Django: Getting Started

by Reindert-Jan Ekker

Start Course

Bookmark Add to Channel Download Course

Course Overview

Course Overview

[Autogenerated] Hi, everyone. My name is Ryan and welcome to my course, Jingo getting started. I'm a longtime developer and educator, and in this course I will teach you about creating Web applications with Django, the Web framework for perfectionists. Jingo is the most popular Web development framework for fighting, and it's awesome to work with. In this course, we're going to build a Web application with Django, and along the way you will learn about all its major components. We will cover templates and your alma things, jingle middle classes and the admin site, using actual forms for user input and much more. By the end of the scores, you will have a firm understanding off all of these and how they work together and you'll be able to create your first website with Django before beginning the course. You should be familiar with bison and the basics of HTML. I hope you'll join me on this journey to learn Web development with bison and jingle, with the jingle getting started course at neuroscience

Starting a Django Project

Introducing Django

[Autogenerated] Hi, This is Ranger Liquor. And welcome to this course about getting started with Django. In this model, I'll give you a short introduction of Django, and I'll also go off the prerequisites for this course. What should you already know to be able to follow the course which software needs to be installed in your system, et cetera. After that, I'll show you a demo off the project well built, and then we'll start our first jungle projects, and we're going to explore the files that it's made up off. We'll run the project and look at it in a browser. We won't be creating our own content yet. That's for the next module. Django is a framework for writing Web applications in Python. It focuses on being very productive without writing lots of coat and on making the code. You have to write as clean and elegant as possible, by the way, just in case you were wondering, Jenga was named after Django Reinhardt, the famous jazz guitar player. So it's got nothing to do with the Tarantino movie. So why would you want to learn Django? Well, Jingo comes with batteries included. There's a huge standard library with lots of functionalities to solve just about any problem. And what's more, there are lots and lots off jingle specific packages to make your life as Web developer easier. All of this makes it possible to be very productive with Jang Go right from the start. Here's some of the most important features that come installed with Jenga. There's a powerful object relational mapping a P I, which allows you to write pure python classes that represent your database tables. This means you won't have to write SQL queries to interact with your database. When you've written the middle classes that represent your data, Jingo can generate an admin interface. This is a Web based user interface for editing content, and it's generated automatically by Django. This very powerful feature can really speed up your development cycle. General also comes with a small but powerful template language for generating your Web pages and a beautiful way to configure the Urals for your site. It will also let you handle HTML forms with a minimum off coat or often without any gold at all. In case there's something you're missing in the temperate language or the standard form classes or any other part of Django. There is a huge amount of third party packages that will probably offer what you need, and that's just the tip of the iceberg. Some other features that General offers out of the box. Our usual authentication ht to be session handling, a powerful high performance guessing system and some very nice internationalization supports.

Prerequisites

[Autogenerated] So now you know why you might want to learn Jango. And here's a short off your things that you should already know before watching this course. First of all, jingle is a biting framework, so you should be familiar with. Bison, at the very least, watched the introductory bite

on course and writes from simple programs. Although Jingle does you some advanced concepts, we're going to stick to the basics so a general on the standing off the core language will be enough to get you started. You should also know how to use Pip to install packages. If you don't know these things, please, which my course on parasite called Managing Part and packages and virtual environments. You should also know the basics of Web development, and that means you should at least know how to write a simple HTML page. It may be helpful if you also know a bit about she's s styling, although you should be able to follow everything even if you don't. We're also going to touch on the http protocol orbit, which is the protocol that enables a browser toe ask a server for a webpage. But if you don't know about 80 to be, Don't worry, it's not hard at all, and I'll explain everything you need to know. Finally, it's also important that you know the basics off databases. We will use the database to store our applications data, and we're going to let Django create and modify our database tables. There's no needs to write any actual SQL statements, but you should be familiar with the concept off tables rose or what an insert or update means. So what do you need to have installed on your system to use Jango? Both Fighting and Geno are cross platform, which means you can work along with this course regardless off your operating system, be it Lennox, Mac OS or Windows. In this course, I will be working with General Three, which supports bison 3.637 or $3 8 So if you're running an older version of bison, be sure to update Then there's something that I think no developers should do without, and that's an editor with bison support for this course. I will be using the community addition off by charm, which is a very full featured piece of software with a lot of features for parts and developments. But there are many, many other editors, and you should be able to follow this course if you use another editor, I will tell you when you will need to take different steps, and I do.

Project Preview

[Autogenerated] Now let me show you a short preview off the replication we are going to build in this course. Let me give you a preview off the project we're going to build in this course. It's a tiny meeting planner weapon application, which you see here is the home page. It says, Welcome to the meeting planner. They're the Children. Overview off all the meetings that already have implant. It's this little list here. There's a form toe. Add a new meeting where can enter the title for a new meeting, and the date and time is will select a room. And if I click okay, the new meeting shows up here in the meetings lists. We could also click on a meeting and see its details and let's go back. Here's the rooms list, which shows all the rooms that we have in our database. So this application includes a database with all the data, and you can also see that all the pages are

styled even though it's kind of minimalists. And one cool thing is that every general application comes with a built in admin interface so I can look in here as an administrator and here we see the default jango administration interface that allows us to manage our room and meetings objects, for example, here, if a go into rooms, I can inspect, edit, add and delete the rooms that are available in my database. And a similar thing goes for all the other data, like all the meetings in the database. So this is what we'll be working towards during this course.

Installing Django

[Autogenerated] Now let's really get started. We're going to start a pie charm project. Install Django creates a jungle project. Run that and inspect the results in a browser. For this course, I will used community addition off by charm. As my editor, you should be able to follow this course if you use another editor, I will tell you when you will need to take different steps than I do. In my case, I start by creating a new projects, and this takes me to the new project screen. It's the location for my project. I choose Django getting started under by charm projects by John will create this folder for me. So if you work with another editor, you might need to create your own folder for your project. But you're also automatically created virtual environment for my project. If I click on the trying or here you see that it will create a virtual environment for me in my daughter, Virtual and Fuller and the name off the environment is the same as the name off the projects. It's based on my latest installed bison version, which is 3 to 6 now. If you're not using my charm you probably want to create your own virtual environment. And if you don't know how to do that, please take a moment to watch the module about that in my course, managing fighting packages and virtual environments. Anyway, I like the default settings, so I'm going to click create on some systems. Creating a pattern project takes a couple of minutes, so you might need to wait a little bit here. When it's done, you can click close on the tip off the day because we don't need that now. This is the empty project by Germans just created for us. But before we can start writing code, I need to install Django to do so. I want to start a terminal in an active virtual environment. Now, the course I mentioned before will show you how to do that in general, both by term, actually has a nice shortcut for us. If I click terminal at the bottom here, it starts a terminal session, and as you can see at the start of the line here, we are already inside our active environments. That means that to install Jingo, I only have to say bison minus M pip installed Django, and this downloads and installs Django. Israel has its dependencies. Now, instead of starting your own project and writing the count yourself, you can also download the demo code from plural site. And the project is also available on Get up. So you can also clone the Get A project, which is available at the location shown here. In that case, you will always have to create your own virtual

environment and install Jingo, and I cannot do that for you. If you don't know the Democrats, the project will look slightly different. Let me show you. So if you don't know the source code and you open this fuller here jungle getting started, there's two extra files which we didn't create ourselves, their own gets and they're called Don't get ignore, which is a thing specific to get projects, and the other one is requirements of 60 and that lists the packages that this project depends on. If you don't know the Democrats, you will get these files. But the project will also work fine without them

Starting a Project

[Autogenerated] Now that we have jingle installed, we can start a jingle project, which is a different thing from a by John Project. So part German city editor and Django is the bison framework that will write our code for we start a jingle project by running a script called Jango Edmon. Now this Jungle Edmund script can do many things, but the most common thing we use it for is to create a new jingle project by saying Start project, followed by the name off the projects. For this course, I'm going to call my demo project meeting planner. And now when I press enter, this will create a new folder with exactly that name. The General Edwin script itself doesn't give any output, but the new folder shoot show up in by charm. But sometimes it doesn't, and in that case you can right click on the project and say Reload from this And now she, our new fuller meeting planner. We can click on the triangle here to see what's in the folder, and there's a file here called Managed by. This is a script we will use from now on to run jangle commands. Let's try it out right now, but to do so, we need to move into the meeting Planner Fuller, where the script is located. We do that by typing seedy space meeting planner and you can see the text Your change to show that we're now in this fuller. We can now run the project by saying bison management by run server. This runs the jingle development Web surfer, and it prints some information for us. The red next year is a warning that will address in a later model we can ignore it. For now, the server also shows us a u r l This is where it actually hosts our replication. Now, when I click this, your l A browser opens, and this is what we get to see the install work successfully. Congratulations. Off course. This is just a simple demo page for an empty weapon up, and we're going to add our own page assume. But for now, let's do some investigation. First of all, we see that the actual address is 1270.0 that one, which is a special be address reserved for your local machine, which means that I am committed to mind computer. But if you talk this in, you will connect to your own computer, and then it is a column, followed by the port number 8000 and that support that the jingle Server is listening on. So if you type this address into your browser, it will connect to your running Jenga Web server now for a moment. Let's just stop the

server. I'm going back to the terminal here and in the terminal. Uncle, a press control plus C. This stops the server, and now you see the directory were in, which means you can now enter a new terminal commands, which is not what we're going to do. I want to go back to the browser for a moment, and if I refresh the page now, I get the message unable to connect. And this is because right now my server isn't running, so there's no process listening to port 1000 at my local address. Now let's take a moment and take a better look at what's in our jingle projects. First of all, there's a new file here called D B Delta Actual like three, and that's a database file that jingle created automatically when we ran the server at the moment. It contention no data at all, and we'll learn more about it in the next model. Then there's a second meeting Platter Fuller in here, and you can basically think off. This folder is the core project Fuller. If you look in sight, we see that it contains a file sitting by. And this is basically where our project is defined. If you double click it, we see the sittings for the meeting planner project throughout the course will use this file to configure our projects. There's also other files here. You're also by which is where we're going to put the coat that assigns you or else to the pages that we will create and their files A S, C I and wst idle by. Those are used when you want to deploy your project toe a production surfer. Finally, there's a file in it. A pie which is empty, and only there to mark this folder as a bison package. So this is what the Jingle Edmund Start Project command created for us. It's mainly come santo this file, setting some pie and you're also bi. Those are the big bone off our Web app and one full or higher. There's the managed by script in the S. U A light database, which was not created by Jingo admin but by running the surfer. The full of structure of this project can be a little confusing at first, so let me just go over each off the folders again. The top level fuller Jenga getting started has been created by my charm, and it contains my get ignore and requirements files Insight. There is the Dangle Project folder called Meeting Planner, and that has a 2nd 4 inside it with the same name, and that contains my Jingle project settings. The structure will slowly become more and more clear as we write out and see howto work with those fellers.

Review

[Autogenerated] Let's go over the main things we've just learned. We started by installing Django by calling Pip installed Jango. It's best practice toe. Always install things inside a virtual environment, and things are no different with Django. If you use by charm, it will actually create an environment for you automatically when you set up a new project. Next we created a jingle projects. You do this with the Django Edmund script that comes with your jungle insulation. You can simply call it like this Django at Mendel by Start Project and then the name off the project. Note that the Jingle Edmund script is installed inside the virtual environment, so make sure that

you're inside your active virtual environment before you try to run this. To run a Jingle project, you move into the project directory with the CD Command and then run the managed off by script with the Run server commands. This will start the built in Jingle Development server. It's a fast and lightweight observer that will reload the changes in your court automatically, which is nice because it really speech up development. By the way, you can stop the server by pressing control. See in the terminal window. Now, one thing I cannot repeat enough is that you should never, ever do this in production. The jungle production server is not secure or performance enough for a real world deployment environment, so you should never do that. Finally, if you want to work along with this course years the Ural to the getup repositories that contains all off the shore scouts or if you prefer, you can download the card for each module from rural sites. And that's it. We've started this course by installing Django, setting up a jingle project, running that and exploring what the project looks like. Let's move on to the next module in which will write some code of her own and create our first webpage with Django.

