perfectly because test number three is able to see the test number one and test, no one is able to see the test number three.

So our logic right now is working perfectly.

And now what we can take care about right now, about the online indicator just to inform the other use to inform the user is this second user is online and after we will do it, we can proceed to the next part of our call switch where we will create the chat functionality.

Then, after the chat functionality, which will create with socket, IO will start creating video group calls the.

# Connecting Online User Indicator

Let's us right now to functionality that one user will be able to see the second user is online how

we can do it.

First, we'll go to our server store in our Discord backend folder.

Servers store.

And here.

We would like to prepare a function right here of the name of Get Online.

Users and that function will just be given us which users are right now online because we can get the

information about the online users from the connected users map.

So what we can do it as we have at the new connector to do you, sir, remove connected to you, sir,

get active connections, so the online user of the specific user I.D. Now we can add to the function

of the name of shared online users and will give us an idea of all users which they are currently online.

And that will be just will prepare an array online users and will iterate right now through our connected

users map for each where first we are getting the value of the object with the data.

And the second we are getting the key.

And now what we'll have right here will just use this online users array, which we have created and

who will just push the object.

Push objective them information of the user of the what socket it is use or have someone be socket tidy

like the key and the user idea will positively, you thought.

User I.D. Because as you remember, Socket ID is the key which we are setting when we are adding the

new element to our map and we can get this user ID from the value and the user I.D., each coming from

the value, which we are setting that value us and object where we have all the user I.D..

Why we are not setting just the user ID, but instead of this, we are using the object because if we

would like to add more fields, which we would like to store more details about with the connected users,

we can add this right now in this object.

But if we would like to add more information than just user I.D. and we will do not use the object,

we would just need to modify more our code to make that work.

But if you add one more field to the object is absolutely fine and the rest of our application will

not crash.

So that's the reason why we are pushing that this data in object and we are getting this right now as

the value to user I.D..

And of course, as we have that, we just need to return this online users.

Now let's go at the bottom and of course, let's take this function, which we have created.

And add this to their functions, which we would like to export now would go to the socket server.

And here what we would like to do it.

We would like to create the function just under a yellow dot use, which will be responsible to emit

online users, so will go for the emit online users like this?

And now we'll get that all of the online users, which we have in our server stores, also online users

servers target online users and now will take our Io connection object.

And we can right now emit to all of the connected users to our socket IO servers.

So right now it not depends of the user ID we can purchase broadcast and even to all of the users using

just I'll emit online users and this event will be admitted to all connected users.

Where will just pass a data, which will be the online users, which we are getting from our store servers

store and now, but we are not executing that anywhere.

So the client side applications in React, so the users which they are online will not receive the list

of the online users.

So we need to manage this somehow that this users will receive that.

So how we can do it?

I think the simplest way.

These if we successfully connect.

If you will successfully just create our listeners what we can do it after that creation of this listeners,

we can add the socket.

Or even after that.

Yes, just at the end.

Well, here we'll create the interval, so that interval will be started when our server will be started

and us will have that interval.

What this interval can do, that interval can emit online users to execute that function.

And when we would like to do it, we would like to use it and emit this evening ever like, you know,

maybe eight seconds.

So it will be going for eight thousand milliseconds like this.

Because one second is 1000 milliseconds, so we can even go like 1000 to get more clear information

that is going about the eight seconds.

So that would be our.

Time where we would like to execute once again that Emmett online users, but something is not fine

right here because I met online users here, we just need to.

We have a typo.

So now we have met online users right here and now let's right in the React application.

And even listener for the online users.

So let's go to our Discord front and folder to the source folder, and we'll be going to the real time

communication where we have socket connection dot J.S..

And now here, sock socket on and who passed this online user where it will be receiving the data.

And that should be.

That's what came in every eight seconds and now we'll just console log and just online users update

came, and let's check if we'll be able to see that in the developer's top in the browser.

Now, let's clear everything.

Refresh our connection.

Have the check if get that in every eight seconds.

Online users update came and if you wait eight seconds more.

The new update of the online users came.

As you see, we have the two information that the event was repeated twice.

Does that same console look?

But one thing which I don't like that we just need to wait for the first update, eight seconds and how we can do it.

We can just go that on the heads.

Go to the server right now to our socket server that on the successful connection here we would like to add mark this user us online and I mean to him, the list of the friends and pending invitations.

And right here at the beginning, we will execute on once Emmit online users because this user is the new online user.

So all of the users will receive the information about this because our map changed and we have new online user.

So now if we'll refresh our application now online users update came.

So we'll have that initial update of the first connection.

