Stop. We get to. Two, his stuff to the server. Obviously you have to have the right to do stuff to the server. Fortunately you do on your local machine. You often won't but. Post and put and and delete are the the methods were going to use to do stuff to the server. Write it reminder for you guys 'cause this is an important way of thinking about it. It's the idea of hoops. Text. This is all about crud. Create. Read update. And delete. And CRUD is implemented in that order. With a. Host is host. And get. Quit delete. since that doesn't make a nice accurate we think of it as how we do this in a Restful API. So the first thing we're gonna do is go back to our if you've all done your Git, Poland upstream main, and you're ready is go down into your. Module 2. Slash what are we at 12? Lecture. Sure. OK, so you're all gonna go down into the server. This is gonna happen each day. 'cause you get used to it, you're gonna go down to the server. And what are we gonna do? We're gonna do an NPM install. This is the setting things up and then NPM start is going to run the server. Now if you think I I think I I mentioned before that the data for the server is is all right here. OK so. Even though we can create things just to bear in mind we don't have a separate database somewhere. So if you stop the server and restart it again, it's gonna start over. It's like your your vending machine, it's going to read from the CSV file. OK, so that's just to bear in mind. You can create stuff for the purposes of this one thing we're not quite to the persistent data storage yet. We will get there shortly. Alright, so and then now I've done this. I'm going to do my MPM. Start. I'm going to. It nicely tells me what are my resources. This is entirely because somebody bothered went and put their console print line out there. It's not. There's nothing inherent in a server doing this. It's going to tell us what we have. Right. And it's gonna tell us how to get to stuff. What you will see sometimes is in this syntax is it'll do like: ID. That does not mean there's a colon there, it's just that's where you would replace the ID and they they don't know how to tell you that. So then like, that's the the syntax you will see sometimes the other way that you will see that syntax and we will see in just a moment. Is with curly braces around the Ivy, so those are the two ways that you'll that they'd let you know it's a variable that you're going to pass in. So what do we do now? We are going to start a postman. Whenever we do stuff, the first thing you should do is do it in postman. Alright. Just make sure I've got my. So I am postman. I am going to create my brand new. You know what? Let's create a collection just because if you're in postman, create the collection and just call it something like you know. Al. Schombing APIs. You can call it Donuts for all I care. It doesn't really matter. She's gotta have a name so we know where to put stuff. Right. And so now when I create. The advantage to doing this. Don't save. I'm just trying to get rid of my. Some of my earlier ones. If I want to create a new thing right, I'm going to do a get still be a little bit like we saw yesterday. I'm going to say, yet I'm going to put HTTP: slash localhost. You can't really. I don't know whether you hear it, but my cat has decided to go nuts behind the. The blinds that are in the window in front of me, and so there's a sort of constant rattling as she runs back and forth. I don't know whether there's an animal out there, but, you know, if she jumps up and down, then there's an animal. OK, so localhost: 3000. And remember when we started this up, it told us, I think, yeah. That the home was Localhost, calling 3000 Arthur and I mentioned that to you, by the way, is that once you guys get to start to writing your server. Then you can actually wind up where it starts multiple servers, and they'll each be at 3000 and 3001 and 3002. So you'll you look sometimes to see what is it? Think that the court is so you know where to call it. But for now? We've got this and we're going to say, what did we say? Hotels? Right. Go out and make sure that our get will work with hotels. And. We are then gonna do well, that's hotels is great and we're going to see hotels slash two, you know and it's gonna give us a particular hotel. What we have now added on top of the hotels which we had yesterday is reservations. So if we check for reservations. It's gonna show us. OK, these are the reservations we have. I'll take a look at what some of these are. The pattern you will get used to. OK. I I will say that one thing I object to in your reading from last night for those who who finished your reading and absorbed every little bit of it. One of the things that that I will tell you that I don't like is they say there's no standard for rest APIs. And that's really not true. There's no official standard, or the only way you can do a rest API. There are very good standards done with what they call a useful called Swagger is now called Open API of an entire set up about how you're supposed to do a rest API and. Part of the rule that of a rest API that we ignore. For these examples, it's a good rest API. When you go and get. Like an array of all the hotels or array of all the reservations. It should in each one of these have the actual URL that you would use as an endpoint to get to that specific one. A well formed rest API. Should allow you to traverse it without programmatically, so that when you get a hotel you can then go and there is a specific way to identify this has been. It's called a trap, which you may see later on. A traffic is the URL for that entity so that you can drill down into the entity without knowing. More about the API, all you have to know is the home of the API and from there you can get all the way down into all the things that we don't do any of that right here, but just so you know, there are. There are actually rules about a well formed rest API. It doesn't much matter for what we're doing here, but but you will see them out in the real world where you can always know we have to look at things like. Well, I can guess that if I'm looking at reservations and it's got an ID. Then there's a darn good chance that I can put in that I ID. And get that particular one. But I shouldn't have to look for what is the name of it and do all that stuff. There should be an HVAC which tells me how to get to it exactly be. That is me. I'll talk to the 13th team telling fish you know up their game here. Emily. Alright, So what do we have? You can also do. So and this is quite common. Remember we. We saw a little bit of this yesterday, but if I have hotels I guess with the star. What was yesterday? OK, so with the reservations, I can go to reservations this way or I can go to reservations and I can do hotels. Which hotel is this? Is this was three, I could say hotels 3 slash reservations. Right, this kind of sequence is quite common. I can either get to all the reservations and and look through it that way, or I can go because reservations are specific to a hotel I can get at it through this other endpoint. Alright, so having done all that now we get into the. How to what else can we do here? Well, now we're gonna play with. Bees, how do we can we post something or put it or delete it? Or Amber Post is create? And put is ugly. And delete is get rid of it. So. Let's start by let's just see if we have a thing like this and we have reservations. Let's go back to the one we give before, which was reservations. 