Step 1.1: NewPostInput.js

```
return <NewPostInput defaultText="New Message" onSubmit={sendMessage} />;
```

Step 1.2: NewPostInput.js

```
class NewMessage extends Component {
  const sendMessage = (value) => {
    console.log(value);
  };

  return <NewPostInput defaultText="New Message" onSubmit={sendMessage} />;
  }
}
```

Step 1.3: SingleMessage.js

```
return (
  <div className={"u-flex u-flex-alignCenter SingleMessage-container"}>
    <span className=" SingleMessage-sender u-bold">{props.message.sender.name + ":"}</span>
    <span className="SingleMessage-content">{props.message.content}</span>
  </div>
);
```

Step 1.4: Chat.js

```
return (
  <div className="u-flexColumn Chat-container">
    <h3>Chatting with {props.data.recipient.name}</h3>
    <div className="Chat-historyContainer">
      {props.data.messages.map((m, i) => (
        <SingleMessage message={m} key={i} />
      ))}
    </div>
    <div className="Chat-newContainer">
      <NewMessage recipient={props.data.recipient} />
    </div>
  </div>
);
```

Step 1.5: Chatbook.js

```
if (!props.userId) {
  return <div>Log in before using Chatbook</div>;
}

return (
  <>
    <div className="u-flex u-relative Chatbook-container">
      <div className="Chatbook-chatContainer u-relative">
        {/* TODO (step 2.2): change data to use our activeChat state */}
        <Chat
          data={{
            recipient: ALL_CHAT,
            messages: TEST_MESSAGES,
          }}
        />
      </div>
    </div>
  </>
);
```

Step 1.6: Chatbook.js

```
const ALL_CHAT = {
  _id: "ALL_CHAT",
  name: "ALL CHAT",
};

const TEST_MESSAGES = [
  {
    sender: {
      _id: 0,
      name: "Alex",
    },
    content: "i love aaron sippy cup",
  },
  {
    sender: {
      _id: 0,
      name: "Nik",
    },
    content: "i spend too much time on piazza",
  },
];

const TEST_DATA = {
  recipient: ALL_CHAT,
  messages: TEST_MESSAGES,
};
```

Step 2.1: (YOUR TURN) Chatbook.js

```
const [activeChat, setActiveChat] = useState({
  recipient: ALL_CHAT,
  messages: TEST_MESSAGES,
});
```

Step 2.2: (YOUR TURN) Chatbook.js

```
return (
  <>
    <div className="u-flex u-relative Chatbook-container">
      <div className="Chatbook-chatContainer u-relative">
        <Chat data={activeChat} />
      </div>
    </div>
  </>
);
```

Step 3.1: message.js

```
const mongoose = require("mongoose");

//define a message schema for the database
const MessageSchema = new mongoose.Schema({
  sender: {
    _id: String,
    name: String,
  },
  recipient: {
    _id: String,
    name: String,
  },
  timestamp: { type: Date, default: Date.now },
  content: String,
});

// compile model from schema
module.exports = mongoose.model("message", MessageSchema);
```

Step 3.2: api.js /message

```
router.post("/message", auth.ensureLoggedIn, (req, res) => {
  console.log(`Received a chat message from ${req.user.name}: ${req.body.content}`);

  // insert this message into the database
  const message = new Message({
    recipient: req.body.recipient,
    sender: {
      _id: req.user._id,
      name: req.user.name,
    },
    content: req.body.content,
  });
  message.save();
  // TODO (step 6): emit to all clients that a message was received
});
```

Step 3.3: api.js /chat

```
router.get("/chat", (req, res) => {
  const query = { "recipient._id": "ALL_CHAT" };
  Message.find(query).then((messages) => res.send(messages));
});
```

Step 4.1: (YOUR TURN) NewPostInput.js

```
const NewMessage = (props) => {
  const sendMessage = (value) => {
    const body = { recipient: props.recipient, content: value };
    post("/api/message", body);
  };
```