Creating a Simple Web Page

Creating a Django App

[Autogenerated] Hi, I'm Angelique and in this model will create our first webpage with Django. Our goal in this module will be towed at our own page to the jungle projects. To get to the point, we need to take several steps. First, we'll create a component called Jango EP, which will contain the goat will right, and our coat will take the form off a so called view function. Then we will add some color to assign a Ural to this you'll function. After doing that, we will run the project and few are nube age. We'll also take some time to make sure that you understand how all these components work together to create a jangle webpage. And I'll go over some off the common problems and pitfalls that you might run into. Let's dive straight in and create a first webpage with Django to do so, we create a new component gold Django EP. In there, we will define a so called view function for our page shine a euro to it and then we can run the project and few of the page. Here we are back in our project and I want to create my first webpage in Django. The best practices to group your coat together into so called Epps. A jingle is just a folder containing python files. So it doesn't have anything to do with a mobile phone app or similar things. I can create a new EP by running a command in the terminal. So I'm starting to terminal. And I checked the text here. It says I'm currently in the top level. Fuller in my projects. Jingo getting start. It's But

the military pie script is one level deeper in meeting planner. Now, please check for yourself. What? The current directory is in your terminal. If you're already in the meeting planet directory, you're fine. Also, if your server is still running, remember, you can press control, see to stop it. But in my case, since I'm not in the right folder, I need to say see the meeting planner to move into the correct location. Now, in case you get confused or the terminal gets messed up, you can always press the little X year to stop paternal and then start a new terminal. This will always start in the top level project folder. So again, I have to start by moving into the correct location. I can now run the minister by script as follows fightin Mitchell by and I give it to command start up, followed by the name off the up I want to create. In this case, I'm adding an epic will contained general pages for my website like the welcome page, a contact page and about Paige. And that's why I'm calling the APP websites now to run its I press enter as just like the start project. In the previous demo, this command creates a new folder with the name we just specified. So there's a folder website. Now let's check the content so military by created a number of files for us at the moment, will focus all of use a pie, and I'm going to remove everything else. But the Minute Opie file, we'll talk about what these files do later. For now, I'm going to remove all these except future by and in it of by even the migrations. Fuller can go. So we left with these tools. The Minute Opie file is empty. It just marks this folder as a heightened package. The views of pie file is where I'm going to write my coat. We see an import statement and the text create your views here. This is where we'll write the actual code for our webpage. But first, I need to make sure that the coat I right here will be seen and run by Django. To do that, I need to open setting some pie in the meeting. Planner Fuller. Remember that this fuller is sort off the core folder for our project, and it contains the project settings or by file. There's lots of settings here, and I'm not going to explain all of that. But if we scroll down a bit, we find a setting called Installed. Epps to make the coat from our new app available in Django, we need to add it to the list here. Good. Now, rep is installed. This part of her projects

Adding a Page

[Autogenerated] let's close sitting side by and go back to views of by. We're going to add our own view here. The view is a jingle component that handles a request for a webpage. In our case, we will write a function called Welcome Like this. So this is a normal python function. It has an argument request which you currently don't use. And in return, something called http Wrist Bones with the string. Welcome to the meeting planner. We call this a view function, and its purpose is to handle on http request for the welcome page of our site. In other words, when a user visits that welcome page, this function will be cold, and it will send the string Welcome to the

meeting planner back so it can be shown in the browser. By the way, I shouldn't forget toe import the http response class at the top off the file to make this work. So now I have a view function. But how do I visit this page for my browser? We need to assign a euro to our view so that we can actually visit it. Let's open your elsewhere by which is in the same Fuller as two sittings module. So here there's a lot of documentation with instructions. How to write your own wrappings. But mainly there's this list called euro patterns, and in this list, we can put all our euro met things. There's already one mapping here for admin, and we'll talk a little bit more about that in the next module. Now I'm going to start by importing the view we just route like this from website of Fuse Import. Welcome. And now I can add a line to the euro patterns leased by calling the function Beth with two arguments here. The first argument specifies to your ill, and the second argument is the actual view function. What this means is that when a user wants to visit the welcomes the HTML page, we want that to be handled by our view function. Welcome. By the way, you might notice that my charm puts little red lines below the import here and hovering the mouse over it, it says unresolved reference. Well, our code is actually correct. It's just by charm that's confused. I can fix this by telling my charm where the route off my jangle project is. I right click on the meeting planning folder. Make sure it's the outer fuller called meeting planner, not the one inside it, and say Mark directory as sources routes. This makes by charm, understands my project structure, and now the red lines go away. Now let's see if this works. I'm going back to the terminal and say Bite unmanageable by run server and now opening a brochure and going to the welcomes HTML page, we see our greeting. So what happens is this because of the Ural 1270 to tear that one sport 1000 points to the jungle server on my local machine, the browser will make a connection with the Jingle Server and sent the request for this. Page welcomes a she male in the look of the server. We can actually see this. The server tells us it's received. Get requests for the welcome of HTML page. Jingo then looks in your else. Opie finds this line here, which matches the Ural welcome with HTML with a few function little handle requests to that you're ill. In other words, this line says, if request comes in for welcome to the H E mail call this function. Welcome. So then the welcome function is cold, and it returns to text that extent sense to the brochure and this place to the user.

When It Doesn't Work

[Autogenerated] now, in case the previous demo didn't work for you. There's a couple of things to check. I'm going to start by stopping the running server by pressing control. See, as shown here. Now, let's go over several things that might go wrong when you follow the previous demo. First of all, check the command you run to start the server. It should be exactly like this. Bison

Mitchell. By wrong server. If you misspell any part of that, he will see an error. The air will be slightly different depending on which part of the spell. But it will always be something like no such file, amount command or something similar. If the commander typing is correct, but it still doesn't work. You might be in the wrong folder. For example, I might be one level too deep. Now I'm inside the Fuller that has the sittings file running the surfer from here also doesn't work. So in this case, I can move a fuller back up by saying seedy space don't thoughts. And now I'm in the right location or if you're not sure, click the ex year to close the terminal and start over completely. There are also other things to check. Check that. You added the new EP name website in the correct place that is in the installed absolutist and not one of the other lists. Check that there's a coma on the line before and it made off. The map is exactly the name s the name off the Effler. Make sure to double check views, Tobias. Well, and make sure that your function is called welcome. It takes exactly one argument called requests. And make sure that you didn't forget the return key word here. Finally, goto your else will buy Check that you're importing correct module, which startled views and the correct function. Welcome. Also, make sure you didn't misspell the name off the function in the cops'll tow path.

Django and the Flow of Control

[Autogenerated] now that we have wrong. Paige, let's go a little deeper. We'll add some extra pages and get some deeper understanding. I'll talk about the way general controls the flow of her application. How jangle MEPs, Urals to view functions and the jungle debunker. Let's take a moment to talk about control flow. Looking at our coat, you might have noticed that it doesn't behave like a regular script. It doesn't simply start somewhere and the run from top to bottom. Instead. What controls the flow of our coat is the jingle server process. But when I run the Jingle Surfer, it doesn't really do anything, because what a Web server does is to wait for incoming requests. In other words, our program will do nothing until a request comes in. To make a request, I have to visit our site whenever I visit the page in my browser like this, the browser sense an http, get request to the server. You can think of this like a tiny text message asking the server for a specific page. Currently, if we go to this address, we get a four or four, and we're going to fix that in a moment to get an actual Paige, we need to enter a euro for which we've defined a mapping like welcomes a she male. So if I press enter, every quest goes out to the server and the server look will now show that the server received to get message for the Ural welcome with a she male. Django looks off the view function meant to that you're ill. So Jenga will call this few methods which returns our text and jingle sense that backs of the browser, which then shows the message. So this also means that every time I refresh the page, a new request is made as we can see in the

look and the view function is executed every single time. Now, to show you that the view is executed every single time, let me add a second page to the application. So this new view function is called dates, and it's very similar to the welcome function in that it returns and htp response with the string this page was served at. But the main difference is that I add data timed up now to the string. So in the new page that we've just created you will see the date that this function was run To make this function work, I also have to import the dime show at the top of the file. I'm going to say from that time in port day time now and you're also bi I can make a new mapping for this function. Let's make it two dates and we also need to import the new view function here Very well. Now the server will automatically detect changes in our coat. You can actually see it in the log file. The service shows that the detective that Uriel stopped by changes and it will reload. So now we can go to this new euro slash date and pressing. Enter. We now see the request Come in four slash dates. Every time I press refresh, you know, see the time shown in the resulting page change. So every time the view function will be run. Now, let's just fix one thing. The welcome page is currently shown at welcomes that she male, but I prefer it to be shown us the index page off my sight at the the faulty where I shown here, which is just a server address in port with nothing after it. We can do so by mapping the euro to the empty string like this. And now if I click this link, the welcome page shows as the default page for my sight Nice.

Adding an About Page

[Autogenerated] I'd now like to add 1/3 page door of a bed and go over some of the possible problems that you might see. If you're working along. Please take a moment to try to implement it yourself. The exercise is as follows. Please add and about page that shows some text about yourself. If you're going to try to implement it yourself, post the video right here and right the coat. I'm going to wait for a couple of seconds before I showed the solution. The way I do this is I just copy the existing welcome function into a new function, and I'm changing the name in tow about. I'll also change the texture to function returns into I'm landed and I make courses for pleural sites Good. The next step is to add a euro mapping in your else Opie, which have already done here. As you can see, it's a lost line, it says Beth, about about so the first argument again. It's a string, and that's the Ural. And the second argument is the actual view function, and you shouldn't forget toe also import if you function. So that's it. We now have a new view function and the server will reload automatically so we don't have to restart or anything like that. Let's go check it out. So if you go to the new euro about this, what you should see, or of course, you should see the text you wrote about yourself. Now let's go over a couple of things that might go wrong here first in euros, look by. You might have missed types. They're about your I something

like this. In this case, if you go to the about Paige, you will get to for a four error because there's no mapping for about because we only have a miss built your ale right now. So let's fix that. And you might also have Miss Stockton. Name off the few function. But if you do that, you will see that the server actually gives you an error because this is not valid bison coat, and this means the service not running, and you will also not be able to connect in the browser. So there's a difference between a running server and asking for a Ural that doesn't exist, and the chauffeur not starting. It'll because you don't have Philip goat Now let's fix this again and let's look at the view for a moment. There are two main things that beginner sometimes do wrong here. First of all, they might forget toe at the return key word in the last line. Let's see what happens if I do that. As you can see, the Jingle server really makes an effort to help you out here. The error is quite informative and tells you that we forgot to return a value. It also tells you that this is about the function website, not views that about, so that will help you find the cart that cost the error. And if you scroll down all the way, Jingo shows you lots of info about its configuration here at the end here, General tells you you're seeing this error because you have D Book is true in your jungle settings file. So our server is running in debug mouth, and that's why it shows the's very detailed ever messages. But you can imagine that in a production environment where your server would be exposed to the dangers off the Internet, leaking this much information is not very safe. So in a production environment you absolutely don't want tohave de bucks. It's true. But during development we do like this setting because it makes fixing errors so much easier going back to review function and fixing the mission return. The other thing that people tend to forget is to function argument request here, please check for yourself. What happens when you leave that out?

Review

[Autogenerated] Let's review what we've learned. We started with creating a jangle Web, using the command bite and military by started, followed by the name of the EP we want to create. Remember that the manager by script is in the Jungle Project folder, which in our case is not the same as our by charm projects fuller. And that means you have to see the into the right folder before you can run this command. After creating a nap, you should make sure it is picked up by Jingo by adding it to the installed absolutist in settings Off by. Let's take a moment to talk about what a Django AB actually is. The Wood EP, when used by a jingle developer, means something very different from, say, an EP for your mobile phone and single rep is a python package that, specifically intended for use in a jingle projects. You can break up the functionality of your project into multiple apps that each act as little with applications off their own with their own views, your

own wrappings, Israel as components that we've yet to learn about, like marbles and templates. So that means that a typical jingle project consists off multiple. Epps, for example, let's say I create a website for a rock bands. I might have an app for showing pages with information and at the church during schedule. Another F for selling tickets. It's a trap set. You can think of it as a way to organize your coat. We put related things together in APS. A nice thing about writing abscess that sometimes you can make them reusable, so I might just reuse the ticket, selling it from another site in the side for my rock bands. Now I have to say that designing and have to be reusable is a little too advanced for this course, so we're not going to do that right now. Structuring your project into multiple Epps helps keep your coat modular and organized, but you're completely Frito. Organize your Eppes anywhere you like, so you could even decide to put everything in one huge step or not to use Epps. It'll but the best practices to follow the unit's philosophy to do just a single thing and do it well. So keep your Epps simple and small. If you need more than one sentence to explain what the purpose of your EP is it's probably too large. Once we've created our first step, we can start adding goat in views it by. We edit a so called view function, which takes single argument requests and returns. And http response The responsibility of this function is to handle so called http requests for our welcome page. But to be able to visit that Paige, we need to assign it to the girl. And this we can do in your l stop I where we import our view. And in the Urals petrus list, we add a cold bath, which is a function that creates a mapping from a euro toe of you. In this case, we leave the euro empty because we want a page to be the root page off our sights. So what happens when I view a simple webpage? Supposed that, like in the demo, we used browser to visit the locally running server at 127.0. Tell Sarah that one Port eight thousands. Well, first of all, the browser would send an http get request to that server if you don't know what that means exactly. Let me put it like this and http request is a message from the browser to the server, where the browser requests that the server does something in the simple case off viewing a webpage. It will send a get requests, which means it asks the server to send it the continent four. From your L. When the jingle server receives a request from a browser, it takes several steps to determine how to handle that request. First, it looks at the euro patterns in your l stopped by to see if there's a euro mapping that matches the request. In our case, the euro is matched by this line here, which refers to review gold. Welcome. This means that Dingo can ask the welcome view to handle this request. So the next step is the general will call the few, which in this case is a function. And all this function does for now, is to send a response with some text back to the browser, which will then be shown by the proffer to the user. And that's it for this module. We've learned how to create a general up at a few function and assign a euro to it. We saw how to run the server and to you the page. And we've also taken time to make sure that we understand exactly how gentle knows which function to call for which

page. And we've looked into solving common problems with that. Let's move on to the next module, where we'll start building our database mobile.

