Build Applications For LLMs in Python

109 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=PTRAKU0ZYEA
https://www.youtube.com/watch?v=ptRaku0zyeA

SUMMARY

The video discusses the development and deployment of a web application using the new FastHTML library, aimed at simplifying the process for Python developers. The session covers building a to-do list application and introduces features like HTMX for dynamic updates, Gradio for easy UI creation, and the use of SQLite for database management, emphasizing the goal of making web app development accessible to non-professional coders.

- *FastHTML is designed to empower Python developers by simplifying web application development.*
- *The session demonstrates creating a to-do list app, showcasing CRUD functionality and dynamic updates using HTMX.*
- *Gradio is highlighted as a tool for creating user interfaces quickly and easily.*
- *The application uses SQLite for database management, allowing for lightweight data storage without a separate server.*
- *HTMX enables asynchronous updates to the UI without full page reloads, enhancing user experience.*
- *The deployment process is discussed, with Railway being recommended as a user-friendly platform for hosting applications.*
- *The FastHTML library aims to provide a seamless transition from simple prototypes to more complex applications without requiring extensive knowledge of JavaScript or other languages.*
- *The session encourages community feedback and contributions to enhance the FastHTML library before its official launch.*

For those of you that have done the fast.ai course before, you'll know that lesson one we teach you how to train a model. And this lesson two, we teach you how to deploy it. And so this is the model that I demonstrate deploying in lesson two. and you can see here, it's a image classifier.

and we do this because you know, what's the point of building a model if nobody can do anything with it. And so a lot of the time the thing you want to do with it is to create an internal or external web application. And so generally speaking, the people doing that course are not full-time professional coders, you know, they they have other jobs and coding is just a thing that they do to get their jobs done.

And most folks in that kind of domain specialist area write in Python. And obviously Python particularly for anybody doing AI is necessary. So we that app is written entirely in Python. and here is that app. And so that app uses a really nifty thing called Gradio. And to create an app in Gradio you basically create an interface.

you say what what basically when you click on the button, what what happens? And the answer is it runs a Python function. what inputs are there? And as you can see, you can say you know, it's an inputs.image. And

what outputs are there? And it's a label. And so then when the function returns, that causes you know, this output to get filled in.

I like Gradio a lot, and it is one of a large number of tools which are generally put in the category of what PyViz calls dashboarding libraries. And you can see here actually Gradio with 7 and 1/2 million downloads a month is certainly the most popular. Bokeh is more specific to visualization, I would have thought. Streamlit though, very very much up there with Gradio. but there's a lot. So there's 37 of them here.

I was like very happy that Gradio, for example, exists. And before Gradio, we showed people how to use Voila. but in some ways it always felt a little bit unsatisfying because you know, if your little prototype web application dashboard thing you create with Gradio is great and popular, what do you do next? You know, how do you go to the next level of expanding it out into a more complete web application?

And basically the answer is like, okay, well now you've got to learn a whole lot of new stuff, you know. You can probably still use Python for the back end. You might use FastAPI, for example, which is fantastic. But then you're going to have to like learn TypeScript or JavaScript and something like React, and you know, hook all these things together, and you're into basically full-time coding job territory at that point. That feels like I'm pretty unsatisfactory. And so what happens in practice is most data scientists, you know, they don't go beyond the kind of Gradio Streamlit level, you know, they create their little prototypes and that's it.

so this is a bit sad for me particularly because my first ever successful YouTube series was called end-to-end web app in under an hour back in 2013. I had never I don't think I'd ever posted a YouTube video in my life. And I was like I wanted to show people that you can create things that are more than just a little demo. And and you know, I've spent months trying to figure out how to like use Angular plus C# to create a web app that a proper real web app you can scale as far as you like in under an hour and I got it working.

And that felt pretty good and much to my surprise it was incredibly popular. It had I don't know, like 300,000 views. Like I basically never done a video before and I thought, "Oh, this is real cool, you know, like a lot of people are interested in this topic of actually building stuff, putting it up on the web that's, you know, flexible and so forth. And the the sad truth is I don't feel like today I could do that anymore. Like I feel like things have got more complicated.

you know, more specialized. and that kind of makes me sad, you know. And I also feel sad that even with that thing I did back in 2013, you know, you had to learn C#, you had to learn open API, you know, you had to learn JavaScript and Angular. Like there's still a lot of things that you had to learn and that felt not great, you know. so Just realized I had the wrong thing recording. Never mind.

Yeah, so we've decided to build something to fix this. and it's called fast HTML. and it's important to mention it's it's it's not released yet. So, quite often in the fast.ai courses I did this, I would I would tell people about a thing that's not released. And then I would ask people, "Please don't share links to this on social media, etc." And people always were good about this. So, I'm assuming that this group of people is as helpful and honest as I've

always dealt with before. the reason for that is that, you know, we want to launch it properly when it's ready. And it's not ready. We're giving you a sneak preview. so, yeah, don't don't share this widely.

we would hate to have to make everything private again if it was if that happened. so, yeah, this is for just for you. And if you want to talk about it, feel free to talk about it here on this Discord or maybe Jono, you could also add a link in the Discord chat to the fast HTML channel. Maybe invite them to the fast HTML channel on the fast.ai server. that's another place that people could talk about this.

so, with that said, what is fast HTML? Basically, it's a library that's designed to bring the the entire like power of the web to a Python coder. and so the thing we're going to build today it's like loaded up.

so in the fast HTML repo, I've got a thing called examples. And so let's run that one. UVicorn DB app colon app oh, move to examples. There we go. And the app is very simple-looking. actually, we're not even going to add auth today, but you'll be able to see in the code how to add auth if you wish.

but basically, it's a very classic like CRUD application. It's a to-do list application. and it's got all about the kind of features that one would expect. Like it's it's it's properly kind of responsive. and like if I add a to-do to it, I can use the keyboard. I can get more details about it to do. I don't have any details to show, but they're there. You know, delete, I can edit. and you can see it like behaves like a modern web application. One with very few features, but like this is all the things you basically kind of need if you want to create you know, Facebook or Instagram. And in fact, it's designed to do everything that Django can do. And Instagram's actually written in Django.

so to give you a sense of like this kind of basic idea of how this is built, you know, is it's it's a totally flexible real web application. but to get started, it's about as easy as Gradio. Okay, so that's the kind of the background.

Jono, did you have anything you wanted to add before I started coding? I think this is good. I think this There's a There's a lot more we could say on the framing of like why this seems like an exciting trade trade-off to us versus some of the other alternatives, but we can do that later or especially once we've seen the functionality. So, yeah, if you want to dive in and show people like the the getting started and then hopefully the the rationale will come out as we talk of like oh this is another neat thing that we can do that might be tricky in a different kind of framework.

Yeah, and because this is built on web fundamentals, you can use any web component library, you can use any CSS library, you know, you can use Tailwind, you can use DaisyUI, you can use anything, you know, so there's a huge world out there of stuff that we can build on. In the past, it's been like JavaScript developers to build on. So, now it's for Python developers to build on. And, you know, you can create really beautiful and and interactive and nice websites with not very much code. And so, the code of what I just showed you, you know, including security, including plenty of blank lines, is 61 lines.

So, that's what we're going to code today. So, there is maybe Jono, you could put in the Discord chat a link to the fastHTML dash shoot repo and in that repo I've created two things, simple and fancy. And they are, you know, they go step by step. You'll see all the steps that we're going to go through today. but today we're going to do it from scratch.

So, oops, I didn't mean to do that. File open recent, reopen closed editor. Can't do that. Oh, thanks very much. Okay. So, I just created an empty folder. So, we're going to start with an empty folder demo. Oh, that doesn't quite work. Hm. Annoying. Hang on a tick, sorry.

It just keeps thinking about how to do this. That's right. I think I can do it from here. Okay, cursor dot I see. Yes, so then I want to go to demo. there we go. Nice. All right. So, we've got a totally empty screen.

So, let's get started. And I guess the best place to get started would be the documentation. And so, pip install python-fasthtml. I've already done that. and that'll install everything that you need, which is not very much, actually. so, let's just go ahead and like copy and paste this minimal app. And so, we'll create a new file demo.py.