706. We can get that. What if we say and let's just say this and you see? I'll call it. She. Is your vacation 706 OK? I will save it in here. So now it's over here and it's saved. So when I came back, I could say, oh, I could run that over and over again and it would just click on it and it would put me back here. I said that's great. Let's try creating another one. And when used the same endpoint. And we're gonna say. Now let's delete it. Right. It uniquely identifies a resource, so I could delete it. OK, so I'm going to go ahead and see it and send that. So let me save it first so that I can. Right. So we're gonna save this. We're gonna do that same thing. We're gonna call it delete. Reservation. 706. OK. And then let's try this because we have this, let's do one more, which is. We will. To a update. So we're going to do a quest and say let us go back to our well. OK, now we're going to actually change the values in it. So when you're gonna put or post something and you're going to use JSON. So always go to raw. Raw means it's not one of these other special things, and once you say that, it is raw. You can say I I can say it, it should look like case. There's nothing enforces it to be Jason, but I can make it look like Jason, OK. Yeah, I don't know if I missed it, but why is the localhost different? Uh, it I think it popped up from a previous one. Thank you. OK, actually make sure that all of them have this. In fact I I will. OK, so I'm going to change from Dave Smith to David Smith because David is very particular about this. Just to sort of see what happens, and I'm gonna save this. I'm gonna say, OK, I'm gonna save this as. Update, 7062 David. The reason I'm saving is just so you like. If you can't follow along during the lecture today, just play with post family. You you always save something into a collection, but the point is that it's then very easy for me to go back and say, OK, I'm gonna look, I'm gonna resend this. I'm gonna say, OK, here's my information. It's now being Smith. If I go to update and I and I know my body is in there and it's got all this now with Steven Smith and I send it. I'm going to get some response back like we're familiar with, and that response is going to be it actually will show me the whole thing, but don't worry about that. So servers don't always do that. The thing that matters is that 200 that 200 says I accepted. Your update. Alright, so now what should happen if I go back to this and call it again? Right. I now have David. I updated the record in there. Now let's say that I say, well, that's great, but David cancelled his reservation. So I'm going to delete and again I'm getting into 200. Almost all the time, what you're looking for is a 200. Again, it shows you content which is really meaningless because there's nothing to come back, right? If I now go and say, OK, now I'm going to go out and I'm going to try to get my reservation. And I pull it up and it's gonna give me a 404 not found. If you remember anything from 400 up to 599 means something went wrong. If it's 400 to 499, it means something went wrong and it's probably my fault. And it's it's 500 to 599. It is probably the server's fault. So that's sort of the thing you remember before 04 is a really common one. It's just the thing doesn't exist anymore. And similarly if I went and I created a new one and I said my. OK so. If I wanted to just get my reservations. Add. I don't wanna say, oh, let's say that. Get list of reservations. I'm still in the same workspace, so I put in there. That means I can if I've done this and I try this and I go to my reservations and I go out and I look. I don't know exactly the order I think they're ordering more or less sequentially. I think I just got rid of, you know, one of my only three reservations. Part of the reason I'm showing you this also though, before we get into the Java is to divide. Restart. If I understand a lot of the logic of this server correctly, which is always in doubt, and I go get my list of reservations. OK now I will have 706 again. The nice thing about doing something like this is that you can say I can develop a series of steps a little bit like when you had a a SQL file and you could say I will execute these things in order in order to create my database or whatever. If you wanted to set up more things in your database. You could go ahead and create something that would post a bunch of, you know, new entries and and and whatnot then. end you

Are you did it again? Yes, absolutely. All you really have to do you go into here to your server that's running, and if you do control C, it will end the server. And then if you just do your NPM start again. It will restart your server. It does not say Localhost, only 3000. And you have a problem and you have to. Kill the previous server. Answer the question about where all this information is. Let's just say you had a black box like like just an NPM start. You add didn't know it was like in the context of reservation or hotels. How would you find out what is in there? Where would you look? Well and again a well formed API you would look at the. There's always a published endpoint. And that's just how you get there at all. And for the for this particular server that happens to be just at the root of localhost at you know port 3000. Some URL but but and when you get there, if you do nothing else but let's let's actually look because it's an interesting question. What it does do and I display obviously don't know. If I go out and I create new one. The only problem with saving things, by the way, is if you you sit here and say, well, I'm just going to change this, you're changing the thing that saved and confusing. So you have to remember to go vote. Let's create a new thing. Um. OK, so I'm gonna say my. And I'm going to do this. What should happen at this point is that it should give you some information and what it would