Step 5.1, 5.2: Chatbook.js

```
const loadMessageHistory = (recipient) => {
  get("/api/chat", { recipient_id: recipient._id }).then((messages) => {
    setActiveChat({
      recipient: recipient,
      messages: messages,
    });
  });
};
```

```
useEffect(() => {
  loadMessageHistory(ALL_CHAT);
}, []);
```

W9 BEGINS HERE

Workshop 9

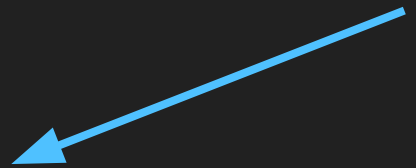
remember in backend, use socketManager instead of socket!

Step 1: define and use initsocket

Step 0.1: api.js (line 99)

```
router.post("/message", auth.ensureLoggedIn, (req, res) => {
  console.log(`Received a chat message from ${req.user.name}: ${req.body.content}`);

  // insert this message into the database
  const message = new Message({
    recipient: req.body.recipient,
    sender: {
      _id: req.user._id,
      name: req.user.name,
    },
    content: req.body.content,
  });
  message.save();
  socketManager.getIo().emit("message", message);
});
```



Step 0.2: Chatbook.js (use message not data)

```
59  useEffect(() => {
60    // TODO (step 0.2): add socket.on for when received message
61    socket.on('messsage', addMessages)
62    return () => {
63      socket.off('message', addMessages)
64    }
65  }, []);
```

Step 1.1: api.js:77

```
75  router.post("/initsocket", (req, res) => {
76    // do nothing if user not logged in
77    // TODO (step 1.1): addUser when init socket
78    if (req.user) {
79      socketManager.addUser(req.user, socketManager.getSocketFromSocketID(req.body.socketid));
80    }
81    res.send({});
82  });
```

Step 1.2: App.js:36

```
32  const handleLogin = (res) => {
33    const userToken = res.tokenObj.id_token;
34    post("/api/login", { token: userToken }).then((user) => {
35      setUserId(user._id);
36      // TODO (step 1.2): make post call to /api/initsocket
37      post("/api/initsocket", { socketid: socket.id });
38    });
39  };
```

Step 2: just some react code to set up ChatList and SingleUser

Step 2.1: SingleUser.js:14

```
const SingleUser = (props) => {
  return (
    <div
      className={`SingleUser-container u-pointer ${props.active ?
        "SingleUser-container--active" : ""
        }`}
      onClick={() => {
        props.setActiveUser(props.user);
      }}
    >
      {props.user.name}
    </div>
  );
}
```

Step 2.2: ChatList.js:16

```
const ChatList = (props) => {
  return (
    <>
      <h3>Open Chats</h3>
      {props.users
        .map((user, i) => (
          <SingleUser
            key={i}
            setActiveUser={props.setActiveUser}
            user={user}
            active={user === props.active}
          />
        ))}
    </>
  );
}
```

Step 2.3: Chatbook.js:37

```
const [activeUsers, setActiveUsers] = useState([]);
```

Step 2.4: Chatbook.js:76

```
const setActiveUser = (user) => {
  console.log(`setting active user to ${user.name}`);
};
```

Step 3 (THEM): pass all necessary props to ChatList

STEP 3.1: Chatbook.js:86 Add Chatlist component to Chatbook (THEIR TURN) - 4 hints

```
return (
  <>
    <div className="u-flex u-relative Chatbook-container">
      <div className="Chatbook-userList">
        <ChatList
          setActiveUser={setActiveUser}
          userId={props.userId}
          users={activeUsers}
          active={activeChat.recipient}
        />
      </div>
      <div className="Chatbook-chatContainer u-relative">
        <Chat data={activeChat} />
      </div>
    </div>
  </>
);
```

SHOW that the chatlist shows up

Step 4 (THEM): create /activeUsers route

STEP 4.1: api.js:103 get (THEIR TURN) - 3 hints

```
router.get("/activeUsers", (req, res) => {
  res.send({ activeUsers: socketManager.getAllConnectedUsers() });
});
```