There we go. Paste. Okay. And actually, those last two lines we don't really need, so I'm just going to delete them. so, I'll save that. And so, this is this is a FastHTML application. I'll go through each line in a moment. And a FastHTML application is built on top of a number of really fantastic technologies. And the first one we'll look at is Starlette. So, a FastHTML application is a Starlette application.

For those of you who are interested in the details, you don't need to know about any of this. so, it's just FYI. Starlette again, these are details which you don't really need to know about, but it it's what's called an ASGI or Asgi toolkit. It creates Asgi applications. the reason that matters to you is that to use FastHTML, you need to run an Asgi server. And the one you probably want is called Uvicorn. So, FastHTML insta- installs Uvicorn for you. So, that's fine.

and so, to run a a an Asgi web application, you type Uvicorn. You then type the name of the .py file that's that you're going to run. And then you type the name of the object which contains the FastHTML application. So, you see I've got FastHTML app equals FastHTML. So, then you type app. Uvicorn demo colon app. and then something I like to add is minus minus reload, which means if I make changes in the in the code, it'll appear automatically. So, you can see this is now running, Uvicorn running. And if I command click on this there it is.

Okay? So, hello LLM course cool people. So, if I press save, you'll see at the bottom it automatically reloads. I still do have to refresh my browser. That's it. Okay. So, now we've got a nice little loop we can start going through. And so, I will endeavor to keep half an eye on the Discord group.

I I can keep an eye on surface things if you'd like. Cool. Yeah, yeah. Don't That's what I was going to say. John I can also do that. There's a comment that this looks like Fast API, which I think is Yeah. Yeah. Yeah. Yeah, so great. So, Fast API is the second thing that we're building on. We don't actually use it at all. but we've built it

because we really like Fast API. We built it to be as similar as Fast API as possible. And so, what I actually did when I started is I opened up the Fast API tutorial.

And I then went through every single one and tried to make it so that exactly the same thing worked. And except for the bit where you return JSON. So, Fast API is designed to create these like two-tier applications where you return JSON and then you create a JavaScript application that uses the JSON. And that can work pretty well if you've got particularly if you've got like different groups of people in your company who are like JavaScript developers versus Python developers and they can each do their own thing.

Personally, I like to like have everything in one place, in one file, in one language. And so, I'm less fond of that particular way of working, but fast for fast API is amazing for that way of working. And so we thought, well, given how many similarities there are, we should try and make that work. So actually, if I copy and paste this exact syntax, obviously I'm not going to return JavaScript. well, I mean, we support JavaScript as well, so we could look look we could return JavaScript.

What the hell? Let's try copying Okay, I haven't actually done this before for a while, so this might not work, but let's try pasting. There we go. That's We're not using fast API, but because I you know, went through the tutorial and tried to make it work the same way, I think if I press save and go back to my browser and reload. Yep, there you go. Okay, so in for some limited set of things, you can actually use the same syntax.

and if you are interested in creating more of that kind of, you know, multi-language two-tier style application, I strongly recommend fast API. All right. So, what happens next? I think the next thing we should do is like have more than one page. So, one one thing that actually stuff like, you know, the the kind of the dashboard type things and often often not great at is actually having more than one page.

So, you'll see a lot of those dashboard apps tend to be, you know, one massive screen full of things. So, let's go a bit fancy here and create a second page, and then I'll explain what all this different syntax means. so, the second page, so I'm just going to go and copy and paste this. And this needs a new path, which we'll talk about in a moment. and so, this one we'll say return title.

actually, what I'll do is we're going to make this super simple. We're going to call this a page. Because I'm actually going to show you two different ways of writing these applications. The first one I'm going to show you is the simplest way. And then I'm going to show you what I generally recommend people actually learn if you've got a bit more time, which is the way to do things much more richly. so our second page. Okay, so the first thing we passed to a page is the title of it.

Another page. Okay. And this one so then they'll have links to each other. So we'll create an a tag. There we go. And this one will also do the same thing down here. All right. And I'm using this kind of somewhat nice IDE Code Cursor, which is basically VS Code with more AI built in. So when you see those little gray things that I just press tab to fill in, that's Cursor being helpful.

I say kind of useful cuz actually quite often I gets in the way and suggest things I don't want to do and doesn't listen to me, but anyhow. So What's that doing? Right. Okay, so if I reload now, there we go. We've got an internal server error, so good to learn about debugging. yes, page is not installed. So for this kind of s- extra simple version of the world, actually we use slightly different. We've got a thing called FastAP.

And in that case we actually go fast app here. We'll talk about that shortly. Save. Okay, let's try that again. Okay. Now, a page automatically uses this as the H1, so we don't need the H1s anymore. So, the page thing tries to make like as little code as possible.

Okay, refresh. There we go. So, click to another page. Go back. Okay, so you got a beautiful fancy multi-page web application. I hope you are all suitably impressed by that. Okay. So, let me explain how this fits together. So, there's you might have noticed that this is a kind of quite a different style of application to the one I told you we're going to build, which is to say like there's totally different pages, you know, rather than things all happening in the same page. so, the first thing I showed you was an SPA, a single page application, and this is a more of a traditional application. These more traditional applications, I think, are a bit easier to write. So, that's why we're going to start with them. And to create these kind of like super simple, separate page per thing applications, fast app is the easiest way to do it.

So, you just say this. so, as I said, you create fast app rather than fast HTML. I will say though that if you like I think I should be able to command click on this. I can't. That's fine. Let me just jump into Vim then. like fast app is literally like 20 lines of code. That's the entire thing. So, it doesn't do much, but I really wanted to make it so like absolutely minimal boilerplate, minimal anything.

so, you can always look at that to see what it's doing as to what page actually is. They create your app. Now, for those of you who haven't done any web programming, you need to know a little bit about how the web works. So, what happens when I go to this link? So, this link here is linking to /page. So, what happens when I click on something that is an A tag that links to /page?

Well, what actually happens is it goes to this server, right, which is my own computer. and it asks that server to get that path. And so, what web application frameworks do is they let you say, "Okay, what happens when a browser asks for that path?" And those are called routes, right? So, one of the things in a route is what is the path that it's for.

So, this function is the function that is called when you get this path. So, those are the two pieces of information you need to provide. One is what is the path and the other is what do they want to do? And as these are called HTTP verbs, and the main and the one that basically happens when you're just like looking at a page is called get. So, you can see I've got two functions called get. One is attached to the root path.

One is attached to the page path. So, if I go in manually and type /page. Right, you can see all it's doing behind the scenes is it's going to and it runs this line of code. and then what this line of code does is it returns a page, which is a special kind of object called an XT object, which we'll be talking about shortly.

and a page takes two pieces of information the title of the page and well, actually, and then as many more things as you like, which is basically all of the things that you then want to appear on the page. So, normally if you've used things like Flask or stuff like that before you've probably created templates, which is where you create a separate file and in that separate file you you write HTML code.

for for this, you don't have to do that. for this, you can use HTML elements as Python functions. and every HTML element works as a Python function. the nice thing is that because they're both just Python functions you can create Python functions that return other HTML elements.

So, container is a Python function. So, this is the source code of fast app which returns a main. That's a that's a type of HTML tag. and it and the main contains an H1. And that's again a tag. so this means that you can have everything in one file and like I find my I'm too much of a simpleton to be able to handle lots of files, so I'd like to have everything in one file and I'm too much of a simpleton to have multiple languages, so I just want to have everything in Python and that means that if you want to you know, create components and stuff like that, they're all just Python functions. And if you want to create a library of components for somebody else to use, you can check that on PyPI and they can now pip install that.

And so actually, fast HTML comes with, in fact, a library of components and it uses a really nice little CSS library called Pico. And it does that because Pico's like super super simple. And so you can see it's got something called components. For example, there's things like a card and an accordion and a drop-down and a group and we'll see all of these things today. We're going to be building them.

and the way Pico does it is that like a card is literally just oh, just use the article tag in HTML and it will render things in a particular way. And if you have a article with a header and a footer, it'll render it in a particular way. So, we have created as you can see, is a card, right? So, here's here's the card function in fast HTML and it just returns an article optionally with a header and a footer.

