Project 6: Learning Curve: Interim Report

Title: Learning Curve
Link to Heroku app: https://the-learning-curve.herokuapp.com/
Members: Ash Duy, Alex Herman, Annikka Turmala

Status summary:
Work done:

- MongoDB database cluster created (Annikka)
- Server side app setup, including automated build and deployment to Heroku. (Ash)
- Room functionality (joining and leaving). (Ash)
- Angular routing and template generation of homepage and playground. (Alex)
- Send user data from dialogs back to their respective components. (Alex and Ash)

Design Patterns

- Singleton – we have been utilizing Angular services as singletons throughout the application, simplifying and unifying features that will be used across components
- Observer – Angular is built with observers as a core feature, so many events (including with the interactions with the Socket.io server) are abstracted into easy-to-manipulate observables that can be subscribed to.
- MVC – Angular also highly encourages the model-view-controller pattern, whereby the html for a component acts as a view, the component class acts as a controller, and Angular services allow us to store models of the current client state (in the UML diagram we submitted, only singleton is marked on the RoomService to make it tidy – on our final diagram we are planning on adding an additional service to handle cards so we will mark that as a model). This separation of application logic makes it very easy to quickly iterate and make changes throughout the application.

Class Diagram: Attached, it is the server-side diagram followed by the client-side diagram.

Plan for next iteration:

Work estimation:
We estimate that 50% of the project is complete.

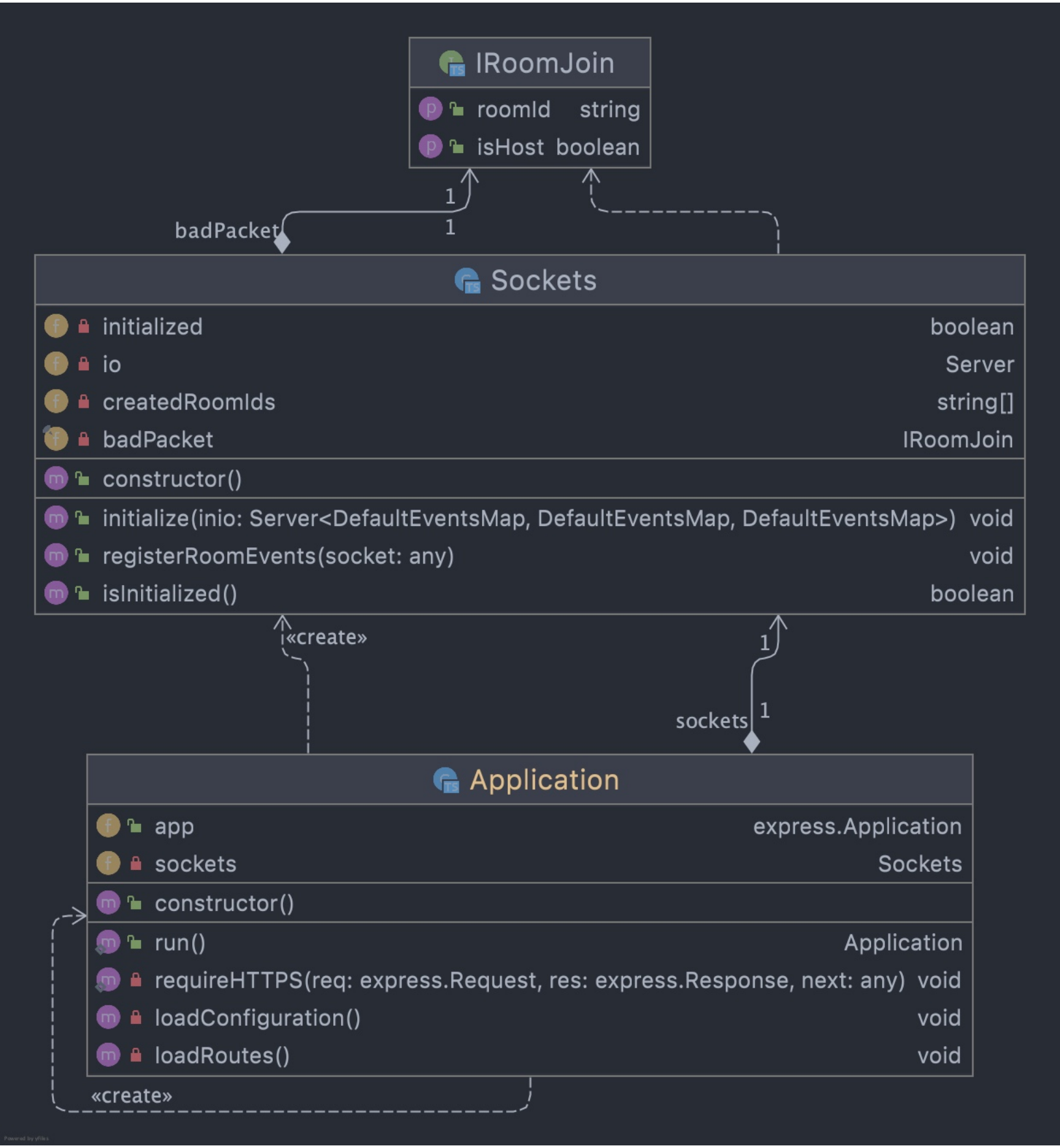
Design changes:
No design changes thus far.