normally gives you is what is the start. This doesn't happen, it gives you. It gives you something else. I don't wanna give you. It's it's. It just says Jason server. I'm gonna mute. That sits down here. OK. Yeah, let's let's try it out then. OK, let's try this out in the browser. It's hard to see in here. If I go to my browser. OK. Congrats. You're doing whatever it's gonna tell you. The resources there usually is some way that if you go to the the endpoint of the server the the initial end point, it will tell you something that might be possible. That answer your question is, I'm not sure that. Like that's how I would know if I didn't know. I I have no way of knowing, I mean like. And also not inherent in in the command line thing where it says slash retail slash reservations. It's also like what's in there. And if we go back to that and just. Well and right and and and that is that the Jason server simply went and read the stuff that it was doing the Jason file that it was reading and and notice it is data generation dot JS. That's because let's let's do this thing. Let us say what is data generation. My right place here. Yes, I am. If I go to. Not sure Sir. And this is worth noting 'cause. It's sort of like the way this stuff works, and we'll open it in Visual Studio code will bring that down here. Let's see if I can actually see it. Probably not. Never mind. The whole thing down, OK. So. This is a jigsaw file. And it has things like hotels. It has the data in it, but what really happens with Jason server is it looks what are the top level entities. And then it just prints out that those are the things nailable. If you wanted to create your own data in here you could and use Jason server and it's actually used by lots of people for different kinds of things. And then it would just, it would take all this and it would. It would then make it available. But if you wanted to put in a new thing after reservations, you would create a new XD, give it an array and call it something else. And then when you started it up. James Server has no special logic on Eric, just says OK, there's another one there and it's called, you know, Donuts. And these are the Donuts we serve at our hotels. And you might, you know, you can set up all this stuff like that. Alright, so let's go back to our code. I wanted just so that you because it starts getting a little more complicated. I just want you to understand where the different pieces are. This is the main app class. Right. And what is the add class gonna do? It's the very first thing. It's gonna have a mean which starts up the app. K. But. Usually have a run method and has the menu. Those things you're somewhat familiar with. One thing I wanted to comment on to you on is that. When you have a method that is just called handle. Handle something or other. In computer terms, that is what we tend to use when we say, look, we could cram all this stuff into this method. But I just want to deal with part of it at a time. It has no other. It's not going to return a value. It's not going to do anything logical. I just want to take some of this code and put it somewhere else. We tend to say handle this thing. So it doesn't feel like a very meaningful method, but what it really means is I just, I'm going to handle where I list the hotels somewhere else and I'll just call that thing again without any arguments, without any, sometimes without argument, but without any parameters or return value. And then I just looked down for where it is handled list hotels. It's just a way otherwise. You could just have put all this code right in that menu selection, but it's sort of separates it out so it makes them short a deal with. When you get to handle list hotels, we sort of looked at this yesterday, but we're going to look at it just a little more. We we have a list of hotels we call our hotel service, which is then going to be in here. Our hotel service is going to do our rest staff. The service, when we have our model is hotel. Our hotel service is then going to be the thing that interacts with the hotel resource, so the hotel service is going to actually go out. It's gonna do the let me request information from my API and bring it back in. In my app, then we're sort of where where we were. I'm gonna get my list of hotels using the hotel service. But my hotel service is all about interacting with the API. So then I take my list and then I pass it to my console service and my console service is the thing that handles interacting with the the console. So then my console services kind of have a print hotels and it's not equal to know. All right? And then you're gonna go into your console service and it's going to do interactive stuff, print hotels. Tell menu, print hotels and that's going to then do this part of the logic which actually prints out the hotels. The way we see it. But this separation of what is the service? What is the thing I'm interacting with? It's a really important part of the way you will. Build programs from now on. If you had a web thing the the Web Part is going to be a service. That console is going to be a service, that API going out to an API resources of service. These are all services. The model is always going to be that the plane objects themselves that we're using in our system. Pogger Plain ordinary Java objects, right? And they will look very much like they did in. Ah, you know, for SQL and once we get to interacting with a web service and SQL amazingly, they'll use the same object because that's, you know, that's what it is. This is our hotel. There's also some util stuff which utility stuff is OK. I have some other stuff that I don't really know how to fit into any of this. It's kind of a library of things and this would be like when I'm logging stuff, don't worry about it for now, but just so you know, that's why there's a utility as well, OK. So we're going to look at this. We're going to say how do we do this thing that we wanted to do, which was we want to, let's say we were lowest something. List isn't great, let's do how do we add a reservation. Because add our reservation is gonna have to handle some of the same logic that we were just doing in posting. Actually the one thing, you know what let's very quickly 'cause mostly I want to copy it over. I'm going to say you know how I did this update 706 I'm going to add a new reservation just so we we see what it looks like. And guess what? It's gonna look just like this, but I'll change the data. So I'm going to say I'm going to create a a new one which is going to be OK. It's gonna be HTTP: and it's gonna be the same things. OK. It's gonna be 3000 because I was using. A different. Thing there pretty OK when you create write