Setting up a Data Model

Introducing Models and Migrations

[Autogenerated] Hi. My name is Rachel Liquor and in this module will learn how to set up a data model in Django. This model is focused on creating a data model, which will allow us to create meeting data and store that in a database. We'll start by learning what jingle middle classes are and how to write them. Then we'll learn about database migrations and how to use those. And finally, we'll see that Jingo provides us with a very nice and powerful interface toe. Enter and manage meeting data. Let's start by introducing to core concepts off this module. First of all, there's general models, models are python classes, and their purpose is to make our data persistent. In other words, they let the store things in a database in such a way that if you stop the jungle server and then restart it, the data will still be there. So when we create a meeting more class and then create a meeting object, we can save that into the database, and we can reload it later after we restart the server in Django, a middle class is maps to a database table, so mobile class called user will result in a user stable and a meeting close, we'll have a meeting stable, then each object off that close can be stored in a row in the table. So if I create a new user and say that in the database, the user stable will contain a Nero with the data for that user. Now, while you are working on a project you're changing and updating your cult old time. This is true for all your python code. So it's also true for our middle class is, for example, we might add, or remove or renamed properties. If I decide I want to be able to store my users age, I will add an age property to my usual class. But that means that I need to add an H column to my database as well. Every time we changed our more class, we will need to change the course bowling database table so that it will have to write columns to store our objects. And that changed to the database structure is automated through migrations. The migration is a python script that changes the database so as to keep the structure off the database up to date with our model classes. Whenever I change my model, let's say I'm adding the age property to my usual class. Jingle will generate a migration script that will update the database table accordingly. Actually, once your project becomes slightly more advanced, Jenga will not be able to generate everything you meet and you will need to write your own migration cult. But we will not do that in this course. In other

words, models are python classes that represent your data and make it possible to say if your data in a database and migrations are scripts that help keep your database structure off to date with your middle classes.

Running Initial Migrations

[Autogenerated] Let's see some off this in practice before we even start writing our arm or classes. We can already run some migrations. Let's do that. I'm going to start by taking a step back and look at the current situation. First of all, looking at sitting side by, we noticed that there are a number of Epps included by default in our jingle projects. We can see them here in the installed absolutist admin, both content types and more. Interestingly, we see the same names pop up somewhere else as well. When we start our server with the Run servic amount like this, it shows this message. You have 17 all applied migrations and chose with names off some maps here at mean both content types, sessions. Each migration is a python script that makes a change to the database corresponding to some piece off fightin coat in a middle class. And there are apparently 17 migrations for these Epps. So apparently, these Epps includes middle classes, and General wants to create database tables to store data from those mobile classes. In the database, I can ask jingle which migrations are waiting at the moment. First, I'm going to stop the server. And then I'm saying bite unmanageable by show migrations. And we see you that the Admin Dept. Has three migrations waiting both their past 11 etcetera. Each of these lines you see here represents some change to the database. And mostly they are about creating the correct table structure. The next step is to apply the migrations with the command Buyten managed by migrate. This runs all currently pending migrations. Can we check? What? The result? Waas? Actually, yes. Remember that our database is stored in this file D beat with extra light. Three. It was created automatically when we were on the server, and at that moment it was empty. But right now it should be different because we just ransom migrations and created some tables can reject that. Actually, yes. We can ask jingle to make a connection with the database using the command Parton militia by D. B Shell. Please note that this command might not work for you because you may need to install it extra light separately. In that case, don't worry. You don't need to be able to do this. I just want to show you what the database now looks like so you can just as well sit back and watch for now. So now I have a connection with the extra light database, and I can ask it's to show the tables that were created, and you can see that there are now 10 tables. All of these were just now created when I ran the migrations. Note that each table name is prefixed with either both or Django most of the time. This is the name off the EP it belongs to, so we see that the both provides us with tables for user's groups and permissions. And of course, this

means it's for each of the tables. There will be a model close in the biting code for the author. There are also a number of tables with the prefix Django. Those don't belong to any specific app. Let's take a short look at an important table. Jingle Migrations. This is an astral select statement. If you don't know SQL, that's okay. You don't need to write any to be able to use Jango. And he was she a line for every migration that has been executed. It shows each migration we just ran with the exact time that it happens. So this is where Jingle records the current state of the database. This makes it possible to compare the middle classes in your coat at any moment with the tables. Now let's exit this. I can't go on to write some more coat off around.

Creating a Model Class

[Autogenerated] So let's write our own mobile close representing a meeting for our meeting planner. But before we start writing a close, we need to add a new jingle app. Remember that we group related components together in APS, the website, every made in the previous model contains simple text pages. Now let's create a new EP that will hold code for dealing with the meeting data for the meeting planner. So I start by running biting Mitchell by start up meetings, which, as you know, creates a new Parton package meetings and let's install the new EP immediately as well. We're going to sitting so by and adding it to the installed Epps, now looking at the files in the meetings we already understand a bit better. With all of these are four migrations will go into the migrations. Fuller will see that soon. The admin is for configuring. The admin interface will see that in a moment as well. Absolute by is for advanced configuration. We're not going to do that in this course. So let's remove that one middle school by will hold our models and views up. I will hold fuse That leaves test stop I for writing unit tests, which I'm also not going to do. So let's delete that one as well. Now the place for our new meeting Mobile will be in models by, as the name implies. So let's open that. And let's start with adding a class called meeting old younger models need to inherit from the base class models Top model. Which makes this an official jangle model, and that means that it will represent a table in the data base. This holds meetings. So how does Jingle know which columns that table will holds? Let's tell it to begin. The meeting has a title and a date, and I'm saying Title is a character field with a maximum length off 200 characters, and date is a date. Fields will add more field soon, but let's start with these note. Our title and date here are a bit different than the object attributes you may be familiar with. These attributes are not set in the dollar, innit? Method like you would do with attributes on a normal python class. These attributes are actually set on the class, so there's something special. Going home General will inspect the attributes of the class and created dollar in and method for us, which creates the properties, title and dates. But it will also help us create a

database table with the right columns so we can save meeting objects in the table. And it's also make sure the database columns have the right type. So that's actually what I'm specifying. Here, please make a title column in the database where we can store text and that they feel American store dates. What the table will look like exactly depends on the type of database you use that might be Mice Trail or Oracle, etcetera, etcetera. In our case, we use SQL Lights jingle. Make sure to generate a Skrill commands to create the tables in such a way that our database understands it. Let's see how that works.

Creating and Running Migrations

[Autogenerated] So we've written a mobile off our own, and we want to be able to store the data from this model in our database. In other words, I want a table for meetings to make this happen. I go to the command line, and rum managed to buy with the command make migrations. What this comment does is it looks at our models and determines what changes we need to do to the database in order to make the database. Mitch are mobile coat. In this case, it finds our meeting class. And of course, there is no meeting table in the database, so the migration should create those for this to work. It's important that our app is in the installed absolutist in settings. Opie, as we've seen earlier. If you forget that jingle will not pick up your models and it won't create immigration, the output off the Make Migrations Command is to file. It mentions here it's interact meetings in the migrations. Fuller called 00001 initialed with pie. Let's open this file to see what's in there, and he received that make migrations generated from fightin called for us that represents steps needed to get our database to the right states. There's a call to create model and the name off the table it creates is meeting. And here are the fields with the types as taken from our middle class. So you see the title and date fields here. But there's also a new field called I D, which is a primary key. This means the database will automatically at a unique number to each new row, so each object to create will have a unique I. D. Running this migration script will result in SQL being executed against the database, and the migration script is not written in SQL because the migration is independent from the actual beckoned. In other words, we might run this against extra light or my school or post press or any other database. And every time it would result in slightly different SQL. Because each database engine has its own SQL dialect to see the actual SQL, this migration will run. I can use the SQL migrate commands. We call this command with two arguments the name of our EP and the name off the migration. You don't have to talk the entire name off the migration. Usually just typing the number it starts with is enough. So here you can see that the migration will cause a create table statement and we see that it creates the I. D column, a title column and a data column now to actually run this SQL and created table. I say

bite unmanageable by my great This will run or currently waiting migrations from all the eps in my installed absolutists. You can see it runs the new migration from our meetings at And if I now look at the data base, we see that we now have a meeting stable noted the name off the table is prefixed with the name of her EP meetings. The part after the underscore is the name of the middle class meeting.

The Admin Interface

[Autogenerated] So we've created the meeting clothes and there's not a database table to store meetings in. But how do we actually create meetings and store them in the database? Well, Jingo comes with a handy auto generated interface called the Admin interface that lets you create and at its model data to use. It will have to register our model with the admin site and configure a super user account to be able to log in. Let's take a look the evidence that comes with Django by default. And as you can see here it is in the instrument packages shutting. There's also some Edmund related called in your also pie. The euro's patterns list, by default, includes this call towpath. This works slightly differently than our own wrappings because the admin interface is different. Let me just say that usually you will not need to make things this way because it's a special thing for the EP in to be able to create and edit meetings in the Edmund interface, we need to change this file in our meetings up. EDS. Mingled by Let's start with importing our Kloss and now I can registration so it will show up in the usual interface. So that's all the setup we have to do. Let's go and start the development server and open a browser. As we saw in the Euro configuration, the admin site is hosted. Honor slash. Admin. So that's the euro I want to visit, and apparently we need to look in here. But we first need a user account. Let's make one. I'm going to stop the surfer end run managed by create super user. Here, you get to choose your user name, filling your email address and choose a passwords. Now Jingle does validate your password to see if it's strong enough. But because this is a development server, I can tell to bypass the password validation and create the user anyway, even though I have a weak passwords now that I have a super user, we can use this account to log in. The interface you see here is completely auto generated by Django. As you can see, there's already three models here. Users, groups and meetings, and the uterus and groups are always present because they're defined in the oath that's installed by default. Now our project doesn't use groups and users right now, So instead, let's click our role model and add a couple of meetings. As you can see, Jingle knows how to generate a nice HTML form for our model fields. So there's a text field here for the title and a date picker for the date fields. Let's say I have a meeting with my project team about my project requirements, and that meeting is tomorrow and I'm going to click, shave and at another. So let's

say I have another meeting on the sixth where I will present the progress off my project to management's. And let's save this. This brings us back to a list off all existing meeting objects. Right now, the way our meetings are listed here is not very informative. Just test meeting object one and two. But we're going to fix that in a moment. For now, you should understand that this number here is actually the unique I d for each objects. Remember the Jingle editor i d column to our meeting table. Well, when you save an object, this idea automatically gets her value, so that's where the one to etcetera comes from. So the Edmund interface is very convenient and powerful It's a very nice way to create and manage data for your site. But the admin interface is called the Hetman interface because it's meant to be used by administrators. This is not your sites. We're not going to use this as the front end to show to our users. Actually, our users will never see this ATM inside, and the next module were actually going to start building a user interface for the end user.

