Hi and Welcome to professional restful API design you flask.

My name is Jorge Escobar and I'll be your instructor in this new exciting course where we're going to

explore how to build efficient and professionally designed API services for anything that you need an

API for it can be to power a platform that you want other people to be kind of like interacting with

or you want to have people to to consume that via a mobile application which is a very common use case

for a device today.

So let's get started.

So what does API mean.

So API stands for application programming interface.

I mean it's a set of routines protocols and tools for building interactions between clients and servers.

And what this means is that you're creating a layer of communication between your application your platform

and something out there that's consuming the data and interacting with it as a matter of fact.

You can have an API that's read only where you're only giving let's say weather information for people

wanting to get weather for their for their specific locations or you can do like I read write API where

you have free you know social application where you're creating users on your reading and writing polls

and things like that.

So it's always this concept of having two separated entities a client on a server that are talking through

these through this layer through the specification or a specified way of communication.

That's called an API.

So what is rest rest stands for representation of representational state transfer needs networked hyper

media architectural style typified by the web which means that let's say you have an API an API can

have any form.

But when you say a restful API the connotation there is that you're using web style approaches to to

be able to serve that content to be able to interact with those clients.

And we're going to see a little bit of more information about that.

But the key thing here is that you think about the objects or the data you're returning as if they were

web pages with some data data for like next and previous page and things like that.

And also you need to think about the way that you interact the client tracks with your platform.

He's using HTP verbs so get those which we've seen in other Fluff's courses and then we have others

like.

And the lead which are not so common unless you are starting to work on an API.

So let's take a look at what the the six constrains for an AP restful API look like.

So you have to have a uniform interface.

He has to be stateless.

It has to be cashable it has to conform to the client server pattern.

It has to be a layered system and optionally it gives code on demand.

So we're going to see each one of these in some detail.

The first one which is unique for an interface I'm going to explain to you in a second lesson because

I think it's the heart of restful API so I want to like go into detail on that one.

So we will go there in a second.

So the next one is stateless.

So what does stateless mean.

It means that the the the the rest request that you're giving to the to your platform are basically

isolated instances.

They don't have a session to go with or they don't know what happened before or what's going to happen.

You know after and it doesn't like you know you cannot identify that those requests by saying all this

is user X Y C statelets means that every time you do a request everything that you need to to serve

or to basically fulfill that request needs to be embedded in that you know outgoing or incoming request

and then you're going to provide back based on those parameters the data that that request is is needing

or or you know requiring.

So it this means that every single request is an entity of its own or it doesn't know anything about

you know Future Past or what's happened so far in terms of the of the of the workflow.

So that stateless.

Cashable that's kind of like you can more or less think about caching which is you know kind of like

storing static copies of request.

This is related to that and what restful API must conform to is that you know what you must tell the

client that you question your content.

If they can cash or not cash don't respond so you're giving them so that you can have a more scalable

platform.

And this is kind of like related to let's say if you have thousands of requests you know per second

blog you could tell the client.

For example let's say I'm requesting John Doe's you know profile that profile is not changing all the

time.

So you can tell the line when you return back the profile information.

Hey you know you can cash these requests or started this request locally for let's say a day.

So that will reduce the number of requests that you're getting from that line because maybe another

user is requesting the same.

You know John those profile and you don't have to regenerate and reheat the database and all that.

You can let the Klein decide that you know they can.

They can cache that information.

And the way you you you give this information to the client is basically through the cash control.

It should be made a data header.

And as you can see in that example you're saying Max h equals 120 which is like second.

So this means that the requests I'm giving you back or the response I'm going to be back you can safely

store for two minutes and there's not going to be changes in the mean time.

So you know we should be confident that you can you're not you don't have to like requesting information

back within 120 seconds.

So the next constraint is client server.

So client server means that there's a separation between the separation of concerns between the client

and the server which means other clients are saying no what the what the servers.

Kind of like set up a slide.

What database are they using or what.

What is the structure of the data.

That kind of thing it's kind of like completely separated and the same way the server is not interested

or doesn't have to know that the client needs an iPhone 6 or Android phone or a an automated system

that's getting information kind of like Twitter you know firehose kind of thing.

