In this course, I'm going to take you from a complete beginner to building powerful noode AI agents. I don't have any coding experience, and you don't need any either. In the past eight months, I've made over half a million dollars in revenue by building and teaching people how to build AI agents. In this video together, we're going to set up your 2e free and end trial. We're going to set up credentials together and walk through step-by-step builds. And by the end, you'll have over 15 AI automations ready to take advantage of this opportunity. All right, so here is a quick look at the highle course agenda. Keep in mind everything will be timestamped below so you can jump around to where you need. But I definitely recommend saving this for later. As you can see, this is extremely comprehensive, packed with a ton of value, so you can come back to this later when you want to explore different chapters. But what we're going to do is start off with talking about AI agents and the opportunity that we are all living in right now. Then we'll move into Foundations. I'll set up a free twoe trial with you guys. I will talk about the UI. We'll get familiar with it and go over some foundational knowledge you'll need. From there, we'll move into step-by-step workflows where we're actually using that knowledge and connecting to different integrations and setting up some pretty cool automations right away. Then, we'll talk about APIs and HTTP request. We'll set up a few common examples together, and you'll see it's not that difficult. Then, moving into the back half of the course, we'll talk about AI agent tools and memory. We will discuss multi-agent architectures. I'll talk about prompting, do a live example, and some other cool tips that I want to share with you guys that could be helpful. We will look at web hooks, what those really mean. And then we'll look at a few example workflows where we've triggered them with web hooks. I'll talk about MCP servers, what that really is, and we'll do a step-by-step self-hosted setup of Naden and connect to some MCP servers. And finally, we'll close off with lessons from my first 6 months of building AI agents. So, if that all sounds good to you guys, let's go ahead and get started. All right, so AI agents, artificial intelligence, whatever it is, there is definitely a lot of hype. There's no denying that. And so the purpose of this section is just to make sure we can cut through all of that and actually understand what is an AI agent at its core. What can they do and why do we need them? You probably heard you know digital employee or virtual assistant all this kind of stuff. But we need to understand what that actually means and what powers them. So at this point I'm sure we are all familiar with something like chatbt which is a large language model at its core. And so we're looking at right here is a very simple visualization of how a large language model works. Meaning right here in green we have a large language model. Let's say it's chatbt and we the user give it some sort of input. So maybe that's like hey help me write an email to John. The LLM would then take our input process this. It would basically just create an email for us and then it would spit that out as an output and that's it. This LLM at its core cannot take any action. It's really not that practical. It just kind of helps you be more productive because at the end of the process, we'd have to take this output and copy and paste it into something that actually can take action like Gmail. And so the power of these large language models really comes into play when we start to expose them to different tools. And tools just means any of these integrations that we use every single day within our work that let us actually do something. So whether that's send an email or update a row in our CRM or look at a database or Air Table, whatever it is, even Outlook, that's a tool. It just means connecting to an actual platform that we use to do something. So now instead of having the LLM help us write an email that we would copy and paste or us exporting a Google sheet and giving it to an LLM to analyze, it can basically just interact with any of its tools that we give it access to. So when we add an LLM to tools, we basically can get two different things and that is either an AI workflow or an AI agent. So right away you can already tell what the

difference is, but you can also see some similarities. So, let's break it down. Starting with an AI workflow, we can see that we have an input similar to like we did up top with our LLM. But now, instead of just going input, LLM, output, we can work in those tools right into the actual AI workflow itself. So, here is an example of what an AI workflow could practically look like. First of all, we have a tool which is HubSpot, and that's going to be the input for this workflow. This will basically pass over a new lead that has been inserted into our CRM. Then we're hitting another tool which is Perplexity which helps us do research. So we're going to do research on that lead. From there, after we get that research, we're going to hit an LLM, which is where this whole AI powered workflow terminology comes in, because we're using that LLM to then take the research, draft a personalized email, and then it can use another tool to actually send that email. And the reason that we do this as a workflow is because this is going to happen in the same four steps in that order every time. new lead comes in, research, write the personalized email, send the email. And so whenever we know a process is linear or sequential or it's going to follow that order every time, it's much much better to do an actual workflow rather than send that off to an AI agent where we have an input, we have the LLM, which is the AI agent. This is the brain of the whole operation and it has access to all of the different tools it can use and then it has an output. So yes, it is true that this AI agent down here could do the exact same job as this AI workflow. We could also over here with an AI agent get a new form and have a new row in our CRM. The agent could then think about it and decide, okay, I'm going to use perplexity to do research and then after that I'm going to send an email with my email tool. But it's not the most effective way to do it because it's going to be more expensive. It's going to be slower and it's going to be more errorprone. So basically the whole idea is AI agents can make decisions and act autonomously based on different inputs. AI workflows follow the guardrails that we put in place. there's no way they can deviate off the path that we chose for them. So, a big part of building effective systems is understanding, okay, do I need to build an AI workflow or am I going to build an AI agent? Is this process deterministic or nondeterministic? Or in other words, is it predictable or is it unpredictable? If something's unpredictable, that's when we're going to use an AI agent with a brain and with different tools to actually do the job. And that's where the whole autonomy comes into play. And I don't want to dive too deep into the weeds of this right now. We'll cover this later in a different section. But real quick, four main pros of AI workflows over AI agents. Reliability and consistency, cost efficiency, easier debugging and maintenance, and scalability. Once again, we have a whole section dedicated to this idea and we're going to dive into it after we've built out a few AI workflows, but I wanted you guys to understand this because obviously the whole purpose you probably came here was to learn how to build AI agents. But before we build AI agents, we're going to learn how to build AI workflows. It's the whole concept of crawl, walk, run. you wouldn't just start running right away. And trust me, after you've built out a few AI workflows, it's going to make a lot more sense when you hop into building some more complex agentic systems. But just to give you that quick fix of AI agent knowledge, and we'll revisit this later when we actually build our first agent together. What is the anatomy of an AI agent? What are the different parts that make one up? So, here's a simple diagram that I think illustrates AI agents as simple as possible. We have an input, we have our LLM, and we have our output like we talked about earlier. But then inside the AI agent you can see two main things. We have a brain and we have instructions. So the first thing is a brain. This comes in the form of a large language model and also memory. So first off the large language model this is an AI chat model that we'll choose whether that's an open AI model or an anthropic model or a Google model. This is going to be what powers the AI agent to make decisions to reason to generate outputs that

sort of stuff. And then we also have the memory. So this can come in the form of long-term memory as well as short-term memory. But basically, we want to make sure that if we're conversating with our agent, it's not going to forget what we're talking about after every single sentence. It's going to retain that context window, and it can also remember things that we talked about a while back. And then the other piece is the instructions for the AI agent, which is also kind of referred to as a system prompt. And this is really important because this is telling this AI agent, you know, here's your role, here's what you do, here are the tools you have. This is basically like your job description. So, the same way you wouldn't expect a new hire to hop into the company and just start using its different tools and knowing what to do, you would have to give it basically some pretty specific training on this is basically your end goal. Here are the tools you have and here's when you use each one to get the job done. And the system prompt is different than the input, which is kind of referred to as a user prompt. And think of it like this. when you're talking to chatbt in your browser and every single message that you're typing and sending off to it is a user message because that input changes every time. It's dynamic, but the system prompt is typically going to say the same over the course of this agent's life unless its role or actual instructions are going to change. But anyways, let's say the input is, hey, can you help me send an email to John? What's going to happen is the agent's going to use its brain to understand the input. It's going to check its memory to see if there's any other interactions that would help with this current input. Then it will look at its instructions and see, okay, how do I actually send an email to John? And then it will call on its tool to actually send an email. So at a high level, that is the anatomy of an AI agent. And I hope that that helps paint a clear picture in your mind. Cool. So now that we've talked about what an AI agent is and what a workflow is and why we want to walk before we run, let's actually get into Naden and start building some stuff. All right. Right. So, before we dive into actually building AI agents, I want to share some eyeopening research that underscores exactly why you're making such a valuable investment in yourself today. This research report that I'm going to be walking through real quick will be available for free in my school community if you want to go ahead and take a look at it. It's got a total of 48 sources that are all from within the past year. So, you know it's real, you know it's relevant, and it was completely generated for me using Perplexity, which is an awesome AI tool. So, just a year ago, AI was still considered experimental technology for most businesses. Now, it's become the core driver of competitive advantage across every industry and business size. What we're witnessing isn't just another tech trend. It's a fundamental business transformation. Let me start with something that might surprise you. 75% of small businesses now use AI tools. That's right. This isn't just enterprise technology anymore. In fact, the adoption rates are climbing fastest among companies generating just over a million dollars in revenue at 86%. What's truly remarkable is the investment threshold. The median annual AI investment for small businesses is just 1,800. That's less than 150 bucks per month to access technology that was science fiction just a few years ago. Now, I know some of you might be skeptical about AI's practical value. Let's look at concrete outcomes businesses are achieving. Marketing teams are seeing a 22% increase in ROI for AIdriven campaigns. Customer service AI agents have reduced response time by 60% while resolving 80% of inquiries without human intervention. Supply chains optimized with AI have cut transportation costs by 5 to 10% through better routing and demand forecasting. These are actual measured results from implementations over the past year. Now, for those of you from small organizations, consider these examples. Henry's House of Coffee used AIdriven SEO tools to improve their product descriptions, resulting in a 200% improvement in search rankings and 25% revenue increase. Vanisec insurance implemented custom chat bots that