So, to create new components, you just write Python and that Python just returns HTML tags or it could return other other functions which return HTML tags. So, like another example is like here's a group, right? So, a group here's an example of a group of a button and an input. And the way they implement that is they create a field set.

And so, I mean, it's it's hardly worth doing, but to make things as easy as possible, I was just like, "Okay, we'll create, you know, a group." And a group just returns a field set with a role of group. And you can see like you can even do it like you can prototype everything if you want to. I quite like doing this. In Jupiter, everything appears correctly, as you see. You know, CSS Pico has a grid a grid object thing, and a grid is actually just a bunch of divs with class grid on them. So, you can see here's a grid with some colors. Okay. So, it's like super super basic.

Okay. So, Jono, just dive in if there's any questions or comments from either everybody in the Discord, or if you have anything you think's worth adding at any point. Otherwise, I'm just going to keep driving on. Then go for

it. All right. So, to create a to-do list, we need data. And this is often one of the challenges with kind of the dashboarding type things is like, you know, you actually want a database.

and actually Python comes with a really great database called SQLite. SQLite is a database which does not require any separate server, but despite what a lot of people think, you can actually create pretty significant applications using nothing but SQLite. the trick is behind the scenes you have to use something called WAL mode, w a l mode. and it actually becomes a pretty capable database. But yeah, you don't have to run a server, which is really cool.

And so, you can use whatever database you like or none at all, and Jono will show examples with no database. but, a lot of the time you'll want to have like tables and rows and things like that. So, we try to make that as simple as possible to do. and you know, I quite like playing around with this stuff a little bit. you know, in in notebooks to fiddle around, so we can kind of do the same thing here from fastai.html.all import star. So, fastai.html comes with SQLite support built in.

but there's no particular, you know, you not you don't have to be particularly attached to that. Okay. So, let's change this. See what's everything here. And Jono, do the gang from Railway know or how many of the gang from Railway know how to come here, by the way? Cuz I wanted to say hello.

Yeah, they have an invite as well. I've been watching my DMs. I haven't heard anything. Cool. All right, no worries. They might have forgotten. So, one of the things to mention is like what kind of like database you use depends a bit on where you want to deploy it. And so, we'll be showing today how to deploy to a really nice thing called Railway, railway.app. but, I spoke to the CEO of Vercel yesterday, so they're building a deployment for Vercel.

I spoke to the product manager for PythonAnywhere, they're going to build one. Hugging Face are also going to build one. So, like there'll be a lot of places you can deploy this. and most of them are going to support this SQLite approach although Vercel have their own database and they're going to actually create their own Vercel binding. but they're all going to look basically the same. So, what you learn here today will be will be fine.

So, I think where do we start? Yeah, let's just keep kind of building out. We can you go back to the Jupyter notebook as well, but maybe we'll just keep working through cursor a little bit. So, with fast app you can see you can pass it a DB path. so, I'm going to go ahead and do that. So, DB path. All right, so let's you know, create a directory for that data {slash} we'll call this I don't know, demos.db.

Okay. and then you can also just pass it a bunch of fields that you want to appear in your table. So, they're like this super simple version is if you just want one table. I'll show you how to have more tables shortly. So, we're going to have an ID field. we're going to have a title field for you out to do. so, for each one you just say what's the name of the field and then you say what's the data type of the field. So, is it done or not?

and then also it's good practice to say which one's the primary key. And obviously in this case we want to use ID. Okay. So, that's enough to create a database. and so, now we can start. So, we could save that and Oh, what happened to our terminal? Do I have to rerun it? It's changed. Did I restart this at some point? Oh, I'm in a Oh, I'm in a different place.

Silly me. I'm going to copy this. We are meant to be in demo.py. All right. So, paste. Right. Okay. So, save that. and Let's see we're reloading. Why isn't it reloading? Okay, then let's go back to our application.

that anymore. Okay. All right. So, back in our application, nothing's happened yet. We've created the database, but it's not actually doing anything. That's what you'd expect. so, that's all fine. So, let's take a look at our database over here as well in our notebook. so, I'm just going to go ahead and copy that path. And so, you can manually create a database or link to a database just by saying database.

And put in the path. Okay. Okay, so there's our DB. you can list the tables using.t t for tables. Okay, and the right place. Notebooks demo. I was expecting there to be a table in there. fast app data slash demos. Might have to restart the Uvicorn thing. Oh, wait. You did.

What's so I was expecting save. Oh, I see. So, now that we've got a database, we actually get returned two things. We get returned the table and we get returned the class that represents that items of that table. So, I need three things.

Okay. There we go. That was silly of me. Okay, so now if we look at this. Okay, so you can see and by default it just create creates a table called items. And so, let's just say to-do's equals that. And then we can have a look at that to-do's. And so, to get the columns, it's just C. what is it C? Is it not C?

Oh, sorry. To-do's equals items. Right, let's try that again. Balance. Okay. So, a So, a database has tables and then you can get any table just like so and you've got even like tab completion, right? So, if you hit tab you see it just fills in items automatically. It's all dynamic. And then I can do the same thing here if I tab completion for columns, right?

and if you want to list out the rows, you just call it. Okay, and there's currently no rows. So, that's not going to make very interesting to-do application. So, why don't we just insert a few rows? and so, behind the scenes, Do remember it created a to-do class for us? We can manually ask for that class just by saying DB Sorry, to do's .data class and that creates a class for it. And so the nice thing about that is now we can insert a to do and we get, as you can see it automatically dynamically lists out the things that we can call. We're just going to use like title here.

show how to use SQLite. so that's going to be the title. Check it and I get tab completion for. All right, and I guess we should say done = false. There we go. And you can see when you insert something, it returns the inserted object. so it automatically has set an ID for it.

And make a cool demo app. Okay, so we've now got a to do table with two things in it. So let's show them. So let's make our get now show those to do's. so we probably want to just create a list, right? So to create a list in HTML, you just use an unordered list, right? And a list contains list items.

a list Oops, let's give that a list. and another list item. Another So I kind of like to do things in really small little steps, as you see. Let's just make sure at each stage it works. You're not actually putting that in the page. That's fair. All right, so let's add going to call that something. Oops-y-daisy.

To-do's. To-do list. Equals. Okay, and then you put it in there. Thank you, cursor. And thank you, Jonno. Okay, so there we've got a list, right? So instead of having that text, we would instead like to have the titles of the to-do's. So we can do like a list comprehension.

And it's actually made a reasonable guess. but remember that to actually list the contents of the table, you have to put brackets, you have to call it. So an unordered list containing Yep, the Single star, right? Cuz we just want to put the Single star, thank you. I always do that, Jonno, Jonno. Okay, an unordered list containing a list item with a title for each to-do. Okay, great. And we don't really need this a tag anymore.

Okay, and now this is a to-do list. And by the way, moderators, how are we scheduled to stop in 10 minutes? I'm guessing we can go over, but is there any sort of time Oh my goodness, I probably started this Keep an eye on. There's no time pressure. Okay. I might have to leave, but please keep going. I'm sorry. That That will That will be here. I I didn't do a trial run, obviously. We're loving it. Keep going.

Okay, all right, so like let's instead make this like the to-do's detail page. So, if you use Flask, this will look pretty familiar. You know, to-do/ ID. Putting things in parentheses in in braces means that in fact you can see Curses guessing at all cuz it's so standard that it's going to be passed as a parameter. it's a good idea to type that parameter cuz it'll automatically cast it to the right type if you can.

and you can see, you know, we can now grab a to-do and actually to grab it cuz it does not actually know how my system works. To grab a single item by primary key, use square brackets. Okay. So, the one in round brackets is where you pass a where clause and or a limit and or an order by. The square brackets just looks up something by primary key. Okay. So, now we actually need to create an A tag which will link to this.

