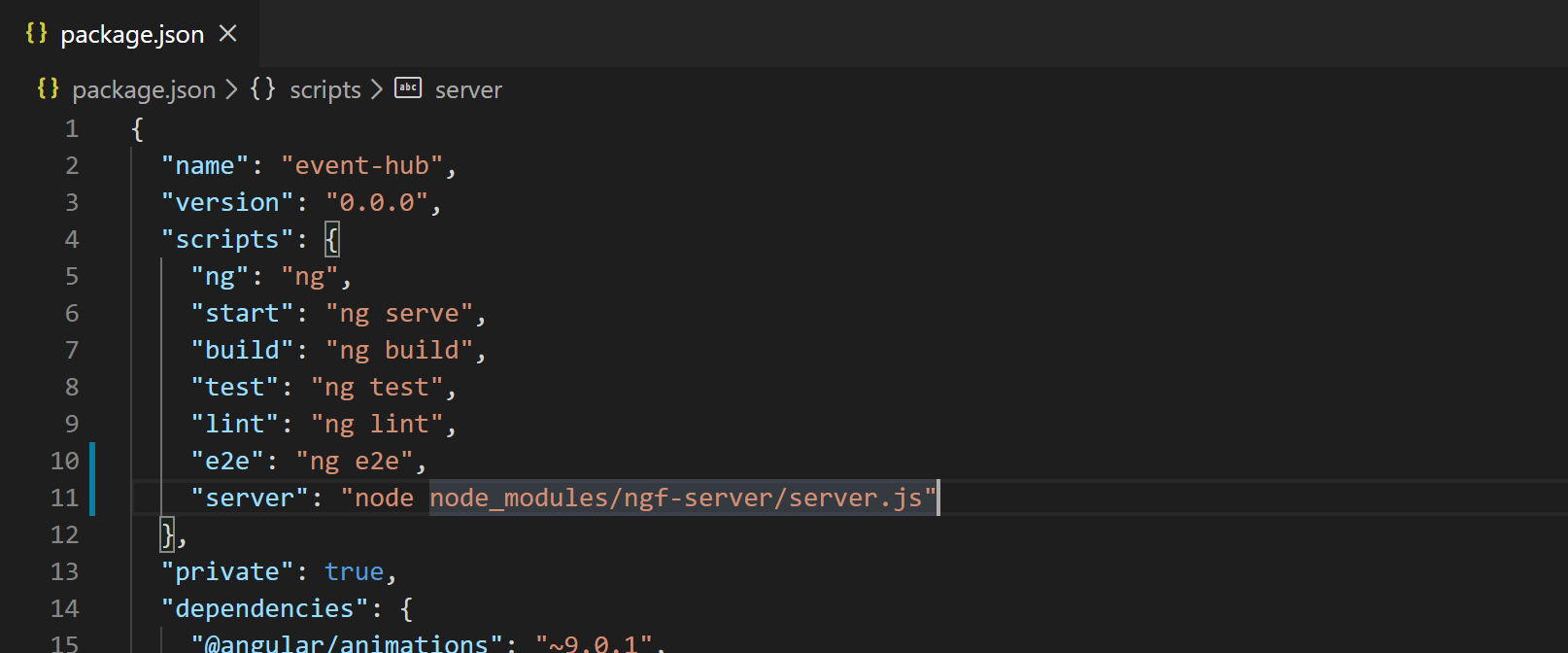
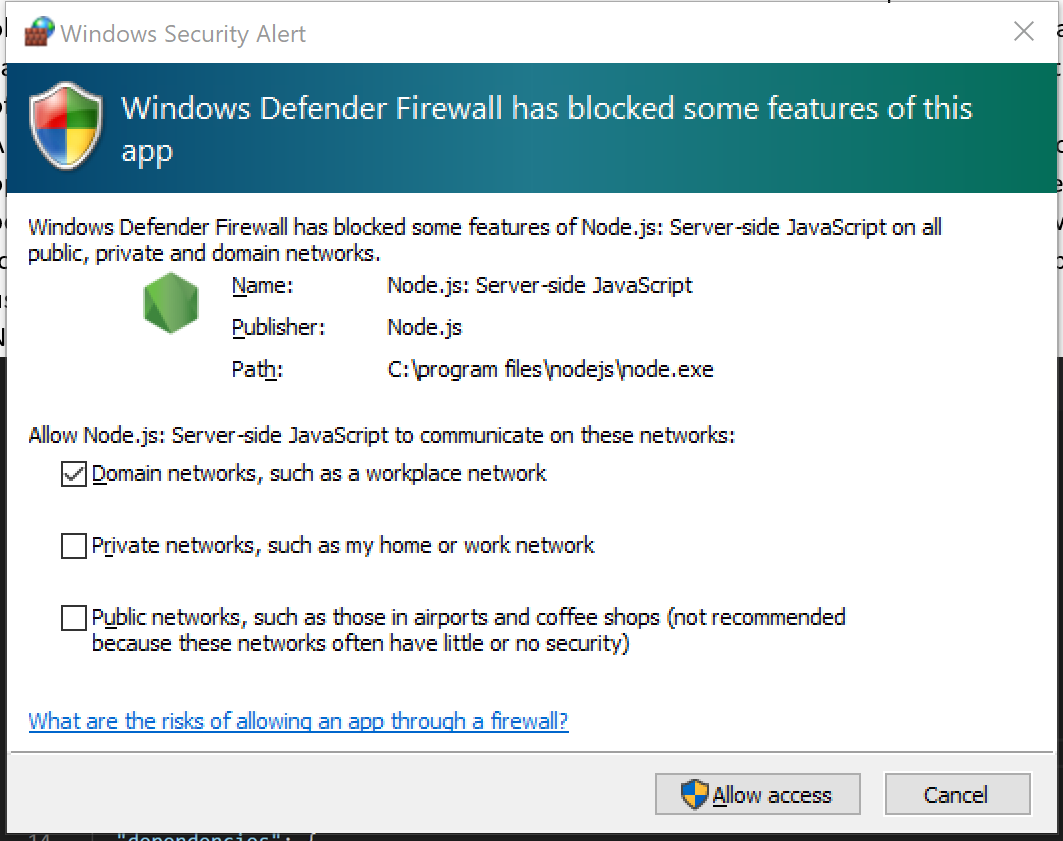
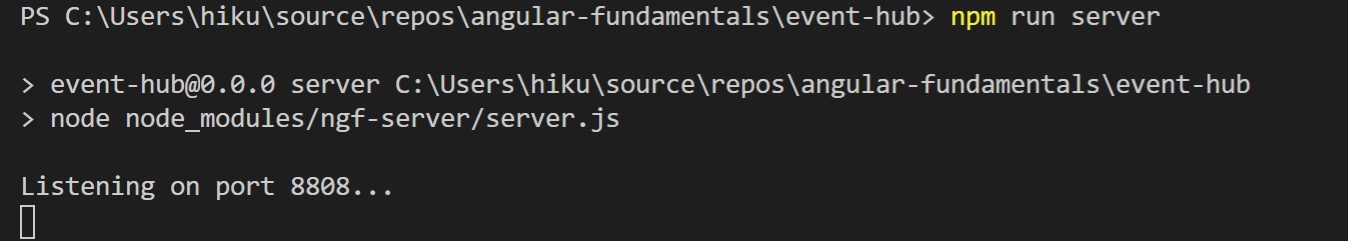
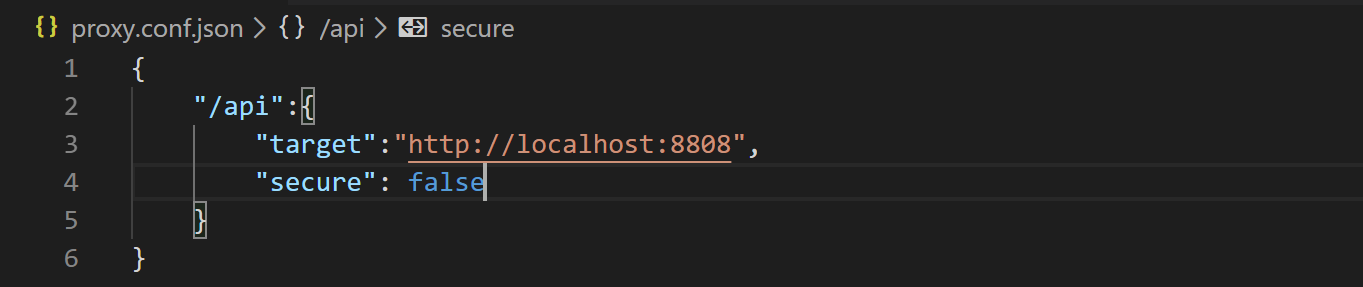
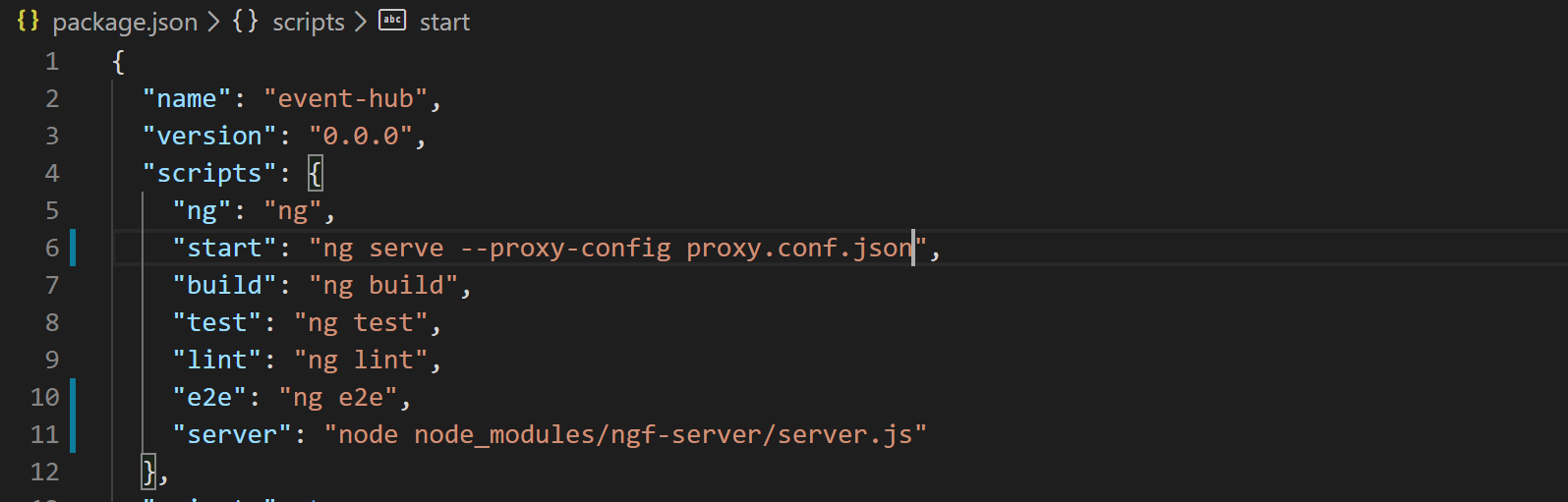
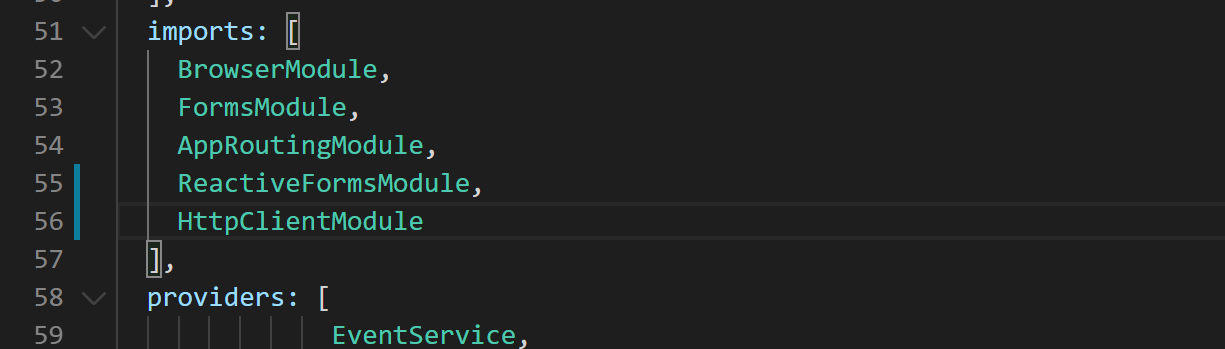
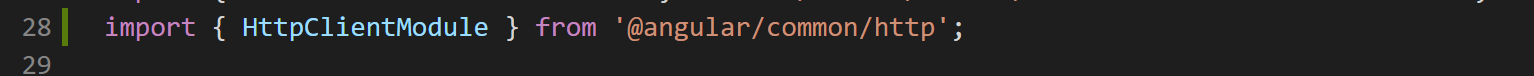