this the end point. Is the collection of resources. When I updated it was 706. And the reason is because when I'm updating I already know the resource, so I should just be dealing with that resource meeting. I am usually creating. I am create creating one or more of the resource. So this is the end point you use and post is what you're going to use for it. And then you're going to give it. Either a single object or an array of objects. For now, we're going to do a single object. Very similar to before, we're gonna go to raw body and then raw and then we make it Jason. And we're gonna make space for it. When I put it in here, and we're going to just put any old thing here, we're gonna, we're gonna say it's reservation 715. Hopefully, we wouldn't be this random when we're creating it from application. We're going to say is for Hotel ID 2. You know, we're gonna say it's. The answer is staying at the hotel and will give the same dates. Beyoncé has an Entourage 20, so I guess that means you're 21. Please be asking herself is gonna stay there. OK, so if we go and we try to post this. Now what we see I'm just this is important is. We get a different return code. 201 means that the thing was created. It was not. It's a brand new resource. It will tell us the resource because Jason does that often. They won't, but sometimes they will. And the reason they sometimes will is think about it for the exact way like when I just did that I gave it what the ID number is. But think about in SQL it's quite likely if I'm creating a new resource. I'm not gonna tell it what the ID is. It's going to come up with a new ID for me. Right. So given those circumstances, it's usually going to return when you do a create, it's gonna return the full thing filled out. So I might not have given it. I had the ID. And it would give me that. I don't honestly know what we have. Let's try it. But what happens if they don't give the idea here and I create yet another one, right? OK, it did. Yeah I did create 716. So yeah, there's logic in there that will say if I don't, if I don't give it the ID or come up with an ID. And so my object is going to have that. Another quick thing to understand, notice that this looks different. The 80s at the end, Jason is not ordered. There's no sequence to it, and similarly it is not ordered. When you when you use our Java code to turn it into an object. And we did that thing that we did yesterday. I don't care what the order that these come in. I just I'm doing it by name, so I don't care. I'm gonna go find them and I'm going to put them in the right place. Matter what order they come in. So we just did all that. If we go back to our list of reservations just to sort of prove our point, here we go back and we have. Those reservations that we used to, we got we have we haven't changed David Smith. So he's now Dave Smith again 'cause we restarted and we have our two new ones that look very, very, very similar except that they have different ID's. For the sake of making sure that I don't think that matters at all, but like I'm going to try to do everything through my application, I am just going to go back and do my quick, you know, restart just so that I'm back to where I was. Now we look at our code. And the way you do a thing about what we did yesterday, we should have some in here. If we go to our hotel service. Um. We get to our hotel service and we did. List hotels or whatever. We did the get for object. What we're going to do is. When we are, if we are putting something, we're going to use put. And if we are creating something, we're going to do post for object. So let's look at an example of add reservation. This one is already filled out. We'll do this first, and then we'll fill out one of our. Add reservation. It's got a new thing. It's got this HTTP headers. All that that is is a way for you to set what you saw over here in postman where I could. Set some of these headers if I needed to change a header. If I don't specify one, then it's gonna seum that it's there aren't any, and mostly I'm going to want to create one, so I'm going to when I post something, I'm going to want the headers and the most common thing that I'm going to do is I'm going to set the content type. And this is where it gets a little tricky, because this is where the rest template stuff merges with the the. Like that whole logic of of of taking an object and turning into Jason or kicking into Jason, turning into an object. That's where this starts to come with play. I don't think I talked about it much yesterday, but it was new reading the process of taking Jason and turning it mapping into an object is deserialization. That's what it's called. You should know that because, again, that is one of those. Somebody may want to throw out at you and interview. They may say, what about deserialization and serialization? And all that means is going from Jason to a Java object or vice versa. Hopefully you remember which direction which is deserialization is going from Jason to the object and serialization is going from the object to Jason. But even if you don't remember that, they probably won't care much if you least know that you're going back and forth between Jason and a Java object. That's that's most of what they hear about. The reason why I tell you, by the way, what they care about and don't care about it is because. It. Most people are their own biggest enemy in the in an interview they are the ones who are scared and. And checking themselves for answering a question wrong, and they all these things which which the interviewer may not care about at all. So partly try not to. They say that the biggest thing you can have in. In an interview. Is confidence. The most important thing you can have is confidence and the second most important thing you can have is humility. Which are weird 'cause they kind of go against each other. So the question is, how are you? Just be confident that what you're answering is the best you can do right now that day. Don't worry about what you could have answered another day or there's a better way or anything else you're doing the best you can to communicate what you know. And always be aware that there's stuff you're not going to know, and they will in fact throw questions at you intentionally. And you won't know. And it's OK. There's a reason why they're doing that. If you, if you were out and you were trying to to map an uncharted land. Part of what you do is you go until you find the edge of the of the Uncharted land and start like OK Now I know where the edge is, right? And then you start building in from there. Well, an interviewer is trying to find the edge. Where is where? What do you know? They don't want you to