All right? So, and okay, Curses guessed correctly, an A tag. It's maybe worth pointing Yeah. out why Curses guessed correctly. Like I find this all mind-blowing because this is a brand new library, right? This has been in existence what, since you wrote it 2 weeks ago or a week ago. So, this is not in the training data, but these language models know how to write HTML. And so, when you're using things that are like these existing HTML primitives, it sees like, "Oh, I guess if I want to pass in an href into a link, that's how I do it." Therefore, like if I want to do, you know, a title or a a name into a form field or a a source into an image, it's going to be the same for. So, it's it's actually very close to something that the language models know extremely well. and so, this is very different to like I've used some very custom libraries where it's all their components, all very like custom, then it's not documented before the language model training, and then the language models know help.

Whereas here, the suggestions are often pretty good.

Yeah, yeah, yeah, for sure. Like Google just released this thing called Me Zap or something which is like in kind of an Angular wrapper which has got some similarities to fast HTML. And you kind of can't do it with that because it's all like a new custom different thing. Whereas this yeah, this is just HTML. so yeah, let's check this out. hopefully Okay, so we get the to-do and then we return a new page. It's going to have the title as the heading.

And it also say whether it's done or not. Okay. So Okay, looking hopeful. So I click on one of these and so you can find inspect this, you'll see that it's linking to {slash} to-do {slash} one. So I click on that. There we go. That's the title and it's not done. All right. And so at this point we probably want our back button. Again. so I have an A There we go. Thank you, cursor.

go back. There we go. Okay, and you can see it's keeping track of the URL correctly. here. Go back. There we go. Make it a cool demo app. Cool. Go back. Cool. Okay. the SQLite thing is a library I wrote called fast light. and you can this one is released. So feel free to talk about it publicly. It's actually very heavily based on SQLite utils written by Simon Willison.

Which is really great. so a lot of the best ideas, if not most of them, come from that. but I added quite a few things to try to make life a bit easier. so yeah, so check out fast light. Maybe somebody can put a link to that in the Discord. And that's what is actually being used here. And so fast HTML.all automatically imports you know, the stuff that you'll need from fast light so forth. that's why we didn't have to import everything manually. You can put stuff manually if you prefer.

okay. So, next step let's make this look a little bit better. so, one thing that I think I quite like to do is for making things look a bit better is to kind of test things out in Jupiter. so, to make stuff work correctly in Jupiter, there's basically two things you have to know to do.

let me just try to remind myself of what those are. Oh, yes, that one. Okay. So, the two things you need to run is show Pico con link and Sorry, what did I say it was? See, I just looked at it 5 seconds ago and I've already forgotten. It's called set Pico class. Okay.

And this is a notebook specific quirk, right? This is if you want to be able to prototype things So, now that I can try that I mentioned earlier about cards, right? So, a card you can say there's the here's a card. And so, by default it shows you the HTML that it would create. And so, the card they said you can have a header and you can have a footer. And that's the HTML that it would spit out. And to see how that would render you just type show.

And okay, that's cool, right? So, I'm going to going to use this. Yeah, so basically without these lines Jupiter's not going to know how to render Pico CSS. So, with these lines, it automatically adds It's got a little bit of JavaScript, basically, that automatically adds the necessary thing to render it correctly. And if you can also if you

put this, and you use Quarto documentation, then Quarto will also correctly render all of your, Pico stuff. And you can create similar things for other CSS libraries, CSS libraries. So, yeah, give it that looks pretty nice. I might just go ahead and, change this now as well.

So, I think what we'll do is we're going to need a way Well, let's just put it in a card first. All right, so, card equals card. and basically it's going to say now to do list. And then we could have like a header. This needs to be H1. This is where it doesn't quite know. It's just guessing.

Not terrible guesses, but header is a header. Wait, is that an argument? Header equals rather than Header equals, yeah. Thank you. This is where Sometimes I find cursor confuses me cuz it's so keen to, you know, this time it's guessed correctly. Okay. And then this is now So, we save that. There we go. So, you can see it's kind of coming together there.

great. So, the next thing we'll do is we need to be able to add a a new to-do. So, what should we do there? So, the first thing actually I might do is I'm going to refactor things in a way that I find helps a lot, which is like the idea of like rendering our to-do as a list item is like something that does that as its own function. It's going to become more and more useful.

So, rendering a to-do is just means turning a to-do into a list item. Okay, and it's actually guessed. So, I got rid of the UL. This is just returning a list item, right? And so, now actually I don't even need a list comprehension. It can just call map. render over to-dos. All right? So, that's going to call this function on every to-do. And here's the function that's making it a list item.

So, I think that looks a little clearer. It still works fine. Okay, good. So, now we're going to add the ability to create a to-do. So, in HTML, the way that you have you know, the ability to get inputs from people is by creating a form. So, let's create a form. Okay. And we're going to use that group thing we learned about earlier. And in the group, we are going to put a an input that's a text box in HTML.

This is all just standard HTML. okay, the name they've guessed correctly, but I want to actually add to it a placeholder. New to-do. I I it's quite a nice way to do things. Okay. So, our form contains a group. Our group contains an input. And it's kind of nice to format things, I think, in a way that's a bit easier to see what's going on. And it's also going to contain a button.

And we'll just call this one add. Okay. and I think that's my form. All right, so then the other thing a form needs is like what happens when you click on the button. So, when you click on the button, it's going to post it somewhere. So, to post things, we're going to talk a lot about this HX thing in a moment. you write HX post. So, basically, whatever you are whatever one of the HTTP verbs you're using, you just write that preceded by HX underscore. So, now we can go ahead and write our route.

So, now it's not get, it's post. All right? And what is the post going to receive? Well, it's actually going to receive,

believe it or not, a to-do object. oh, look, and it's guessed correctly. Thank you. That's very impressive. the reason it's going to receive a to-do object is because it receives the contents of the form. And whatever the name of the inputs are is going to become the attributes of the object. And in this case, we're not even going to be, you know, everything's not going to be done to start with, and done is set to false by default. So, we're going to get a to-do. And so, to add a to-do, render guessed wrong here. It's actually insert.

Okay. And you do this all in one line. Let's just return. And so, because to do start insert returns the to do that's inserted, I can just say return that. Now, that's actually not quite true because this is going to be a separate page. We're going to use an approach like that later. Actually, what we're going to do is we're going to actually just say return. And we're going to take all this out. Move it into a separate function called home.

All right. And then this is just going to return home. And this is also going to return home. Okay. So, that's the kind of what makes this page-based approach pretty convenient is like after you do something, you just return the whole page again. the form's not actually returning the page.

We're going to put it up here in the header. Okay. There we go. So, if I now say create, delete, and edit. Thing is, Hey, okay, it's working. So, we have a to do list as long as you never to be able to set anything to done, and as long as you never delete anything, it's This is my kind of to do list. Something you only add things to.

Okay. so, this HX thing is using something called which is really kind of the secret behind a lot of fast HTML. HTMX is basically something that makes browsers work the way they always ought to have worked. So, normally with browsers, they can only do two things without JavaScript. They can provide hyperlinks that when you click on them, it sends a get request to that place, or you can create forms where if you click on the button, it sends a post request to that place.

And then it can only do one thing with the result returned by the server, which is replace the entire page with the result. So, HTMX changes it so that you can send so that anything you click on, move, touch, wave over, anything at all you do on the page can send any method to any endpoint, and it can do anything you like with the thing that comes back from the server. And so, you don't have to write any JavaScript to do those things. It just makes it so that's how the browser works now. So, we can just pretend that that's how the browser always worked cuz fast HTML always includes HTMX.

So, all the HTMX things start with HX underscore. So, this just says, "Okay, when you when you submit this form by clicking on the button, it should post to this route, and so this route has a post function. Okay? And that's how HTMX works. And because we're in this like simplified full page version, then the the thing that comes back replaces the whole page. Okay.

So, we've got that working. great. we should probably add a delete button, I guess. So, let's put that in the details screen. So, contents equals Grab that.

Okay, so we're going to have a back link. And let's also have a a delete. Oh, cool. And you can see it's filtered all in automatically. That was nice of it. And so now start contents, good.

Jamie, HX delete I don't think says trigger a HTTP delete. I think that's delete whatever That's actually Yeah, you're you're right. It's done it wrong. Yeah. That way around. Yes. Thank you. Great. so, yeah, all of the HT All the HTMX things you say what method to call and what endpoint to call it on. So, they're the two things that you provide. so, this is going to call the delete method there. And you know, maybe these should be buttons.