cut client query resolution time from 48 hours to just 15 minutes. Small businesses using Zapier automations saved 10 to 15 hours weekly on routine data entry and CRM updates. What's revolutionary here is that none of these companies needed to hire AI specialists or data scientists to achieve these results. The economic case for AI skills is compelling. 54% of small and medium businesses plan to increase AI spending this year. 83% of enterprises now prioritize AI literacy in their hiring decisions. Organizations with AI trained teams are seeing 5 to 8% higher profitability than their peers. But perhaps most telling is this. Small businesses using AI report 91% higher revenue growth than nonAI adopters. That gap is only widening. So the opportunity ahead. The truth is mastering AI is no longer optional. It's becoming the price of entry for modern business competitiveness. those who delay risk irrelevance while early adopters are already reaping the benefits of efficiency, innovation, and market share gains. Now, the good news is that we're still in the early stages. By developing these skills now, you're positioning yourself at the forefront of this transformation and going to be in extremely high demand over the next decade. So, let's get started building your first AI agent. All right, so here we are on Naden's website. You can get here using the link in the description. And what I'm going to do is go ahead and sign up for a free trial with you guys. And this is exactly the process you're going to take. And you're going to get two weeks of free playing around. And like I said, by the end of those two weeks, you're already going to have automations up and running and tons of templates imported into your workflows. And I'm not going to spend too much time here, but basically Nitn just lets you automate anything. Any business process that you have, you can automate it visually with no code, which is why I love it. So here you can see NIDN lets you automate business processes without limits on your logic. It's a very visual builder. We have a ton of different integrations. We have the ability to use code if you want to. Lots of native nodes to do data transformation. And we have tons of different triggers, tons of different AI nodes. And we're going to dive into this so you can understand what's all going on. But there's also hundreds of templates to get you started. Not only on the end website itself, but also in my free school community. I have almost 100 templates in there that you can plug in right away. Anyways, let's scroll back up to the top and let's get started here with a new account. All right. All right. So, I put in my name, my email, password, and I give my account a name, which will basically be up in the top search bar. It'll be like nate herkdemo.app.n.cloud. So, that's what your account name means. And you can see I'm going to go ahead and start our 14-day free trial. Just have to do some quick little onboarding. So, it asks us what type of team are we on. I'm just going to put product and design. It asks us the size of our company. It's going to ask us which of these things do we feel most comfortable doing. These are all pretty technical. I just want to put none of them, and that's fine. And how did you hear about any? Let's go ahead with YouTube and submit that off. And now you have the option to invite other members to your workspace if you want to collaborate and share some credentials. For now, I'm just going to go ahead and skip that option. So from here, our workspace is already ready. There's a little quick start guide you could watch from Eniden's YouTube channel, but I'm just going to go ahead and click on start automating. All right, so here we are. This is what Eniden looks like. And let's just familiarize with this dashboard a little bit real quick. So up in the top left, we can see we have 14 days left in our free trial and we've used zero out of a,000 executions. An execution just basically means when you run a workflow from end to end that's going to be an execution. So we can see on the lefth hand side we have overview. We have like a personal set of projects. We have things that have been shared with us. We have the ability to add a project. We have the ability to go to our admin panel where we can upgrade our instance of nodn. We can turn it off. That sort of stuff. So here's my admin

panel. You can see how many executions I have, how many active workflows I have, which I'll explain what that means later. We have the ability to go ahead and manage our nen versions. And this is where you could kind of upgrade your plan and change your billing information, stuff like that. But you'll notice that I didn't even have to put any billing details to get started with my twoe free trial. But then if I want to get back into my workspace, I'm just going to click on open right here. And that will send us right back into this dashboard that we were just on. Cool. So right here we can see we can either start from scratch, a new workflow, or we can test a simple AI agent example. So let's just click into here real quick and break down what is actually going on here. So, in order for us to actually access this demo where we're going to just talk to this AI agent, it says that we have to start by saying hi. So, there's an open chat button down here. I'm going to click on open chat and I'm just going to type in here, hi. And what happens is our AI agent fails because this is basically the brain that it needs to use in order to think about our message and respond to us. And what happens is we can see there's an error message. So, because these things are red, I can click into it and I can see what is the error. It says error in subnode OpenAI model. So that would be this node down here which is called OpenAI model. I would click into this node and we can basically see that the error is there is no credentials. So when you're in NADN what happens is in order to access any sort of API which we'll talk about later but in order to access something like your Gmail or OpenAI or your CRM you always need to import some sort of credential which is just a fancy word for a password in order to actually like get into that information. So right here we can see there's 100 free credits from OpenAI. I'm going to click on claim credits. And now we just are using our NEN free OpenAI API credits and we're fine on this front. But don't worry, later in this video I'm going to cover how we can actually go to OpenAI and get an API key and create our own password in here. But for now, we've claimed 100 free credits, which is great. And what I'm going to do is just go ahead and resend this message that says hi. So I can actually go to this hi text and I can just click on this button which says repost message. And that's just going to send it off again. And now our agent's going to actually be able to use its brain and respond to us. So what it says here is welcome to NINDN. Let's start with the first step to give me memory. Click the plus button on the agent that says memory and choose simple memory. Just tell me once you've done that. So sure, why not? Let's click on the plus button under memory. And we'll click on simple memory real quick. And we're already set up. Good to go. So now I'm just going to come down here and say done. Now we can see that our agent was able to use its memory and its brain in order to respond to us. So now it can prompt us to add tools. It can do this other stuff, but we're going to break that down later in this video. Just wanted to show you real quick demo of how this works. So, what I would do is up in the top right, I can click on save just to make sure that the what we've done is actually going to be saved. And then to get back out to the main screen, I'm going to click on either overview or personal. But if I click on overview, that just takes us back to that home screen. But now, let's talk about some other stuff that happens in a workflow. So, up in the top right, I'm going to click on create workflow. You can see now this opens up a new blank page. And then you have the option up here in the top left to name it. So I'm just going to call this one demo. Now we have this new workflow that's saved in our N environment called demo. So a couple things before we actually drag in any nodes is up here. You can see where is this saved. If you have different projects, you can save workflows in those projects. If you want to tag them, you can tag different things like if you have one for customer support or you have stuff for marketing, you can give your workflows different tags just to keep everything organized. But anyways, every single workflow has to start off with some sort of trigger. So when I click on add first step, it

opens up this panel on the right that says what triggers this workflow. So we can have a manual trigger. We can have a certain event like a new message in Telegram or a new row in our CRM. We can have a schedule, meaning we can set this to run at 6 a.m. every single day. We can have a web hook call, form submission, chat message like we saw earlier. There's tons of ways to actually trigger a workflow. So for this example, let's just say I'm going to click on trigger manually, which literally just gives us this button where if we click test workflow, it goes ahead and executes. Cool. So this is a workflow and this is a node, but this is a trigger node. What happens after a trigger node is different types of nodes, whether that's like an action node or a data transformation node or an AI node, some sort of node. So what I would do is if I want to link up a node to this trigger, I would click on the plus button right here. And this pulls up a little panel on the right that says what happens next. Do you want to take action with AI? Do you want to take action within a certain app? Do you want to do data transformation? There's all these other different types of nodes. And what's cool is let's say we wanted to take action within an app. If I clicked on this, we can see all of the different native integrations that Nin has. And once again, in order to connect to any of these tons of different tools that we have here, you always need to get some sort of password. So let's say Google Drive. Now that I've clicked into Google Drive, there's tons of different actions that we can take and they're all very intuitive. you know would you want to copy a file would you want to share a file do you want to create a shared drive it's all very natural language and let's say for example I want to copy a file in order for nitn to tell Google drive which file do we want to copy we first of all have to provide a credential so every app you'll have to provide some sort of credential and then you have basically like a configuration panel right here in the middle which would be saying what is the resource you want what do you want to do what is the file all this kind of stuff so whenever you're in a node in nen what you're going to have is on the left you have an input panel which is basically any data that's going to be feeding into this current node. In the middle you'll have your configuration which is like the different settings and the different little levers you can tweak in order to do different things. And then on the right is going to be the output panel of what actually comes out of this node based on the way that you configured it. So every time you're looking at a node you're going to have three main places input configuration and output. So, let's just do a quick example where I'm going to delete this Google Drive node by clicking on the delete button. I'm going to add an AI node because there's a ton of different AI actions we can take as well. And all I'm going to do is I'm just going to talk to OpenAI's kind of like chatbt. So, I'll click on that and I'm just going to click on message a model. So, once that pulls up, we're going to be using our NEN free OpenAI credits that we got earlier. And as you can see, we have to configure this node. What do we want to do? The resource is going to be text. It could be image, audio, assistant, whatever we want. The operation we're taking is we want to just message a model. And then of course, because we're messaging a model, we have to choose from this list of OpenAI models that we have access to. And actually, it looks like this N free credits only actually give us access to a chat model. And this is a bit different. Not exactly sure why. Probably just because they're free credits. So, what we're going to do real quick is head over to OpenAI and get a credential so I can just show you guys how this works with input configuration and output. So, basically, you'd go to openai.com. You'd come in here and you'd create an account if you don't already have one. If you have a chat GBT account and you're on like maybe the 20 bucks a month plan, that is different than creating an OpenAI API account. So, you'd come in here and create an OpenAI account. As you see up here, we have the option for Chatbt login or API platform login, which is what we're looking for here. So, now that you've created an account with OpenAI's API, what you're