And if we have this right now, we have ideas of the all of online users and we can right now check if any, our friend is the same idea as the online user interface will just scroll this green indicator that he's online.

So let's do it in the next video.

# Online Indicator at React Side

Let's show right now the green indicator, if the user is online, what we would like to do it like

the same with different invitations and the friends list.

If the online users update came, we can remove that will go for store dispatch.

And here will set online users and here will pass the.

Online users, which they are coming into data like this content online users, and we can get them

from the data which is coming from the server.

And we'll go and just said this as an online users like this, why we are sending always their project,

not just, for example, the online users instead of the object, because if we would like to add some

fields, some more information like their additional properties, which we would like to connect with,

that even you can add this on just directly to this object.

So your logic still will be working and you will not need to adopt this if you will be just adding new

values to this object.

And now as we have that.

What we can do it we can just create this action because we have not created because we would like to

dispatch and that would create the helper function, which will be responsible to define the name of

the action, which we would like to dispatch set online users.

And also, they would be coming from their friends actions set online users.

Now we would go to the Friends Actions file in our star folder in our application.

And here we have pre parody before the name of the action.

But now we just need to create that helper function export concert set on line users where we will be

accepting the online users.

And now we'll just return the type of this which will be friends actions not set online.

Users here let us past the online users, and now it will be it will be able to dispatch this action

successfully of the name of their friends actions set on users.

Let's check our friends, read user.

Here we have online users and if the action of this name would be dispatched, the new online users

will be set.

So it's looking that it's working correctly.

And let's go right now to the our friends list, where we'll be able to check if the user is online

and how we can do it.

Let's go to the Friends list in our React application, which is in the React application Discord frontend

source.

Here we have the Dashboard Friends list and we have Friends list and here we are setting.

If this user is online, but we are mapping through their friends and friends, does not have that is

online field.

If we are getting them from the server because now we need to check if the idea of the friends is the

same as the one of the ideas of the online users.

And if yes, we will mark this as this user is online and how we can do it, just not passing the friends,

but will right with the properties with their custom function.

I will just take this and I will write that check.

Online users and here I will pass the list of the friends and I will pass the list of the online users.

We can get the list of the online users, which contains an array of the online users IDs from the same

red user, which is the friends and we can the friends list and online users.

Of course, we need to define this function.

Check online users where it will be accepting their.

Friends.

All right.

And just define the default value as an empty array and also the online users, let's define this as

the empty array like this.

As the initial value and now how we can do it.

Now we can check if this user is online, so we'll go.

Or even at the beginning.

Let's map for every friend which we have on our list.

So we'll go for the friends, for each and will receive every friend, which we have in this friends

era.

And now for every friend, we would like to check if he's online and how we can do it.

Let's define each user online.

And we can do it to find online users.

Find.

And will find in the era of the online users, if the idea of the online user like this.

We will get every user which is in the online users and will find if the user ID of the online user,

will it be the same as their.

Idea of the front, as we said before, and now, if that idea would be the same, what we can do it,

we can mark that at the new friend field, which was to be a friend, is online and now will check its

user online, which we have just checked it, that if that will be true with the mark, this as a true

and if not, will mark this as a false.

So we can pass it right here because if the.

If there will be no online user of the specific friend I.D., which we have right here, friend I.D.

and this you said it would be not I.D. This friend will just get the undefined from defined function,

which is being executed on online users.

All right.

And as you see if it will be not found undefined issues instead.

And now, as we have thought, what we can do it, we can just return the friends, which we are getting

from our story.

Like this and everything should be fine right now.

And so that check if that will be working, that will be able to see if the user is online.

Let's open this.

Let's refresh our application for now.

We have only one online user, which is test number one.

So let's log in after test number two.

So I will open the new private window where I will open my application, which is a localhost three

thousand.

Let's go after test number two at Gmail dot com.

Just remember to use the different browsers because then they can be a conflict with the local storage.

So we'll have an an issues to log in on the different accounts and now before logging.

We are seeing that online indicator test user test.

Number two is seeing that this second user is online and let's check what will happen if you will just disconnect as this user lets close.

And now on the next update, we should have gotten the information that this user is no longer online, as you see.

New update After time interval of eight seconds gains, this user has been removed at the server store of the on map of the connected users, and this is not a longer market as the online user.

So what we have right here down in this part of the schools that we have the full functionality of eight

friends that we can reject or accept the invitation, we have real time at least of the friends real time list of the pending invitations.

And the next part of the course will be related that if we press.

The name of the friend, the new chat window will be open and will be able to exchange their messages directly with this user.

And after will just finish that chat functionality will add this video call rooms.