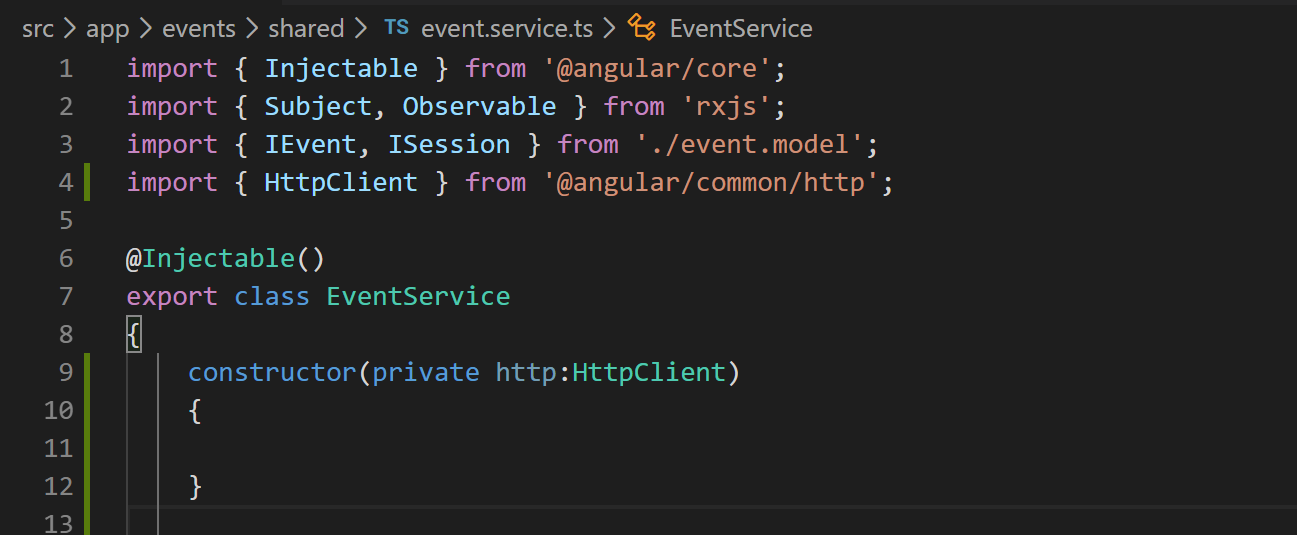
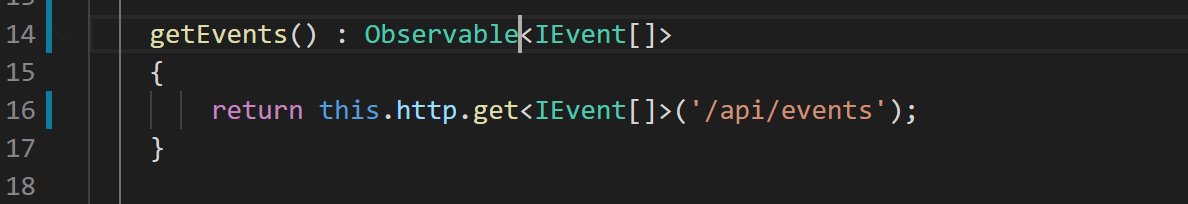
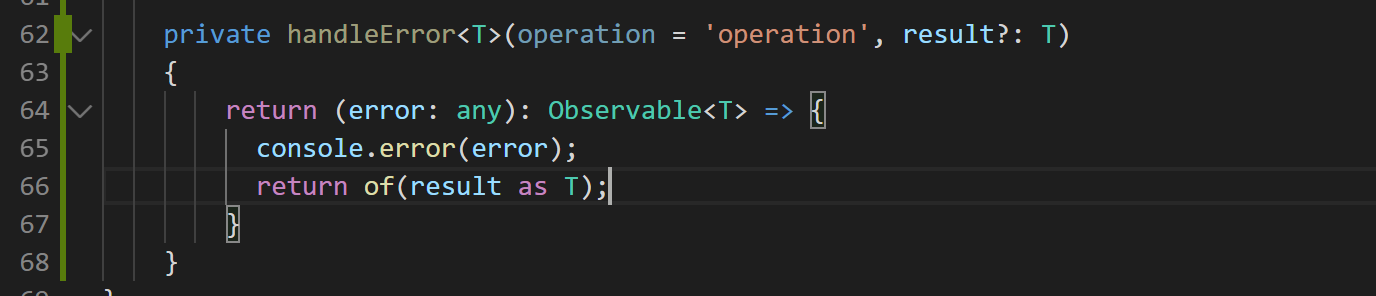
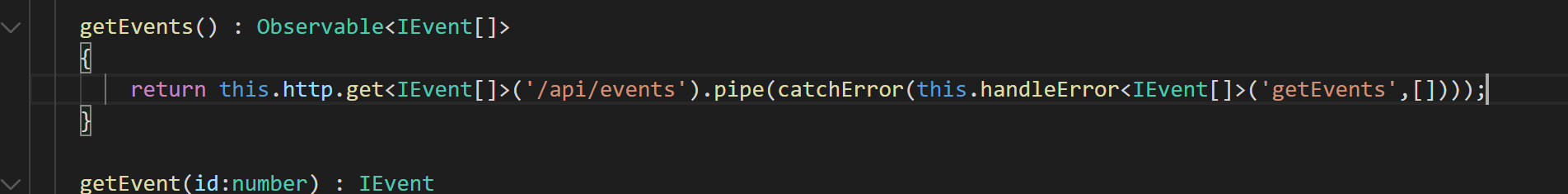
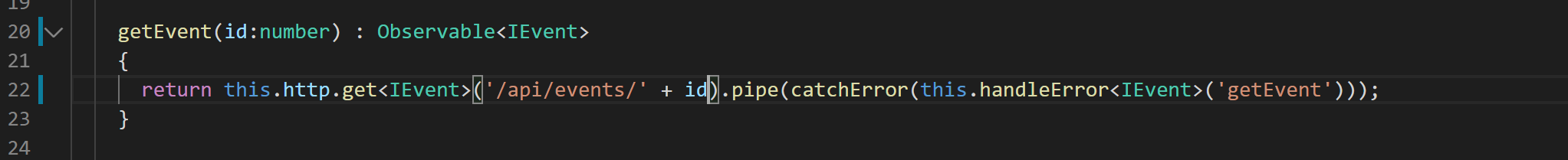
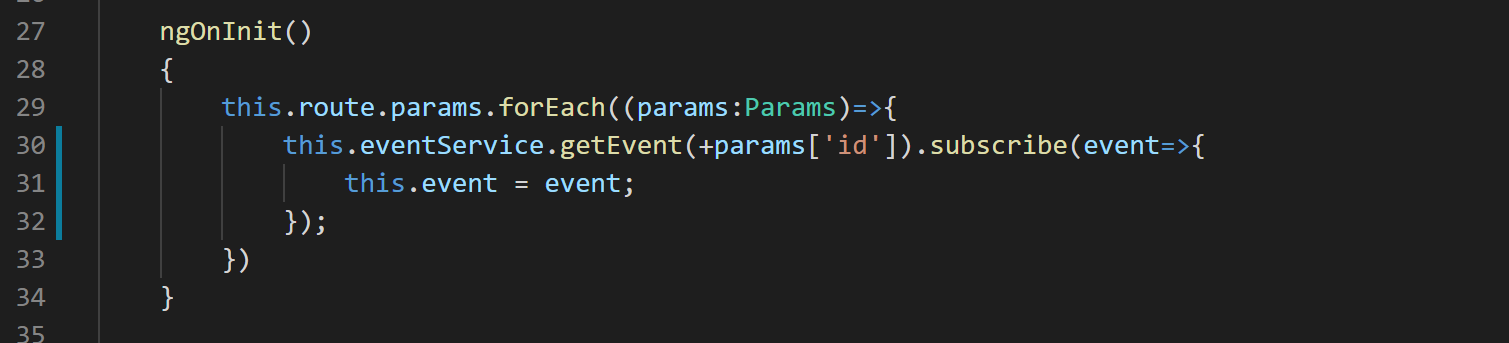
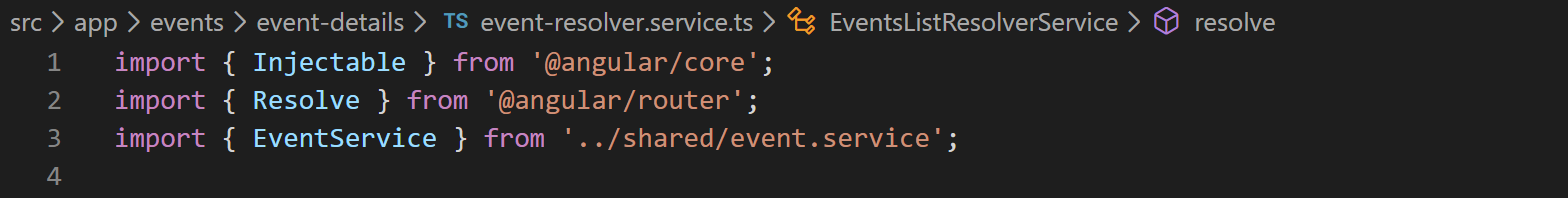