Step 5: GET active users

Step 5.1: Chatbook.js:69 get

```
useEffect(() => {
  get("/api/activeUsers").then((data) => {
    // If user is logged in, we load their chats. If they are not logged in,
    // there's nothing to load. (Also prevents data races with socket event)
    if (props.userId) {
      setActiveUsers([ALL_CHAT].concat(data.activeUsers));
    };
  });
}, []);
```

SHOW that the chatlist now populates, but only if we switch pages (maybe not if reload because the getActiveUsers request completes before we finish logging in so the server thinks we are not logged in)

Step 6 (THEM): socket receive active users

Step 6.1: Chatbook.js:85 (THEIR TURN)

```
useEffect(() => {
  const callback = (data) => {
    setActiveUsers([ALL_CHAT].concat(data.activeUsers));
  };
  socket.on("activeUsers", callback);
  return () => {
    socket.off("activeUsers", callback);
  };
}, []);
```

SHOW that the chatlist now updates live

Step 7 (THEM): write setActiveUser

Step 7.1: Chatbook.js:96 (THEIR TURN)

```
64    useEffect(() => {
65      loadMessageHistory(activeChat.recipient);
66    }, [activeChat.recipient._id]);
```

```
96    // TODO (step 7.1): Set the state "activeChat" to the new recipient (user)
97    // and empty array for messages.
98    // Then, make sure that the message history for this user is loaded (might
99    // involve writing code outside of this function)
100   if (user === activeChat.recipient) {
101     return;
102   }
103   console.log(`setting active user to ${user.name}`);
104   setActiveChat({
105     recipient: user,
106     messages: [],
107   })
108   // loadMessageHistory(user);
109 };
```

SHOW that we can now switch chats, but we get the wrong loaded chat (ALL_CHAT)

Step 8: api.js query DMs instead of just all chat

STEP 8.1: api.js:83

```
router.get("/chat", (req, res) => {
  let query;
  if (req.query.recipient_id === "ALL_CHAT") {
    // get any message sent by anybody to ALL_CHAT
    query = { "recipient._id": "ALL_CHAT" };
  } else {
    // get messages that are from me->you OR you->me
    query = {
      $or: [
        { "sender._id": req.user._id, "recipient._id": req.query.recipient_id },
        { "sender._id": req.query.recipient_id, "recipient._id": req.user._id },
      ],
    };
  }

  Message.find(query).then((messages) => res.send(messages));
});
```

SHOW that we now load the correct chat, but our DMs get broadcasted

Step 9.1: don’t emit DMs to everyone, only the sender and receiver (ex. Send message to self, see in other user’s all chat),

STEP 9.1: api.js:113

```
  if (req.body.recipient._id == "ALL_CHAT") {
    socketManager.getIo().emit("message", message);
  } else {
    socketManager.getSocketFromUserID(req.user._id).emit("message", message);
    if (req.user._id !== req.body.recipient._id) {
      socketManager.getSocketFromUserID(req.body.recipient._id).emit("message", message);
    }
  }
});
```

SHOW that DMs don’t get broadcasted but allchat shows up in DMs

Step 10.1: filter out messages on front end (ex. Send all chat, see in dm)

STEP 10: Chatbook.js:75

```
useEffect(() => {
    const addMessages = (data) => {
      if (
        (data.recipient._id === activeChat.recipient._id &&
          data.sender._id === props.userId) ||
        (data.sender._id === activeChat.recipient._id &&
          data.recipient._id === props.userId) ||
        (data.recipient._id === "ALL_CHAT" && activeChat.recipient._id === "ALL_CHAT")
      ) {
        setActiveChat(prevActiveChat => ({
          recipient: prevActiveChat.recipient,
          messages: prevActiveChat.messages.concat(data),
        }));
      }
    };
    socket.on("message", addMessages);
    return () => {
      socket.off("message", addMessages);
    };
  }, [activeChat.recipient._id, props.userId]);
```

SHOW that all chat doesn’t show up in DMs