The server is is doesn't know anything about the client.

And every time a request comes in you're just going to like present information based on the request

back to the client so you don't you're not concerned about what the client looks like.

The next constraint is Ledyard's system and this is talking about you know a client cannot tell what

are the layers or what is the set up behind the request that you're giving them.

And this can kind of like allow you to do like load balancing and have multiple applications servers

or even have ready skosh servers that are serving cached versions of the information the client doesn't

know.

So for example in this in this graph you can see that there's like you know the cloud is ancient it

and the black box on the on the left hand side is a client.

And you know these kind of like making older requests to the server the servers the black box on the

right and then the server has a database that can be like serving those requests or can be serving those

requests through like like I said ready skosh layer.

So it doesn't really matter how you are presenting that information back.

And that gives you the the ability to scale your API platform without breaking you know kind of like

a restful API constraints.

So the last one and this one is optional and to be honest it's not used a lot.

It's called On The man and the man is basically you know how like before you know in the web like maybe

a few years ago there was like Java blitz and these apps were like self-contained little programs that

were running your in your browser and it was like a little maybe a little watch or a clock you know

or a little game or mini game.

So the API restfully API

spec tells people that they can you know provide not only data objects but they can also provide snippets

of code and I could see this like being like nowadays maybe a flash a mini flash application or something

like that but it's not very use and this one constraint is not really kind of like something that you

are going to use often.

So that is that is that for the you know bottom five constraints so I'm going to talk to you about the

uniform interface in the following lesson so stick around.

Autoscroll

**Course content**

**Overview**

**Q&AQuestions and answers**

**Notes**

**Announcements**

**Learning tools**

**About this course**

Learn the fundamental concepts and build a RESTful API application from scratch

By the numbers

Welcome back.

So we're going to continue looking at the the constraints and we're going to look at the uniform interface

constraining which I feel like it's the most important.

So let's check out what that looks like.

So the unifor interface is divided in four parts.

One is resource based.

The second is money of resources through representations.

The third is self-descriptive messages and the fourth is hyper media as the engine of application state

or halos.

So let's go through each one of those.

So for the resource based This is basically every resource is empty fight using your rights as a resource

identifiers.

So what this kind of translates into is that the actions that you're going to take or the that you're

going to make are ordered or are have a very key and structure through that you are right and then you're

right.

You might remember is that part of the you are l of your L that doesn't have the HTP whatever or whatever

server that com part.

So you take that out the first flash.

And on that state you are right and that is is going to be telling of what type of requests or what

are you requesting.

And you're going on the Sanur API you're seeing those your L's or your eyes as your structure.

So let's see an example for example if I hit API slash V-1 slash order.

So that is a new try but what you're looking at here is that you are hitting the API section of the

Web site you're hating the version 1 of the API and you might have you know many multiple versions and

now you're saying telling the API that you want to get a list of the orders.

So as you can see here there are some you know it's kind of like here key and there's a structure in

there that the API is defining through a good set structuring of the of your eyes.

Are you're going to use to request information.

But let's see another example.

So this one is now I'm looking at AP IB One orders 50 15 44.

So as you might imagine 1044 is an order ID.

So what you're saying here is that give me the detail of this or this order alone these order 15 44

and you can start to see there's there's a structure there.

If you had orders only you're going to get probably a list of all the orders even hate orders 50 44

you're going to get only the details of that of that of that order.

So elegant and professionally the URIs are set up that way your and the way I like to think about is

that orders the flask app where you have you know the folder orders the inside views models tasks etc.

that you'd call more or less match.

The first part of your eye.

And then the second one is more like like an ID like you know it's kind of like a view or a controller

kind of situation where you're all you if you're passing that idea and then just return back that I.D..

So we're going to see these things in practice when we start working on our FBI.

So the one thing that you need to think also in this in this country is the resource constraint is that

there are resources that you're that you're sending to the client are separate from the river from the

representations or the structure that the resource has inside your API.