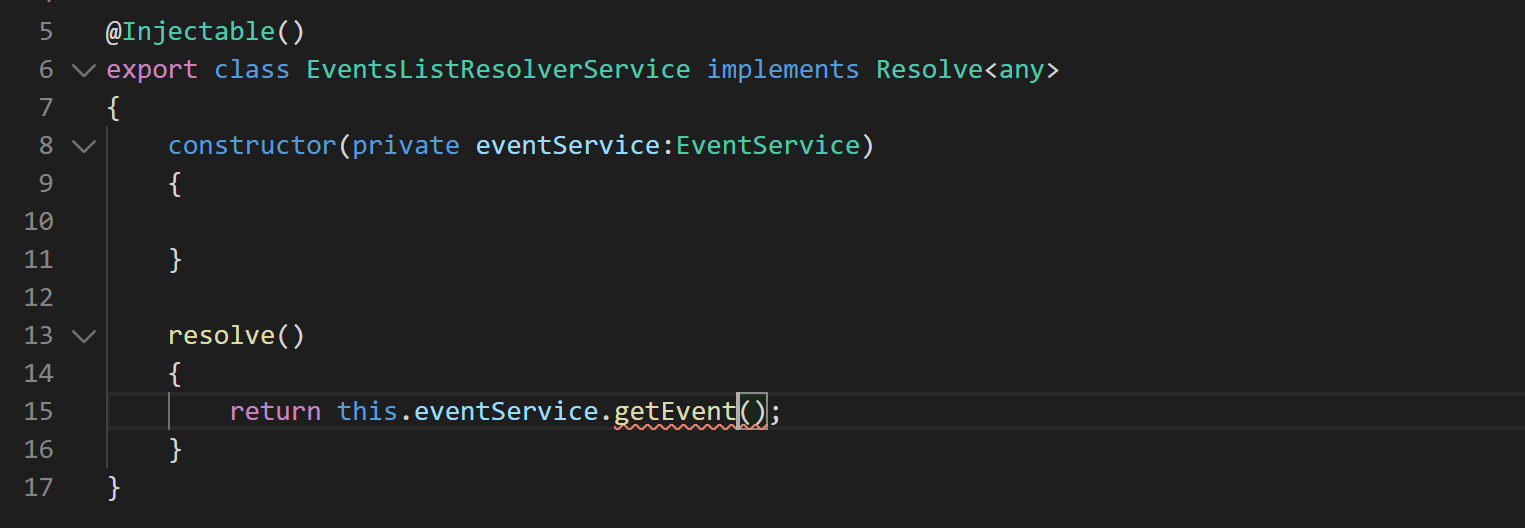
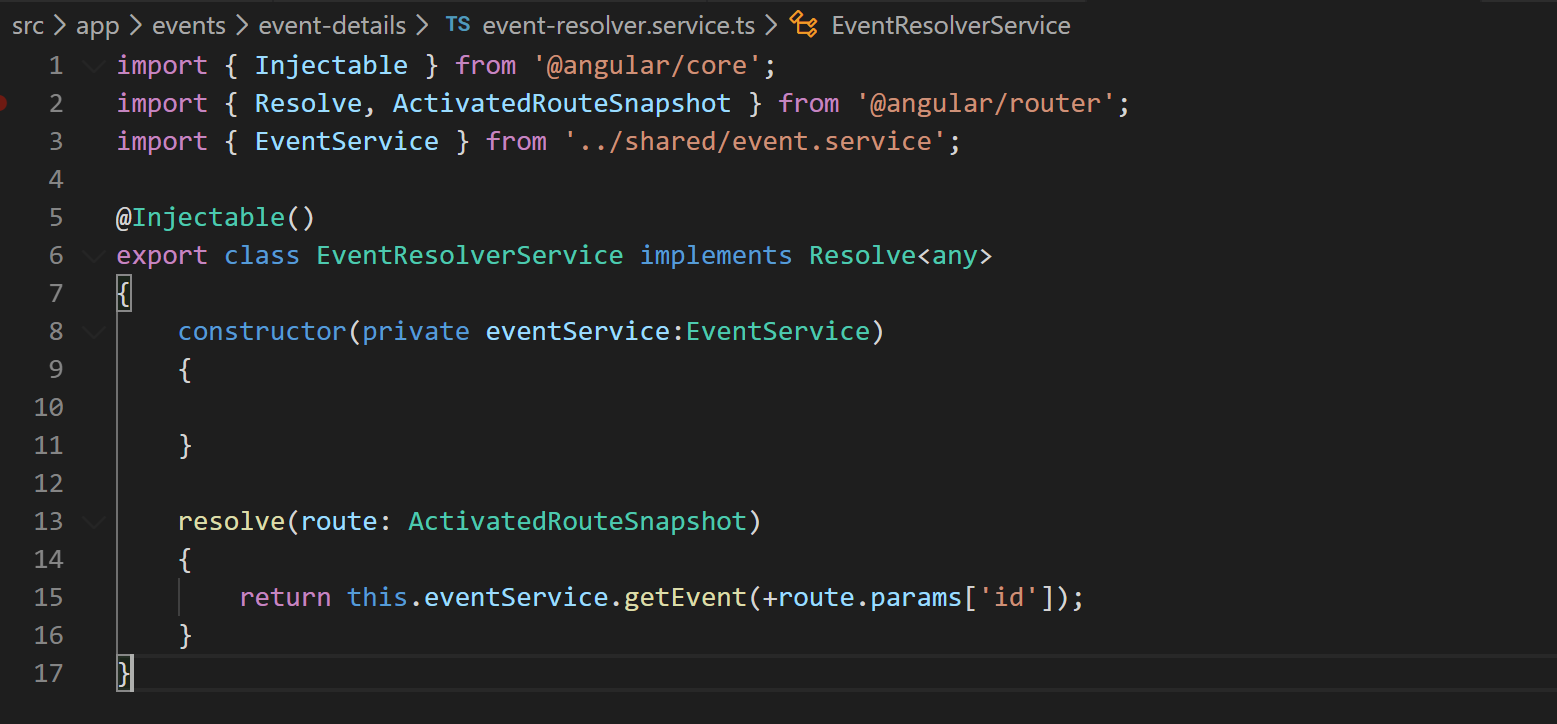
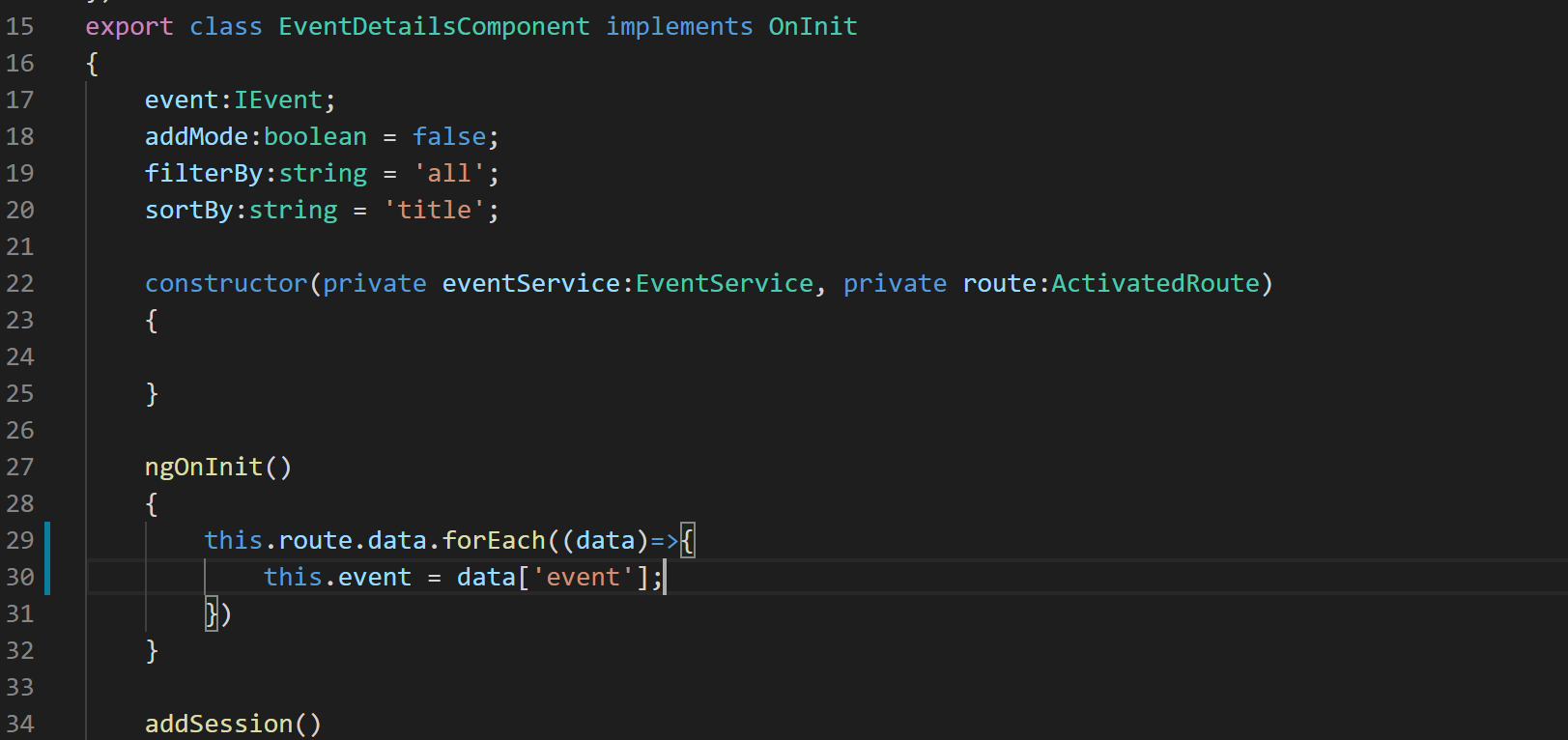
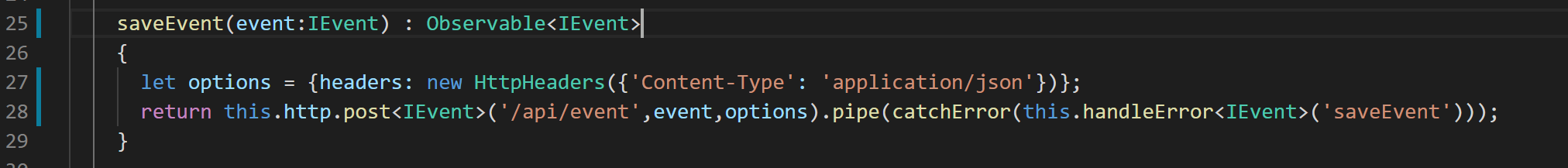
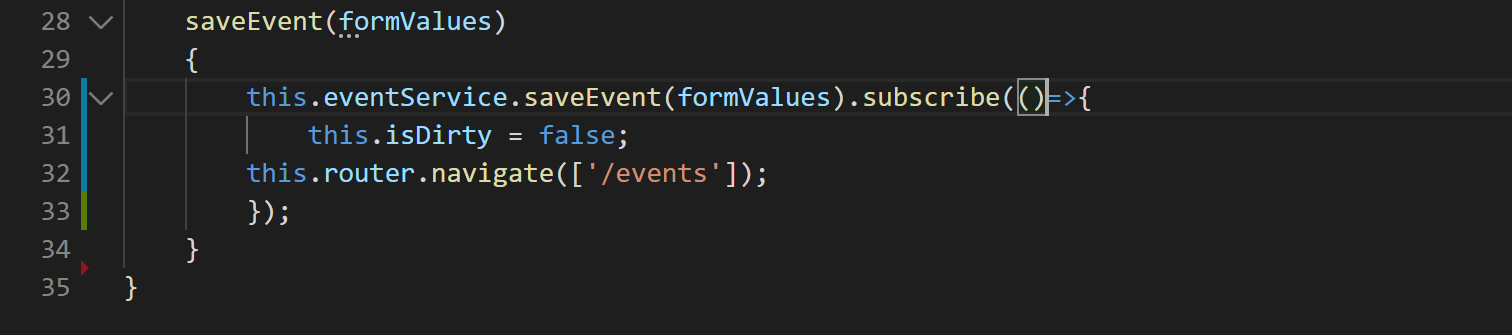
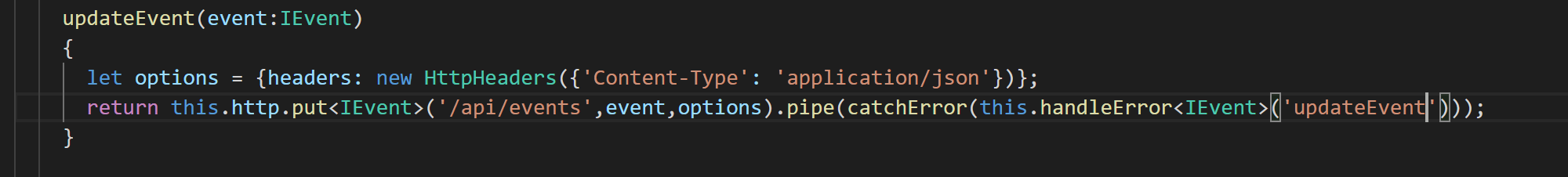
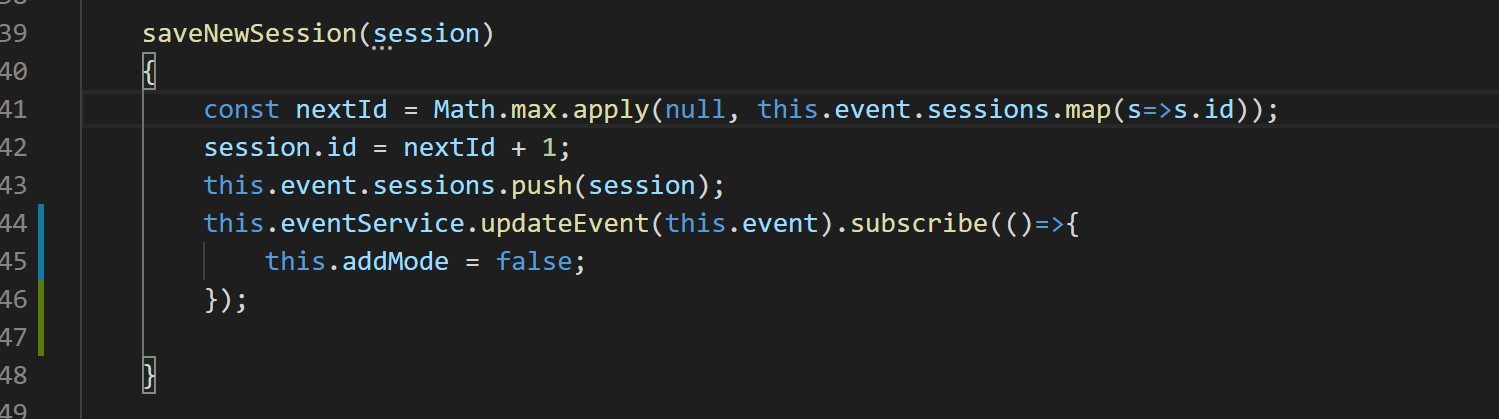