Things that do things feel like they should be buttons. okay. So, now we obviously need a delete. So, I can copy that. Delete. And so, this time because we're calling {slash} ID, it has correctly guessed all of Okay, that all looks correct. So, I think Kester guessed correctly there. All right. Let us Okay, I think we've done the first one.

So, let's delete it. There we go. Okay, that works. Okay. So, we're probably ready to get rid of these annoying things. I don't want those there.

right. So, what else have we got now? It's looking pretty good. All right, so I guess we should Oh, we need to make edit work. Okay. So, to edit something, we're going to need to go to a new page.

It's going to need to have a form on it for editing our to-do. So, let's create that page. And so, we'll be editing some particular to-do. So, we'll need that. And it'll be edit. And this one, we're going to return a form. Okay. And the thing we want to edit, okay, it's going to come in as an ID, so we're going to need to grab that to-do.

Just like we before. and then we'll create the form. Let's not return it just yet. Let's create it first. Form equals Okay, the form is going to need let's create a group just like we did before. Oh, look at that. Name, title, button, save. Yeah, that's pretty good, actually. Okay. Yeah, look at this. It's even guessed. So, put is what you normally use as an HTTP verb to edit things. So, he just correctly guessed that I want to put.

So, that's good. Although let me think. Yes, that's right. but we don't need Actually, we can just put directly because it's a form. All of the information is going to be there already. so, this time we do want the ability to be able to change done. So, we'll have a checkbox.

Okay. and ID is done. So, it sets the Actually, we could we could I it's not a bad idea to always use ID. Fast HTML always sets the name to the same as the ID if you don't specify both. So, that's not a bad way to do things. so, we should give it a label. Okay. Do you want that in the form? Sorry?

Would you like that in the form? Oh, it is in the form. My my bad. it's in the form. Yeah, I I've actually made this overly confusing. So, I should do, yeah, that. That's the group. And there's also a checkbox. And then that's the form. I think that's more correct. Now, then, when you submit the form, it also needs to know which which to-do ID this is, cuz we're editing a to-do rather than adding one.

Oh, dear. It's This is what I don't like about Cursor. It adds stuff I don't want it to add. we also need to tell it So, we add a hidden And so, that's just going to pass back the ID. Now, the thing that you might be surprised about is that we've created this form, but it's not in any way It hasn't got any of the information from the to-do in it.

And that's because we have a very simple, but rather nifty little thing where you can simply say fill form. Don't know why we've lost all of our fill form fill this form with this to-do. And that just goes ahead and puts the correct information. So, now we don't really even need this on a separate line. And there's no magic here. It's just simply going through and See, it keeps adding stuff I don't want.

it just goes through and adds each attribute that it can find in the to-do to each named field in the form. And that actually returns the form. So, we can keep simplifying. Okay. that's probably about as simple as I think we can make it. So, let's try that. Okay, not surprised that's broken.

Okay. So, Oh, did I Okay, now what I've done wrong. post And yeah, that should be Oh, That's port. So, there's a question Oh, I've got to do /1, that's why I need to go back to.

Yeah. Yeah, one doesn't exist anymore. you've got, you know, two different things that have the root for slash there or maybe even Yes. and so someone is saying, "Oh, like how does how do we define like which method is used? And is it is it related to the HX get HX delete part?" so maybe you can clarify the roots bit there. Yes. Fair enough. yeah, so basically whatever appears after HX is the name of the HTTP verb that's called. And whatever HTTP verb is called is whatever function is called. So, HX delete calls the delete function.

And if we're adding one, HX-post calls the post function. and the kind of default in this special simplified version in FastAPI is to call get if you just have a normal HF. But we in the future we'll be using Ajax get. Yeah. Okay. So all right, so we actually need a way to get to this edit page.

So we can actually quite simply do that by just adding another link here. And oh, it's guessed correctly. So edit. And then edit link is going to go to {slash} edit. So we're return. Okay. we probably want a little bit of something between those two. So let me just go bump bump bump.

There we go. Okay, so I can still go to places to see it, go back, edit. There's my form. Okay. Make it called demo, not a demo app. Save. Oh, yes. And now the problem is that after I'm done with that is I don't want to return actually that, do I? I want to fill the form. And then when I'm done, actually I want to return home.

Cuz this is the multi-page version. Okay. Hell no, no, no. That's wrong. what am I trying to do here? We've got a fill the So we've got this is so this is this is showing the form. Yeah, we need a put version as well. Yes, yes, yes, yes. This is showing the form. Sorry, my bad. Right. It was fine. So we need a put version, yes, which we don't have. Thank you, Jono.

Okay. Put. And at the end of that it's going to return home. So, this is going to receive a to-do. And then, yes, we

just go to to-dos.update with the to-do and return home. That's exactly right. So, Curser guessed that one correctly. All right. So, make a call. Demo. Save.

There we go. That works. and it's not actually done, but just to test. And actually, we should have something here. but at the start has a checkbox.

I don't think we need an X if it's not done. That's a bit over enthusiastic. Okay. See how that looks. There we go. Okay. So, we're not really done with that. So, not done. Great. and then the only other thing Well, I think Yeah. I guess that's it, actually. A few questions from the audience on this one. One is like how does it know which to do to update?

So, in our edit form, we had a hidden. And remember I said that all of the values in the form just get used to fill in automatically the attributes of the thing that's passed to the put method. So, it's going to get the title. It's going to get the done. And it's going to get the ID. So, it's hidden. So, in fact, let's look at it.

This is a really useful trick, right? Is Is for these things is just inspect cuz it's all just HTML. So, here you go. Hidden create an input type equals hidden value equals two. ID equals ID name equals ID. And There's a a slightly less smart way that you could do this as well, right? Which is that you could take the take the individual things as arguments to your function. So, you could instead of taking a to-do, you could take an ID, a name, and a done as arguments. And then you could Okay, I'll go look up the to-do with that ID and then I'll change the other things to match.

it's just more work. And so, if you you have this data class ready of like this is what a to-do can have, an ID, a name, and a done, put that as the argument and then it'll go and find those things in the form. Yeah. Yeah, you could do that. Mhm. Hopefully you wouldn't. then there's another question that maybe is a good lead on to other things. Someone's saying, "Oh, so HX_ can only be used by buttons? Or like what events and browser elements can this apply to?"

Yeah. Every single JavaScript event, every single element can have an HX_ thing on it. So, for example, if you wanted to add validation as you like say, "Oh, enter your, you know, credit user. Enter your username." Every time you type, it can call an endpoint, check whether that user exists, and fill in a little thing underneath saying like user does, you know, user already exists, whatever. You can have something that when you wave your mouse over it, you know, something happens. So, yeah. Full interactivity of everything that JavaScript can do.

Okay. So, then I guess we should deploy this. and actually I'll use a slight shortcut here, which is that Although I haven't actually deployed this before, I don't think. So it'll be an interesting test. I'm going to go to the FastHTML just tutorial where we already have main.py in simple.

and it's should be roughly the same code as I just wrote. But it's got two extra lines at the bottom, which if you remember, were actually the two lines I deleted from the documented suggestion. And these are just the ones

that allow me to run it without having to write Uvicorn. So let's just cancel this. So if I when you've got those two lines at the at the bottom, it it means you can just say Python main.py.

And it runs. so the reason we have that is because we now want to deploy this to Railway. and by default and I love doing things that work by default, it'll run something called main.py. So I just dump the standard file into my directory in the same place as main.py. and I've never had to change this yet. I'm not even sure if you need it at all, but that's just what I put there.

And then I have a requirements.txt listing the things that I need. I don't think I need anything except the first line, actually. But anyway, I know this works. And then there's a tiny little Do I have it here? Yes. There's a tiny little script deploy which just runs these steps. Railway init, so that will create the project. Railway up, that will upload your project. Railway domain, that will add a domain to your project. FH railway link is will link the current directory to your project.

And this is really cool. Railway volume add creates a persistent volume. And the the app always will run inside {slash} app on railway. And we created our data in data {slash}. So this is means that we're going to It's going to work exactly the same way, but it'll be persistent. So let me tell you about railway and why we think this is is amazing. Is And by the way, Jeremy, I think Jake from railway is here.

