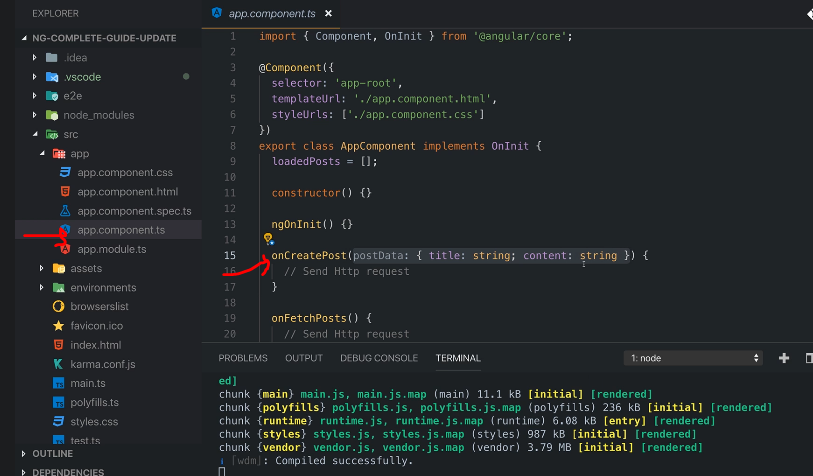
* Narrator: Now attached you'll find this project here.

Graphical user interface, application

Description automatically generated

* It's an Angular project which I opened here a very simple project where we have a forum a template driven forum where we can enter a title content and then send that post here, like a very simple post and a simple block ***to store it on Firebase.***
* And later we also want to add logic to fetch all posts that we stored on Firebase and to clear all posts there.



* But for now, let's go with creating a new post.
* And for that, if we have a look at our code you only have the app component and there in the app component ts file you have on create post which will actually get some post data in this format.

Text

Description automatically generated

* Now let's first of all log that to validate that this is correct and that this works for that I opened the developer tools here and now let's write a new post with some content and click send post.
* And we see that here.

Graphical user interface, text, application

Description automatically generated

* So this seems to work.
* That's perfect.
* Now of course we could reset that form here too if you wanted to, but I'm fine like this.
* *The goal now of course, is to not just lock it here* ***but to instead send an HTTP request.***
* And for that we need Angular's help.
* To be precise, we need to unlock a new feature.
* So to say In the app module we have to add a new core Angular module or a new module baked into the Angular framework I should say which is the **HTTP client module.**
* For that, you simply import HTTP client module here from **@Angular/common/http.**
* So that package name is @Angular/common/http and you import the HTTP client module.
* Now you add that HTTP client module here in your imports array.

Text

Description automatically generated

* Just like that.
* This now unlocks the HTTP client Angular offers in your whole project.
* And with that***, we can now start sending requests in our app component here*** in there we now just need to inject our HTTP client and I'll store it in a property which I just name HTTP and it's the HTTP client, which you need to import from @Angular/common/http.
* So make sure to add this import at the top of your path.
* Now with it imported, we can use it to send HTTP requests.
* And how does this work now? Now let's say here in on create post once we got that post data, we want to send it to Firebase.
* For this, we can use that injected HTTP service.
* And there you have a couple of methods.
* These methods are mostly named like the HTTP warps.
* So you have get, you have delete, you have post and that will allow you to send different kinds of requests.
* Now here, since I wanna store data I'll start with a post request.
* ***Now you call this method and it takes a couple of arguments.***
* **The first argument is the URL** you wanna send this request to, and that is the URL you can find in your Firebase realtime database.

Graphical user interface, text, application

Description automatically generated

* This URL or at least this URL is a part of the URL we'll send the request to.
* So paste that in here, but this is not the entire URL.

Text

Description automatically generated

* Now, on a customer or on different REST APIs you're communicating with you would have clearly defined endpoints like slash posts, ad, or something like this to which you have to send your requests and the official docs of the API you're working with would tell you which API endpoint to send the request to.
* For Firebase base, it's a bit different.
* *You have this starting point URL here and then you can add your own segments after that.*
* *And this will get replicated as folders in this database.*
* So to say, and this looks like you're directly talking to the database, but keep in mind that Firebase only abstracts this away from you.
* You are still communicating with a REST API provided by Firebase.
* *They just translate your path, you're sending it to to a folder structure in your database.*
* ***You are not communicating directly with a database here.***
* ***And I'm stressing this because it is important to keep in mind that you never communicate directly with a database from inside your Angular app.***
* So let's say here we wanna have a note, a folder so to say named Posts.
* Well then we would simply add posts as a segment here.
* And important for Firebase, you need to add dot Jason.
* This simply is a Firebase requirement.
* It's not an Angular requirement, not a REST API requirement.
* Only a requirement by that Firebase REST API.