Review

[Autogenerated] Let's do a bit off review off what we've learned so far. This whole module is centered around jangle models, and the entire purpose of models is to make it possible to save our part in objects in a database so that the data keeps existing after the server stopped running. We've seen that our middle class is our maps. Two tables in the database. In other words, when I create a meeting model, this will result in a corresponding meeting table in the database and in the table. The columns correspond to the fields off my mobile close like the start time and the title. This allows us to work with the database just by writing python coat. Jenga will make sure to generate the SQL for what we want to do and run that against the database. It creates the tables and columns for our mother classes. When we run migrations and execute the SQL, we need to update, insured or delete data when we manager data through the admin interface in the Standard, General installed their support for several databases. Both dress really be my Skrill Oracle and issue light, but those are not your only options You can store your data with many other database engines as well through third party libraries, which give you support for IBM D B two mikes, off sequel server and many other beckons. We've seen how to write a middle class for which Jango can generate a database table, and these classes usually go into a file called Model Stop by and then have to inherit from the General Model Super Class, which is in the jungle Doctor be models module. Any classes inherits from that super close who'll be mapped to a database table. The name off the table is taken from the class itself, although you can set a different name if you want to. Then in the body of the class, you define the middle fields. And in the example here we have a character field called name and an interview field Gold H. So how do

you know which types of model fields there are and what options they take in this course? I'm not going over all the possibilities because we have too much ground to cover. But let me show you what you can do if you want to know more. I usually just go to a search engine and search for jungle model fields, and this takes me straight tow the reference documentation that tells you everything about the different kinds of fields and the options they take scrolling a bit down on the right year, you see a long list of all the different kinds of fields you can create with Django. When you're writing code for your models, you will be changing your database A lot, as we've seen general automates is for you. But make sure that you have installed your app by adding it to the installed absolutist in the project. Settings or otherwise, Django will not see your models, so the workflow usually goes like this. Suppose you make a change to a middle class. You then call the Make Migrations commands to generate a migration script that reflects the change in your model. It's always wise to check the coat jingle generated to see if it does what you had in mind. Sometimes you want to see an overview of all migrations and which ones have already been run. You can do this with the show migrations commitments. Finally, you can run the pending migrations with the migrate commands. So to register a model with the Edmund sites, we only have to add a very simple piece of cult in at Mendel. By off course, you need to make sure that you import both e edmond package and you're our mobile class. And then you can simply say at Mendel cited register and pass your model as an argument off course. If you want to look into the admin, don't forget to create a super user with managed by create super user.

Bringing It All Together

[Autogenerated] So how does all off this translate to the daily work off a developer? Let's see some more tea. Most. Let's go over the entire work flow again. We're going to create a new middle class run migrations and make it editable in the admin interface. But this time we'll also add a foreign key, which enables us to store a reference from one class to another in the database. Let's add some more fields for meeting class and see I'm opening model. So bye. And here I've had two new fields. A time field for picking the start, time off the meeting and a duration in hours, which is an interview field. Now we've changed the model. I need toe at these two fields as columns to the meeting table in the database. So the next thing I do is add a migration with Bison Mitchell by make migrations, and now we get a message here you're tryingto at a no notable field. Start time to meeting without a default. Why is this a problem? Well, when we add the new column for the start time, we also need to provide a value for the new field for every road that is already in the database. So we already created two meetings in the admin interface, and those will also get a start time and a duration field and empty fields are disallowed by Django by

default. So Jingo is now asking me how I want to solve this. Well, I'm going to press two to quit and let's add some default values to our fields. First, let's imported the time module, and then I'm shutting the default starting time to nine o'clock, and the default duration for a meeting will be one hour. So you can set the default value for a field by saying default equals, followed by an appropriate value show, a time for a time field and an interview for an interview. Fields. Now let's try to create a migration again. Now we see a new migration is made and, of course, that this will be a new file in Immigration folder and we can open it, and you received that the script contains two at field operations getting a duration and a start time, including the default values. We just add it now. The last step, of course, is to actually run our migration by saying fightin managed by migrates, and this will actually at the columns to the database. Now, before we check out the results in the Edmund interface, let me add a string representation to our class by defining the dollar strewn with its that returns to string with the title, the start time and the date off the meeting. Now you might wonder if we also need the migration after adding this function. Let's see, I'm going to run, make migrations and now this tells me that nothing has changed. This is because we have not changed any off the model fields. We only edit some behavior, and that's not part of what will be stored in the database. So let's run the server and going back to the admin interface and reloading. I'm going to click here all my meetings and the way our models are now displayed age a bit friendlier. If I click at meeting, we now see all four of our fields and the default values are filled in already. So before we continue, feel free to explore the admin interface and see what you can do with it.

Adding a Second Model Class

[Autogenerated] Now let's practice the whole models and migrations workflow Another time. First, I want to ask you to try something for yourself, please. As a mobile, Close called room to present a meeting room, and the room has a name of floor number and a room number. If you're going to try to do this, please post the video and writer coat. I'm going to wait for a couple of seconds before I show you the solution So used to coat. I wrote for this. The classroom, just like the class meeting inherit from Models model, and we give it three fields a name, which is a character field, so that's equivalent to saying it's a string and I give it a mix length off 50 characters. We also have a phone number and a room number, both of which are interview fields. I also edited dollar strewn with it so that the rules we create will look nice in the admin interface and like we've done before, we create a migration for this and run it very well. Now, when I plan a meeting, I want to be able to select a room to meeting to do so. I'll add an extra field to the meeting class. So here I say, room is model stood for in key, and this adds a foreign key relation

from the meeting close to the room class. Foreign key is a concept from SQL databases, and I'm not going to explain the details about it. But what it comes down to is that this field will hold the I. D off the room object that this meeting references in a moment we'll see in the jungle Ataman. This will make it possible for us to select a room when we create a meeting. Now the only lead argument here is required, and this is also a database specific thing. It determines what happens when a room is to remove from the database by saying Cascade here. I'm saying that if a room is deleted, all meetings for that room will also be removed, and that seems sensible. There are also other options, but that's very much a database topic, so I'm not going to discuss that in death right now. We're not allowed to create a foreign key in general without specifying the on the lead behavior, so that's why I put it in here Now again, let's make a migration. And here's the same warning that we've seen before. I'm adding a column room without setting a default value. So jingles asking, What do you want to do with the meetings that are already in the database? Which value do you want to put in this column for those meetings? Well, actually, I don't have a good answer to that. So let's say two for quit. And instead of fixing this problem, I'm going to make my life a bit easier. First, I'm going to throw away all existing migrations. Make sure not to remove the innit Opie file here. So I'm just removing the migration files from the Migrations folder in the meetings up. But the in it'll pie file stays next. I'm also going to remove the database. This gives me a completely clean slate for the database. Now I can say make migrations and because right now we have no database. It will see no existing database table and also no existing meeting objects, so it will create one single migration file with both my model's meeting and room and all their fields in a single file here. This shower. Looking at that, we now have a single migration that contains called for both the room and the meeting model. Now let's run the migrations, and I see that all the default migrations for an empty project run again. Because, of course, we're running it against an empty database. So it has to rerun all the migrations, so everything runs without any annoying questions or warnings. But the downside to this is that you lose all the data you entered. Now, when you're working on a new project like this, that's not a big problem at all. Losing all data also means we have to recreate the super User. And now, before I run the server again and start creating new rooms and meetings, we also need to make sure that we can actually use our room close from the admin interface. So we have to at the new room close to the admin interface in Edmund by So here I am not also importing the new room class and registering it for use in the admin site. Now, finally, we can run the server again and let's go to the admin interface and let's go and create a meeting and then here we can now select a room. But of course there is no room yet in the database, but the little plus sign here allows you to create one on the fly. So let's do that. And let's say we have a large conference room, which is room 21 on for two, and I click safe. And now I can select this room here now clicking safe. This is the over few off all my meetings,

going back to the home page and selecting rooms you can now see. We just created a room. So this shows you that by adding one line to our model, namely this foreign key here. The admin interface gives us a very convenient way to create model data and link those objects together in the next model will start creating a user interface for the actual users. For our meeting planner, they will not be able to create or edit rooms, just meetings. So that shows you already for whom the administer men's. It's for the administrator who can actually at the rooms, and that brings us to the end of this module. We've learned how to set up a data model. With Young Go, we've created two mobile classes, created and run the migrations to shut off a database and used the admin interface to create an edit data. Now let's move on to the next module, where we'll learn about templates and the model template view better.

Combining Model, View, and Template

Introducing Templates and the MTV Pattern

[Autogenerated] Hi, My name is Rachel Liquor and in this majogo at template Sure project and learn about the MTV pattern. Just about every modern work framework and that includes python frameworks follow the same basic pattern called model template view. You can see this. There's a set of best practices for how to organize your coat. There are three types of components. Models, templates and views, and each have a clear responsibility in this module. I'm going to teach you how to implement this better in Django. By the way, you might know the same pattern by the name model vehicle controller, and that is actually the same thing by another name in the world. Off Python Web frameworks. People are just used to calling it MTV. Instead, off M V C now views or Vue functions and middle classes we've already seen. So to make our application full of the MTV pattern, we have toe add templates, and then we need to make all three components work together. We'll see how to use our data model from a view function and call a template from a view function, and we'll see how we can use your else to pass parameters to review functions and how to return for a four error when that's necessary.

A Template for the Welcome Page

[Autogenerated] Let's start by implementing a new kind of component called template. Templates are the components that are responsible for displaying our data to a user in a Web application. This playing data means creating a Web page, so we'll use these templates to generate HTML That's the browser can display. We'll see how to call these templates from the view and howto past the data we want to show from the view to the templates. Let's look at our welcome view function. So far, we've written views that return strings, and those strings are the content of our pages. But in a real application, you would want to create beautiful pages with a Gmail and that H Milken become quite large and complex and putting large amounts off complex H E mail in a string in your fightin coat. He's not really nice. It's better to create a separate component called a template that will generate that a she male while keeping your parts and coat nice and small and clean. Now, Django has a default location where it expects these templates, Toby, and we need to make this ourselves. So let's create a folder here. It has to be in our rep. And let's call it a template. Now, make sure to call these templates. Don't forget the s here or this will not work. Now, the best practice is not to put your files directly in the template folder, but to create another folder with the name off your EP. So here I create another four called website. We do this to prevent name clashes. Several Epps might have templates by the same name, but putting them in a fuller with the name off your EP. Make sure we can tell them apart. So in here I will create a new HTML file. Is your editor doesn't have this option. Simply create a text file with this name. Welcome with HTML. So here we see some coat that's given to us by by charm. It's a very minimal HTML file and it has all the basic building blocks for HTML. The first line is a dark type which marks this is an age male five file. Then we see the HTML tech which reps the entire document and we see a head and the bullet part the head contains metadata. So that's data about this document. For example, here. You see the character set for this file and you can add information about the author or the language, etcetera, etcetera. In this case, I'm going to set a title here, and this will show up in the brow for step. Let me fill it in. I'm gonna set it toe Welcome. And the body will contain the actual content for our page. I'm going to copy Paste from coat in here And what we see here is an H one Tech which will show in the brows as a header saying, Welcome to the meeting planner, followed by a paragraph Tech containing some text. This is the demo application for the course getting started with the angle on plural sites. There's a link here pointing to the parole side home page, which will be clickable. Now to make this page actually show up in the browser, we have to use this template from a view function. Let's go back to our views module and in the view function. Instead of just returning the text for the page here, I will not go a function called Render. By the way, this is already imported in the line at the top. This line was edit when we generated our air. Now the first argument would give to render is request, which is also the first argument off our view function. This is an object that holds all kinds of

information about the request that the browser sent like cookies. Http headers and more. Right now, we will not use it except to pass it through to the templates. The second argument issue Name off the template file that we want to render when the function is called, Jingle will look for a file called Welcome with HTML inside a folder called Website inside the template folder, and it will find the template we just created. At the moment, our template file is simply a plane H e mail file, and this function call will just return the contents off our template file. In turn, we returned that value from the few function, and the general server will make sure to send it to the browser. So going to the browser, we see this page with exactly the content from R H E mail page. Now, in case you're working along and you get an error, let me go over some pitfalls. First of all, what happens a lot again is that you forget to use the return key word here. This happens to the best of us. We've already talked about this mistake in the previous module and you would see the same error as before. I just wanted to remind you that you shouldn't forget to say return here. Another thing that happens frequently is that the templates folder is not named correctly. So make sure it's called templates with an S at the end and that it's inside your rep and insight template. There should be another fuller with the name of my AB websites. The argument to rent a template should match the path to the template file relative to the templates. Fuller. Sure. Starting from the temperature fuller, we go into the full of rip site and find welcome today. She male as an example of what can go wrong. Here, let me remove something. So removing a letter here now we see another error in the browser template does not exists. So if you see this error, you may have a problem with the name of your templates. Now let me fix that. And before we move on, there's one less point I want to make about templates.