going to do is come up to your dashboard and you're going to go to your API keys. And then all you'd have to do is click on create new key. Name this one whatever you want. And then you have a new secret key. But keep in mind, in order for this key to work, you have to have put in some billing information in your OpenAI account. So, throw in a few bucks. They'll go a lot longer than you may think. And then you're going to take that key that we just copied, come back into Nitn, and under the credential section, we're going to click on create new credential. All I had to do now was paste in that API key right there. And then you have the option to name this credential if you have a ton of different ones. So I can just say, you know, like demo on May 21st. And now I have my credential saved and named because now we can tell the difference between our demo credential and our NAN free OpenAI credits credential. And now hopefully we have the ability to actually choose a model from the list. So, as you can see, we can access chat GBT for latest, 3.5 Turbo, 4, 4.1 mini, all this kind of stuff. I'm going to choose 4.1 mini, but as you can see, you can come back and change this whenever you want. And I'm going to keep this really simple. In the prompt, I'm just going to type in, tell me a joke. So now, when this node executes, it's basically just going to be sending this message to OpenAI's model, which is GBT4.1 Mini, and it's just going to say, "Tell me a joke." And then what we're going to get on the output panel is the actual joke. So what I can do is come up right here and click on test step. This is going to run this node and then we get an output over here. And as you can see both with the input and the output we have three options of how we want to view our data. We can click on schema, we can click on table or we can click on JSON. And this is all the exact same data. It's just like a different way to actually look at it. I typically like to look at schema. I think it just looks the most simple and natural language. But what you can see here is the message that we got back from this open AAI model was sure here's a joke for you. Why don't scientists trust atoms? Because they make up everything. And what's cool about schemas is that this is all drag and drop. So now once we have this output, we could basically just use it however we want. So if I click out of here and I open up another node after this, and for now I'm just going to grab a set node just to show you guys how we can drag and drop. What I would do is let's say we wanted to add a new field and I'm just going to call this open AI's response. So we're creating a field called open AI's response. And as you can see it says drag an input field from the left to use it here. So as we know every node we have input configuration output on the input we can basically choose which one of these things do we want to use. I just want to reference this content which is the actual thing that OpenAI said to us. So I would drag this from here right into the value. And now we can see that we have what's called a variable. So anything that's going to be wrapped in these two curly braces and it's going to be green is a variable. And it's coming through as JSON message.content which is basically just something that represents whatever is coming from the previous node in the field called content. So we can see right here JSON message.content we have message. Within message we have basically a subfolder called content and that's where we access this actual result this real text. And you can see if I click into this variable, if I make it full screen, we have an expression which is our JSON variable. And then we have our result, which is the actual text that we want back. So now if I go ahead and test this step, we can see that we only get output to us OpenAI's response, which is the text we want. Okay, so this would basically be a workflow because we have a trigger and then we have our nodes that are going to execute when we hit test workflow. So if I hit test workflow, it's going to run the whole thing. And as you can see, super visual. We saw that OpenAI was thinking and then we come over here and we get our final output which was the actual joke. And now let me show you one more example of how we can map our different variables without using a

manual trigger. So let's say we don't want a manual trigger. I'm just going to delete that. But now we have no way to run this workflow because there's no sort of trigger. So I'm just going to come back in here and grab a chat trigger just so we can talk to this workflow in Naden. I'm going to hook it up right here. I would just basically drag this plus into the node that I want. So I just drag it into OpenAI. And now these two things are connected. So if I went into the chat and I said hello, it's going to run the whole workflow, but it's not really going to make sense because I said hello and now it's telling me a joke about why don't scientists trust atoms. So what I would want to do is I'd want to come into this OpenAI node right here. And I'm just going to change the actual prompt. So rather than asking it to tell me a joke, what I would do is I'd just delete this. And what I want to do is I want OpenAI to go ahead and process whatever I type in this chat. same way it would work if we were in chatbt in our browser and whatever we type OpenAI responds to. So all I would have to do to do that is I would grab the chat input variable right here. I would drag that into the prompt section. And now if I open this up, it's looking at the expression called JSON.input because this field right here is called chat input. And then the result is going to be whatever we type anytime. even if it's different 100 times in a row, it's always going to come back as a result that's different, but it's always going to be referenced as the same exact expression. So, just to actually show you guys this, let's save this workflow. And I'm going to say, "My name is Nate. I like to eat ice cream. Make up a funny story about me." Okay, so we'll send this off and the response that we should get will be one that is actually about me and it's going to have some sort of element of a story with ice cream. So let's take a look. So it said, "Sure, Nate, here's a funny story for you." And actually, because we're setting it, it's coming through a little weird. So let's actually click into here to look at it. Okay, so here is the story. Let me just make this a little bigger. I can go ahead and drag the configuration panel around by doing this. I can also make it larger or smaller if I do this. So let's just make it small. We'll move it all the way to the left and let's read the story. So, it said, "Sure, Nate. Here's a funny story just for you. Once upon a time, there was a guy named Nate who loved ice cream more than anything else in the world. One day, Nate decided to invent the ultimate ice cream. A flavor so amazing that it would make the entire town go crazy." So, let's skip ahead to the bottom. Basically, what happens is from that day on, Nate's stand became the funniest spot in town. A place where you never knew if you'd get a sweet, savory, or plain silly ice cream. And Nate, he became the legendary ice cream wizard. That sounds awesome. So that's exactly how you guys can see what happened was in this OpenAI node. We have a dynamic input which was us talking to this thing in a chat trigger. We drag in that variable that represents what we type into the user prompt. And this is going to get sent to OpenAI's model of GPT 4.1 Mini because we configured this node to do so. And the reason we were able to actually successfully do that is because we put in our API key or our password for OpenAI. And then on the right we get this output which we can look at either in schema view, table view or JSON view. But they all represent the same data. As you can see, this is the exact story we just read. Something I wanted to talk about real quick that is going to be super helpful for the rest of this course is just understanding what is JSON. And JSON stands for JavaScript object notation. And it's just a way to identify things. And the reason why it's so important to talk about is because over here, right, we all kind of know what schema is. It's just kind of like the way something's broken down. And as you can see, we have different drill downs over here. And we have different things to reference. Then we all understand what a table is. It's kind of like a table view of different objects with different things within them. Kind of like the subfolders. And once again, you can also drag and drop from table view as well. And then we have JSON, which also you can drag and drop. Don't worry, you can drag and drop pretty much this

whole platform, which is why it's awesome. But this may look a little more cody or intimidating, but I want to talk about why it is not. So, first of all, JSON is so so important because everything that we do is pretty much going to be built on top of JSON. Even the workflows that you're going to download later when you'll see like, hey, you can download this template for free. When you download that, it's going to be a JSON file, which means the whole workflow in NN is basically represented as JSON. And so, hopefully that doesn't confuse you guys, but what it is is it's literally just key value pairs. So what I mean by that is like over here the key is index and index equals zero and then we have like the role of the openi assistant and that's the key and the value of the role is assistant. So it's very very natural language if you really break it down. What is the content that we're looking at? The content that we're looking at is this actual content over here. But like I said the great thing about that is that pretty much every single large language model or like chat gbt cloud 3.5 they're all trained on JSON and they all understand it. So, well, because it's universal. So, right here on the left, we're looking at JSON. If I was to just copy this entire JSON, go into ChatgBT and say, "Hey, help me understand this JSON." And then I just basically pasted that in there, it's going to be able to tell us exactly like which keys are in here and what those values are. So, it says this JSON represents the response from an AI model like chatbt in a structured format. Let me break it down for you. So, basically, it's going to explain what each part of this JSON means. We can see the index is zero. That means it's the first response. We can see the role equals assistant. We can see that the content is the funny story about Nate. We can see all this stuff and it basically is able to not only break it down for us, but let's say we need to make JSON. We could say, "Hey, I have this natural language. Can you make that into JSON for me?" Hey, can you help me make a JSON body where my name is Nate? I'm 23 years old. I went to the University of Iowa. I like to play pickle ball. We'll send that off and basically it will be able to turn that into JSON for us. So here you go. We can see name Nate, age 23, education, University of Iowa, interest pickle ball. And so don't let it overwhelm you. If you ever need help either making JSON or understanding JSON, throw it into chat and it will do a phenomenal job for you. And actually, just to show you guys that I'm not lying, let's just copy this JSON that chat gave us. Go back into our workflow and I'm just going to add a set field just to show you guys. And instead of manual mapping, I'm just going to set some data using JSON. So I'm going to delete this, paste in exactly what chat gave me. Hit test step. And what do we see over here? We see the name of someone named Nate. We see their age. We see their education. And we see their interest in either schema table or JSON view. So hopefully that gives you guys some reassurance. And just once again, JSON's super important. And it's not even code. That is just a really quick foundational understanding of a trigger, different nodes, action nodes, AI nodes. You have a ton to play with. And that's kind of like the whole most overwhelming part about NIN is you know what you need to do in your brain, but you don't know maybe which is the best nen node to actually get that job done. So that's kind of the tough part is it's a lot of just getting the reps in, understanding what node is best for what. But I assure you by the time your twoe trial is up, you'll have mastered pretty much all that. All right, but something else I want to show you guys is now what we're looking at is called the editor. So if you look at the top middle right here, we have an editor. And this is where we can, you know, zoom out, we can move around, we can basically edit our workflow right here. And it moves from left to right, as you guys saw, the same way we we read from left to right. And now, because we've done a few runs and we've tested out these different nodes, what we'll click into is executions. And this will basically show us the different times we've ran this workflow. And what's cool about this is it will show us the data that has moved through. So let's say you set up a workflow