know everything. You know everything. Then you're obviously not like a beginner developer, but they just want to know where the edges are. So sometimes I'll ask question intentionally to find out whether it's beyond your edge. And that's OK. Don't beat yourself up. You know what? You know, you're all going to be do just fine in these, but that's so. Anyway. Little firefield there, but here we go. So you create a headers. You set whatever headers you need to set, and usually the only thing you're going to set for for the purposes here is going to be. You know this this line you don't have to memorize it, you can go, you know, find it and and you will soon copy and paste them million times 'cause this whole thing is pretty standard. The important part here is that like a collection or something else, the HTTP entity has one of these diamond brackets and you put in the type of the thing that you're doing. The object. This is reservation. If it were hotel, it would say hotel. It is the your class object, OK. That's what you're creating. You're putting in here. add then your new is gonna look yeah it's gonna be whatever the thing you already have the thing that got passing of that This is what I was talking about. OK, so you've got the HTTP headers. Alright so. And it's got this one line that is the line trucking. You're going to create your new HTTP editors, and you're gonna put it in your gonna set the content type to Jason. And that's how you do it. And you don't really need to understand what or how or anything else. That's just how it works. But this is the important part that the HTTP after entity Van where did you copy that from? Other scratch pad where I have lots of code and stuff that I can. So. Here we've got we are creating our HTTP energy envy with the square. The diamond brackets for the type of thing we're doing. It would be like a list of reservations would say you know arraylist and then whatever. And this is. You're just gonna use these later on. We may create some new headers and for each one of them it's gonna be headers, dot, set, whatever that type of header is, and then the the value put in. But mostly this is what you're going to use, and it's just OK. I've got my behaviors now. I'm going to do a try. Catch, OK. And I'm going to do try. So now I'm going to say. What we want to call it. Before we return. We are returning a reservation. So. I am going to create my reservation. I could have done it first, but I'm just going to create my reservation. And is going to be called returned reservation. Has returned as passing one in as well. OK, so I'm gonna do and I'm gonna set it to null. Charlie, we don't have one yet. We're gonna try to get one. So I go into my Tri loop. And I'm going to set that return reservation. Be closed. This part will be familiar. I'm going to use my rest template. Instead of doing query for object or I'm going to do post for object. Post for object. I'm going to do my API base URL like before, so all of this is very much the same thing we just did. And it's going to be reservations 'cause remember when I showed you in the over here? The the the. You always use the endpoint of the collection, so that's all I'm doing is doing this in Java. So I put my reservations now this is where. The first argument is going to be that endpoint that you are lend point should, should reservations be plural. Yes, it should. Thank you. OK. The next thing I pass in is this entity that I just created. This HTTP reservation entity. I'm gonna pass that in. That is the when I talk to you about the request and response, that is the whole bundle together. It it is the request. Right and. Well. it's hard to read someone you enter. Start. I do hate intelligence. So now I have that and then the. The next thing that I'm going to pass in is and this is very similar to what we did with the query for object is. I've got to tell it what class it is. Now when I talked a minute ago about serialization and deserialization, the way that HTTP template knows too. Take this reservation. Or two to post this. And and deserialize it. Get it right. I'm sorry. Serialize it from the Java object that is there into the. Jason, which is going to post the way it knows to do that is that I tell it what the class is. If I gave it the wrong class here, which is quite common to do. Then what I'm going to do is it's going to try to serialize it from that other thing. That's not gonna work. Thank you. Alright, so that would catch and this is where this is how we handle. Errors that come through. Through our API. You catch the thing you catch is the rest client sponse? List client responds. Exception exception. Toggle call Eva 'cause. That's all I have. All right. So that is going to be. If I have an error then I and I want to know what that error is. That I can call and this is where. Again, don't worry too much about the exact thing I could do a console writeline, you know thing here it doesn't matter what I'm gonna use is there longer because that's a better way of doing it. So they're they're trying to get you used to the idea that everywhere you go, they're going to have some kind of thing for logging errors. You just learn whatever the syntax of it is. In this case it's just basic Blogger dot log and give the strength and the string that I'm give it is E dot get status code. Wait. Yeah, I guess I'm gonna do this first. Applause.: stop this. Text. Well, I'm going to say is if I have an error, I'm going to log that error. OK. That's the client response. And you have to just like, yeah, these are the two you're gonna deal with most, which are the rest. Rest note resource. Access. Exception. He. And it's gonna look almost exactly the same. It's just gonna this one has is simpler basic logger. Dot log. I'm just gonna be eat out again. Message. Messenger. This first one says I actually made my. I made my rest request. And I did something wrong. So like maybe I didn't have authorization, maybe I wasn't allowed to post new reservation. Right. That's going to give an error and the status code is going to be instead of like 201. It could be 404 or whatever else. It's gonna be a status code and then there's always a message text which comes along with it, which pretty much says the same thing, but it might give you a little more useful information. I know what the server has. So this is your I have my rest problem problem. The resource access exception is basically saying I wasn't able to get out there and make my rest request. All it failed for some reason. So. If I make it through here, if I get past my thing, what does that tell me? It tells me that my. Try worked. And I must have a returned exception. So our group returned reservation, so