Template Variables and Dynamic Content

[Autogenerated] this template file looks like a regular H e mail file, but it can actually be a bit more than that. Let me insure something now. This year, the structure with double curly braces is called a template variable, and it's a place holder that will be filled in by Django. When the template is rendered, we can supply the value for this variable in the view method by adding addict is an extra arguments. This ____ does Geico message with this string as their value. So now every time Jingle receives a request for this page, the view function will run and it will pass this dictionary to the template, and this string will be placed here in the she mill. Soon we'll see that these values might come from a database, and this allows us to have dynamic HTML pages where the content off the page is not always the same, but changes with circumstances depending on who's logged in or a form you filled in or a product you selected. So we call this file welcome with HTML jingle template because it's not actually a she male file at all. It's a template

for an HTML file only after Django feels in these placeholders with the values provided from our view function. Do we have an actual HTML to send to the browser. Now a question I get asked a lot is, Can I use these templates variables everywhere in my email? And the answer is yes. You see, as far as Django is concerned, this is just a text file with placeholders. Everything that's not a place or or some kind off special temperature in Texas is left alone by Jingo Incense to the browser, as is, if I want to set a placeholder somewhere in the middle of an 80 mil tec, let's say here in the link, that's totally fine. Let's also give a value for this new variable X in the view and now refreshing the browser. First of all, we see the message here. This data was sent from the view to the template. So this the output from the template very bowl calls message. We also see that the link to parole site is now broken. Why is that? Well, let's look at the source code for this page here in the tech for the link is to value 42 which was inserted, thereby Jango. And this obviously isn't correct, a she male. So that's why the link doesn't work. This goes to show you that you can use templates, variables everywhere in the text. General doesn't really care that this is supposed to be a shame l and just fills in the place. All those you give it, although soon will see that it can do a lot more than that. Let's remove the coat that broke the link as well as the part that's in the dictionary. So the files in the templates directory dynamically generate HTML pages based on the data that are few functions sense to these templates.

Review: Templates

[Autogenerated] We've just learned about templates, and they are the components from the model template view pattern that we used to display our data. Jingo includes its own template ING system, which is a way to generate text files, and in our case we use it to generate the HTML files that are sent to the browser and this place to the user. The templates that we write our also text files containing special place holders for variables where data can be filled in. And that's how the actual a she male is generated. Actually, jingle templates can do much more than that they support if statements, four loops and many more things will get back to that later. So here's an example off a general template for HTML. You can think of it as an incomplete H E mail document containing placeholders to be filled. In the place. Holders are the parts highlighted in orange. Their names are variables in double curly braces. The values for these base haulers are looked up in the template context. This is like a dictionary containing names and values, all the other texts outside off the jungle variable blocks. It's just copied to the h E mail are put today's. So in this way we can create dynamic HTML pages. The content off this page will very depending on the value off the variables. In the context, simply creating the template is not enough. It's the responsibility off the view function toe call the correct template for a page to do so we call the render function and

you pass it. The request object is realism. Name off the template file you want to render. The third argument is a dictionary containing names and values off variables to be passed to the template as the template context. Jenga will look for the file welcome with H E mail in a four calls templates. No, that s at the end. And this folder should be inside the EP that also contains the view out General will feel all the place for us in the template and return the html page ready to be sent to the brow for. And don't forget, toe actually return the H e mail contents from the view function. If you don't, you will see an error

Completing the MTV Pattern

[Autogenerated] Let's see how we can combine the moral view in template. We'll start with a relatively simple page that retrieves so meeting data from the database and since that from the view to the template so we can show it to the user, then we'll apply the same pattern. In a slightly more realistic and useful example, I'm going to implement a so called detail page for meetings. This means we will take meeting objects from the database and pass it to the templates. But we also want to make it possible for the user to select which meeting to show. We'll learn how to change the euro mapping so that it takes a perimeter to select a specific meeting i d. This will also make it possible to select a meeting that doesn't exist. So for that we need to write error handling cards to return before or four error. So let's say my template looks like this. It shows the user of the number of meetings currently the database using a variable called numb meetings, now looking at the view like before years or render function, and I'm still passing it a dictionary and this dictionary is what we call the template context, and it's where Jenga will look for the NUM meetings. Variable the court. Here is where I retrieved the number off meetings currently in the database. Each middle class, like meetings, has a property called objects that lets us retrieve information from the database. One thing you can do with objects is to go count, which will return the number off roads currently in the meeting stable, which we then Pashto the templates off course. We also need to import the meeting class from the meetings up, and I'm doing that here at the top from meeting Stop Models Import meeting. So refreshing the page in a browser. We see the information that would retreat from the database in an HTML page generated by a template. This means that we have now built a page with the complete model template for you Design pattern

Taking a Parameter from the URL

[Autogenerated] Now let's apply the same pattern again to create a page that shows the details for a meeting. I'm going to go into the meetings up and open views toe by at the moment. This is quite empty. Let me add a view function. This function is called detail, and it will be the few function for the detail page for a meeting. In other words, a page showing all the detailed information for a single meeting objects. There's two new things in this view function. First of all, it takes to arguments. So not only a request, but also an argument i d. This will be the idea off. The meeting will show on the page. The other one is this function. Call here meeting the objects. Don't get the meeting of objects Attribute we've seen before. It's actually called the mobile manager for the meeting model class, and it allows us to retrieve things from the database earlier. We use this to get the number off all meeting objects in the table. Right now, I want to get a single meeting object, and I do so by calling meeting dot Objects don't get and facing the idea is an argument like this. P k is i d. Pick a year means primary key, which to our database means the row with this unique I d. And the result is that the database will find the row with given I d return the data in the throw. Jenga will then convert that back into a real python meeting objects which stand assigned to the variable meeting here. Next we call the render function and pass it. Our meeting objects in a dictionary. Sure, it can be shown in the template, and the name of her template files is meeting slash details of the female. Now all of that is nice, But how is the usual goto input the value for the I. D? Well, we can put that information in the euro off the page. So here's our you're also bi. Remember, it's in the Meeting Planner folder and let's start by importing our new view function. So I say from meetings that fuse important detail. And now let's add a euro by calling path like we've done before. So, like we're used to the first argument here is the u R l and the second taste of you function. But the Ural now has an interesting new piece of Syntex. It says Meetings slash and then a less than sign, followed by the text into Colin I. D. And ending with a greater than sign. This is a special piece of Syntex that jingle can understand. What we're saying here is that we expect an integer so a whole number after the slash and the value of that should be best to the view function as an argument named I. D. So now the user can go to the euro meeting slash, too, and that should return the meeting with index, too. But before we can test this, we need to look at one more file that templates, which is in meetings, slash templates than meetings again and then detailed HTML. So this is a very similar page. Tow the welcome page we saw before the meeting. Variable will hold a meeting objects, and we use it in double curly braces to retrieve the title off the meeting and use that as the page title. We also get the date in time and the room as well. So let's go in test whether this works. If we go to meetings slash one. We see our first meeting and meeting slash two gives us the other meeting. So the number we put into the euro here is matched by this part off the euro mapping and then passed

to the view function as the I. D arguments. The view function retrieves the meeting with that I d and passes it to the template.

Returning a 404 Error

[Autogenerated] when we enter an I D. That doesn't exist, though we get a debunk few showing a does not exist error. Let's see what happens here. So when the user requests of page meetings slash one, this matches the Ural Rule for our view function, and the I. D printer gets the value. One, which causes the meeting with index. Want to be retrieved from the database and shown to the user. That's all well and good. When I use an index, it doesn't exist in the database. This line here will throw an index error. You could, of course, decide to handle this by wrapping this line in a try except Brooke and then handle the index area yourself. But Django actually has a nice utility to help us with this. We can import it on the first line, and it's called Get Object or four or four, and we can replace are called tau objects. Don't get with a call to this function. This function will not call. Objects don't get for us, but we have to pass in our middle class meeting as its first arguments. It focal objects don't get on the meeting clothes and pass it. The second arguments Piquet is i d. The nice thing is that it also handles the case where the I D. Doesn't exist in that case, because the user is asking for something that can't be found. The correct response is to return an 80 to be four or for not found eric out. So that explains the name off the function. It will either get an object for the given class with a given I D or it will return for a four error to the browser. So this result within all existing index now results in a 404 not found. Paige. It's still an error, but a more correct one. By the way, if you were to enter something that isn't an end, this also results in a for a four page, and the same is true if you leave out the index altogether. This is because looking at our coat, we have specified that the euro, for this view is meeting slash, followed by a number So none of these other cases matches this pattern, and that's why Jenga returns for 04 No Euro mapping can be found for those your else

Review

[Autogenerated] the model template few pattern makes of the core structure off a jingle replication. It consists off three types of components. The view which defines the behavior. For Paige, it's responsibility is to determine how to send the appropriate content to the client. The view's we've seen so far are functions which we met through a Ural, using a better in your I So bye The template has the responsibility to create a user friendly presentation, and it does so by generating an H E mail page to be shown in the brow For. And then there's the model, which

represents the data that lift in our application in general, we right middle classes, which are meant to database tables. All of this might seem familiar to you. You might know this better already by another name model. Few controller, as it's known pretty much everywhere outside of the biting world. So how do these components worked together to serve a webpage? Well, as you know, General application does nothing on its own. Everything is to start with a request from a browser. So let's say I go to the ural slash meeting slash one in my browser. What happens when I press enter is that the browser sense and HDP get request for this You're ill to the jungle server. Jenga will look up this your I in your Elsa Pie and see that it's maps to our view function. The view function is responsible for determining how to form a response, and that means it needs to determine whether to call template or the model in what order, etcetera, etcetera. So in our case, we first call the database, which returns to data for our meeting. And then we sent the data to the template, which returns in HD Milk page with the data fielding, which he then sent back to the browser, and the browser shows to the page to the user. So this is the whole flow through the complete MTV pattern in a general application. And in my opinion, understanding this flow and the responsibility off each component is one off the main keys to becoming a proficient Web developer in any framework. When you really understand the order in which each component does its desk and how information flows between them, you are already a step ahead of the competition interview functions. We've now written called to retrieve model data from the database. And we've seen that middle classes have adult objects attributes which will allow us to retrieve the data. Let's go over some functions that you can use on the stock objects attributes. First of all, there's dark objects, not all. And this will retrieve all right from the table and return a list of all the objects. So saying, meeting that objects not all will give you a list off meeting objects that represents all the roads in the meeting Stable. Now, we haven't seen this function in practice yet, but we'll see that soon in the next module. What we did see is meeting the objects that count, which will give you the count off all the roads in the table and machine meeting the object gets, which will retrieve one specific object by i. D. From the database. Our meeting Detailed view has an argument. I d. So we can tell it to which meeting to show. We try to retrieve a meeting with that i D. And if there's no Sutro in the database, we want to show an error. This is such a common thing to do that jingle has a very nice shortcut method for it called Get Object or four or four. It does exactly what the name implies, which is retrieving, object or show off for a four error. If you look at the last line here, you'll see how to use this method we call get object or for a four, and we pass it as the first argument, the middle class that we want to retrieve an instance off. Next we say Piquet is I d. So we're retrieving the object with a given i D as its primary G. This will either result in a 44 error being shown to the user or a meeting instance being assigned to the meeting variable. We also saw that it's possible to add parameters to your your l's. An example of

this is shown in the coat example at the top. The part in orange says into X between angle brackets. This expression will match a number in the Ural and assign that number to the view parameter called X. In other words, if the usual browsers to the page slash example slash five, the my view function will be called with the argument. Ex having the value. Five. So this is a way to use parts of your l's as inputs for our view function. By the way, if a request comes in, that doesn't match this like slash example slash Hello. This doesn't match the euro mapping, and you will receive a four or four error in this module. We learned about the model template, few pattern and at a temperature project to make the pattern complete. To make all the components work together, we learned how to call the model and the template from our view functions. We also learned how to take few parameters from murals and return for a four error when appropriate.