So for example when you request orders and you know without without the I.D. you're not going to get

like a Querrey or you're going to get like all the like the fields that are on the database you're going

to get some structuring formation and that usually has a a structure that is fine depending on what

the client looks like if it's if it's a computer or if it's a mobile application or what they need except

as specific as an example you can return the email x amount or Jason.

So the thing to think about here is you're not just going to like dump you know the whole Querrey when

you got the orders or are all the older fields what you're going to like you know return something that

structure and that you're going to be defining in your in your API definition.

So the second one is the money politian of resources through representation.

And this this kind of like is tied with how the clients will ask for what type of representations are

going to be thrown back at them or returned based on the on the accept Tippi Hedren that they're sending.

So for example a client that has a mobile application might say I need the you know orders 1044 but

I want it in Jason format your API should be intelligent enough to read that accepte HDP heterodyne

they're sending you and return back the information in the in the format that they're asking.

Whereas maybe like a web based or a browser based application or client might be requesting their mail

and you're going to you know throw back instead of the orders in some form and maybe a snippet of age

female Wi-Fi with the different you Fields of that order in ex-teammate format.

So your API needs to be flexible enough to accommodate and throw back or return information or data

using the the data up that the client is requesting.

So the third one is self-descriptive messages and what this entails is that the

the requests themselves are represented via the HTP vervet that the client is asking.

And we went through a little bit of this on our flask courses which is you know of a get request remember

that when you do I get a request you send all the fields in the actual Urals or you have like a question

mark you know the field name equals field data ampersand et cetera.

Right that's a get request whereas the post was more like within the body of the of the request you

were sending you know the actual fields that you were you were posting to the to the to the server right.

Well in an API world get Post put and the lead are actually there what we call the crud the Create read

update and delete operation so that means I'm going to send a request to get something from from the

from the API.

If I send a post it means I'm creating a brand new record and I'm sending that data for that for that

record in that post you know data out payload of used to be a request port is and one that we haven't

used a lot.

And you don't use normally lacking in web applications party may be as you do use which is an update.

So you're not creating a new record but you're basically saying update this this record this order with

this new information that say you know invoice startle or you want to modify the items within that order.

So you use it to do that update on the lead which is an actual HTP work.

Is it you know you didn't tell toread it reverences you want to delete a record.

And of course you need to have the proper authentication and you have to have the authorization to leave

records.

But if you're in if you're implementing that and you have the proper authentication then you're able

to delete records.

So that's self-descriptive messages.

And the last one is hyper media as the engine of application state or haoles which is a weird kind of

like word.

But what this means when I read this I was a little bit confused and I could understand what they were

talking about but it's in a nutshell what this this is that you or your API if it's a restful API it

needs to act like a web page in the sense that you can send not only when you get the orders you know

request.

You're not only dumping the orders back to the client but you're also putting some made links that say

there are total of you know 3900 orders.

This is page number one.

And if you want to look at the next page this is where you were either you need to read.

So I envision that when you when you when you hit orders even though returning Jasen information which

is what we're going to be use we using mostly.

Think of it not only as just data but I think it's an application it's a web page and you have like

a table where you see the order.

Like darm but it's only the first 10 results and you have like a previous next on the bottom and you

also have some sort of like maybe bet bread crumb on the top.

So that's what Halo's means is that your API should kind of like use hyper media and made links and

things like that to kind of like tell the client Hey I'm here.

But you can also do these things or if you want to update the Bannon's you're going to do this operation

etc..

So that's kind of like an example of that.

So here we have a a get request to get account number 1 2 3 4 5.

So the top line is kind of like what the client send to the API and the HTP two hundred OK is what the

API is returning.

And as you can see there you have light you know the account number or you have to balance currency

and you could stop there.

You could say OK that's all.

But since you're doing haloes you're also having this link rail which is an actual like valid HTP you

know or extremal tagging and you say if you want to pass it you can do a count 1 2 3 4 5 slash the and

if you want to withdraw you know you get the idea you're kind of like helping the client to maybe make

decisions and that includes transferring or closing the account.

But there's another layer to this so let's say for example that now the account is is you know overdrawn

and as you can see on the balance on the currency or on the balance field you see that it's minus 25.

So they apply here should be intelligent enough to say wait a minute.