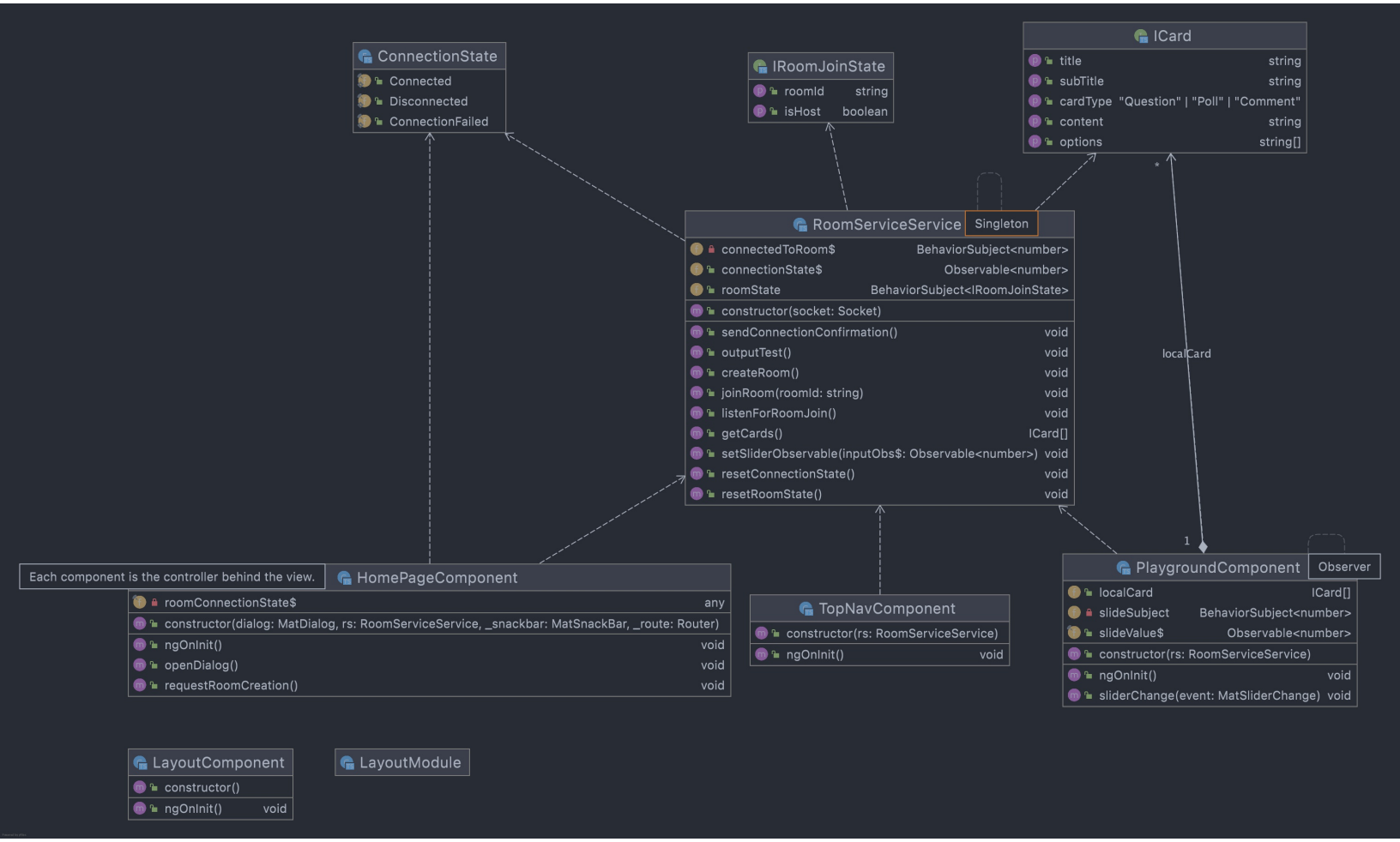
Plans for next iteration:
We plan to break up the work starting on the 29th, when we will begin getting the card cache to work on the server side and following up with rendering them from the cache on the client. After that we plan to flesh out the card functionalities (and types) and enable peer-to-peer updates based on editing those cards. Finally, we plan on getting database saving working (if no clients