Urls and Link Building

Project Structure Overview

[Autogenerated] Hi, my name is trying to liquor in this model will get some deeper understanding of your ALS and learn about link building. This model is focused around your else and the best practices for doing your your own wrappings in Django. One thing we're going to look at is linked building, which is the practice of automatically generating murals for the links in your pages based on your your own wrappings. To be ableto doing building, we need to name our your own wrappings, and we'll also reorganize our euro. Make things a bit to follow best practices for working with our jingle EPPS Along the way, one of the things will learn is how to use a four loop in a template. Now, before we start writing code, let's go over all the files that we currently have in our project because, frankly, the whole jangle project structure can become confusing very quickly. So what do we have at the top level? There's Django getting started folder. I created this with my charm, and it's a fuller that the holes of the Jungle project. So it's not the jingle project itself. Inside there, there's the Meeting Planner folder and this is her actual jangle projects, which we created with the Jingle Edmund script saying Start projects. So this is where all our jingle files live. Now, inside the Meeting Planner folder, there's another Meeting Planner folder, and this is our core project. Fuller. Let me open it. It contains setting so pie with all the project settings, including the installed EPPS lists, where in store EPPS and you're also by which contains all our your ailment things. By the way, we're going to break that up into multiple files in this module. Then there are two EP folders, meetings and website. The website Fuller only has the view so pie,

which contains a welcome view, date you anything about few anything's the templates for some off those pages as well. Let's look at our other meetings, and the idea of the meetings up is that it should contain all coat about meetings. So, first of all, there's models by here. We also have a view, so by insight to meetings up the meeting's air passage, own templates fuller with the template for my detailed view because of my meetings at pairs a model supply. It also contains a Migrations folder that contains the database migration scripts. Finally, in the top Most Meeting Planner folder, there's debate that assure, like three, which contains our data. And there's the Mitchell by script, which allows us to run commands like managed by start up run server. Good. I hope that gives you a clear overview of all the files we have and that we're now ready for the next demo.

A Template with a For Loop

[Autogenerated] to make ourselves a little more you differently. It would be nice if there was a way to navigate to the meetings from the home page. To do so. We need to add links to our home page, and to do that we need something called Link Building, which means we're going to generate the euros for our links from our Ural. Nothing's one of the things we need to make that work is names for your l's so well at those, and we'll see how to implement a four loop in a template. So let's add some links to the home page. Here's my welcome view, and at the moment it gets account off all the meetings in the database. Let's change the line a little and get all the meeting objects instead. Instead of calling count, I'm now calling objects. Not all. And let's change the variable name into just meetings. What happens in this case is that Jingle will retrieve all objects from meeting table and store them as a list in this variable. Well, actually, it's a little more complicated, since General doesn't actually retrieve anything from the database until we start displaying the contents off the list in the template, but I don't want to go very deeply into that right now. It's a good thing to be aware, though, that generous doing here is more complicated under the hoot than you can see from the coat right here. Let's move to the templates now before we continue. If you like me. You're not wondering what happens if I used this template right now because it uses no meetings and we've just removed that from the view So the NUM meetings variable doesn't exist anymore. Well, we get an error or not. Let's check in the browser, just to be sure. And this is what we see. There are currently meetings in the database, so a temporary doesn't given error. If you use a variable that doesn't exist, it just prints nothing. So here this expression, no meetings. Right now, it just does nothing. The variable doesn't exist, but that's not really a problem. The template just ignores it, so you should be aware of that. If you use a variable in your template and you don't see any output, it might just be a misspelling off your

variable name. Now I'm going to remove this paragraph because we don't need it anymore. And I'm gonna copy paste from coat in here And what a ferret Here is a header Meetings followed by a U L Tech and on ordered lists. And he will see a new piece off dangle templates in Tex Curly braces and percent signs. This is the jingle templates in text for a statement like, if or four in this case, I'm making a four statement. This works very similarly to a normal pipe and four statement. But there are some differences before statement in a template Needs to have an end for, as you see here, marking the end of the block, we say for meeting in meetings, looping over the meetings very bowl, which will hold all meetings from the database because that's what I've passed from the view function. Each of these meetings will be assigned to the new variable meeting one by one. And every time we go through the four loop, the entire block gets evaluated and the resulting text guests included in our page. So this will generate as many Litex and links s. There are meetings in our database inside each link tech. We have the address attribute, which takes the Ural to link toe Here we construct the euro for each meeting objects. So here I'm saying slash meeting slash And then I use a template variable expression with two curly braces toe at the I. D off the meeting objects So this will generate the euro for the detail Page off that meeting. Let's test this on the homepage. We now see a list of our meetings. Clicking on one off them takes me to the page for that meeting, so that also works nicely. So this list we see here was generated by the full Lupin the template Looking at the source Scout, this is the part of the H E mail that was generated by the four loop each. L E Tech with the link inside corresponds to a single meeting and to a single iteration through the four loop. The text is exactly the same every time, except for the index in the euro and the text inside the link. And here's the four loop again for you. Note that the ul tech that creates the list is outside the four loop because we only need that attack once inside it. I start my four loop and inside the four loop, I create an l E Tech and attack for every meeting. The parts of the text. The change are created by the temperate expressions with double curly braces. And don't forget toe end the four loop with an end four statements.

Link Building

[Autogenerated] now here's something else to think about currently. Our view and templates are tightly couples, and that's a bad thing. What do I mean by that? Well, if I decide to change this Eurail mapping for the detailed view to something else, this breaks the template because the template has, ah heart card link. And if you have hard coded links in all pages on your site, you will make it very hard for yourself to maintain your templates. When you're else change. Think about a scenario where you have 100 pages that link to the meeting detail page. Then if you change this

your own mapping, you will have to update all those pages. But there's a better way. Django defines a special template tech called your L, which looks up a specific your own mapping. We use a curly brace and percent signed to call it, and we passed the name off a specific Ural mapping. In this case, it's the detail for you in the meeting said. After that, we can add any arguments to pass to review function. So here's the meeting. I D. Jenga will know looking. It's your own wrappings and determine the correct your l to put in the HTML to create a working link. So let's test this out. Unfortunately, we get an error here. No reverse match. And that is because we didn't give our euro mapping in name yet. Let's go two euros a pie. And here I can give my euro mapping and name by adding name is detail. Let's re lock the home page and he would see our list with links again. Let me click one, and this works. Now if I decide to change my euros now, let's say by adding something different in front of the Ural. Now, if I really loved the home page, the link still work. But as you can see, the euro off the detail page has now also changed. So that means the actual euro in the links on the welcome page has changed as well. And that's because, in the template, we're using the Ural Tech to dynamically generate your l's instead of heart goading them. Of course, now, before we continue, let me remove this nonsense from our your l's. Now I have a little exercise for you. Here's the template for the meeting detail page and I'd like you to apply what we just saw. So here's a little exercise please at a home link to the welcome view here using the U. R L template tech shall please take a moment to try toe at the h e milk out for a link that links to the welcome view. Using the Ural Template Tech, you can post a video. I'm going to wait a few seconds and here is my solution. I hope you got something similar. If you couldn't work it out, that's OK as well. Just keep practicing. At some point you'll get it right, too. So here's my coat. I used the HTML atec to create a link with the H ref attribute, and it takes the actual Ural to link toe and to generate the euro. I linked to hear I say curly brace percent sign your Earl. This is a jingle template, syntax, and I say I want to link to the welcome view. Note that I say welcome between single quotes. Then I close the jingle template tech saying percent sign Cory Brace and then I say a home. And that will be the clickable text for the link now on its own. This is not going to work right now. If I go too well off my detail pages, I get this error. No rivers match, and it says reverse for welcome, not found. Welcome is not a village view function or better name. Well, this might ring a bell, so it's not a village name. Why is that? Because we didn't give our Ural mapping and name. So looking at your else took by what I have to do here is toe add name to the euro mapping for my welcome page. So that looks like this name is welcome. Now, if I refresh the page, here's my diesel page and now I have a home link and if I click it, I go back to the welcome page. So again, here we have our euro mapping it now has a name, and that enables us to use the U. R L Tech here to generate the correct euro for the welcome page. We call this link building

Listing All Rooms: An Exercise

[Autogenerated] the next part we build I would like to do together with you is another exercise. But this time it's an exercise in which all parts come together, so it's going to be a bit more complex. It goes like this, please, as a new page that shows a list off all room objects before we start. It's a good idea to go into the admin interface and create several rooms to give an example years my list off rooms. So make sure to go into the admin interface and add some rooms for yourself, so we have some data to practice with. So the idea is to create a page that shows this data to the user to be clear. It's okay if you just show a bit of text for each room. There's no need to create links for every room. So to do this exercise, you'll need three things of you function, a template and a Eurail mapping. Now this is quite a bit more complex than the other exercises we've done so far, so just try to do this step by step. If there's any part you cannot figure out that's okay, just what your bit of the solution and then try the next step. It's okay if you cannot do it all, you're learning after all. So now you may post the video if you like. I'm going to wait a couple of seconds before I start showing the solution. So let's begin with the first step. And that's the view function. Here's my solution. I called this function rooms list, and, as always, it takes an argument requests, and it calls the render function passes the request object and then the name off a template. Meetings slash rooms list dot html It's okay if you name your function or your template differently, but it's best practice to put your template inside the meeting's folder because we're currently working in the meetings off course to make this work, don't forget to import the room class at the top. Next, we have a dictionary, which is also passed to the render function, and I call room the objects of all. And this is the core functionality off this fuel function to retrieve all room objects and pass it to the template. I assigned it to the G rooms here, but if you called it something else, like our That's finest. Well, just make sure that you use that name in the template as well. The next step is the template, which, in my case, I'm going to put in meetings, templates, meetings and then rooms list with a she male. The H E mail here has the familiar structure that we've seen before. An H email header with a title tech where I say rooms and a body. Most of these things can be slightly different or even completely absent, and everything will still work. Year in the body is where I do the actual work. I have a UL unsorted list with a four tech where I say four room in rooms. Rooms here refers to the key in the dictionary that had passed from the view function. Then in the four loop for every room, I create a list item and print the room. Now, if you use something else year like room dot name, that's also fine. As long as you print some information about each room in this loop. The third step is the Ural moving. Let's open your l stopped by and here I'm going to add a mapping for our new view function for the you're ill at best, A string rooms. Then I passed the view function, which is named

rooms list, in my case again, if yours his name differently. Make sure to use your function name here. And I've also added a name for this. You're as well calling it rooms. If you didn't name your Ural, add the name right now because we're going to use that in a moment. Now to make this work, make sure toe also import your view function. Okay, so that's all three of our components. A template of you and the euro mapping. Let's test this going to the euro rooms. We now see a list of all our rooms goods. Now, finally, I want to make a link from here the home page to our new rooms Page, if you like, you can try your hand at this as well. Just pause the video. So here's what I do in the welcome template which is in the website air. I add this line. So this creates a link with the Ural Tech which references the name off my Ural napping rooms. And now we have a link on the homepage in clicking. It shows us the new room mist

Best Practice for Urls and Apps

[Autogenerated] Let's talk a bit more about your I'm avenge. You see, currently we're not following the best practice from mapping our euros to views. What we're going to do is to add a Eurail supply inside our meetings up and change the main euros that pie so that it includes the EPS, your own wrappings and adds a prefix that sounds kind of abstract. So let's see how that actually looks. Usually an app will contain multiple views, and most of the time you want to put all those shoes under the same prefix. What I mean with that is the following. I have a couple of different pages in my meetings up my meeting detail page and the room's list, and soon I will add another page as well. Now a very common thing to do is to make all the Urals for all pages in the same hep to start with the same prefix, let's say meeting, slash, detail, meeting, slasher rooms and so on. The best practice to handle this is as follows we create a new file called You're also bi Inside are up and let me copy base from code in here. So now my meetings. EPA has its own. You're also bi and it looks like this. I start by importing the Beth function and the import of use function from the up itself as well. Then I create a new variable called neural patterns, which, just like in the Euro, so by we saw before is a list off your I'm epics. So in here I have your own wrappings for all the views inside my head. Now, to explain how this works, let me also show the Eurostar by from the meeting planner folder. So this is my project white, you or else we'll buy. And here are the old your alma things for the hetman and a rope from Paige. And then here are the meetings for the views in my meetings up the detail page and the room's this page, and I'm going to remove those and replace them with a new line. So here, instead off mapping single view instead, I'm including the Urals for my meetings up and includes method will look in that module and find the euro potential is there. Interestingly, in this case, we passed the name off our your I's module. Meeting's off. Your I's as a string, so we don't need this import statement anymore

because we don't need to import the views from our meetings up anymore. Also, I have to import the include methods. So to explain again, exactly what this line does is it includes the euro patterns on the right year from meetings slash. You're also bi in the project. What you're also by with the prefix meetings. So the actual weapons we now have is meetings, slash rooms and meetings slash, followed by a number I D. So what will happen when I open my browser and go to meeting slash one? Is that this first argument off the culture? You're all here will match the first part of the Ural meetings. Slash Jungle chops that off off the euro. So then left with just a number one and include causes it to look in my meetings. You're also bi and then we fight a match for the remaining number one. So now reloading the home page and going toe the rooms list. You see that the rooms list is now hosted under meeting slash rooms, but the link still work. So what we've done is to create a euros a pie in the meetings up so that the meeting's at can define its own your own wrappings. All those weapons will be included under the prefix meeting Slash Because we're including them here in this line in the project of White, you're also bi.