So the only action that the person can do here is to pass it.

Meaning get this account back to like positive right.

But they shouldn't be able to withdraw.

They shouldn't be able to transfer or they shouldn't be able to close these accounts.

So you know that great line is what we're going to like make disappear.

And this is what you're going to get if they're the account balances is a negative.

So this is Halo's right there it's like you're not only returning data you're also like reacting to

the data that you're kind of like using so that your client is able to make the right decisions.

And and you're like kind of like guiding for guiding them through the proper channels.

So that's that that's that's that's the whole thing.

I know there's a lot of theory here and unfortunately I was wrestling before do I just you know give

the students kind of like this concepts as we go along.

But I think even if you didn't get the whole thing it's fine you can review this later when you've done

the course or maybe you know review it at a later time but this will give you kind of like a very good

handle of what a professional Well the sign restful API looks like.

So let's start coding our API application.

Welcome back.

So we're going to continue looking at the the constraints and we're going to look at the uniform interface

constraining which I feel like it's the most important.

So let's check out what that looks like.

So the unifor interface is divided in four parts.

One is resource based.

The second is money of resources through representations.

The third is self-descriptive messages and the fourth is hyper media as the engine of application state

or halos.

So let's go through each one of those.

So for the resource based This is basically every resource is empty fight using your rights as a resource

identifiers.

So what this kind of translates into is that the actions that you're going to take or the that you're

going to make are ordered or are have a very key and structure through that you are right and then you're

right.

You might remember is that part of the you are l of your L that doesn't have the HTP whatever or whatever

server that com part.

So you take that out the first flash.

And on that state you are right and that is is going to be telling of what type of requests or what

are you requesting.

And you're going on the Sanur API you're seeing those your L's or your eyes as your structure.

So let's see an example for example if I hit API slash V-1 slash order.

So that is a new try but what you're looking at here is that you are hitting the API section of the

Web site you're hating the version 1 of the API and you might have you know many multiple versions and

now you're saying telling the API that you want to get a list of the orders.

So as you can see here there are some you know it's kind of like here key and there's a structure in

there that the API is defining through a good set structuring of the of your eyes.

Are you're going to use to request information.

But let's see another example.

So this one is now I'm looking at AP IB One orders 50 15 44.

So as you might imagine 1044 is an order ID.

So what you're saying here is that give me the detail of this or this order alone these order 15 44

and you can start to see there's there's a structure there.

If you had orders only you're going to get probably a list of all the orders even hate orders 50 44

you're going to get only the details of that of that of that order.

So elegant and professionally the URIs are set up that way your and the way I like to think about is

that orders the flask app where you have you know the folder orders the inside views models tasks etc.

that you'd call more or less match.

The first part of your eye.

And then the second one is more like like an ID like you know it's kind of like a view or a controller

kind of situation where you're all you if you're passing that idea and then just return back that I.D..

So we're going to see these things in practice when we start working on our FBI.

So the one thing that you need to think also in this in this country is the resource constraint is that

there are resources that you're that you're sending to the client are separate from the river from the

representations or the structure that the resource has inside your API.

So for example when you request orders and you know without without the I.D. you're not going to get

like a Querrey or you're going to get like all the like the fields that are on the database you're going

to get some structuring formation and that usually has a a structure that is fine depending on what

the client looks like if it's if it's a computer or if it's a mobile application or what they need except

as specific as an example you can return the email x amount or Jason.

So the thing to think about here is you're not just going to like dump you know the whole Querrey when

you got the orders or are all the older fields what you're going to like you know return something that

structure and that you're going to be defining in your in your API definition.

So the second one is the money politian of resources through representation.

And this this kind of like is tied with how the clients will ask for what type of representations are

going to be thrown back at them or returned based on the on the accept Tippi Hedren that they're sending.

So for example a client that has a mobile application might say I need the you know orders 1044 but

I want it in Jason format your API should be intelligent enough to read that accepte HDP heterodyne

they're sending you and return back the information in the in the format that they're asking.

Whereas maybe like a web based or a browser based application or client might be requesting their mail

and you're going to you know throw back instead of the orders in some form and maybe a snippet of age

female Wi-Fi with the different you Fields of that order in ex-teammate format.