now I can return that return reservation. So again, think about this in terms of what we've been talking about before. I'm going to move this just 'cause I think. I I always liked the idea of just putting at the very top, OK? i have a method supposed to return to reservation very thing i'm and then I try to post it using this. So that's my logic. If it fails, this is how I handle a failure. By the way, you could. And we probably will in the future. You could put this trycatch around a get as well. Right. Because you could have a client response exception. So in the future as we build this, I just didn't want to complicate things yesterday. But from now on we will put this around a again as well. It would just be inside here, you would have that query for object. Ben, could you add comments into some of those as to what you're doing with what it does, if you could please? Excellent idea. So you know. Create the thing reservation. By the way, just as a note here. I'm passing in a reservation and then I'm passing out a reservation. The reason why they are different is that that example that I showed you in postman where with the second one where it created the. The one before post right there. Where it created the ID for me, the one that got passed in was the was the body that I sent into the the message. It looked like this. It might be that all that has changed is the ID, but there might be other things that are changed as well, and an example of something that might change is there might be a stamp about when the thing was created. And I don't I can't set it in the first part because I I don't know the exact timestamp, it might be in here. So the reason why you will always do this this logic in Java where you pass in an object and then return that same kind of object but not the same one. Is because when you post it's gonna return the stuff that that that was newly created. Again it might have a different ID, it might have other things that are different about it, but it's the same type of thing anyway. So that's where, yes, yeah. Real quick question in doesn't matter the. Order of like of these exceptions like are they on an equal level? Because in the reading they have on the other way I just wanted to is the progressive flight. Interesting they are. If you have two different things that are basically the same level, and in this case they're basically the same level, it doesn't matter which one you do first like. It matters that you have a more specific one before a more general one, but in this case you they're not, neither one is more or less general, so it doesn't really matter. Yeah, do it in any order. For this particular case. Whereas if I had another catch here, which was just catch, you know, exception, I would have to put that last. But I'm I'm never gonna get resource acception is not access. Inception is not a subclass or a you know some type of of of rest. Which is when you need to to do that, but it's a good question. Package. Errors. and it's in here. That should bold put these in all of them, but that one should layout the different places and what everything means. That's good. Thank you. Alright. OK. So that was in theory that is our ability that is our our thing. So let's try running the app and see if we can. Add a reservation. So we can list our hotels. We did that before. I'm just getting used to what it does. OK we can list reservations for hotel, so we're going to do. The hotel we're going to do is that Cleveland downtown doesn't have any. There are no reservations. OK, well, then we should add or reservation for that. So let's create a new reservation for. Hilton downtown and we should enter it as a comma separated list containing the hotel ID, the full name, the check-in date, the checkout date, number of guests, blah blah. OK, so when do this I'm going to say the hotel we want is 2. Made show with spaces, so I assume we we can allow spaces and we're going to try our Beyoncé example. OK, we're going to say the reservation is for tomorrow since. Why not? So it is 3 slash 1 slash 2022, that's one that's today. Yes, he's not coming until tomorrow. OK, the checkout date is going to be 3 slash 5 slash 2. We're going to have 21. Yes, OK. So if we did that right, this is a very big if. Now when we list the reservations for the hotel. That too. So we're going to say. I will do it for the Hilton downtown. OK. So there's one reservation. It's ID 707. I could guess because the previous one that we had was 706 because remember we got rid of the the two that we created ourselves. So it got the next number. So it returned that. That's why this is different than what I gave it. Turn it is 2 until ideas. Two Beyoncé dates 21 when we have created our reservation. So now if we want to. I did the create first because it is in some ways the most challenging. But let's take a look. Let's actually exit so that we don't. Stop. Exit. I restarted it. Alright, so let's update a reservation and it's gonna look. Quite similar, OK. When we update the reservation, we do not return a new object. The assumption is that when you update, I don't need to return back a full object because it's just did it update with any value that. I will tell you and if you can return an object, and sometimes people do because. There are things, as I say, the latest timestamp. There are things that can change, but generally speaking, when we update, we're not going to do this. So I'm going to model the thing we would normally do. Well, OK, So what do we normally gonna do? We are going to update, we need to. Just because we're returning a Boolean, let's set a Boolean. And we will call it, you know, turn value. We may or may not need it 'cause. We may just do it by the fact that we return true false. We'll get there. OK, but let's say we do. We're going to say yes, do our. Package up. Are. It should be request. So what we need to do here we need to do that HTTP headers. Equals headers. It is new. Cannot type today. OK, new headers. That's really all it is. We're going to say our headers dot set content type. We're going to put in our media type application, Jason. It's again, all very helpful. Intelligent. It's gonna tell you the stuff that you're normally going to do. It's it's, it's easy to do. This is the one that is a little harder to get right because you have to make sure of this. We are still dealing with reservations. Add. Entity. Always calls it. Actually it's. You've got two entities. Would be entity 2 or something like that. But everyone always calls it and me and say new. Which is. Fishing. We need to pass in the. Argument that came in to us. The object which is going to be serialized into our Jason, so the updated. Reservation. And