Review

[Autogenerated] When you configure a euro for your few, you can add 1/3 argument name in this example. The name is detail. We do this so we can refer to our euros by name. Like here in HD Milling. I'm using the Ural Tech to refer to the meeting detail page by its name. I can also add arguments for the Vue functions. So here I'm passing the idea off the meeting to the View function as well. To understand why we do this. Imagine a project with 100 pages with links to the meeting detail page. Now if I need to change my euro met things that's not a problem. I simply change the euro mapping and all my links will keep working. This way. I make my coat more maintainable. Now Don't forget the quotes around the view name in the Ural Tech because if you forget them, it won't work. And there's another piece of temperature index machine as well. If we use curly braces and percent signs, we can use the so called template tax. The example we've seen is the four tech, which works just like a python for oh, so here I'm looping over meetings, which is a variable from the template context, and the HTML that's between the four tech and the end for tech will be repeated for every meeting in meetings. Then we used the double curly braces, toe prints, the I D for each meeting. If you haven't up with his own views, the best practice is toe. Also give it its own neural configuration inside a file called Your Elsa Pie in the folder. Generally speaking, this file is not that different from the eurozone by we've seen before. We import the path, function and the views we want to map, and then we create a variable called neural patterns, which is a list, of course, to the path function. So then, in the project of White, you're also bi. We want to use the euro conflict from the F, and to do that, we have to start

importing not just a path method, but the include method as well. Next again, we have a euro patterns very well containing a list off course tow path, and he would see a different use of the bath function again. The first argument is a you're ill, but the second argument is now called tau includes, which takes a string argument. And that's the name off the module that contains the Ural Confit. So meetings with your L's refers to the file your l stop by in the meetings. This will look up the Ural Petrus least defined in that file, and all your own wrappings in that list will be included in the projects white, your own mapping with the first arguments that would give here as a prefix. So all your else from meetings of Urals will in this case, start with meetings slash In this model, we've learned a lot about you or else we saw how to do link building generating Urals in templates. Using the names we give to our your l's. We also saw how to reorganize our neural structure to give Epps their own neural configuration. Along the way, we learned how to make a full up in a template. I hope to see you in the next module about styling and template, inheritance

Templates, Styling, and Static Content

Adding Styling with CSS

[Autogenerated] Hi, I'm Veronica Liquor and in this module will learn howto add styling to our site using template, inheritance and how to use aesthetic Golden's. What I want to do in this module is to apply some styling to our side to make it look a bit nicer. Now I'll admit I'm not very good at making things pretty, but I can at least explain the technologies that you would use for that. The most important one is a language called CSS and will not just write from CSS. Styles will see how to create a simple file with some styles and apply that to the whole side at once using a technique called template inheritance. One side effect of this will be that it also makes your html a bit nicer and cleaner. So let's get Golding. I'll start by creating a small seaside style sheet and applying that to a single HTML file. Next will use template inheritance to apply the styles to the whole site and restructure our HTML code to be a bit cleaner. In the process, I'm going to add some CSS to our projects. Seizes like H e mail is a thing for the front end, and what I mean by that is that CSX files are sent to the browser and that the brow should reach them. H Mill defines the contents to show and, she says, defines how to style it. So it's about colors and phones and layout. So you might think that seizes files go in the templates fuller with the HTML templates. But they don't we make a new fuller here in the website epic called Aesthetic and in There Again, a website folder. So the

structure here is similar to templates. Remember in templates. We also have a website fuller within their the actual templates and same goes for aesthetic. We have static, and in there we have a folder with the name of her M so in there, creating new text file with the name style diseases. And here's a little CSS I wrote. For those of you not familiar with Jesus, this is how it works. We have a selector here, its body and a list of, she says. Rules applied to that selector. And this win said toe everything in the body tech in my html page, which is just about everything on the page. We will apply the rule that we want to use a sensor reformed, make the text color cornflower blue, and we set a background color as well. So let's see how it can actually apply the Caesars to our site. Let's go to the welcome template for a moment. What I can do to apply our styles to this page is Adam Link element to the head like this. I know that this has to be inside the heads tech. If you put it anywhere else, it won't work. So adding this will cause the style street to be loaded and the season's styles to be applied to the page. Now, if I go to the welcome page and reload, we see nothing changes. And that's because sometimes when you do this, you need to restart your jungle server. So let's do that. I stopped the server and started again. And now going back to the welcome page and reloading, we indeed see that the color rules have now been applied. The court has not been applied to the links, but that's a sees this issue, and this course is not really about CSS, but about Django. By adding more rules to your CSS file. You can make this page just look wonderful, but that's not what we're here for. I just want to show you that we can actually do this. By the way, let's make our coat a little bit better. At the moment, I've Heart called it the link to our style sheets, but we know from the previous module that that's a bad idea because it's much better to generate these kinds of your l's now with the links we used a Ural Tech. But for aesthetic content, there's a special tech and it's called static. So this will look up the file website slash stylus uses in my aesthetic fuller and generate the correct your L. Now to enable this to work. We also need another tech which needs to be at the top off the fire. So here I'm saying, Load static between percent signs and curly braces, and this enables to use off the static tech. Now taking one more look at my CSS file. This is called static content because it doesn't change. We sent this exact same file to the browser every time. There are no template, placeholders or things like that in here. This file is served as it is and this rights called aesthetic content, and it's separate from templates which dynamically generate different content every time. And the reason we use aesthetic tack here is that it's very well possible that when our application grows at some point, we want to change the way we host all our static content. Using this tech, we won't have to change our templates. When that happens, we will just configure Django so that it will generate the correct Urals for where are static content actually will be hosted. And in that way it's very similar to using the Ural text for links because it makes your templates more maintainable.

Adding an Image

[Autogenerated] just to show you we can. You static content for more than just she's s. I've put an image in the static folder here called Calendar Does P and G. If you want to work along, you can put any image you like in There. Doesn't matter if it's a gift or a J. _____, P and G et cetera, and then in my h e mail, I can at the following line image sources followed by the static tech and in there I say website slash calendar. Don't B and G. Make sure to use single quotes around the name of your static file so this will generate the U. R L for my image and note the quotes around aesthetic block. Those were part of the HTML north of the static rock itself, and I also added, with is 100 pixels here to make the image scale nicely. Otherwise, you will have a huge Aiken on your page now refreshing the welcome page who see that the image has now been included in the page. So you conserve images, aesthetic content as well a seizes and actually any kind of file you'd like to house in your project. In many products, you will see a JavaScript Files anaesthetic folder, but it might also contain Jason or even sound or video. Now, I don't really like the way this looks, so I'm going to remove the image deck from my H E mail. But I just wanted to show you that we can do this.

Template Inheritance

[Autogenerated] now, if I go to another page, we see the CSS is not actually applied here because there's no link to the style sheet in the template for this page. So the obvious solution would be to copy Paste the coat that loads the style sheet from the welcome template in tow. Every other template. Right now we only have a couple of templates, but in a larger project you will have dozens or more so repeating the same link to the style sheet everywhere. It's tedious, and if something changes, you will have to go over every template to duplicate the results everywhere. But there's a better way I'm going to use templates. Inheritance First, let me create a new template which I call based off H E mail. I'm putting this directly in the template fuller here in the website. The reason for that is that it is a template, but I will use it from my other abscess. Well, you're not grouping it inside the website Fuller there, So creating a CI mill file calling it based on H e mail. This is my base templates, and it defines the base structure for our other templates. All the coke that is repeated in every temperate over and over. I cannot just put in here. So you see, there's a dark type and a she male tech, a head and the body tech. All these elements had come back in every page. There's also the large static tech at the top and the link to our style sheet. Now there's also a new template structure called a block off, which I've defined to Block Called Title and a bloke called Content. You can give these blocks any name you like. These blocks marked the parts off the H E mail that can be filled in by the templates that inherit from

this template. The Children. Let's see how that looks. I'm going to start with the welcome template. What I'm going to do is to remove the whole header, including the body tech, and replace it with. This extends based on HTML, and this tells Django that we want based on th e mail. Toby, the parents template off this template everything that used to be inside the body tech. We now surround it with a block content and we replace the closing tax at the end of the page with an end block. So now we have a page that says we extend from base of th e mill and here's to coat that goes inside the content block. So looking at base of the HTML again, the resulting A she male will start with the HTML header, start a body tech, and then this block declaration will be filled in with the content from the welcome template. After that, we close the body and a she male Tex. Now we have a second block called title. Let's fill that in as well, and this will make sure that the browser step for this page will say Welcome. So reloading Welcome page. We see that the result is the same. So the welcome page is still generated correctly, even though we've now broken it down into a parent and a child template. We can repeat the same procedure for all other pages, but I've already done that. So let's see. Here are the template for my meeting pages, and again you seethe same structure and extend statement that title block and the content Brock. Now, in my opinion, you can really say that the H Mel gets cleaner and simple by using inheritance. All the ugly declarations at the top of the HTML have moved to the base template, and we're only left with the parts that are specific for this page. Looking at the rooms list, it's the same thing one more time. Now, if we revisit our site and click a meeting, we see that although the content of each page has stayed the same, we know if the styling applied to every page. The same is true for the rooms lists. Because we use inheritance, we can apply styling to the entire site at once with this one tech here.

Review

[Autogenerated] So we've seen that you can do template inheritance with Django, but starting your template with an extensive statements. In the example, we extends from a template gold based off a she male. So that will be the parents template. Alleged female in that template will be included in this template, but we can use blocks to fill in parts of the S E mail from the parent. With content specific to this page, these blocks have to be defined in the parent template. So if the parents template defines a block title, I can use a block title in this page and replace that part off the parent template with my content. We've also seen how to link to a style sheet from H. De Mille with the Link Tech. Now my style's he disserved aesthetic file. So is there a draft a tribute? I could use a stick Your l like static slash style the CSS. But the preferred way is to use aesthetic tech, which generates to your ill when you give it the name off the setting file. This allows us to

use static files in such a way that when my setup changes and I want to host my aesthetic files another way than simply from the aesthetic fuller. We won't have to change our templates. So we've learned about styling and sees this and how to apply that styling tore H e Mail. We also saw that toe apply our style to the whole side at once. We can use template inheritance. A nice side effect of this is that this also makes our edge email cleaner. Good. That's our module about styling and inheritance. Let's go on with our final module about whip forms for creating meetings.

Adding User Interaction with ModelForms

A Template with a ModelForm

[Autogenerated] welcome. I'm Lando Tucker. And in this final module off this course, we'll see how to add form store application to enable, usually in boots. I would like to enable our users toe ad meetings themselves, and we can do so by adding forms for application. On the template side of things, we do so by implementing a she male form. But we don't write the card for that ourselves. Instead, we let jangle generated by using a so called mobile form in the corresponding view function will also have to add some code, specifically will have to process and validate the usual input. And if everything is okay, we want to create a new meeting object and saved to the database. And once that's done, we will learn how to redirect the user to another page to show that the operation was successful. Finally, I will talk briefly about forms, validation and security. Let's start by adding a page for adding new meetings to our application. I will start by creating a template for a whip form that the usual can feel in using a model form object, and they will write the view functionality to process the usual input, _____ the new meeting and redirect the user to another page. I'm going to create a new view function called New, and all it does is render a template called nudes H E mail. The next step is, of course, toe. Add the corresponding template _____ of HTML, and this goes in. The temperatures are inside the meetings up and inside the meeting shoulder. In there. The structure here is familiar. We shot by extending from the base template. And then we have a title block that says New meeting followed by content broke inside the continent. Look, we see a header plan, a new meeting and a form tech. Now this is something new. The form tak by itself does nothing but one way to think of it is that it creates a grouping off Tex. The text inside it will together create a form. Now you could write your own HTML here to create form inputs like a text field for a meeting description, input for a starting time and another

one to pick the room. It's a try, but the nice thing in Jingo is that jingle can generate the HTML for this form based on our middle class. And that's why I have this stable tack here and the curly braces saying Form. This means that the template will need a form variable passed from the view. So let's go back to our view. And here I'll start by adding an import statements from general forms, import moral form factory and right above the definition off the new view. I'm goingto call this function. We just important like this. What's happening here is slightly advanced, so let me explain. The model form factory method can generate a new class for us called a model form, and the middle form can help us create and process H e mail forms. We tell the function to generate a model form close based on our meeting, Klaus, bypassing our meeting close as the first argument. I also tell it that I want to see all the fields off my meeting mobile in the HTML form by saying exclude is the empty list. After this function of goal, the value of meeting form will be the newly generated model form class. So meeting for miss a class, not a regular object, and that's also why I'm writing meeting form here with capital letters to show that it's a class now because this is a close. We can use it in the view function by saying form is meeting form and this will create a new form objects that is an instance off the meeting form class we just created. So for Muir is a mobile form object for our meetings. I didn't best this object to the template by adding this dictionary tour called Surrender form Form Off course. We shouldn't forget to add a Eurail mapping. So let's go to your Elsa pie in the meetings up. So this is where I've already met the other views for our meetings. And here I add a new line mapping our new function to the Ural New and giving it the name new as well. And now we can visit the new bench in the browser. So the euro, for our new view, is meetings slash new, and he was she a form for creating a new meeting. All the inputs here have been generated by the middle form object based on our meetings bubble. So let's see those side by sides. So what we can see here is the jingle tries to generate an appropriate input field for each field. In your model, for example, the title field is a character field, and for that jungle generates a text field. The duration is an interview fields, and for that we also get a text field. But it has a number picker with these two little _____ here, the rooms is a foreign key and jangle generator dropped down for us, which even shows the existing rooms in the database. Now some fields are a bit too complex for Jang out to generate beautiful inputs out of the box, for example, date and time are just text fields, but you can customize this and I'm going to get back to that.