So your API needs to be flexible enough to accommodate and throw back or return information or data

using the the data up that the client is requesting.

So the third one is self-descriptive messages and what this entails is that the

the requests themselves are represented via the HTP vervet that the client is asking.

And we went through a little bit of this on our flask courses which is you know of a get request remember

that when you do I get a request you send all the fields in the actual Urals or you have like a question

mark you know the field name equals field data ampersand et cetera.

Right that's a get request whereas the post was more like within the body of the of the request you

were sending you know the actual fields that you were you were posting to the to the to the server right.

Well in an API world get Post put and the lead are actually there what we call the crud the Create read

update and delete operation so that means I'm going to send a request to get something from from the

from the API.

If I send a post it means I'm creating a brand new record and I'm sending that data for that for that

record in that post you know data out payload of used to be a request port is and one that we haven't

used a lot.

And you don't use normally lacking in web applications party may be as you do use which is an update.

So you're not creating a new record but you're basically saying update this this record this order with

this new information that say you know invoice startle or you want to modify the items within that order.

So you use it to do that update on the lead which is an actual HTP work.

Is it you know you didn't tell toread it reverences you want to delete a record.

And of course you need to have the proper authentication and you have to have the authorization to leave

records.

But if you're in if you're implementing that and you have the proper authentication then you're able

to delete records.

So that's self-descriptive messages.

And the last one is hyper media as the engine of application state or haoles which is a weird kind of

like word.

But what this means when I read this I was a little bit confused and I could understand what they were

talking about but it's in a nutshell what this this is that you or your API if it's a restful API it

needs to act like a web page in the sense that you can send not only when you get the orders you know

request.

You're not only dumping the orders back to the client but you're also putting some made links that say

there are total of you know 3900 orders.

This is page number one.

And if you want to look at the next page this is where you were either you need to read.

So I envision that when you when you when you hit orders even though returning Jasen information which

is what we're going to be use we using mostly.

Think of it not only as just data but I think it's an application it's a web page and you have like

a table where you see the order.

Like darm but it's only the first 10 results and you have like a previous next on the bottom and you

also have some sort of like maybe bet bread crumb on the top.

So that's what Halo's means is that your API should kind of like use hyper media and made links and

things like that to kind of like tell the client Hey I'm here.

But you can also do these things or if you want to update the Bannon's you're going to do this operation

etc..

So that's kind of like an example of that.

So here we have a a get request to get account number 1 2 3 4 5.

So the top line is kind of like what the client send to the API and the HTP two hundred OK is what the

API is returning.

And as you can see there you have light you know the account number or you have to balance currency

and you could stop there.

You could say OK that's all.

But since you're doing haloes you're also having this link rail which is an actual like valid HTP you

know or extremal tagging and you say if you want to pass it you can do a count 1 2 3 4 5 slash the and

if you want to withdraw you know you get the idea you're kind of like helping the client to maybe make

decisions and that includes transferring or closing the account.

But there's another layer to this so let's say for example that now the account is is you know overdrawn

and as you can see on the balance on the currency or on the balance field you see that it's minus 25.

So they apply here should be intelligent enough to say wait a minute.

So the only action that the person can do here is to pass it.

Meaning get this account back to like positive right.

But they shouldn't be able to withdraw.

They shouldn't be able to transfer or they shouldn't be able to close these accounts.

So you know that great line is what we're going to like make disappear.

And this is what you're going to get if they're the account balances is a negative.

So this is Halo's right there it's like you're not only returning data you're also like reacting to

the data that you're kind of like using so that your client is able to make the right decisions.

And and you're like kind of like guiding for guiding them through the proper channels.

So that's that that's that's that's the whole thing.

I know there's a lot of theory here and unfortunately I was wrestling before do I just you know give

the students kind of like this concepts as we go along.

But I think even if you didn't get the whole thing it's fine you can review this later when you've done

the course or maybe you know review it at a later time but this will give you kind of like a very good

handle of what a professional Well the sign restful API looks like.

So let's start coding our API application.