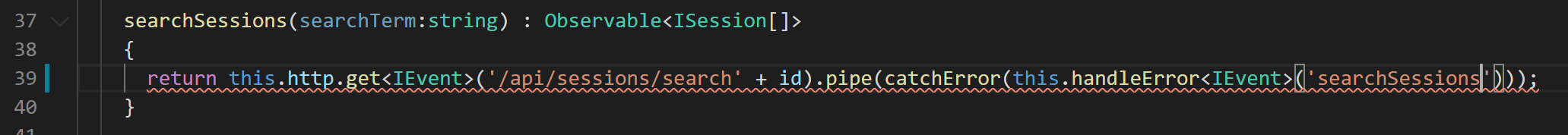
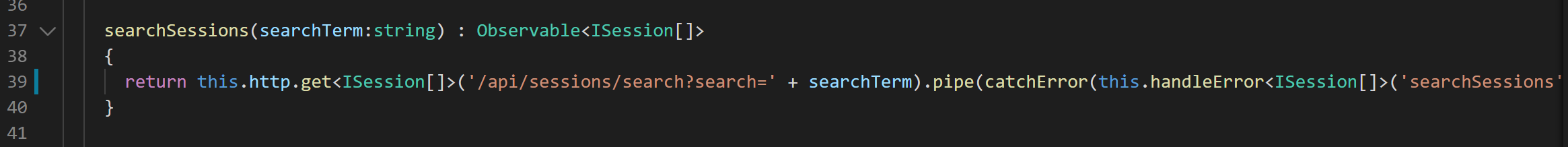
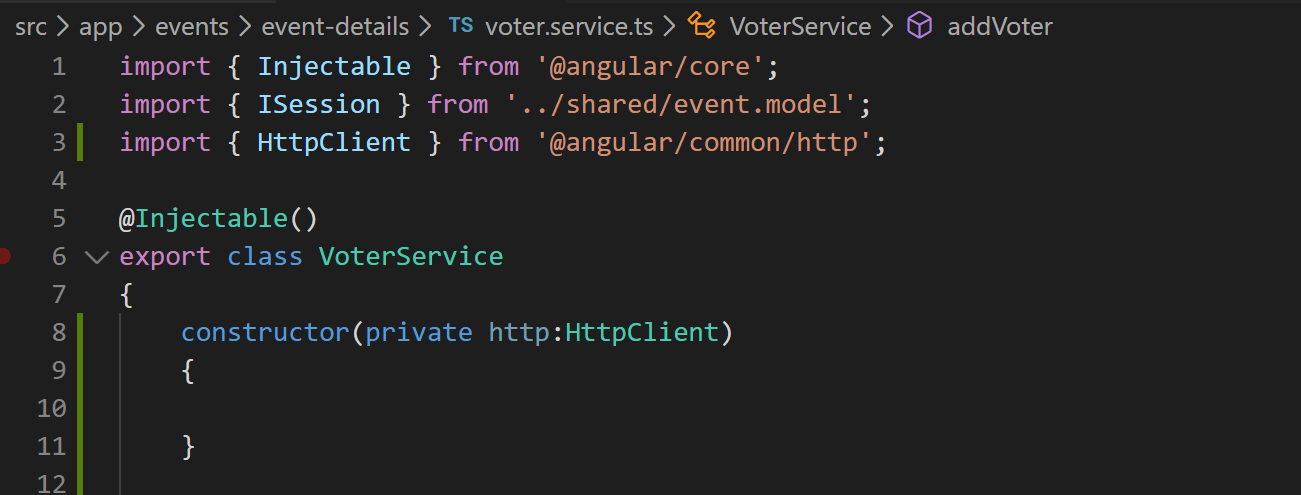
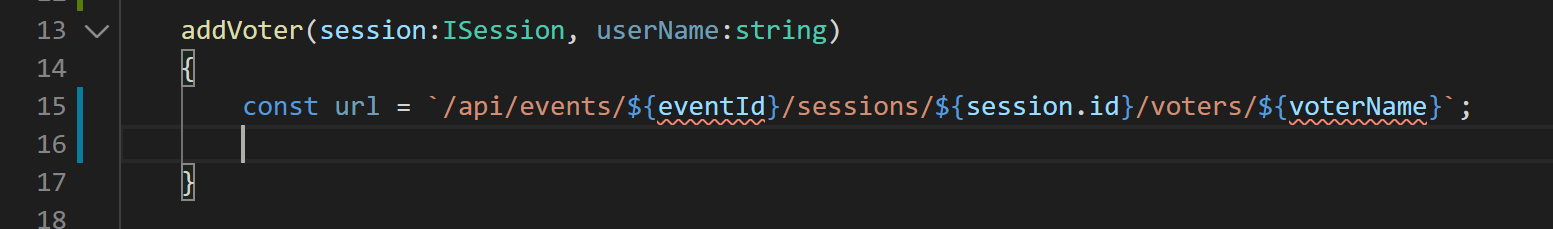
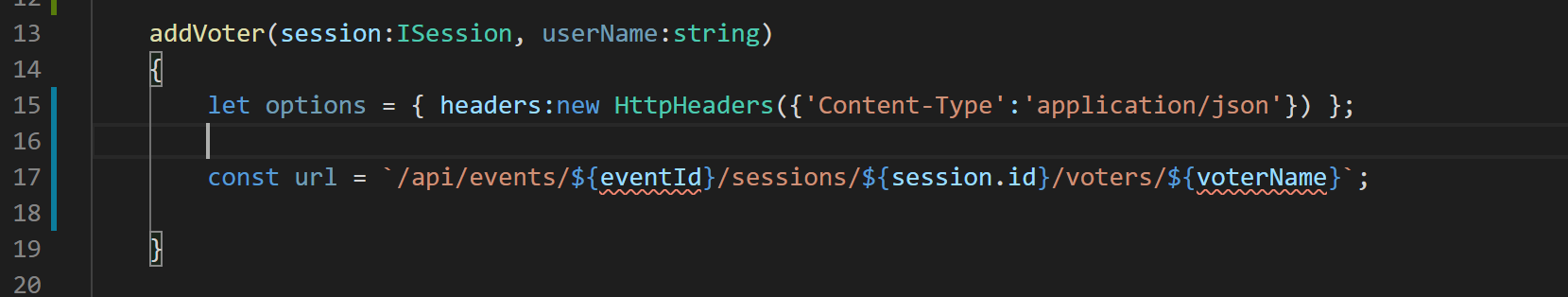
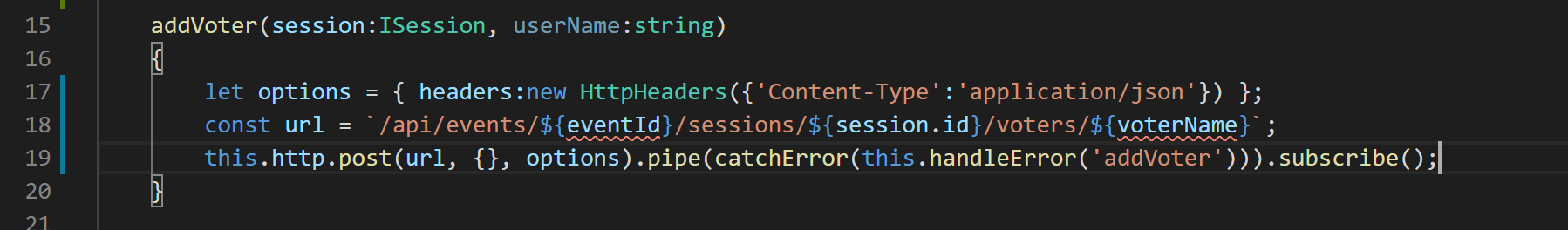
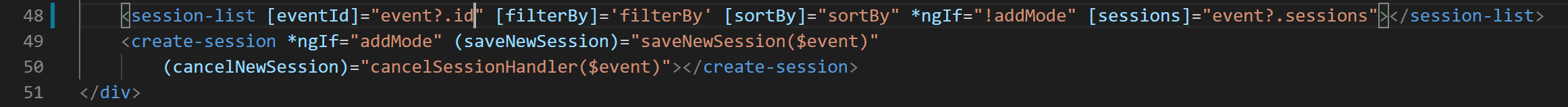