that is so not the way you focus razor. No, I can't. I'm. Vision. OK, there we go. And you passing your entity. Sorry. Passing your headers. OK, so I have now packaged up the HTTP request, so now I'm going to do my try catch. I'm going to say my. Try. Say rest template. I'm gonna say put because pot is what we use and after that it's gonna look similar. But here's the one thing that is different. Oops, not there. Oh. I called my API basting. Yeah, but my base value in I'm going to put my reservations. This is like when we were typing out in before. I need to then pass in the ID. Because I I need to make it part of the URL. When we update, it's not. Even though it is inside the thing, when we update we need to have. What is the actual resource for updating? So even though it's in here, we need to put it in here. Updated. Reservation dot get ID. OK, so I will do that and then we're going to pass in the entity. How much? Henry has reservations. We don't need that. Don't need to put in the class for this. Might catch whatever I'm going to. Say whatever. OK. And then return false well, where should I? What should I do here I could say. I really should do is I. Should I? I can do two different things here. What could I do? I could either set return value. Return value equals true and then down here I can return return value. That would follow our normal pattern. Or if you just don't want to do that for whatever reason, I could just return true here and not save a Boolean and then return false here either one is OK. I'm gonna leave it like this because that's, you know, but that's. Because at the end of my try catch I don't mean to do anything else, so I can just return. I want the only advantage and I will mention this because I'm trying to get you to think about things like try catch is is if I had a finally clause which would do always execute, I would not want to do a return. Hi so this should allow me to update my reservation OK. And so I will run the thing again. I will actually look and see why is nothing working. Look at my problems and it will say explicit type reservation can be replaced with whatever. That's saying. And I really need that now. I don't know why it complains of me, but that's not that was just a warning. Anyway, I couldn't be ignored that, but I do need is I need this. I put my reservation back in. I like being explicit. OK, now I'm going to run my app. And this is the same kind of thing you're going to do for homework today. It's gonna be very similar to this. It's just. I can't find where my ferret is. OK, so I'm doing my thing. I'm gonna. Now. And this is. Thing to remember is the Beyoncé reservation still there. This is where you have to remember I restarted my application but what I restarted was my client. I never changed my server. And I can prove that to myself by going out and looking here and saying. This is where I posted that reservation and then I got things, but I never restarted the server, so my fiance reservation should still be there. And does anyone remember it was a it was 707. And so that I can go now, I'm going to list my reservations for. The Hilton. OK, there's that one reservation. So now that is reservation 707. So I'm going to say, let's update that reservation that we just created. OK, there it is. Which one am I gonna do? I'm gonna do 707 because it is the only one here. And now there are the details, and now I'm going to enter reservation date. Now. This is another thing to know when you are updating. The way we are updating, there another way to do it, but when you are updating. You're using put and you're using all the information you're giving it all the information. So you have to replace everything. If I don't give it all the pieces of information. Then they it will fail. I mean it will either fail if it's critical piece of information, it won't let it do it, or if it isn't a critical thing, it will replace something that used to be a valid date with a dollar, something like that. So unfortunately, the way this interface works, I have to type everything over again. So let's do that thing too., Beyoncé don't have to do that like that. OK. And then I'm going to do my 3 slash 2 Slash 2022 and let's have Beyoncé check out two days later, 3 Slash. 7 slash 2022. And the number of guests is now going to be 23 as one, right? If I have managed to do that right? Yeah, and it never asked notice. It never asks for the ID. And the reality is, even if you set the ID to something different, a well behaved server won't change it because that's is part of the resource. Right. This my reservations hotel. And there it is, and it goes on different dates. So I've done all that stuff. And then the final thing and this is. 'cause. I'm a part of this. It's going to be just how do we delete? Like before, it's gonna have some return value. OK, so I will say. Boolean. Successful heart successfully delete it. We're going to we do not need for delete, we don't need to have any headers because we don't need to have any headers. We don't need to to set up a HTTP entity. We do need a try. Yeah. We're gonna say rest template. That delete. We're gonna give it our. API base URL. Flash. Now, like before, we have to specify the exact thing we're going to get rid of. So we have to say our what is it called? Just the ID. It's going to be our. RRURL. And that's that's it. We will say success. It's true. Say false. Out here with success. And once again, I'm just gonna steal. Just try that again. Put down here. You could do something different in these if it's a delete or an update or whatever else, but frankly, just use the same thing. It's got the status code, it's the status text. You don't need to do be fancy here, so this is just going in and saying look. If I was successful at doing my my delete, it will be it will return true and otherwise it will return false and if it fails it will give an error. OK, so let's. Start, rerun. Alright, we still have our reservation. Let's actually let's see if. OK, we're going to delete a different reservation just for the sake of things. OK, so here are reservations. We are gonna do our. We're going to say let's get rid of reservation 550. Now I'm going to do here is now I'm going to switch back to postman just to remind you that all of these reservations that this is a persistent data store on the server side. So even though I'm now in a different client, if I go out and get my list of reservations, it's going to include the one that I just created and then updated from for Beyoncé. And now what I'm going to do is I'm going to get rid of 550 over here in my client. Both clients are going against the same server. Think about it. If you and hundreds of other people are all