that every time you get an email, it's going to send some sort of automated response. You could come into this workflow, you could click on executions, and you could go look at what time they happened, what actually came through, what email was sent, all that kind of stuff. So if I go all the way down to this third execution, we can remember that what I did earlier was I asked this node to tell us a joke. We also had a manual trigger rather than a chat trigger. And we can see this version of the workflow. I could now click into this node and I could see this is when we had it configured to tell us a joke. And we could see the actual joke it told us which was about scientists not trusting atoms. And obviously we can still manipulate this stuff, look at schema, look at table and do the same thing on that left-hand side as well. So I wanted to talk about how you can import templates into your own NN environment because it's super cool and like I said they're all kind of built on top of JSON. So, I'm going to go to NN's website and we're going to go to product and we're going to scroll down here to templates. And you can see there's over 2100 workflow automation templates. So, let's scroll down. Let's say we want to do this one with cloning viral Tik Toks with AI avatars. And we can use this one for free. So, I'll click on use for free. And what's cool is we can either copy the template to clipboard or since we're in the cloud workspace, we could just import it right away. And so, this is logged into my other kind of my main cloud instance, but I'll still show you guys how this works. I would click on this button. it would pull up this screen where I just get to set up a few things. So, there's going to be different things we'd have to connect to. So, you would basically just select your different credentials if you already had them set up. If not, you could create them right here. And then you would just basically be able to hit continue. And as this loads up, you see we have the exact template right there to play with. Or let's say you're scrolling on YouTube and you see just a phenomenal Nate Herk YouTube video that you want to play around with. All you have to do is go to my free school community and you will come into YouTube resources or search for the title of the video. And let's say you wanted to play with this shorts automation that I built. What you'll see right here is a JSON file that you'll have to download. Once you download that, you'll go back into Nitn, create a new workflow, and then when you import that from file if you click on this button right here, you can see the entire workflow comes in. And then all you're going to have to do is follow the setup guide in order to connect your own credentials to these different nodes. All right. And then the final thing I wanted to talk about is inactive versus active workflows. So you may have noticed that none of our executions actually counted up from zero. And the reason is because this is counting active workflow executions. And if we come up here to the top right, we can see that we have the ability to make a workflow active, but it has to have a trigger node that requires activation. So real quick, let's say that we come in here and we want a workflow to start when we have a schedule trigger. So I would go to schedule and I would basically say, okay, I want this to go off every single day at midnight as we have here. And what would happen is while this workflow is inactive, it's only actually going to run if we hit test workflow and then it runs. But if we were to flick this on as active now, it says your schedule trigger will now trigger executions on the schedule you have defined. These executions will not show up immediately in the editor, but you can see them in the execution list. So this is basically saying two things. It's saying now that we have the schedule trigger set up to run at midnight, it's actually going to run at midnight because it's active. If we left this inactive, it would not actually run. And all it meant by the second part is if we were sitting in this workflow at midnight, we wouldn't see it execute and go spinning and green and red in live real time, but it would still show up as an execution. But if it's an active workflow, you just don't get to see them live visually running and spinning anymore. So that's the difference between an active

workflow and an inactive workflow. Let's say you have a trigger that's like um let's say you have a HubSpot trigger where you want this basically to fire off the workflow whenever a new contact is created. So you'd connect to HubSpot and you would make this workflow active so that it actually runs if a new contact's created. If you left this inactive, even though it says it's going to trigger on new contact, it would not actually do so unless this workflow was active. So that's a super important thing to remember. All right. And then one last thing I want to talk about which we were not going to dive into because we'll see examples later is there is one more way that we can see data rather than schema table or JSON and it's something called binary. So binary basically just means an image or maybe a big PDF or a word doc or a PowerPoint file. It's basically something that's not explicitly textbased. So let me show you exactly what that might look like. What I'm going to do is I'm going to add another trigger under this workflow and I'm going to click on tab. And even though it doesn't say like what triggers this workflow, we can still access different triggers. So I'm just going to type in form. And this is going to give us a form submission that basically is an NAND native form. And you can see there's an option at the bottom for triggers. So I'm going to click on this trigger. Now basically what this pulls up is another configuration panel, but obviously we don't have an input because it's a trigger, but we are going to get an output. So anyways, let me just set up a quick example form. I'm just going to say the title of this form is demo. The description is binary data. And now what happens if I click on test step, it's going to pull up this form. And as you can see, we haven't set up like any fields for people to actually submit stuff. So the only option is to submit. But when I hit submit, you can see that the node has been executed. And now there's actually data in here. Submitted at with a timestamp. And then we have different information right here. So let me just show you guys. We can add a form element. And when I'm adding a form element, we can basically have this be, you know, date, it can be a drop down, it can be an email, it can be a file, it can be text. So, real quick, I'm just going to show you an example where, let's say we have a form where someone has to submit their name. We have the option to add a placeholder or make it required. And this isn't really the bulk of what I'm trying to show you guys. I just want to show you binary data. But anyways, let's say we're adding another field that's going to be a file. I'm just going to say file. And this will also be required. And now if I go ahead and hit test step, it's going to pull up a new form for us with a name parameter and a file parameter. So what I did is I put my name and I put in just a YouTube short that I had published. And you can see it's an MP4 file. So if I hit submit, we're going to get this data pulled into N as you can see in the background. Just go ahead and watch. The form is going to actually capture this data. There you go. Form submitted. And now what we see right here is binary data. So this is interesting, right? We still have our schema. We still have our table. We still have our JSON, but what this is showing us is basically, okay, the name that the person submitted was Nate. The file, here are some information about it as far as the name of it, the mime type, and the size, but we don't actually access the file through table or JSON or schema view. The only way we can access a video file is through binary. And as you can see, if I clicked on view, it's my actual video file right here. And so that's all I really wanted to show you guys was when you're working with PDFs or images or videos, a lot of times they're going to come through as binary, which is a little confusing at first, but it's not too bad. And we will cover an example later in this tutorial where we look at a binary file and we process it. But as you can see now, if we were doing a next node, we would have schema, table, JSON, and binary. So we're still able to work with the binary. We're still able to reference it. But I just wanted to throw out there, when you see binary, don't get scared. It just basically means it's a different file type. It's not just textbased. Okay, so that's going to do it for just kind of setting up the foundational

knowledge and getting familiar with the dashboard and the UI a little bit. And as you move into these next tutorials, which are going to be some step by steps, I'm going to walk through every single thing with you guys setting up different accounts with Google and something called Pine Cone. And we'll talk about all this stuff step by step. But hopefully now it's going to be a lot better moving into those sections because you've seen, you know, some of the input stuff and how you configure nodes and just like all this terminology that you may not have been familiar with like JSON, JavaScript variables, workflows, executions, that sort of stuff. So, like I said, let's move into those actual step-by-step builds. And I can assure you guys, you're going to feel a lot more comfortable after you have built a workflow end to end. All right, we're going to talk about data types in Nadn and what those look like. It's really important to get familiar with this before we actually start automating things and building agents and stuff like that. So, what I'm going to do is just pull in a set node. As you guys know, this just lets us modify, add, or remove fields. And it's very, very simple. We basically would just click on this to add fields. We can add the name of the field. We choose the data type, and then we set the value, whether that's a fixed value, which we'll be looking at here, or if we're dragging in some sort of variable from the lefth hand side. But clearly, right now, we have no data incoming. We just have a manual trigger. So, what I'm going to do is zoom in on the actual browser so we can examine this data on the output a bit bigger and I don't have to just keep cutting back and forth with the editing. So, as you can see, there's five main data types that we have access to and end it in. We have a string, which is basically just a fancy name for a word. Um, as you can see, it's represented by a little a, a letter a. Then we have a number, which is represented by a pound sign or a hashtag, whatever you want to call it. Um, it's pretty self-explanatory. Then we have a boolean which is basically just going to be true or false. That's basically the only thing it can be represented by a little checkbox. We have an array which is just a fancy word for list. And we'll see exactly what this looks like. And then we have an object which is probably the most confusing one which basically means it's just this big block which can have strings in them, numbers in them. It can have booleans in them. It can have arrays in them. And it can also have nested objects within objects. So we'll take a look at that. Let's just start off real quick with the string. So let's say a string would be a name and that would be my name. So if I hit test step on the right hand side in the JSON, it comes through as key value pair like we talked about. Name equals Nate. Super simple. You can tell it's a string because right here we have two quotes around the word Nate. So that represents a string. Or you could go to the schema and you can see that with name equals Nate, there's the little letter A and that basically says, okay, this is a string. As you see, it matches up right here. Cool. So that's a string. Let's switch over to a number. Now we'll just say we're looking at age and we'll throw in the number 50. Hit test step. And now we see age equals 50 with the pound sign right here as the symbol in the schema view. Or if we go to JSON view, we have the key value pair age equals 50. But now there are no double quotes around the actual number. It's green. So that's how we know it's not a string. This is a number. And um that's where you may run into some issues where if you had like age coming through as a string, you wouldn't be able to like do any summarizations or filters, you know, like if age is greater than 50, send it off this way. If it's less than 50, send it that way. In order to do that type of filtering and routing, you would need to make sure that age is actually a number variable type or data type. Cool. So there's age. Let's go to a boolean. So we're going to basically just say adult. And that can only be true or false. You see, I don't have the option to type anything here. It's only going to be false or it's only going to be true. And as you can see, it'll come through. It'll look like a string, but there's no quotes around it. It's green. And that's how we know it's a boolean. Or we could go to

schema, and we can see that there's a checkbox rather than the letter A symbol. Now, we're going to move on to an array. And this one's interesting, right? So, let's just say we we want to have a list of names. So, if I have a list of names and I was typing in my name and I tried to hit test step, this is where you would run into an error because it's basically saying, okay, the field called names, which we set right here, it's expecting to get an array, but all we got was Nate, which is basically a string. So, to fix this error, change the type for the field names or you can ignore type conversions, whatever. Um, so if we were to come down to the option and ignore type conversions. So when we hit ignore type conversions and tested the step, it basically just converted the field called names to a string because it just could understand that this was a string rather than an array. So let's turn that back off and let's actually see how we could get this to work if we wanted to make an array. So like we know an array just is a fancy word for a list. And in order for us to actually send through an end and say, okay, this is a list, we have to wrap it in square brackets like this. But we also have to wrap each item in the list in quotes. So I have to go like this and go like that. And now this would pass through as a list of a of different strings. And those are names. And so if I wanted to add another one after the first item, I would put a comma. I put two quotes. And then inside that I could put another name. Hit test step. And now you can see we're getting this array that's made up of different strings and they're all going to be different names. So I could expand that. I could close it out. Um we could drag in different names. And in JSON, what that looks like is we have our key and then we have two closed brackets, which is basically exactly what like right here. This is exactly what we typed right here. So that's how it's being represented within these square brackets right here. Okay, cool. So the final one we have to talk about is an object. And this one's a little more complex. So if I was to hit test step here, it's going to tell us names expects an object, but we got an array. So once again, you could come in here, ignore type conversions, and then it would just basically come through as a string, but it's not coming through as an array. So that's not how we want to do it. And I don't want to mess with the actual like schema of typing in an object. So what I'm going to do is go to chat. I literally just said, give me an example JSON object to put into naden. It gives me this example JSON object. I'm going to copy that. Come into the set node, and instead of manual mapping, I'm just going to customize it with JSON. Paste the one that chat just gave us. And when I hit test step, what we now see first of all in the schema view is we have one item with you know this is an object and all this different stuff makes it up. So we have a string which is name herk. We have a string which is email nate example.com. We have a string which is company true horizon. Then we have an array of interests within this object. So I could close this out. I could open it up. And we have three interests. AI automation nadn and YouTube content. And this is, you know, chat GBT's long-term memory about me making this. And then we also have an object within our object which is called project. And the interesting difference here with an object or an array is that when you have an array of interests, every single item in that array is going to be called interest zero, interest one, interest two. And by the way, this is three interests, but computers start counting from zero. So that's why it says 0, one, two. But with an object, it doesn't all have to be the same thing. So you can see in this project object project object we have one string called title we have one string called called called status and we have one string called deadline and this all makes up its own object. As you can see if we went to table view this is literally just one item that's really easy to read. And you can tell that this is an array because it goes 012. And you can tell that this is an object because it has different fields in it. This is a one item. It's one object. It's got strings up top. It has no numbers actually. So the date right here, this is coming through as a string variable type. We can tell because it's not green. We can tell