are left in the room) and persistent sessions for leaving/rejoining clients. It's a lot to do, but we think it'll make a great app!

What we plan to have done by 12/18:

- Create database connections
- Homepage have animated svg
- persistent users
- Cards in 3 columns with optional virtual scrolling
- Adding moderator capabilities
- Cards to be editable after you place them (like delete functionality)
    - By admin
    - By author
- Database writes current state of room when everyone leaves room
    - If someone rejoins empty room, the room is repopulated with cards & sliders
- Functioning cache


IRoomJoin
roomId string
isHost boolean
1
1
badPacket
Sockets
initialized boolean
io Server
createdRoomIds string[]
badPacket IRoomJoin
constructor()
initialize(inio: Server<DefaultEventsMap, DefaultEventsMap, DefaultEventsMap>) void
registerRoomEvents(socket: any) void
isInitialized() boolean
«create»
1
sockets
1
Application
app express.Application
sockets Sockets
constructor()
run() Application
requireHTTPS(req: express.Request, res: express.Response, next: any) void
loadConfiguration() void
loadRoutes() void
«create»


ConnectionState
Connected
Disconnected
ConnectionFailed
IRoomJoinState
roomId string
isHost boolean
ICard
title string
subTitle string
cardType "Question" | "Poll" | "Comment"
content string
options string[]
RoomServiceService
Singleton
connectedToRoom$ BehaviorSubject<number>
connectionState$ Observable<number>
roomState BehaviorSubject<IRoomJoinState>
constructor(socket: Socket)
sendConnectionConfirmation() void
outputTest() void
createRoom() void
joinRoom(roomId: string) void
listenForRoomJoin() void
getCards() ICard[]
setSliderObservable(inputObs$: Observable<number>) void
resetConnectionState() void
resetRoomState() void
localCard
*
1
Each component is the controller behind the view.
HomePageComponent
roomConnectionState$ any
constructor(dialog: MatDialog, rs: RoomServiceService, _snackbar: MatSnackBar, _route: Router)
ngOnInit() void
openDialog() void
requestRoomCreation() void
TopNavComponent
constructor(rs: RoomServiceService)
ngOnInit() void
PlaygroundComponent
Observer
localCard ICard[]
slideSubject BehaviorSubject<number>
slideValue$ Observable<number>
constructor(rs: RoomServiceService)
ngOnInit() void
sliderChange(event: MatSliderChange) void
LayoutComponent
constructor()
ngOnInit() void
LayoutModule