hitting the same server. Updating, deleting, doing whatever else you're going to see each other stands because it's all the same store on the server. So let's go ahead and delete our thing. OK, I wanna say bye. We're going to complete. Our shows us all of them. We're going to say 550. OK, let's try to lean 505 just to see what happens. The error occurred. Check the log for details. We'll do that in a minute. But right now, let's delete since I don't even know where the log is. We're going to delete 5:50. Did it sting if I list the reservations for the hotel? For one. It. reservations for. I guess that other one was also for one. So it it no longer has the 551 and similarly if I go back out to postman. Just do the same now it doesn't have that that reservation. It used to be right here. This is in some ways tedious. Going through the process. But it's really important to get into your head. This is the there's a service out there. It stores a bunch of stuff, whether I have different clients, the same client and different locations. Any of those things going inside service are all interacting with that persistent data store. All I am doing is dealing with usually one object or one list of objects at a time. Don't list any below. And did you say if you restarted the server then none of the new information that would that you put in won't be there. This particular server comes off, you know it loads up the way. The way your vending machine did, so if I stop it and start, it's gonna go back to where it was when it started. So if I right now I go ahead and restart it. OK. And then I go back and look in the postman. Now I'm gonna have. I'm still going to have 5:50 because that that, you know, was part of the original. And I'm not going to have. I'm going to have this one. I'm not going to have the Beyoncé one because I haven't created. So the server when you restart this particular server, this one always be true, but when you restart this particular server, it's going to start from the same place. Let's go back and see if there's anything else with it. Anything else that I need to fill out. Don't modify any methods below. Well, that's good. You don't have to modify any methods flow. OK, let's see if anything else I need to. One second while I look at my notes, see it's in the else I have to cover that I've forgotten. Do that sometimes. OK so. Having looked at all this. Let's go back to just. As much as you might not be able to create it from scratch yourself right now. If you had this model in front of you, do you think you're going to be able to go and do this again for the query using the yesterday's logic for the the three things for today? Or one feels like they could follow this through? Your phone no 'cause. You're going to go out and you're going to have an exercise. Amazingly enough, it's gonna be this exact same stuff. But I am gonna put out the fully filled out version. You know, with the comics and stuff, I'll put it out as a lecture final so you can go to that. And I'm going to let you guys go today because you have your starting technical or you're starting interviews this afternoon, got employer showcase with a bunch of stuff. One question for you in terms of the whiteboard, we will probably continue to do whiteboarding in the mornings. OK. So first of all, I. I'm not going to want the same people doing it, so it's going to be new volunteers, but you do get to choose whether you have easier medium or hard. So that's up to you. But think about it. Let me know if you want a four class if you want to be a volunteer so that I can, we can short, you know, short circuit that time when we're sitting around trying to get somebody, you know, everyone is going not it. Um, so we can do that? Aside from the practice that you should do with each other. And you should do it just a little bit every day if you can with somebody else, do a little bit of white board, ETA near headed brat. There's not a lot of time, but if you guys wanted, I could do kind of like an evening session or something like that where I just had to come in and different people life board and it's, you know, you're all cheering each other eyes where when people like that OK probably won't be tonight. It'll probably be tomorrow night and I will probably just pick sometime after dinner and. You know anybody who wants to drop in and do it? The other thing I was thinking about doing is I don't want it. And I wanted to come across in a negative way, but one of the things I was thinking about doing is recording myself, going through the process one time, just so you can. Is it like this is a good model of how you talk the code, how you talk the problems, whatever else would that be helpful to people? I'll. I'll try to get that one done tonight. Probably. Alright. Well, good luck with those of you who have your your mock interviews and. He will now post the the exercises shortly. I don't think these will be. Yeah, there. There's not a lot of problem solving. This is this is really can I do the can I follow the steps. So I think that this homework should be pretty straightforward. But if you have questions you know come find me also there was something undo that I did want to mention. As far as like if we're doing whiteboarding in class, I think there's two tech elevator channels that are set up for that. It just making sure everyone knows about them and if they want to be applying that, they're in there. I know there's one which I posted about in the Java channel at some point, which is it's a code problem every day. And those are good to look at. You can use those and can't remember. I do you if you did, and if you know what the other one is, you know, posted in the chatter in the slider. Chandler. Whatever, like. Yeah, I'll throw it in Java. I think it's some. Oh, Sir, I'll find it days of code spring 22. Those are the problems of day. And then there's another one. Yeah, I know that when I don't know the other one, but yeah, and do those, there's there again. There are ones out on the web there, you know it's. It's, I will tell you sort of at least 75% of those questions that you were going to face are going to somehow involve an array. You know, they might involve a list. But that's the it's gonna be something to do with that. That's what they all are. And still so you should be ready for that without memorizing the the solutions. Just be ready for dealing with. How do I even try to create if I just list, say, I don't even give you the problem. Create a method signature which takes. Array Avengers returns a boolean. and and you should be able to do that i've told you nothing else about it and you know you're gonna look through it somehow OK so right for loops