because it has double quotes around it. And we can also tell because in schema it comes through with the letter A. But this is just how you can see there's these different things that make up um this object. And you can even close them down in JSON view. We can see interest is an array that has three items. We could open that up. We can see project is an object because it's wrapped in in um curly braces, not not um the closed square brackets as you can see. So, there's a difference. And I know this wasn't super detailed and it's just something really really important to know heading into when you actually start to build stuff out because you're probably going to get some of those errors where you're like, you know, blank expects an object but got this or expects an array and got this. So, just wanted to make sure I came in here and threw that module at you guys and hopefully it'll save you some headaches down the road. Real quick, guys, if you want to be able to download all the resources from this video, they'll be available for free in my free school community, which will be the link in the pinned comment. There'll be a zip file in there that has all 23 of these workflows, as you can see, and also two PDFs at the bottom, which are covered in the video. So, like I said, join the Free School community. Not only does it have all of my YouTube resources, but it's also a really quick growing community of people who are obsessed with AI automation and using ND every day. All you'll have to do is search for the title of this video using the search bar or you can click on YouTube resources and find the post associated with this video. And then you'll have the zip file right here to download which once again is going to have all 23 of these JSON N workflows and two PDFs. And there may even be some bonus files in here. You'll just have to join the free school community to find out. Okay, so we talked about AI agents. We talked about AI workflows. We've gotten into NADN and set up our account. We understand workflows, nodes, triggers, JSON, stuff like that, and data types. Now, it's time to use all that stuff that we've talked about and start applying it. So, we're going to head into this next portion of this course, which is going to be about step-by-step builds, where I'm going to walk you through every single step live, and we'll have some pretty cool workflows set up by the end. So, let's get into it. Today, we're going to be looking at three simple AI workflows that you can build right now to get started learning NAND. We're going to walk through everything step by step, including all of the credentials and the setups. So, let's take a look at the three workflows we're going to be building today. All right, the first one is going to be a rag pipeline and chatbot. And if you don't know what rag means, don't worry. We're going to explain it all. But at a high level, what we're doing is we're going to be using Pine Cone as a vector database. If you don't know what a vector database is, we'll break it down. We're going to be using Google Drive. We're going to be using Google Docs. And then something called Open Router, which lets us connect to a bunch of different AI models like OpenAI's models or Anthropics models. The second workflow we're going to look at is a customer support workflow that's kind of going to be building off of the first one we just built. Because in the first workflow, we're going to be putting data into a Pine Cone vector database. And in this one, we're going to use that data in there in order to respond to customer support related emails. So, we'll already have had Pine Cone set up, but we're going to set up our credentials for Gmail. And then we're also going to be using an NAN AI agent as well as Open Router once again. And then finally, we're going to be doing LinkedIn content creation. And in this one, we'll be using an NAN AI agent and open router once again, but we'll have two new credentials to set up. The first one being Tavi, which is going to let us search the web. And then the second one will be Google Sheets where we're going to store our content ideas, pull them in, and then have the content written back to that Google sheet. So by the end of this video, you're going to have three workflows set up and you're going to have a really good foundation to continue to learn more

about NADN. You'll already have gotten a lot of credentials set up and understand what goes into connecting to different services. One of the trickiest being Google. So we'll walk through that step by step and then you'll have it configured and you'll be good. And then from there, you'll be able to continuously build on top of these three workflows that we're going to walk through together because there's really no such thing as a finished product in the space. Different AI models keep getting released and keep getting better. There's always ways to improve your templates. And the cool thing about building workflows in NAN is that you can make them super customized for exactly what you're looking for. So, if this sounds good to you, let's hop into that first workflow. Okay, so for this first workflow, we're building a rag pipeline and chatbot. And so if that sounds like a bunch of gibberish to you, let's quickly understand what rag is and what a vector database is. So rag stands for retrieval augmented generation. And in the simplest terms, let's say you ask me a question and I don't actually know the answer. I would just kind of Google it and then I would get the answer from my phone and then I would tell you the answer. So in this case, when we're building a rag chatbot, we're going to be asking the chatbot questions and it's not going to know the answer. So it's going to look inside our vector database, find the answer, and then it's going to respond to us. And so when we're combining the elements of rag with a vector database, here's how it works. So the first thing we want to talk about is actually what is a vector database. So essentially this is what a vector database would look like. We're all familiar with like an x and yaxis graph where you can plot points on there on a two dimensional plane. But a vector database is a multi-dimensional graph of points. So in this case, you can see this multi-dimensional space with all these different points or vectors. And each vector is placed based on the actual meaning of the word or words in the vector. So over here you can see we have wolf, dog and cat. And they're placed similarly because the meaning of these words are all like animals. Whereas over here we have apple and banana which the meaning of the words are food more likely fruits. And that's why they're placed over here together. So when we're searching through the database, we basically vectorize a question the same way we would vectorize any of these other points. And in this case, we were asking for a kitten. And then that query gets placed over here near the other animals and then we're able to say okay well we have all these results now. So what that looks like and what we'll see when we get into NAND is we have a document that we want to vectorize. We have to split the document up into chunks because we can't put like a 50page PDF as one chunk. So it gets split up and then we're going to run it through something called an embeddings model which basically just turns text into numbers. Just as simple as that. And as you can see in this case let's say we had a document about a company. We have company data, finance data, and marketing data. And they all get placed differently because they mean different things. And the the context of those chunks are different. And then this visual down here is just kind of how an LLM or in this case, this agent takes our question, turns it into its own question. We vectorize that using the same embeddings model that we used up here to vectorize the original data. And then because it gets placed here, it just grabs back any vectors that are nearest, maybe like the nearest four or five, and then it brings it back in order to respond to us. So don't want to dive too much into this. Don't want to over complicate it, but hopefully this all makes sense. Cool. So now that we understand that, let's actually start building this workflow. So what we're going to do here is we are going to click on add first step because every workflow needs a trigger that basically starts the workflow. So, I'm going to type in Google Drive because what we're going to do is we are going to pull in a document from our Google Drive in order to vectorize it. So, I'm going to choose a trigger which is on changes involving a specific folder. And what we have to do now is connect our account. As you can

see, I'm already connected, but what we're going to do is click on create new credential in order to connect our Google Drive account. And what we have to do is go get a client ID and a secret. So, what we want to do is click on open docs, which is going to bring us to Naden's documents on how to set up this credential. We have a prerequisite which is creating a Google Cloud account. So I'm going to click on Google Cloud account and we're going to set up a new project. Okay. So I just signed into a new account and I'm going to set up a whole project and walk through the credentials with you guys. You'll click up here. You'll probably have something up here that says like new project and then you'll click into new project. All we have to do now is um name it and you you'll be able to start for free so don't worry about that yet. So I'm just going to name this one demo and I'm going to create this new project. And now up here in the top right you're going to see that it's kind of spinning up this project. and then we'll move forward. Okay, so it's already done and now I can select this project. So now you can see up here I'm in my new project called demo. I'm going to click on these three lines in the top left and what we're going to do first is go to APIs and services and click on enabled APIs and services. And what we want to do is add the ones we need. And so right now all I'm going to do is add Google Drive. And you can see it's going to come up with Google Drive API. And then all we have to do is really simply click enable. And there we I just enabled it. So you can see here the status is enabled. And now we have to set up something called our OOTH consent screen, which basically is just going to let Nadn know that Google Drive and Naden are allowed to talk to each other and have permissions. So right here, I'm going to click on OOTH consent screen. We don't have one yet, so I'm going to click on get started. I'm going to give it a name. So we're just going to call this one demo. Once again, I'm going to add a support email. I'm going to click on next. Because I'm not using a Google Workspace account, I'm just using a, you know, nate88@gmail.com. I'm going to have to choose external. I'm going to click on next. For contact information, I'm putting the same email as I used to create this whole project. Click on next and then agree to terms. And then we're going to create that OOTH consent screen. Okay, so we're not done yet. The next thing we want to do is we want to click on audience. And we're going to add ourselves as a test user. So we could also make the app published by publishing it right here, but I'm just going to keep it in test. And when we keep it in test mode, we have to add a test user. So I'm going to put in that same email from before. And this is going to be the email of the Google Drive we want to access. So I put in my email. You can see I saved it down here. And then finally, all we need to do is come back into here. Go to clients. And then we need to create a new client. We're going to click on web app. We're going to name it whatever we want. Of course, I'm just going to call this one demo once again. And now we need to basically add a redirect URI. So if you click back in Nitn, we have one right here. So, we're going to copy this, go back into cloud, and we're going to add a URI and paste it right in there, and then hit create, and then once that's created, it's going to give us an ID and a secret. So, all we have to do is copy the ID, go back into Nit and paste that right here. And then we need to go grab our secret from Google Cloud, and then paste that right in there. And now we have a little button that says sign in with Google. So, I'm going to open that up. It's going to pull up a window to have you sign in. Make sure you sign in with the same account that you just had yourself as a test user. That one. And then you'll have to continue. And then here is basically saying like what permissions do we have? Does anyone have to your Google Drive? So I'm just going to select all. I'm going to hit continue. And then we should be good. Connection successful and we are now connected. And you may just want to rename this credential so you know you know which email it is. So now I've saved my credential and we should be able to access the Google Drive now. So, what I'm going to do