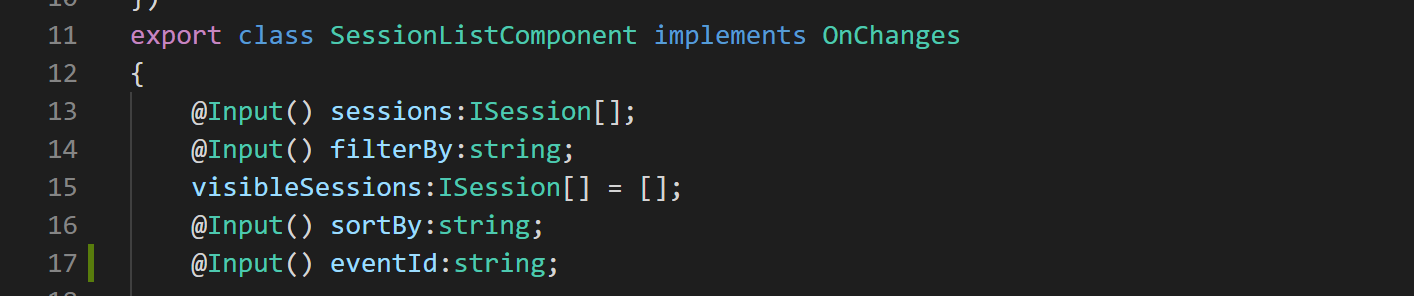
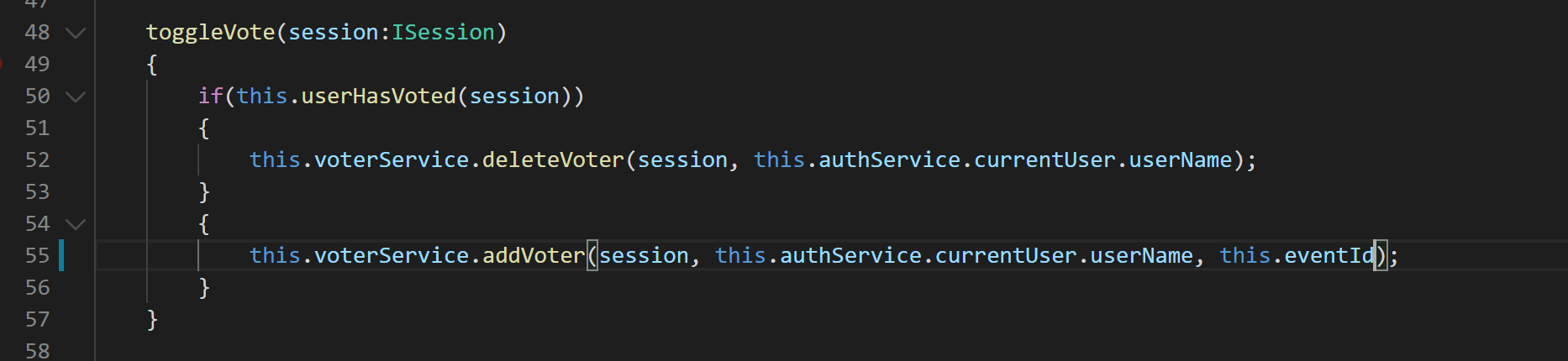
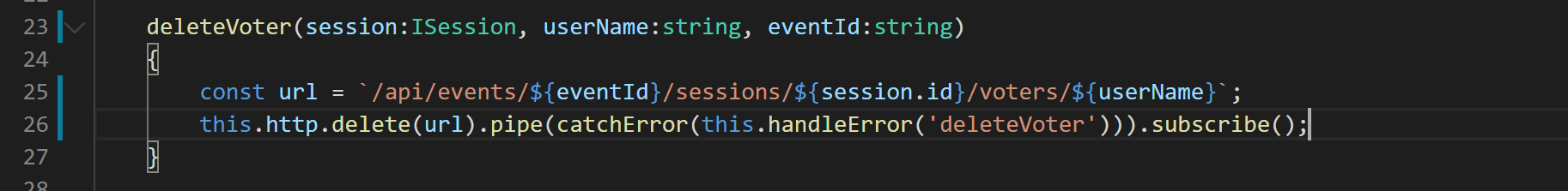
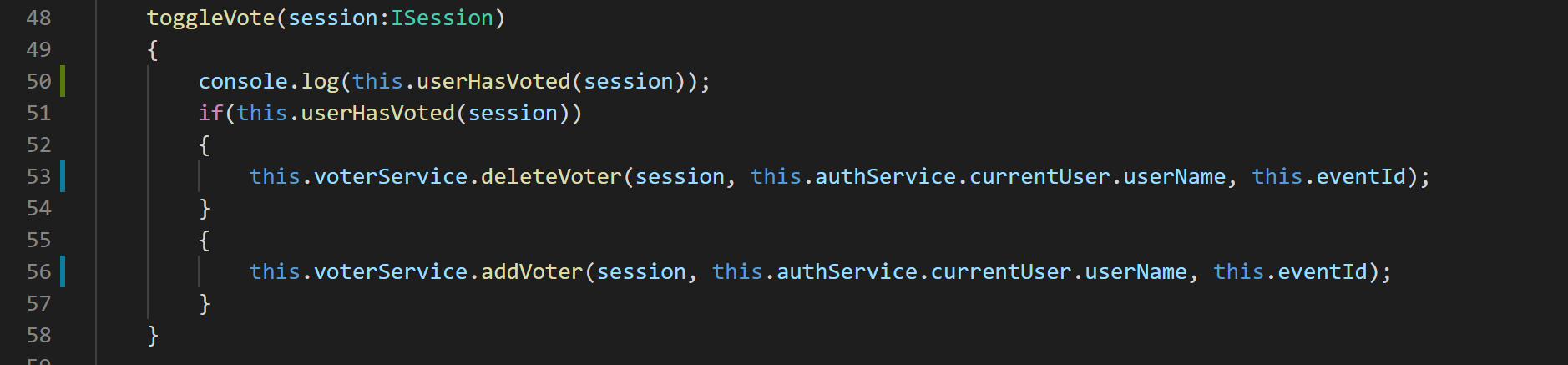
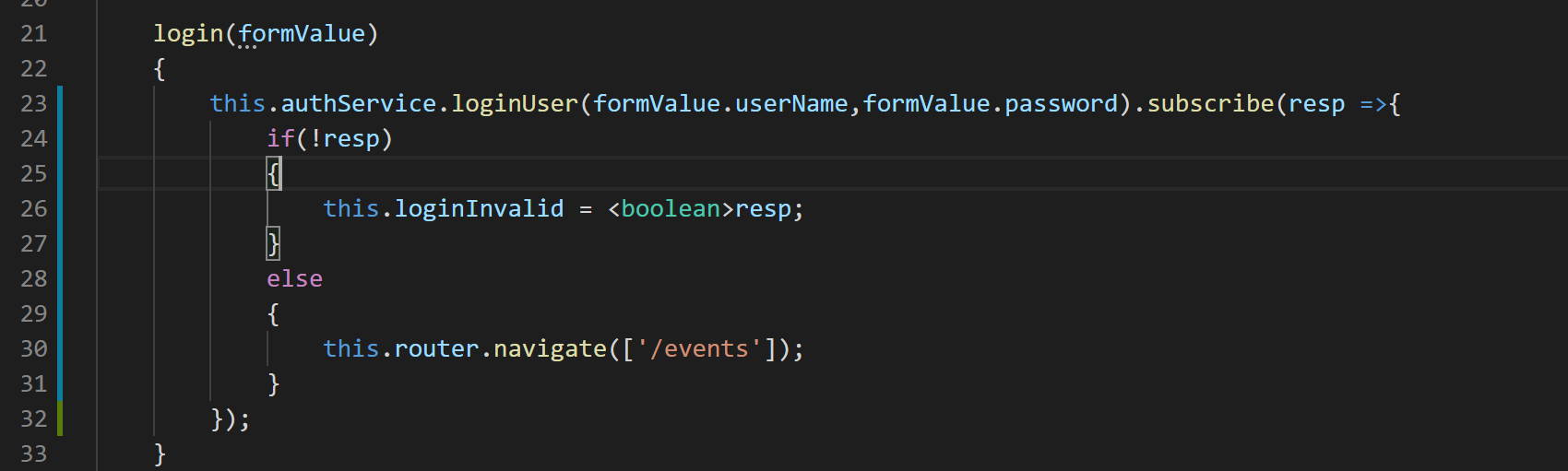
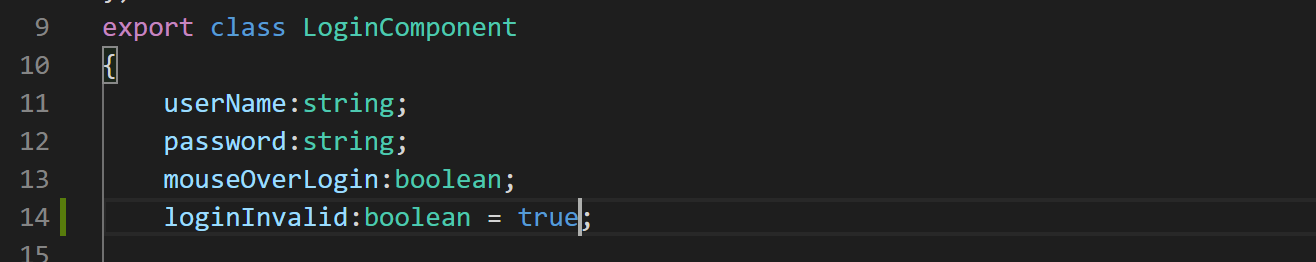
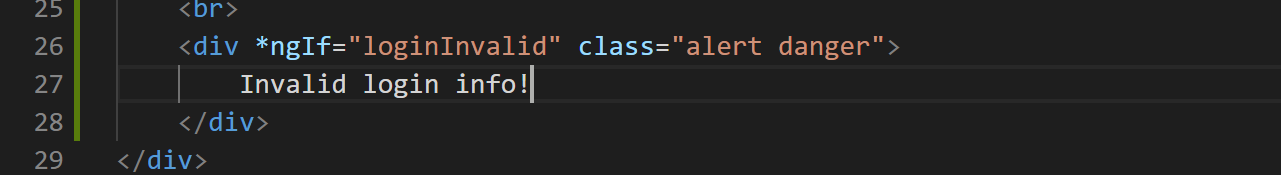