Oh, great. Okay. So hello Jake from railway. I'm glad I was about to say nice things about your product. So Jake is the is the CEO of railway. And I I'm not suggesting railway because I know Jake and therefore wanted to suggest it. instead, I tried lots and lots of different options for deployment. Found that railway was the best of all the ones I could find and then reached out to Jake and say like, "Hey, you guys are cool. What do you think about, you know, us deploying stuff onto your platform?"

And I had a couple of minor suggestions for things that could be improved, I thought. He had them fixed within a day. and so let me tell you what I was looking for. I wanted this to be really hobbyist friendly. so it's like the only one I found that's that you pay five bucks a month and then you can create as many deployments as you want, as many volumes as you want, as many domains as you want. And so I've been using it a lot over the past few days, you know, a week or two, as you can imagine, to like try everything out cuz I'd never used it before. And my total cost so far for all of my projects is 3 cents.

and of course the five bucks a month, but the five bucks a month gives you five bucks of credits. So I've still got $3 sorry, $4.97 worth left. and so this is like totally amazing to me is you can and these things like they they spin down quietly in the background. Is Is Jake just in our text chat or is he able to join the Zoom so he can say hello? he should be able He's muted right now, but Okay, great. So what I'm going to do is I'm going to run this deploy.

and then while it's running, Jake can tell us more about what's happening here. So I'm going to run deploy. I'm going to give it a name. Let's call it simple demo. And that'll take two or three minutes. So if Jake wants to say hello. Hey, how's it going? I'm looking for the video thing, but I guess I'll just be a a voice in the crowd. yeah,

Jeremy reached out to us a little while ago. I guess a couple of days ago.

and yeah, didn't didn't really know him but I had a I had a time I like I asked a couple of friends. I was like, "Man, this guy seems like, you know, quite large in the ML community." and so I'm mostly in the distributed community and surprised there wasn't a ton of overlap there generally. But yeah, we're we're really really stoked and we've just tried to build a really really awesome quick and super easy to use deployment platform. and and like Jeremy mentioned, really catering towards students, hobbies in general.

we can join as a panelist. Hey, there we are. Okay, beautiful. Awesome. Yeah, so so probably disconnected there, but yeah, Jeremy reached out and then yeah, I wanted to make sure that you know, we We to make sure that platform was really really student-friendly and also could scale you know, to infinity. So, we have the $5 plan. So, as mentioned, you can you can host things on there. If you turn on we call like app sleeping, it'll basically automatically scale down your workloads for you.

So, you can host a lot of stuff on this platform for for that kind of initial $5 a month. And Jake, just before I'm just going to interrupt just so it can run well in the background. I just tried to deploy it and if I look at my screen, this has never happened before. did I do something? no, that's if you're if you're able to open that link, it should also just tell you what the build logs Oh, yeah, this is quite nice actually.

So, you can click on here. And it doesn't look like it actually So, if you click the build logs, it should give you all the builds and then there's the deploy logs as well. So, it looks like the command line for some reason just failed to fetch the logs. I can definitely have a look at that in general. Okay, no worries. So, I should just be able to go ahead and click on this. And there you go.

okay, nothing here yet. Something Sometimes that happens for a couple of minutes. sometimes it takes a couple of minutes. domain propagation we're we're now really pushing the the edges of Envoy's state updating. So, we've had to build our own kind of distributed state updating system. So, we're rolling out a new proxy on on Friday that'll make this quicker to to update the initial kind of edge. but as of right now, it should take you know, about 60 seconds, sometimes slightly longer for for it to kind of like roll out there.

Okay. It was like so fast this time. Like normally it takes two or three minutes, but this time it took seconds. So, that was why I was a bit surprised. there's automatic caching as well with this. So, if you've built something before and you've gone and and pushed it in general, it will just reuse any of that old caching without you having to configure anything. So, I think your The not It's taking quite a while. So, I will I will not have people have to look at that while it's happening, but yeah. So, okay, so the fact that it takes a couple of minutes for the domain to get linked up That's definitely not not intentional.

We're building a whole system to to fix that, which we should be doing. And yeah, normally the screen fills up with the actual logs of the deployment in the CLI, and this is the first time that hasn't happened, but that's the nature of demos. It's the classic classic demo experience, you know, right? Like you just you you you plan and then it works for everything, and then the moment you demo it, it's it's everything kind of explodes for you, so.

Okay. You know, I'm inclined to move on to Jonno's demo rather than do the fancy version. I'll just mention that you know, cuz this took about three or four times longer than I hoped. so, instead what I might just do is like show you all like the fancy version. I was going to give you a quick run through, right? So, the fancy version, we don't use fast app. Right? We just use fast HTML. Fast app really is just for people who want to copy and paste the page boilerplate and do something as as simple as Gradio, you know.

And so then if we're not using fast app, you do the steps we did in the notebook. You create the database. You go your database.that's tables.todo's. If the to-do's is not in the tables, then create it. Right? And then create the class for it. So, those four steps are the things that fast app does. then create the fast HTML app. this is the same as before. Where it gets interesting though is post. So, just like before we've got our group with a new to-do and a button.

And it doesn't HTMX post. But we're not using pages anymore. It doesn't return a page. It just returns a to-do title. And maybe I just run this so that you can see what that looks like, right? So, unicorn fancy 03 colon app. So, what's happening here is that the button I click on is going to post in just this before, but this time I got to think about a target ID.

The target ID is details, and that is the name of my footer. And what that does it means is that the result that's returned by this post is going to be placed into this element. Which is not at all what we want. It's just to show you what that looks like. You're live now, by the way, on your deploy. Okay, great. so, if I add a to-do here, ahoy, add.

See, it's popped down here. Right? So, I didn't This is not This is like the stage where, okay, we're now doing JavaScript stuff, right? It's not a full page refresh or not a full screen replacement. It's just replacing something in the DOM. And so, we got to say what verb do you call and what happens with the result. So, it goes into here, right? So, then in fancy 04, we do something even more fancy, where the the form Okay, the target is now the to-do list itself.

And so, then this is kind of the last new piece of thing that you have to learn, which is that there's also a thing called swap. Normally, if you say there's a target, it replaces the contents of that. But we don't want to replace the whole contents of the to-do list with one to-do. So swap says what do you want to replace? This says, "Oh, just stick it in before the end." Right? So with this version, go to the end.

There you go. See, it's put it down at the end. Right? So that's this key idea like the one extra thing I wanted to explain is that yeah, you can have a target and a swap value. And this now means you can, you know, basically do whatever you like. So actually, it might be an interesting to look at if I click one, it pops down at the bottom and the delete appears down there. And that's because clicking one has now does an HX get, right? So it's causing HTMX's get.

and it's sticking the result in the details, right? And so HX get is returning that path. It's got a button in it. That button calls delete. And delete does something interesting. Delete returns nothing at all. So that is going to

replace that target ID, which is the list item itself, with nothing at all. So if I press delete on another one, which is here, it disappears, right? It didn't refresh the page at all.

And so you end up with a really nice thing with this approach. So if we have a look at the full version in main.py, our full version actually has if anything less code than the simple version. Like it's not harder. It's just like a few more things to learn about. and you can see like yeah, so there's a port, and so the port now simply returns to do to do to do's.update. the post simply returns to do's.insert.

yeah, so it's it's actually, I think, in many ways a lot simpler and and clearer when you do it this way. And there's also, one extra piece of functionality that if you take your class to do and patch into it something with a special name, this is always called whenever you render a to do. XT stands for XML tag, that's what these data structures are. So, that's why I can just return my inserted to do. I don't even have to render it, you know.

and I can just return my updated to do. And over here, I can create just an unordered list of to do's. They're all automatically rendered with this special function. Okay. So, that's my fancy version, and then, Jake said this now works as well. So, we just go if we go railway open, that will pop open the browser.

Just have to go over here. There we go. Oh, what just happened? Browser. So, we should be able to click here. Click here. There it is, okay? And so, this is persistent. And you'll also see in the, fast HTML repo, there's also a version of this with authentication and per user to do lists. so, like, it's something you can actually deploy, and it's got user creation and login and all that kind of thing, and I think that's an extra two lines of code.