Text

Description automatically generated

* This now allows us to send a post request to that URL.
* Now since it's a post request, it doesn't work just like this.
* ***However, a post request also needs a request body.***
* Now this post method here works such that it takes more than one argument.
* ***It has at least two required arguments.***
* **The second argument being the request body.**
* So we pass one additional argument here and that could be our post data.
* So the data, we wanna store this data.
* Now very important, ***you normally send Jason data when interacting with a restful API.***
* **And actually that will happen here as well.**
* **But the Angular HTTP client will take our JavaScript object here and automatically convert it to Jason data for us.**
* So it will still send Jason data and we'll see this in a second when we inspect that request in the browser developer tools.

Text

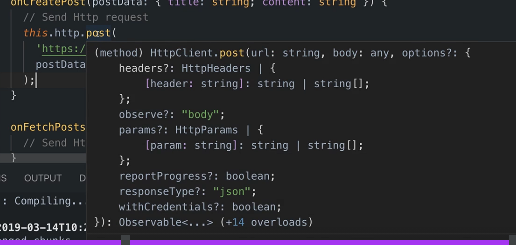
Description automatically generated

* But it will, Angular will transform this automatically for us.
* Now with this we have a request that should work.
* It has all the core data it needs, the URL and the request body.
* And for now, we don't need to set any our data.
* Now let's save this and let's give it a try, shall we? So let's go back to our application.
* Enter some content here and click send post.

Graphical user interface, text, application, email

Description automatically generated

* Hmm, did something happen? Let's check Firebase.
* Well, I don't see any data here.
* And we would see it here.
* Let's check the network tab.
* So in the browser developer tools and the Chrome developer tools go to network tab here.
* You see a bunch of network requests.
* Actually, these are all the files that are downloaded as part of your Angular application and all the JavaScript files is on.
* Let's clear that.
* Let's simply send another post and we don't see any request here.
* So somehow this request is not being sent and that happens on purpose.
* ***Angular heavily uses observables as you already learned.***
* And **HTTP requests are also managed via observables** because they are a perfect scenario or use case for observables.
* **We can wrap them and we can then subscribe to them to get informed about the response and to handle errors and so on.**
* The thing just is, Angular is smart, so to say.
* If you're not subscribing to that *prepared HTTP request* you could save.
* So to that observable, that wraps your HTTP request.
* *If you're not subscribing to it, then Angular and RXJS know that no one's interested in the response and therefore the request doesn't even get sent.*
* Because if no one's interested in the response then it won't send the request.
* So Post Indeed will return an observable.



* As you can also see here, it does not give us the response or anything like that as a return value.
* ***It gives us an observable that wraps your request.***
* And to get access to the response you have to call **subscribe** here.
* ***And this will then be your response data, actually.***
* Because Angular, the HTTP client I should say, will do even more than just giving you the response ***it will automatically extract the data attached to the response, so the response body, and give you that.***
* Though you also have ways of accessing the full response as you'll learn later.
* So for now, let's simply console lock that response data to get a feeling for whether that works and what we get here.

Text

Description automatically generated

* So let's save this.
* And now with the subscription set up and with us subscribing here, by the way, ***you don't need to manage the subscription and unsubscribe because it will complete after being done anyways.***
* And it's an observable provided by Angular for which you never need to manage subscriptions.
* But that's just a side note.
* So now let's simply test this.
* Here let me clear all network requests again.
* Well, actually, let's first of all open the console again and let's now try this again.
* Send a post.
* And now we get a log here, which looks good.
* Let's ignore what's in there for now.
* It's a JavaScript object, we can see that.

Graphical user interface, text, application

Description automatically generated

* But it has a name and then some cryptic key here.
* Let's have a look at the network tab.
* And there we also see that request.

A picture containing text

Description automatically generated

* Now we actually see two requests even two requests to the post and point.
* Now that's just a characteristic of browsers when sending post requests so not to slash posts, but of type post.
* Then they always send two requests.
* The first one, if you click on that, is of type options.

Graphical user interface, text, application

Description automatically generated

* ***That's a different HTTP warp that will first of all check whether the post request is allowed to be sent.***
* *And if that gets a success response it will send the actual request, which is this one.*
* And there you see that was off type post.

Application

Description automatically generated with medium confidence

* This was the URL that sent it to it's succeeded.
* These are the response headers we got back by Firebase.
* And these are the request headers we sent.
* And for example, this one and the content type headers were added automatically to the request sent for you by Angular.
* You didn't add these headers, they were added by Angular.
* And here in the request payload you also see that Jason data that was attached for you.

Graphical user interface, application

Description automatically generated

* You can also of course see the response here which is the response we lock to the consult.

Graphical user interface, application

Description automatically generated

* So this is how we send HTTP requests.
* And besides the general syntax, the core takeaways ***that requests are only sent when you subscribe.***