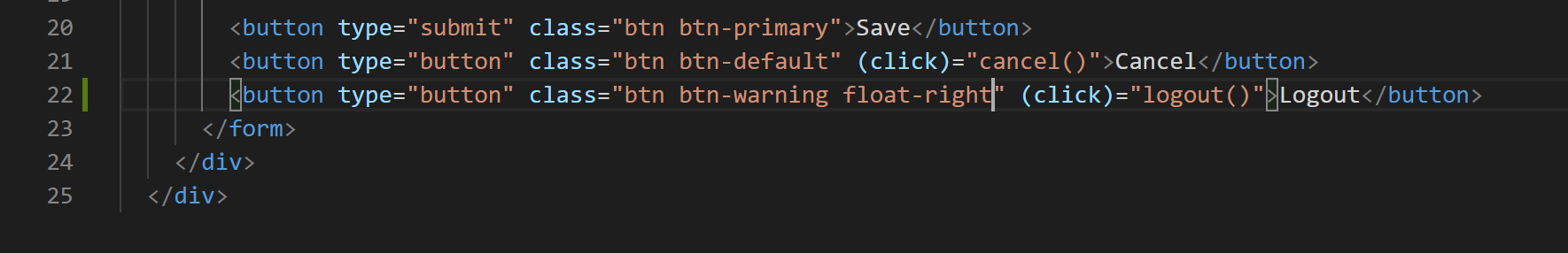
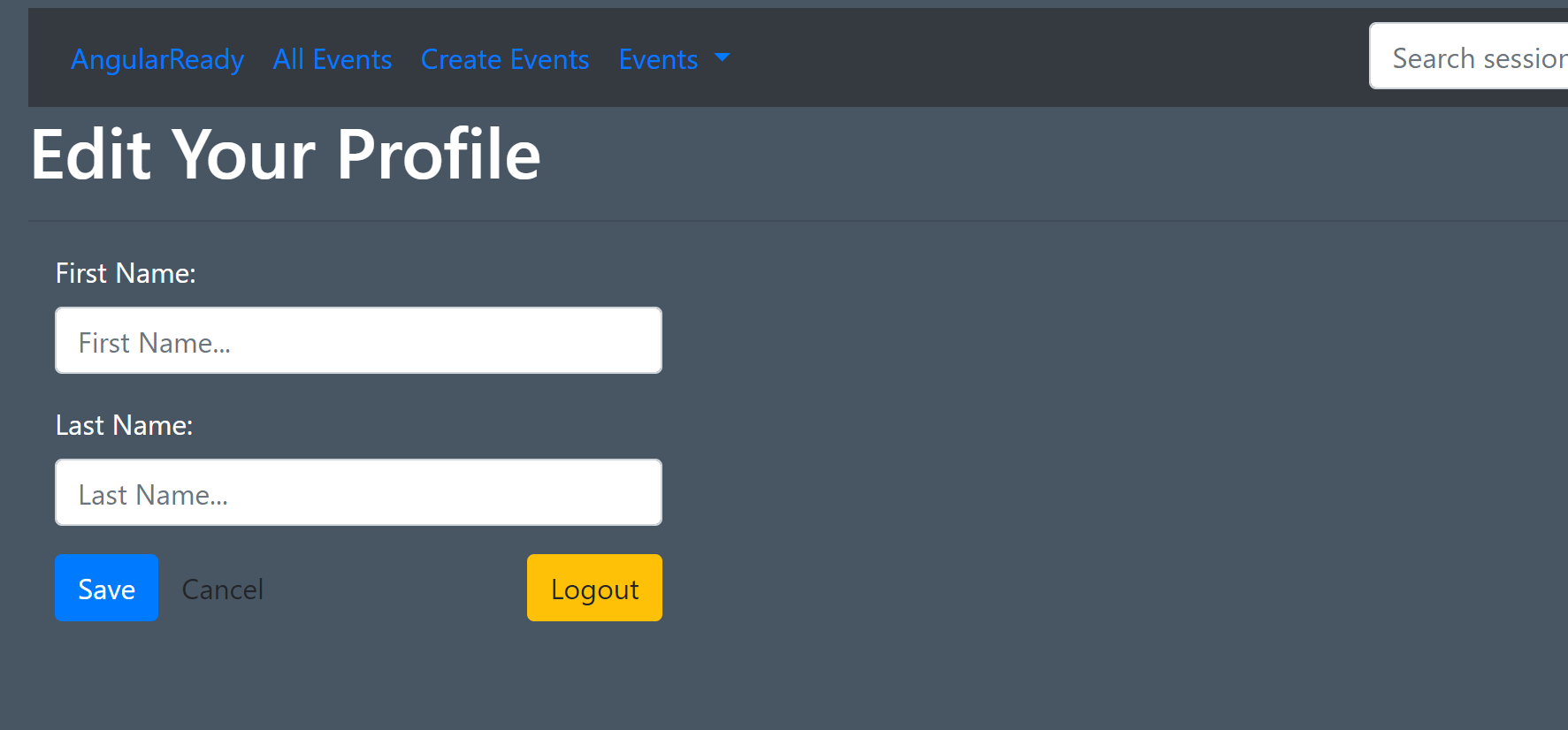
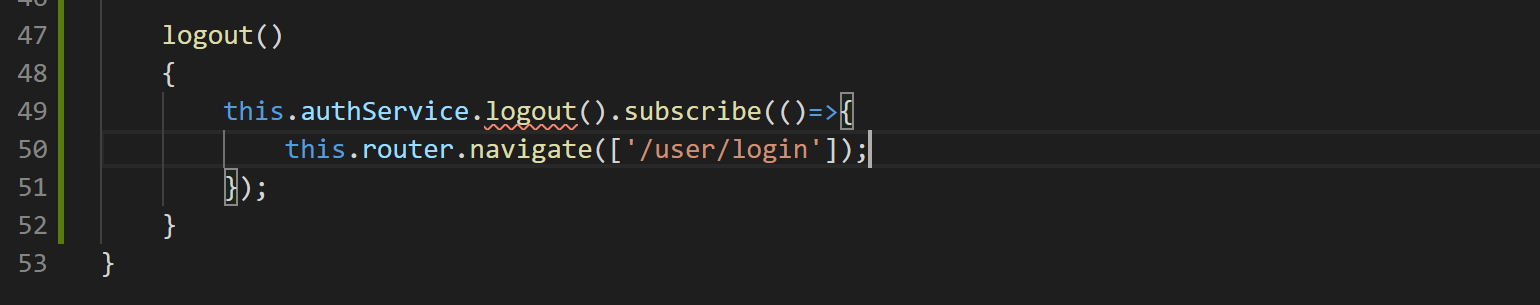
1. In this module we will move all the data to the server and communicate with the server using RxJS observable. Although we can very well use Promises but since Angular by default works with observables we will work with observables. So far difference between promises and observables go one key difference is that Promises returns one value in future and observables can return more than one value in future. Basically, with observables you can work with streams of data.