So, it doesn't get very complicated. Anyway, sorry, Jono, that went a while, so I will hand it over to you. All right, Thank you, Jeremy. oops. I think I can share my screen now. All right. So, what I'm going to do is talk through a couple of new examples. and before I do, I'm going to link to this in the Discord. I'm trying to write up some like illustrations of ways to do things. And so, Jeremy's to-do example will eventually become a section here. The image example that I'll show will be a section here with explanations that you can like come back and revisit.

but yeah, so the goal of this section is to just demo, okay, we've seen the mechanisms in Jeremy's built like from scratch the to-do app. I just wanted to show a couple of other things. And these are all bits that I built this week. Didn't take much time putting together these same pieces. And so, let's start with this image generation UI here, right? This is again, if we're talking Gradio demos or whatever kind of classic UI.

If I type in a prompt and hit enter or click generate, that's going to get added and we have this nice little user interface. This is responsive and pretty. So, I kind of thought we'd get through more of the more of the demo before we got to this point. And so then, I was going to talk about how we do the the visuals. And I will still do a little bit on that, but I don't want to get too caught up in like, "John, how did you do that grid?"

More of the general picture like, how did we build this thing? And so, if we look at this, oh, and I can refresh my

browser. I've gone like don't don't don't keep adding at this point. Whoever here is you know, a hardcore nerd. all right. so, let's look at this app. Yeah, people agreeing with you in the Discord there. We we can refresh the page and I still get my images that I've generated. and by the way, I think I shared earlier a link where you can go and generate and there's some credits.

so, let's think what it would take to build this before we look at the code. so, we know how to construct some HTML, right? So, this is a form just like Jeremy's to-do form. There's an input and a button. And a title. there's some way of rendering our generations, right? And this looks a little fancier than just the raw HTML image. So, there's going to be some kind of wrapper around this. we're not refreshing the page. So, if I generate something else, it's not refreshing the whole page. and so, we're going to be using HTMX to selectively like update with our generation.

and then there's one extra trick, which is that actually generating these images takes time. And so, you can see here we've got this kind of like preview that says, "Oh, a generation is coming." And then without me refreshing the page or anything like that, once the model behind the scenes comes back with the image, it'll populate in here. And so, I want to also show how you can do things like that and start to build up the the pieces that you need to build something that's interactive and maybe can update on the back end and on the front end.

yeah, so let's look at the code for this. and then I'll quickly show a couple of other apps without talking about how it's made, but we can reason through what are the pieces we need. okay, so this is my image app. There's multiple versions of this scattered around the place, so don't stress if the code here is not exactly the same. I'm going to be using replicate. You've seen them in the course before to actually generate the image. This is not an image generation tutorial, but this is some API that I can go and call.

Now, unlike Jeremy's demos, I thought I'd make the database explicitly. And you can imagine here, like if this was a deployed app, I might have a users table and keeping track of how many credits everyone has, what subscription plan they're on. Then I have generations and what date they were generated, what prompt they used, if they used any extra bells and whistles and settings. and so, this is creating a database that's going to store that information.

well, not quite there, but there is a the the user app.py in fastHTML/examples has a bit more of that stuff, but yeah, we should create some more examples of multi-table. Right. But it'll just be the same thing, right? It'll just be basically if something else not in tables, then something else.create to create your multiple things. okay, so starting with the styling, in case that's going to throw people off, I wanted this nice grid and Pyco CSS, which is like our default favorite, is pretty minimal.

but they recommend using some other layout options if you want to do something like a grid. And so I come to this page and it's got some examples. And also any page that has things, you can right click, you know, inspect and see how they actually do it if you want to get a feel for it. but this especially where you get these long strings of CSS classes, I think this is using, you know, maybe it's Tailwind or something like that. I don't know. As a Python developer, I kind of want to stay away from those.

but it has examples. It's also a web page and so you could do hey ChatGPT, I'm trying to build this. You need some way to put the fastHTML in context, which I can share later. but you could show it this example. Or what you could do is take this and and this is something I've done, copy and paste this HTML and then put this in a notebook. So somewhere here when we come to the image generation app, yeah, this is what I'm going for. I can construct this with our fastHTML approach of using tags that are Python objects.

And one thing we didn't mention, Jono, is because the word class is a special word in Python, we can't say class equals, so we say cls equals. Right, yeah. so if you inspect the HTML, you'll see what it translates into. and that's div class equals properly spelled out. yeah, so to cut a long story short, I've taken this layout library that has some examples. I've turned it into a effectively a Python version. I have to tell it to go and get the CSS. And so this is a link just like you would have the little link tag at the top of your page. and so I'm going to use this to layout my images. So back in the app, I'm including the link to that CSS and I'm telling fast HTML put that in the headers when whenever you make a page for me.

and then this is my home. So before we were constructing the the form and then the to-dos and then the to-do details. Here I have the input for the prompt. And I'm putting that in a form along with a button. And this is going to call a HX post. This is going to send a post request. and it's going to tell me what I'm trying to generate. Then I have a list of previews which we'll look at just now. This is the actual boxes that are going to have the image in them.

I have a div to contain those. I've given it an ID. And so if you're thinking back to Jeremy's demo just now, we're thinking HTMX could like request a preview and then put it right at the top of the the list of gens, right? So I've got this nice container. Whenever I send a prompt, what I'm going to get back, I'm going to put at the top of this list. And sure enough, yeah, HX swap equals after begin, that's exactly what that's saying. Right at the top, the beginning of this list, add whatever HTML you get back when you send this post request.

cool, and I can give it a title and so on. so let's look at the post request. I'm getting in a prompt. And I'm going to generate an image using that prompt. and then I'm going to return back a preview. there's a few other things going on. One, I want to reset the prompt field. And so this is something that Jeremy hasn't talked about yet, but you can use an extra thing. So in addition to the this piece which is saying put this at the start of the list, I'm going to say, "Look, I'm actually going to pass back some extra elements."

this one here is a new version of the input field. but it doesn't have anything in it. So, we're going to clear out the old prompt. And the reason it knows to do that is that I set this special thing called HX swap OOB, standing for out of band. Right? And so, when HTMX gets back several different chunks of of HTML, the first one is going to go in wherever the target was. Right? So, we said the target is right after the start of the generation list div.

Right? But, any extra pieces that have this HX swap out of band, they're going to go and replace whatever has that ID with this new version. So, our original, prompt input here, we're just going to replace that. And that's why when you type something in the, in the interface, you don't see that same prompt sticking around. you'll instead see this nice clear thing. wonder why, oh. Okay, yeah. sometimes the generations take a while.

cool. So, that's the that's like the the framework. there's a magic trick happening because we're not passing back the image straight away. We don't have the image straight away. Instead, what we're doing, and this is something that you'll see a lot, this is in processing that needs to happen and it takes time, but I want my user interface to update like straight away. Right? I'll show a chatbot demo just now. The chatbot might take a while to respond, but you want to start showing the response straight away.

And so, there's a pattern that you can do, which is to say, "Look, if the image has been generated, I'm going to return back my nice pretty image box. you don't have to worry about the div card image, etc. This is just like the nice pretty version that I prototyped in a notebook. but if the image isn't ready yet, I'm instead going to return a temporary thing. And the reason it's temporary is that this is also going to have an HTMX trigger on it.

Specifically, it's going to trigger every 2 seconds. Right? So, we said there's some events like click or whatever, but you can also set HX trigger to be, you know, on hover or on change or if it's dragged around or in this case, just every second or every 2 seconds it's going to be polling. it's going to be sending a get request to this root which we can define. It's going to replace the whole thing. Like it's going to replace itself with whatever it gets back.