is I'm going to click on this list and it's going to show me the folders that I have in Google Drive. So, that's awesome. Now, for the sake of this video, I'm in my Google Drive and I'm going to create a new folder. So, new folder. We're going to call this one um FAQ. Create this one because we're going to be uploading an FAQ document into it. So, here's my FAQ folder um right here. And then what I have is down here I made a policy and FAQ document which looks like this. We have some store policies and then we also have some FAQs at the bottom. So, all I'm going to do is I'm going to drag in my policy and FAQ document into that new FAQ folder. And then if we come into NAN, we click on the new folder that we just made. So, it's not here yet. I'm just going to click on these dots and click on refresh list. Now, we should see the FAQ folder. There it is. Click on it. We're going to click on what are we watching this folder for. I'm going to be watching for a file created. And then, I'm just going to hit fetch test event. And now we can see that we did in fact get something back. So, let's make sure this is the right one. Yep. So, there's a lot of nasty information coming through. I'm going to switch over here on the right hand side. This is where we can see the output of every node. I'm going to click on table and I'm just going to scroll over and there should be a field called file name. Here it is. Name. And we have policy and FAQ document. So, we know we have the right document in our Google Drive. Okay. So, perfect. Every time we drop in a new file into that Google folder, it's going to start this workflow. And now we just have to configure what happens after the workflow starts. So, all we want to do really is we want to pull this data into n so that we can put it into our pine cone database. So, off of this trigger, I'm going to add a new node and I'm going to grab another Google Drive node because what happened is basically we have the file ID and the file name, but we don't have the contents of the file. So, we're going to do a download file node from Google Drive. I'm going to rename this one and just call it download file just to keep ourselves organized. We already have our credential connected and now it's basically saying what file do you want to download. We have the ability to choose from a list. But if we choose from the list, it's going to be this file every time we run the workflow. And we want to make this dynamic. So we're going to change from list to by ID. And all we have to do now is we're going to look on the lefth hand side for that file that we just pulled in. And we're going to be looking for the ID of the file. So I can see that I found it right down here in the spaces array because we have the name right here and then we have the ID right above it. So, I'm going to drag ID, put it right there in this folder. It's coming through as a variable called JSON ID. And that's just basically referencing, you know, whenever a file comes through on the the Google Drive trigger. I'm going to use the variable JSON. ID, which will always pull in the files ID. So, then I'm going to hit test step and we're going to see that we're going to get the binary data of this file over here that we could download. And this is our policy and FAQ document. Okay. So, there's step two. We have the file downloaded in NADN. And now it's just as simple as putting it into pine cone. So before we do that, let's head over to pine cone.io. Okay, so now we are in pine cone.io, which is a vector database provider. You can get started for free. And what we're going to do is sign up. Okay, so I just got logged in. And once you get signed up, you should see us a page similar to this. It's a get started page. And what we want to do is you want to come down here and click on, you know, begin setup because we need to create an index. So I'm going to click on begin setup. We have to name our index. So you can call this whatever you want. We have to choose a configuration for a text model. We have to choose a configuration for an embeddings model, which is sort of what I talked about right in here. This is going to turn our text chunks into a vector. So what I'm going to do is I'm going to choose text embedding three small from OpenAI. It's the most cost effective OpenAI embedding model. So I'm going to choose that. Then I'm going to keep scrolling down. I'm

going to keep mine as serverless. I'm going to keep AWS as the cloud provider. I'm going to keep this region. And then all I'm going to do is hit create index. Once you create your index, it'll show up right here. But we're not done yet. You're going to click into that index. And so I already obviously have stuff in my vector database. You won't have this. What I'm going to do real quick is just delete this information out of it. Okay. So this is what yours should look like. There's nothing in here yet. We have no name spaces and we need to get this configured. So on the left hand side, go over here to API keys and you're going to create a new API key. Name it whatever you want, of course. Hit create key. And then you're going to copy that value. Okay, back in NDN, we have our API key copied. We're going to add a new node after the download file and we're going to type in pine cone and we're going to grab a pine cone vector store. Then we're going to select add documents to a vector store and we need to set up our credential. So up here, you won't have these and you're going to click on create new credential. And all we need to do here is just an API key. We don't have to get a client ID or a secret. So you're just going to paste in that API key. Once that's pasted in there and you've given it a name so you know what this means. You'll hit save and it should go green and we're connected to Pine Cone and you can make sure that you're connected by clicking on the index and you should have the name of the index right there that we just created. So I'm going to go ahead and choose my index. I'm going to click on add option and we're going to be basically adding this to a Pine Cone namespace which back in here in Pine Cone if I go back into my database my index and I click in here you can see that we have something called namespaces. And this basically lets us put data into different folders within this one index. So if you don't specify an index, it'll just come through as default and that's going to be fine. But we want to get into the habit of having our data organized. So I'm going to go back into NADN and I'm just going to name this name space FAQ because that's the type of data we're putting in. And now I'm going to click out of this node. So you can see the next thing that we need to do is connect an embeddings model and a document loader. So let's start with the embeddings model. I'm going to click on the plus and I'm going to click on embeddings open AAI. And actually, this is one thing I left out of the Excalaw is that we also will need to go get an OpenAI key. So, as you can see, when we need to connect a credential, you'll click on create new credential and we just need to get an API key. So, you're going to type in OpenAI API. You'll click on this first link here. If you don't have an account yet, you'll sign in. And then once you sign up, you want to go to your dashboard. And then on the lefth hand side, very similar thing to Pine Cone, you'll click on API keys. And then we're just going to create a new key. So you can see I have a lot. We're going to make a new one. And I'm calling everything demo, but this is going to be demo number three. Create new secret key. And then we have our key. So we're going to copy this and we're going to go back into Nit. Paste that right here. We paste it in our key. We've given in a name. And now we'll hit save and we should go green. Just keep in mind that you may need to top up your account with a few credits in order for you to actually be able to run this model. Um, so just keep that in mind. So then what's really important to remember is when we set up our pine cone index, we use the embedding model text embedding three small from OpenAI. So that's why we have to make sure this matches right here or this automation is going to break. Okay, so we're good with the embeddings and now we need to add a document loader. So I'm going to click on this plus right here. I'm going to click on default data loader and we have to just basically tell Pine Cone the type of data we're putting in. And so you have two options, JSON or binary. In this case, it's really easy because we downloaded a a Google Doc, which is on the lefth hand side. You can tell it's binary because up top right here on the input, we can switch between JSON and binary. And if we were

uploading JSON, all we'd be uploading is this gibberish nonsense information that we don't need. We want to upload the binary, which is the actual policy and FAQ document. So, I'm just going to switch this to binary. I'm going to click out of here. And then the last thing we need to do is add a text splitter. So, this is where I was talking about back in this Excal. we have to split the document into different chunks. And so that's what we're doing here with this text splitter. I'm going to choose a recursive character text splitter. There's three options and I won't dive into the difference right now, but recursive character text splitter will help us keep context of the whole document as a whole, even though we're splitting it up. So for now, chunk size is a th00and. That's just basically how many characters am I going to put in each chunk? And then is there going to be any overlap between our chunks of characters? So right now I'm just going to leave it default a,000 and zero. So that's it. You just built your first automation for a rag pipeline. And now we're just going to click on the play button above the pine cone vector store node in order to see it get vectorized. So we're going to basically see that we have four items that have left this node. So this is basically telling us that our Google doc that we downloaded right here. So this document got turned into four different vectors. So if I click into the text splitter, we can see we have four different responses and this is the contents that went into each chunk. So we can just verify this by heading real quick into Pine Cone, we can see we have a new name space that we created called FAQ. Number of records is four. And if we head over to the browser, we can see that we do indeed have these four vectors. And then the text field right here, as you can see, are the characters that were put into each chunk. Okay, so that was the first part of this workflow, but we're going to real quick just make sure that this actually works. So we're going to add a rag chatbot. Okay. So, what I'm going to do now is hit the tab, or I could also have just clicked on the plus button right here, and I'm going to type in AI agent, and that is what we're going to grab and pull into this workflow. So, we have an AI agent, and let's actually just put him right over here. Um, and now what we need to do is we need to set up how are we actually going to talk to this agent. And we're just going to use the default N chat window. So, once again, I'm going to hit tab. I'm going to type in chat. And we have a chat trigger. And all I'm going to do is over here, I'm going to grab the plus and I'm going to drag it into the front of the AI agent. So basically now whenever we hit open chat and we talk right here, the agent will read that chat message. And we know this because if I click into the agent, we can see the user message is looking for one in the connected chat trigger node, which we have right here connected. Okay, so the first step with an AI agent is we need to give it a brain. So we need to give it some sort of AI model to use. So we're going to click on the plus right below chat model. And what we could do now is we could set up an OpenAI chat model because we already have our API key from OpenAI. But what I want to do is click on open router because this is going to allow us to choose from all different chat models, not just OpenAIs. So we could do Claude, we could do Google, we could do Plexity. We have all these different models in here which is going to be really cool. And in order to get an Open Router account, all you have to do is go sign up and get an API key. So you'll click on create new credential and you can see we need an API key. So you'll head over to openouter.ai. You'll sign up for an account. And then all you have to do is in the top right, you're going to click on keys. And then once again, kind of the same as all all the other ones. You're going to create a new key. You're going to give it a name. You're going to click create. You have a secret key. You're going to click copy. And then when we go back into NN and paste it in here, give it a name. And then hit save. And we should go green. We've connected to Open Router. And now we have access to any of these different chat models. So, in this case, let's use let's use Claude um 3.5 Sonnet. And this is just to show you guys you can connect to