2. Alright so before we can communicate with a server we need a server. We cannot really code all operations in a server side tech like ASP.Net Web API or any other open source server side tech because that will prolong the length of the course. So we have a quick and simple server written for this application which has enough features to act as a server so lets install that server by using command: ”npm install --save ngf-server”.
3. Now lets add the command to start the server in “package.json” file like below:
4. 
5. Now go to “Terminal” menu in VS Code and click “New Terminal”.
6. In this new terminal enter command “npm run server”. A Windows security dialog box might appear like below depending on you PC settings. If it does click “Allow access” button on it:
7. 
8. You should see this kind of output in terminal after you click “Allow access” button.
9. 
10. Now we need to configure Angular CLI so that when we make a request to our API endpoint it goes and talk to our local server running on port 8808 rather than try to find the endpoint on internet. We do that by adding a proxy.conf.json to root of our application like below. Right click on root of the application and add a file called “proxy.conf.json”.
11. 
12. Now we need configure “ng serve” command so that this file gets loaded when Angular server starts so lets modify the command in “package.json” like below:
13. 
14. Now if you are running the Angular server stop it and start again using “npm run start” command.
15. Now your Angular CLI server will redirect application specific URLs to application server and any URL starting with “/api” to our temporary server.
16. Now that we are ready to use HTTP communication in our application let's import a HTTP client module to our app module like below:
17. 
18. Add the necessary import statement like below:
19. 
20. Now your VS Code terminal might start throwing lot of errors as you press Ctrl+S. That’s because of a conflict with the temporary server we are running. So stop the temporary server and run ng serve command again and it will all work out.
21. Ok now lets browse the application just to check if everything is fine.
22. Now lets start using server communication with “event.service.ts” Currently all our events are hardcoded as constant in the same file. We want to read the events from the server rather than from this file. So lets inject “HttpClient” class in our service like below:
23. 
24. Remove all the code in getEvents method and replace it with code like below:
25. 
26. Notice we changed the return type of the method from Subject to Observable.
27. Now we want to handle any error that might occur while we talk to the server so lets add a generic “handleError” method that will log the error on console like below:
28. 
29. Lets now use this handleError method to handle any error in our getEvents method like below:
30. 
31. Lets now modify the events-list-resolver.service.ts to use this modified getEvents method all we need to do is remove the call to “map” function. That’s because Angular’s resolver automatically subscribes to the Observables we don’t need to call the “subscribe” method on an observable.
32. Now we should be able to fetch events from server. If not then stop the Angular CLI server and start it again with command: “npm run start”.
33. Now you should start to see events being returned by the server.
34. Now lets modify the getEvent method that returns a single event like below:
35. Remove all the code from getEvent method.
36. Copy the code from getEvents method and paste in getEvent method’s body:
37. 
38. Notice we had to change the return type from IEvent to Observable<IEvent>.
39. If you notice we are getting an error in event-details.component.ts that’s because we now need to subscribe to the observable being returned.
40. 
41. Now if you go to the application you will see event data being returned by the server.
42. There is a problem though if you remember we implemented route guard on this route to prevent user from visiting the page if event did not exist. That route guard is no longer working because we are now returning observables. Lets remove that guard and lets now have route resolver instead which will resolve the event before the route gets rendered.
43. So lets start with app-routing.module.ts like below:
44. 
45. If you notice we have used the “resolve” key on route object to specify the “EventResolverService”.
46. Lets create the “EventResolverService now. Right click the “event-details” folder and create a file named: “event-resolver.service.ts”.
47. Lets copy the code for this service from “events-list-resolver.service.ts” file:
48. After pasting the code replace the import statement for “EventService” with this statement:
49. 
50. Now we need to call the “getEvent” method rather than “getEvents” method like below: like below: 
51. getEvent method takes a parameter so lets pass one. The parameter is already available to us from the ActivatedRouteSnapshot. Lets inject “ActivatedRouteSnapshot” in “resolve” method like below:
52. 
53. Now our resolver is ready lets add it providers array in app module like below. Also we will remove the entries (import and provider) for the route guard.
54. 
55. Don’t forget to add the import statement for “EventResolverService” in “app-routing.module.ts” file.
56. Ok now we need to fix the event-details.component to read from the resolved data rather than call the event service so lets go to event-details component and modify the ngOnInit method like below:
57. 
58. If you notice we are reading from resolved “data” from the ActivatedRoute. Just like ActivatedRoute has “params” property for all the parameters it has property for all resolved “data”.
59. Ok with this now event-details component is ready to do. So go check it out in the browser.
60. Now lets modify the saveEvent method to use server API.
61. So remove all the code from saveEvent method and modify it like below:
62. 
63. Now lets modify the create-event method to consume the observable being returned by modified saveEvent method like below:
64. 
65. Now you can go to the application and try and create a new event and it should all work just fine.
66. Now lets fix the updateEvent method to update the event on the server using HTTP PUT verb like below. So go to event-service.ts and modify the updateEvent method like below:
67. The “put” method takes exactly same parameters as “post” so we can copy the code from saveEvent method and change the method name that’s it.
68. 
69. Now lets go to the event-details page and subscribe to the observable being returned like below:
70. 
71. Alright now lets try use it.
72. If you notice the Developer tool bar we are getting an Http 404. That means the Http PUT implementation was not found on the server endpoint. There are two ways to fix it one we add the put implementation and two we just live by using post method.
73. Lets try to add the “put” implementation. So go to folder “node\_modules” and look for a folder called “ngf-server”. If the folder is not visible then close VS Code and open again. Now in this folder look for a file called “routes.js”.
74. In this file copy the line number 17 which is 
75. And paste it just below and change the post method to put like below:
76. 
77. We are not going to change the handler method “events.saveEvent” to save time.
78. Now that we have changed server code lets start the server again by inserting command “npm run server”.
79. OK now if you try to add a session it will work.
80. Let now update the searchSessions method. So delete all the code in searchSessions method and copy the code of getEvent method and paste in here.
81. Lets modify the parameters and URL of the method to hit ‘/api/sessions/search’ endpoint like below:
82. 
83. Now let us modify the query string parameter and return type like below:
84. 
85. Since we have not changed the method signature and return type we don’t need to go and fix the consumer of this method in the nav-bar component.
86. Now lets go to application and try search to see if it still works.
87. Now that all our methods have been updated to talk to server lets remove the “events” array constant.
88. We are now ready to try and implement Http DELETE. So open “voter.service.ts” file:
89. And inject HttpClient like below:
90. 
91. Lets fix the addVoter first as that is a HTTP Post and we are familiar with that:
92. To form the URL to post to we will use string interpolation like below:
93. 
94. Now lets define the HTTP options object like below:
95. 
96. Before we make the call to “post” method lets add the handleError method. Lets copy it from event.service.ts. Make sure to add import statements for ‘of’, Observable and ‘catchError’.
97. Now lets make call to post method like below. Let copy the code to “post” from event-service.
98. 
99. Alright now we are ready to define the eventid. It has to be an input property on “session-list” component. From there we will pass to the addVoter service method call. So lets add the input property in session-list.component.ts file and bind it on <session-list> element in event-details.component.html file like below:
100. 
101. Now lets pass it to addVoter method in toggleVote method like below:
102. 
103. Alright now lets try to vote. Go to the application in browser and try to vote. You would be able to vote but the heart icon won’t toggle just yet as we need to update other methods too. Open the Developer tool bar and go to Network tab to see the API called.
104. Finally lets update the “deleteVoter” function like below:
105. 
106. Alright lets pass the eventid from session-list component like below:
107. 
108. Ok you might have noticed that the heart symbol is not toggling that’s because we have not updated the “userHasVoted” method to fetch the status from server. Its just going to be a simple “get” call so we will park it for later.
109. Ok now lets fix the auth.service.ts file to login using server. So open the file and inject HttpClient in it.
110. Lets now make a post call to URL ‘/api/login’ like below:
111. 
112. So as you can notice we have defined the regular content type header and made a call to our api URL. But once the call returns we are not subscribing to it we are rather tapping into it using the “tap” method and assigning the “currentUser” as we need. Then we are handling the error and in case of an error for example user typed wrong password or user name we are returning an observable containing Boolean value of ‘false’.
113. Now lets update the consumer of this method in login.component.ts:
114. 
115. 
116. We are setting a property called “loginInvalid” in case of a failed login. We will use it on “login.component.html” to show an error message like below:
117. 
118. Lets see if it works. So go to the application and try to login with some random values.
119. Now lets implement the logout functionality:
120. Let start with HTML template. So open “profile.componen.html” and add a button called logout.
121. 
122. Lets see how this looks in browser:
123. 
124. Alright it looks exactly how we want it to be.
125. So lets add the logout method on “profile.component.ts” button like below:
126. 
127. Ok now that we have this in place lets a add the logout method on auth.service.ts which will post to URL ‘/api/logout’ with empty body parameter.
128. 
129. So lets go and test this feature it should be working fine.
130. We have added a lot of functionality on our app.