Right? So, every 2 seconds it's coming and hitting this and what it gets back is just the same preview again. So, the image doesn't exist. We do this temporary thing. After 2 seconds, it sends a get request to replace itself and it replaces itself with another preview. Which again, maybe the image doesn't exist still. We do another temporary preview. So, it's just this like polling and once the image is available, we can now return a div. This has no trigger on it. So, this is going to stop querying and querying and querying.

but it's now going to show the actual image. and so that's why we can go from this little temporary generating thing to oh, I guess maybe my replicate is Maybe you guys have used up all my replicate credits. That's a possibility. yeah. Okay, so that's that's what's going on. a final thing to note is the reason this happens at all, like the reason I can afford to like return some stuff when the image hasn't finished generating it, is this generate and save function is running in a separate thread. So, when I'm responding to the post request, I basically start the generation and then respond straight away with the temporary preview. this is a very useful pattern because then is a decorator from Fast Core, by the way. If you chuck it on a function, it just makes it run in a thread. There's You don't have to use it. It's just a convenience.

Yeah. Yeah, so you could do this with your preferred like way to start a new a new thread in Python. but in the background it's sending a request to replicate with the prompt and all the other settings, waiting for the image to come back and then saving that. yeah, so that's our image generation app. some pieces to call out. One, I didn't start by writing this whole app like this, right? You start by saying, "Let me first get a page with a form."

Okay, now when I send the form, I'm going to try and get the prompt and print it out. Okay, now I'm going to return the prompt. Okay, now I'm going to call replicate and return an image. Okay, that's taking a while, so now I'm going to do this little trick. So, you build it up piece by piece, and each of those individual bits of functionality isn't too terrible on its own. and then for things like styling, this kind of thing, fortunately

have trained on a lot of web development and a lot of web development content, asking what this means, trying it out in a notebook, figuring out, "Oh, all this is saying is on really big screens, I want each column to be three units wide, and there's 12 units total, although no one really tells you this, unless you happen to like already know this. On a medium screen, each one should be four units wide, and again there's only 12 total, so there's only going to be three columns now.

On a small screen, everything should be two units or six units wide out of 12. and so that's how we get this nice responsive thing, and that's a quirk of the specific like the specific grid library, and every library's going to be different. but you can generally break it down to say, "Okay, I get all these complicated examples, what's actually going on? Oh, it turns out I need a div with this class, and inside there should be divs with that class, and I can construct this in Python, I can test it out in a notebook, then I can start putting that in my app, and gradually tweaking, 'Hey GPT, you know, I want my my chat box to always be the right height. How do I do that with Tailwind?' Oh, you should use this special class."

like well, over units and splitting them up using classes like this seems fairly universal. I think it might have been Bootstrap that first invented it, but like Yeah, I see it pretty much everywhere now. Yeah. but the point is like hopefully you can get away without necessarily knowing that perfectly. because you can Yeah, get around it with these these models that know it or you can find a way to convert that esoteric looking HTML into some component that you can use.

so another demo that we don't necessarily need to look at the code. oops. I guess I already had something there. but yeah, we get a nice little little bit of data back. So this again like this is not the perfect way to do this cuz I haven't got my head around web sockets yet. but this particular chat's piece here has But it works, which is really cool. Like like you were able to use your existing knowledge and in this case you've used I guess this is a you know some component library of you found. You didn't write all this Right. Right. Yeah, so that's dialog thing.

That's something we should look at, right? So Daisy Oh my goodness. Live coding. Daisy UI and many other like collections of components shadcn there's there's lots of work that's already been done on the web to make beautiful components, right? And they're designed to be used in HTML and JavaScript and often in some other library. But when you go and look at them, okay, cool. Here, this is a very complicated thing and there's lots of different classes, but I can find some examples and say I want to emulate this.

This is what I need. Copy and paste this, replicate that in a notebook, figure out how to load this library, which is usually like they'll be some information. Here's how you install it or here's how you get it with a a link at the top of your page. Just that that warning about the CDN like I just always do things from CDN cuz we're not production, you know? Like when you've got 100,000 users, then you can look at the warning. But yeah, just just look at the CDN one and just do that.

Yeah. but yeah, so this is the general pattern. Think of what behavior you code where you like use this Daisy UI

thing and Sure. I'll show it a couple of places. One is So in the I This is a very much a work in progress document, by the way, but I show okay, here's the the code and then we can run this. and I should switch to this tab where you can actually see the outputs here.

Sorry, just a second. Got to get everything so you can see it. So just while you're doing that, you know, the way I would do this, and I don't know if you've done it yet, is I would create a function called like chat or a function called chat header, and it would, you know, return those things. And then you could like have a thing pip installable thing, you know, where you could like pip install fast HTML chat, you know, and then everybody could use those functions.

But all they're actually doing is just going div class equals, you know. Exactly. Yeah, it's all about like defining that. so this is exploring it in a notebook. And then the next step up is Let me get rid of that. Okay. I think I have the chatbot one here. No, this is definitely not the final Yeah, here Yeah, something like this, right? So a chat box that takes a list of messages, and a single chat message that has text and a role, and it's going to tell you how to return that.

So like that's the general pattern. And so then now, you know, in my my main, I'm saying up, I'm going to have a chat box, and it's going to start out with no messages, and then I'm going to make a chat message out of my message and a chat message out of the model's response something along those lines. And that's going to give me the the nice little pretty UI. And as you hit things like, "Oh, okay, I want it to scroll and be a fixed size rather than you know, doing whatever it does by default." I'm going to chat GPT, "Hey, I've got this thing. It looks like this. What magic Tailwind text do I type after this to set it to some percentage of the window height?"

Cool. Someone asking by the way is playing. yeah, I guess Is there any other things? Oh, I mean, I can say like you don't have to build apps that look fancy at first first blush, right? You can start with the behavior that you want and then you can add the pretty pictures. Pretty much everything I've looked at so far will say, "Look, here's our library. It's a CSS file and some scripts that you need to include. And then you can copy that." So, here's my version of the image generation that just says, "Look, what do I actually need to do this?" Like right click inspect their source.

And then you'll see, okay, they have you know, a special class on whatever the the field set is. And they have a special bit of CSS in the header. Nice thing with the web, you can just do this on any website. Like, "How are they doing this? Oh, they're importing a script. And then they're just putting some special decorators or they're following a specific format." And so, yeah, we'll ship with We'll say, "Look, here's how you do it with something that looks nice in PyCon, but people who are better designers than us can say, 'Oh, I really like my Bootstrap components or whatever.'" And it shouldn't be too hard to replicate that.

Nice. Jonathan, that's so cool. I love it. Okay. so, I think that's kind of everything we had. So, like I know a lot of companies have like given out hundreds of dollars of credits or whatever for using these things. I honestly for Railway, given it's five bucks a month, but just go and pay the five bucks, you know, I don't you know, it's I the amount of time it would take Jake to create a script if you five bucks it just honestly doesn't seem worth it.

It's so damn cheap. And the extra three cents to actually use it. Yes, mate, go. And if you are not happy with your $5, we are very very much here. We are happy to make any sort of changes. We're super open to feedback. You can find me on Twitter or if you want to call it x.com or whatever now at just Jake. So, that's Justin and Jake. My DMs are open. You can comment on random posts. You know, we're just looking to make the best possible deployment experience ever, you know, so.

If you don't like it, Dan will refund your $5 for you, right, Dan? Yeah, well, I mean, we'll refund it or Dan will, you know, we can do They made enough from this course. They can afford to. Yeah, and feel free to So, the the repo is just answer.ai/fasthtml. feel free to you know, put suggestions there. and particularly feel free to talk about the library in the fasthtml channel on the fastai server.

And then yeah, like early to mid-July, we'll launch it. And so, if you want to have pip installable modules of components ready by then, then yeah, you know, that's going to be great. And or if you've got suggestions for things for us to add, that's all good as well. and then yeah, hopefully mid-July, we'll launch and people will be excited. And you know, one of the things I want to do is kind of create something that works just like radio, for example, but it's written in fasthtml. So, when people want to expand beyond that, they just like gradually build up with the complexity rather than having some kind of cliff where you have to learn a new language.

So, that's the plan from here. Thanks for having us, Dan. Hey, this was awesome to see. you probably saw in Discord people have love loved everything you've done in the past and are quite excited about this. So, thanks so much. Thanks, man. All right. And thanks for joining us, Jake. Thanks, Jono, for your amazing help. Yeah. Thanks for having me. Appreciate it. Bye, all. See you. Cheers, everybody. Bye.