Submitting the Form

[Autogenerated] now, this is what we have. Right now, all thes inputs here have been generated by this single line where we ask our model form object to generate HTML inputs for our middle

class. But we're missing one important thing. You see, the form we have doesn't do anything. It will allow the user to fill in these fields, but not to submit the form and send it to the server. To do that, we have to add the submit button. Now, when you get this button, make sure that you add it inside the form and let me reload the page. Here. There's the submit button. Now, when the user clicks this bottom, this will cause the entire form to be submitted. It will send everything the user typed into the server. So let's try that out. I'm going to fill in the form. And as you can see, unfortunately, we need to fill in the date as a string. So here I have to say 2020 or 301 for the first of March. And I'm just going to keep the start time in duration and let's pick a room. And now I'm going to submit it by clicking the create button. Now we didn't write any kowtow. Handle the form on the chauffeur yet. But let's just look at what happens here. The form gets sent to the same mural that we were already looking at, and you can see that the browser adds a question mark. And then it adds the usual input in the form off key value pairs. And you see the name off each form title data, etcetera, etcetera, and then an equal sign and the data that the usual filled in. But actually, this is not the usual way. You want to send form data to the server. In most cases, we prefer to send the data back using a different kind of http message called a post and we can at this is an argument to the form attack. So here I'm saying, method is post. So now here I am, back in my form. First, I have to refresh this to make sure that the browser have seen the new method exposed in the form tech. So let me just remove all of this from the girl and I'm pressing enter to reload the page and let's fill in the form again and submitting. And now we get an error. CSRS verification failed. This is because Django protects us from a class off a text called she S R F or cross site request forgery. To make a form work, we need to add another line of code to the templates C s R F token in curly braces and percent signs. It's important that you get this inside the form tech is well, now what this does is it in search of a piece of code into your HTML form that allows jingle to check that the data submitted into the form is not coming from a She's our effort. Tech are not going to explain this exactly. But you should at least know that you need this tag in all your jungle forms here in the brows, for I'm gonna press Beck and I'm gonna resubmit the form and we now see that the error has disappeared. Now the next step is to actually process. They usually input. And for that we need to take another look at the view function

Understanding the Program Flow for Processing a Form

[Autogenerated] Whenever you have a form in a replication, there are three basic steps, the first we've already seen. We show the form, which is a set off a similar imports, which are generated by a model form close based on our middle class. When we visit the page that contains the form, that's a regular page few, which means that the browser since http gets requests, get requests

are the most common type of HTP requests and the browser since those when you visit a page by clicking a link, for example, shows user clicks on the link to create a new meeting, the browser sense of get records to our new meeting. Ural, this ghost of you and it creates a model form instance and sends back the XML for our form, and we get to see the page with the Forum. Then the usual filled out the form and click shipments, and that's the next step. Most of the time, we will send the usual input back with a Post requests. So now the View will receive a post request containing the data the user has tapped into the forum. We need to validate it to see that all fields are filled in correctly, and based on the result of the validation, we either show the form again. With errors like this, field cannot be empty or this field has to be a number et cetera, or, if the form has been filled in correctly, we process the data for example, saving the data to the database. And then we redirect to another page that shows a success message to the user.

Processing User Input

[Autogenerated] inside the few methods. I can now add an if statement to test for the HDP method off the current requests. If request up method equals post now, if the request method disposed, this means that the user has clicked, submit and that we can process the usual input now before doing so. Let's also add an else close if we end up in the else close. This means that the request method has Toby get, and in that case, the usual has not submitted the form yet. In other words, get request means that the users simply wants to retrieve the form so that he can see it and fill it in. So in that case, we simply render the templates. So what do we do in the Methodist posts? Well, let's start by retrieving the data that the usual typed in. There's an attribute of the request called Request the Post and this contention data that the user filled in. Our goal is to create a new meeting, objects from the data and store it in the database. And this is another thing that the model form class can help us with. I'm going to say form is meeting form and then best this request of post and this will create another meeting form object. But this one uses the data that the usual filled into the form. Next, I say, if form not his village, which asks the mobile form whether the things that the user filled in our actually valid data this is a very important step ejects whether all necessary fields are filled in whether they contain valid data like an actual number for an interview field or a villa date for a date field. And you should never forget to validate your input before you save it into the database. Not just because you might get database errors, but also because of security. Now engage. The former has been correctly filled in. We go for middle safe, which creates a meeting instance for us and saved it into the database So effortless. Cole. There should be a new row in the database with a new meeting. Next, I say return, redirect welcome. And instead of rendering an HTML template here, the redirect function

will send a special return call to the browser that tells it to redirect to a different page. I'm passing the name off of you here. Welcome. And Jenga will sense the browser, the euro, off my welcome page and tell it to go there. So after successfully submitting the form and inserting a meeting into the database, the user will see the welcome page with the list off meetings. And the new meeting should show up there. And it will be a confirmation to the user that everything went well. But in case the form is not valid, I want something else. Now, please pay attention here. I'm goingto d IND end the last line off the function. So what happens here is that this line gets executed in two cases. Either the request method was not post and we created an empty meeting form and they're showing that to the user or the user already submitted the form. So we have opposed. But the input was not village. And in death case, we also end up here. But in this case, the form variable holds to form with the data that the usual fielding so we can show that again. And actually, the village method above will have added errors for invalid fields. Now, before I show you how that looks, I'm going to have to import the redirect method. So when I goto slash meeting slash knew, this will be get requests and we see an empty form. And when we fill it in now I'm going to use an invalid ate here to trigger an error the 30th of February. Now, when I will click, submit, This will be a post requests and formed off is valid method will be cold and that will detect that the data is not village, so we will not save anything in the database. But instead we will jump to the last line in the function and rear ender the templates. But this time Jango shows us the validation errors in our import. So here it tells me that my date was invalids. So let's set a real date Instead, let's take the 29th because this is a leap year. So that works and now formed what is real. It is true the new meeting gets created and shaped in the database and we get to redirect it to the home page and on the homepage. We now she are meeting in the lists

A Note About Validation and Security

[Autogenerated] Now I want to make a point about the validation off the form. Let's try to leave the title field empty and submit the form. Now, in this case, I get a warning that looks different than the validation error we got from Django a moment ago. And that is because this validation is done by the browser and it won't even let me submit the form. So if you would check the jingle looks, you wouldn't see a post request coming in right now. So here the browser is validating these fields, and it prevents me from actually submitting the form. And the message you see here will look differently depending on the brow for your use, so browsers can actually check your input. But they cannot check everything. So, for example, this pressure I have year Firefox will not check whether the date I feel in its correct. So suppose I do feel in title and I select a room. But let's type in some nonsense for the dates and in this case I can't submit the form and the actual

validation off the input is done by AARP. I think out. And that's why, in this case, we see an error from Django now, because this behavior varies between browsers. So not all browsers will validate your fields in the same way. And because it's actually possible to submit a form like this without using a browser so an attacker can submit any form data they like. That means that you should never, ever trust the validation from your browser. And you should always make sure to validate your usual input in your bison card on the server. If you don't do this, your coat can never be secure.

Customizing Form Fields and Validation

[Autogenerated] as a final demo. I just want to show you that we can actually make our forms behave a little better here. I've added to file called forms by and in here. I've created my own model form class. I don't want to go too deeply into this because I just want to show you that you can do these things. But actually go into details is a little bit beyond this getting started. Course, I'm specifying that we need mortal form for the meeting class and that we want all field in the female form. But here, I'm also doing some more advanced things. I'm setting widgets for some off my fields. And this is a way to control exactly what kind of HTML input Jenga will show in the form. Here. I'm setting a date, and time is types for my date and time fields Israel as a minimum and maximum for the duration fields. Also, I've written a custom validation function again. I'm not going to go into the details, but I wanted to show you that we can do this here. I'm checking that the date the usual fielding is not before today. So it's not in the best now in view. So bye, I'm not importing this new meeting form class. Instead of calling the mobile form factory methods, the rest of the God is exactly the same. It's just using my own mobile form class now instead of the generated one. So now if I refresh my form, page one off, the things you get is a nice date picker for the date. Now, how this actually looks depends on your browser, but most modern browse air support. This the Time field, unfortunately, is still a bit basic. So this just shows you that you can actually customized exact imports that you can show on the form. Now, if we pick a date in the best and submit the form, this will get validated by my own custom validation coat. And we get this error that I wrote myself. Meetings cannot be in the past, so that's just to show you that we've only seen the tip of the iceberg. Jingo has a huge number of features and gives you lots of control over all aspects of your application. In this course, we really were just getting started

Review

[Autogenerated] we started by creating a template to show a form. There are several things that you need to do for this to work. If you leave any of those out, your form won't function properly. First, we need a national form check that tells to browse or how to send the form later to the server. We tell it that the method is boast. Then, inside the form tech, we display the form itself by using the form object inside the double curly braces, which will generate input for all foreign fields. And then there's two seats are left open. If you don't include this line in your forum, General will not accept the post request when you submit the form this for security purposes, and it's just something you will need to do for every four. Finally, you need to provide the submit button so that the user can actually submit the form. To make this work, you need to provide a form object to show in the template as a preparation. We call moral form factory to create a model form class based on our meeting middle class, this newly generated class I call meeting form then I write a view function. I create a new meeting, form instance and then I call render passing the name off the temperate and the form objects. But we want our view method to be ableto handle the form shipmates. And to do that, we need to add a little logic. The basic pattern you see in a lot of general code is this. We start by checking if the current request is a post request. If it is, the user has filled in the form and posted it. And that means we can validate the input and either process it by, let's say, saving something in the database and redirect to a success page. Or if the validation gives an error, we can show those errors to the user. But if it's not a post request, that means it's a get request. In other words, the browser asked for empty form to display to the user with form validation. We mean checking the user input to see if it's acceptable. If the user forgot to fill in a required field or filled in some value. That's not village. We shown error message. We start by creating a form instance, but in this case, we best get the data that was posted. So the form fields now contain the data that the usual put into the form. Next, we can call formed off his village to see if the user input is okay. If it is, we go for not safe. We shall save the input from the form as a model instance and then call redirect to another page that shows that the action was successful. If the form isn't village, we render the temperate again and form we show in the template now contains the data from the user as well as the validation errors. So the user will be presented with his own input. Israel is the error messages that tell him which feels should be corrected. And there we are. This was the final module off this course and we've learned how to add a whip from tower application that enables to use your toe at meetings. Creating such a web form takes several steps. We started by creating a template that shows the HTML form and the actual HTML code for the input is generated by a mortal form class that we generate based upon our middle class. Then we wrote a view function that validates to import. And if the usual importation okay, we create a new meeting object and safe to the database. And then we redirect back to the home page. We also had a short moment to talk about forms and security. So that's

all for this Django getting started. Course. Thank you for watching. I hope you enjoyed it. And I wish you happy coating with Django. This was radical anchor for pleural sites.

Course author

Reindert-Jan Ekker

After studying Artificial Intelligence, Reindert-Jan has worked in many different roles in the software development life cycle, from tester and developer to Scrum Master. Then he decided to go...

Course info

Level	Beginner
Rating	★★★★★ (36)
My rating	★★★★★
Duration	2h 34m
Released	4 Mar 2020

Share course