different ones. But anyways, now we could click on open chat. And actually, let me make sure you guys can see him. If we say hello, it's going to use its brain claw 3.5 sonnet. And now it responded to us. Hi there. How can I help you? So, just to validate that our information is indeed in the Pine Cone vector store, we're going to click on a tool under the agent. We're going to type in Pine Cone um and grab a Pine Cone vector store and we're going to grab the account that we just selected. So, this was the demo I just made. We're going to give it a name. So, in this case, I'm just going to say knowledge base. We're going to give a description. Call this tool to access the policy and FAQ database. So, we're basically just describing to the agent what this tool does and when to use it. And then we have to select the index and the name space for it to look inside of. So the index is easy. We only have one. It's called sample. But now this is important because if you don't give it the right name space, it won't find the right information. So we called ours FAQ. If you remember in um our Pine Cone, we have a namespace and we have FAQ right here. So that's why we're doing FAQ. And now it's going to be looking in the right spot. So before we can chat with it, we have to add an embeddings model to our Pine Cone vector store, which same thing as before. We're going to grab OpenAI and we're going to use embedding3 small and the same credential you just made. And now we're going to be good to go to chat with our rag agent. So looking back in the document, we can see we have some different stuff. So I'm going to ask this chatbot what the warranty policy is. So I'm going to open up the chat window and say what is our warranty policy? Send that off. And we should see that it's going to use its brain as well as the vector store in order to create an answer for us because it didn't know by itself. So there we go. just finished up and it said based on the information from our knowledge base, here's the warranty policy. We have one-year standard coverage. We have, you know, this email for claims processes. You must provide proof of purchase and for warranty exclusions that aren't covered, damage due to misuse, water damage, blah blah blah. Back in the policy documentation, we can see that that is exactly what we have in our knowledge base for warranty policy. So, just because I don't want this video to go too long, I'm not going to do more tests, but this is where you can get in there and make sure it's working. One thing to keep in mind is within the agent, we didn't give it a system prompt. And what a system prompt is is just basically a message that tells the agent how to do its job. So what you could do is if you're having issues here, you could say, you know, like this is the name of our tool which is called knowledgeb. You could tell the agent and in system prompt, hey, like your job is to help users answer questions about the um you know, our policy database. You have a tool called knowledgebase. You need to use that in order to help them answer their questions. and that will help you refine the behavior of how this agent acts. All right, so the next one that we're doing is a customer support workflow. And as always, you have to figure out what is the trigger for my workflow. In this case, it's going to be triggered by a new email received. So I'm going to click on add first step. I'm going to type in Gmail. Grab that node. And we have a trigger, which is on message received right here. And we're going to click on that. So what we have to do now is obviously authorize ourselves. So we're going to click on create new credential right here. And all we have to do here is use OOTH 2. So all we have to do is click on sign in. But before we can do that, we have to come over to our Google Cloud once again. And now we have to make sure we enable the Gmail API. So we'll click on Gmail API. And it'll be really simple. We'll just have to click on enable. And now we should be able to do that OOTH connection and actually sign in. You'll click on the account that you want to access the Gmail. You'll give it access to everything. Click continue. And then we're going to be connected as you can see. And then you'll want to name this credential as always. Okay. So now we're using our new credential. And what I'm going to do

is if I hit fetch test event. So now we are seeing an email that I just got in this inbox which in this case was nencloud was granted access to your Google account blah blah blah. Um so that's what we just got. Okay. So I just sent myself a different email and I'm going to fetch that email now from this inbox. And we can see that the snippet says what is the privacy policy? I'm concerned about my data and passwords. And what we want to do is we want to turn off simplify because what this button is doing is it's going to take the content of the email and basically, you know, cut it off. So in this case, it didn't matter, but if you're getting long emails, it's going to cut off some of the email. So if we turn off simplify fetch test event, once again, we're now going to get a lot more information about this email, but we're still going to be able to access the actual content, which is right here. We have the text, what is privacy policy? I'm concerned about my data and passwords. Thank you. And then you can see we have other data too like what the subject was, who the email is coming from, what their name is, all this kind of stuff. But the idea here is that we are going to be creating a workflow where if someone sends an email to this inbox right here, we are going to automatically look up the customer support policy and respond back to them so we don't have to. Okay. So the first thing I'm actually going to do is pin this data just so we can keep it here for testing. Which basically means whenever we rerun this, it's not going to go look in our inbox. It's just going to keep this email that we pulled in, which helps us for testing, right? Okay, cool. So, the next step here is we need to have AI basically filter to see is this email customer support related? If yes, then we're going to have a response written. If no, we're going to do nothing because maybe the use case would be okay, we're going to give it an access to an inbox where we're only getting customer support emails. But sometimes maybe that's not the case. And let's just say we wanted to create this as sort of like an inbox manager where we can route off to different logic based on the type of email. So that's what we're going to do here. So I'm going to click on the plus after the Gmail trigger and I'm going to search for a text classifier node. And what this does is it's going to use AI to read the incoming email and then determine what type of email it is. So because we're using AI, the first thing we have to do is connect a chat model. We already have our open router credential set up. So I'm going to choose that. I'm going to choose the credential and then I'm for this one, let's just keep it with 40 mini. And now this AI node actually has AI and I'm going to click into the text classifier. And the first thing we see is that there's a text to classify. So all we want to do here is we want to grab the actual content of the email. So I'm going to scroll down. I can see here's the text, which is the email content. We're going to drag that into this field. And now every time a new email comes through, the text classifier is going to be able to read it because we put in a variable which basically represents the content of the email. So now that it has that, it still doesn't know what to classify it as or what its options are. So we're going to click on add category. The first category is going to be customer support. And then basically we need to give it a description of what a customer support email could look like. So I wanted to keep this one simple. It's pretty vague, but you could make this more detailed, of course. And I just sent an email that's related to helping out a customer. They may be asking questions about our policies or questions about our products or services. And what we can do is we can give it specific examples of like here are some past customer support emails and here's what they've looked like. And that will make this thing more accurate. But in this case, that's all we're going to do. And then I'm going to add one more category that's just going to be other. And then for now, I'm just going to say any email that is not customer support related. Okay, cool. So now when we click out of here, we can see we have two different branches coming off of this node, which means when the text classifier decides, it's either going to send it off this branch or it's going to send it down this branch. So let's quickly hit play. It's going to be

reading the email using its brain. And now you can see it has outputed in the customer support branch. We can also verify by clicking into here. And we can see customer support branch has one item and other branch has no items. And just to keep ourselves organized right now, I'm going to click on the other branch and I'm just going to add an operation that says do nothing just so we can see, you know, what would happen if it went this way for now. But now is where we want to configure the logic of having an agent be able to read the email, hit the vector database to get relevant information and then help us write an email. So I'm going to click on the plus after the customer support branch. I'm going to grab an AI agent. So this is going to be very similar to the way we set up our AI agent in the previous workflow. So, it's kind of building on top of each other. And this time, if you remember in the previous one, we were talking to it with a connected chat trigger node. And as you can see here, we don't have a connected chat trigger node. So, the first thing we want to do is change that. We want to define below. And this is where you would think, okay, what do we actually want the agent to read? We want it to read the email. So, I'm going to do the exact same thing as before. I'm going to go into the Gmail trigger node, scroll all the way down until we can find the actual email content, which is right here, and just drag that right in. That's all we're going to do. And then we definitely want to add a system message for this agent. We are going to open up the system message and I'm just going to click on expression so I can expand this up full screen. And we're going to write a system prompt. Again, for the sake of the video, keeping this prompt really concise, but if you want to learn more about prompting, then definitely check out my communities linked down below as well as this video up here and all the other tutorials on my channel. But anyways, what we said here is we gave it an overview and instructions. The overview says you are a customer support agent for TechHaven. Your job is to respond to incoming emails with relevant information using your knowledgebased tool. And so when we do hook up our Pine Cone vector database, we're just going to make sure to call it knowledgebase because that's what the agent thinks it has access to. And then for the instructions, I said your output should be friendly and use emojis and always sign off as Mr. Helpful from TechHaven Solutions. And then one more thing I forgot to do actually is we want to tell it what to actually output. So if we didn't tell it, it would probably output like a subject and a body. But what's going to happen is we're going to reply to the incoming email. We're not going to create a new one. So we don't need a subject. So I'm just going to say output only the body content of the email. So then we'll give it a try and see what that prompt looks like. We may have to come back and refine it, but for now we're good. Um, and as you know, we have to connect a chat model and then we have to connect our pine cone. So first of all, chat model, we're going to use open router. And just to show you guys, we can use a different type of model here. Let's use something else. Okay. So, we're going to go with Google Gemini 2.0 Flash. And then we need to add the Pine Cone database. So, I'm going to click on the plus under tool. I'm going to search for Pine Cone Vector Store. Grab that. And we have the operation is going to be retrieving documents as a tool for an AI agent. We're going to call this knowledge capital B. And we're going to once again just say call this tool to access policy and FAQ information. We need to set up the index as well as the namespace. So sample and then we're going to call the namespace, you know, FAQ because that's what it's called in our pine cone right here as you can see. And then we just need to add our embeddings model and we should be good to go which is embedded OpenAI text embedding three small. So we're going to hit the play above the AI agent and it's going to be reading the email. As you can see once again the prompt user message. It's reading the email. What is the privacy policy? I'm concerned about my data and my passwords. Thank you. So we're going to hit the play

above the agent. We're going to watch it use its brain. We're going to watch it call the vector store. And we got an error. Okay. So, I'm getting this error, right? And it says provider returned error. And it's weird because basically why it's erroring is because of our our chat model. And it's it's weird because it goes green, right? So, anyways, what I would do here is if you're experiencing that error, it means there's something wrong with your key. So, I would go reset it. But for now, I'm just going to show you the quick fix. I can connect to a OpenAI chat model real quick. And I can run this here and we should be good to go. So now it's going to actually write the email and output. Super weird error, but I'm honestly glad I caught that on camera to show you guys in case you face that issue because it could be frustrating. So we should be able to look at the actual output, which is, "Hey there, thank you for your concern about privacy policy. At Tech Haven, we take your data protection seriously." So then it gives us a quick summary with data collection, data protection, cookies. If we clicked into here and went to the privacy policy, we could see that it is in fact correct. And then it also was friendly and used emojis like we told it to right here in the system prompt. And finally, it signed off as Mr. Helpful from Tech Haven Solutions, also like we told it to. So, we're almost done here. The last thing that we want to do is we want to have it actually reply to this person that triggered the whole workflow. So, we're going to click on the plus. We're going to type in Gmail. Grab a Gmail node and we're going to do reply to a message. Once we open up this node, we already know that we have it connected because we did that earlier. We need to configure the message ID, the message type, and the message. And so all I'm going to do is first of all, email type. I'm going to do text. For the message ID, I'm going to go all the way down to the Gmail trigger. And we have an ID right here. This is the ID we want to put into the message ID so that it responds in line on Gmail rather than creating a new thread. And then for the message, we're going to just drag in the output from the agent that we just had write the message. So, I'm going to grab this output, put it right there. And now you can see this is how it's going to respond in email. And the last thing I want to do is I want to click on add option, append nadn attribution, and then just check that off. So then at the bottom of the email, it doesn't say this was sent by naden. So finally, we'll hit this test step. We will see we get a success message that the email was sent. And I'll head over to the email to show you guys. Okay, so here it is. This is the one that we sent off to that inbox. And then this is the one that we just got back. As you can see, it's in the same thread and it has basically the privacy policy outlined for us. Cool. So, that's workflow number two. Couple ways we could make this even better. One thing we could do is we could add a node right here. And this would be another Gmail one. And we could basically add a label to this email. So, if I grab add label to message, we would do the exact same thing. We'd grab the message ID the same way we grabbed it earlier. So, now it has the message ID of the label to actually create. And then we would just basically be able to select the label we want to give it. So in this case, we could give it the customer support label. We hit test step, we'll get another success message. And then in our inbox, if we refresh, we will see that that just got labeled as customer support. So you could add on more functionality like that. And you could also down here create more sections. So we could have finance, you know, a logic built out for finance emails. We could have logic built out for all these other types of emails and um plug them into different knowledge bases as well. Okay. So the third one we're going to do is a LinkedIn content creator workflow. So, what we're going to do here is click on add first step, of course. And ideally, you know, in production, what this workflow would look like is a schedule trigger, you know. So, what you could do is basically say every day I want this thing to run at 7:00 a.m. That way, I'm always going to have a LinkedIn post ready for me at, you know, 7:30. I'll post it every single day. And if you wanted it to actually be automatic, you'd

have to flick this workflow from inactive to active. And, you know, now it says, um, your schedule trigger will now trigger executions on the schedule you have defined. So now it would be working, but for the sake of this video, we're going to turn that off and we are just going to be using a manual trigger just so we can show how this works. Um, but it's the same concept, right? It would just start the workflow. So what we're going to do from here is we're going to connect a Google sheet. So I'm going to grab a Google sheet node. I'm going to click on get rows and sheet and we have to create our credential once again. So we're going to create new credential. We're going to be able to do ooth to sign in, but we're going to have to go back to Google Cloud and we're going to have to grab a sheet and make sure that we have the Google Sheets API enabled. So, we'll come in here, we'll click enable, and now once this is good to go, we'll be able to sign in using OOTH 2. So, very similar to what we just had to do for Gmail in that previous workflow. But now, we can sign in. So, once again, choosing my email, allowing it to have access, and then we're connected successfully, and then giving this a good name. And now, what we can do is choose the document and the sheet that it's going to be pulling from. So, I'm going to show you. I have one called LinkedIn posts, and I only have one sheet, but let's show you the sheet real quick. So, LinkedIn posts, what we have is a topic, a status, and a content. And we're just basically going to be pulling in one row where the status equals to-do, and then we are going to um create the content, upload it back in right here, and then we're going to change the status to created. So, then this same row doesn't get pulled in every day. So, how this is going to work is that we're going to create a filter. So the first filter is going to be looking within the status column and it has to equal to-do. And if we click on test step, we should see that we're going to get like all of these items where there's a bunch of topics. But we don't want that. We only want to get the first row. So at the bottom here, add option. I'm going to say return only first matching row. Check that on. We'll test this again. And now we're only going to be getting that top row to create content on. Cool. So we have our first step here, which is just getting the content from the Google sheet. Now, what we're going to do is we need to do some web search on this topic in order to create that content. So, I'm going to add a new node. This one's going to be called an HTTP request. So, we're going to be making a request to a specific API. And in this case, we're going to be using Tavly's API. So, go on over to tavly.com and create a free account. You're going to get a,000 searches for free per month. Okay, here we are in my account. I'm on the free researcher plan, which gives me a thousand free credits. And right here, I'm going to add an API key. We're going to name it, create a key, and we're going to copy this value. And so, you'll start to get to the point when you connect to different services, you always need to have some sort of like token or API key. But anyways, we're going to grab this in a sec. What we need to do now is go to the documentation that we see right here. We're going to click on API reference. And now we have right here. This is going to be the API that we need to use in order to search the web. So, I'm not going to really dive into like everything about HTTP requests right now. I'm just going to show you the simple way that we can get this set up. So first thing that we're going to do is we obviously see that we're using an endpoint called Tavali search and we can see it's a post request which is different than like a git request and we have all these different things we need to configure and it can be confusing. So all we want to do is on the top right we see this curl command. We're going to click on the copy button. We're going to go back into our NEN, hit import curl, paste in the curl command, hit import, and now the whole node magically just basically filled in itself. So that's really awesome. And now we can sort of break down what's going on. So for every HTTP request, you have to have some sort of method. Typically, when you're sending over data to a service, which in this case, we're

going to be sending over data to Tavali. It's going to search the web and then bring data back to us. That's a post request because we're sending over body data. If we were just like kind of trying to hit an and if we were just trying to access like you know um bestbuy.com and we just wanted to scrape the information that could just be a simple git request because we're not sending anything over anyways then we're going to have some sort of base URL and endpoint which is right here. The base URL we're hitting is api.com/tavaly and then the endpoint we're hitting is slash search. So back in the documentation you can see right here we have slash search but if we were doing like an extract we would do slash extract. So that's how you can kind of see the difference with the endpoints. And then we have a few more things to configure. The first one of course is our authorization. So in this case, we're doing it through a header parameter. As you can see right here, the curl command set it up. Basically all we have to do is replace this um token with our API key from Tavi. So I'm going to go back here, copy that key in N. I'm going to get rid of token and just make sure that you have a space after the word bearer. And then you can paste in your token. And now we are connected to Tavi. But we need to configure our request before we send it off. So right here are the parameters within our body request. And I'm not going to dive too deep into it. You can go to the documentation if you want to understand like you know the main thing really is the query which is what we're searching for. But we have other things like the topic. It can be general or news. We have search depth. We have max results. We have a time range. We have all this kind of stuff. Right now I'm just going to leave everything here as default. We're only going to be getting one result. And we're going to be doing a general topic. We're going to be doing basic search. But right now, if we hit test step, we should see that this is going to work. But it's going to be searching for who is Leo Messi. And here's sort of like the answer we get back as well as a URL. So this is an actual website we could go to about Lionel Messi and then some content from that website. Right? So we are going to change this to an expression so that we can put a variable in here rather than just a static hard-coded who is Leo Messi. We'll delete that query. And all we're going to do is just pull in our topic. So, I'm just going to simply pull in the topic of AI image generation. Obviously, it's a variable right here, but this is the result. And then we're going to test step. And this should basically pull back an article about AI image generation. And you know, so here is a deep AI um link. We'll go to it. And we can see this is an AI image generator. So maybe this isn't exactly what we're looking for. What we could do is basically just say like, you know, we could hardcode in search the web for. And now it's going to be saying search the web for AI image generation. We could come in here and say yeah actually you know let's get three results not just one. And then now we could test that step and we're going to be getting a little bit different of a search result. Um AI image generation uses text descriptions to create unique visuals. And then now you can see we got three different URLs rather than just one. Anyways, so that's our web search. And now that we have a web search based on our defined topic, we just need to write that content. So I'm going to click on the plus. I'm going to grab an AI agent. And once again, we're not giving it the connected chat trigger node to look at. That's nowhere to be found. We're going to feed in the research that was just done by Tavi. So, I'm going to click on expression to open this up. I'm going to say article one with a colon and I'm just going to drag in the content from article one. I'm going to say article 2 with a colon and just drag in the content from article 2. And then I'm going to say article 3 colon and just drag in the content from the third article. So now it's looking at all three article contents. And now we just need to give it a system prompt on how to write a LinkedIn post. So open this up. Click on add option. Click on system message. And now let's give it a prompt about turning these three articles into a LinkedIn post. Okay. So I'm heading over to my custom GPT for

prompt architect. If you want to access this, you can get it for free by joining my free school community. Um you'll join that. It's linked in the description and then you can just search for prompt architect and you should find the link. Anyways, real quick, it's just asking for some clarification questions. So, anyways, I'm just shooting off a quick reply and now it should basically be generating our system prompt for us. So, I'll check in when this is done. Okay, so here is the system prompt. I am going to just paste it in here and I'm just going to, you know, disclaimer, this is not perfect at all. Like, I don't even want this tool section at all because we don't have a tool hooked up to this agent. Um, we're obviously just going to give it a chat model real quick. So, in this case, what I'm going to do is I'm going to use Claude 3.5 Sonnet just because I really like the way that it writes content. So, I'm using Claude through Open Router. And now, let's give it a run and we'll just see what the output looks like. Um, I'll just click into here while it's running and we should see that it's going to read those articles and then we'll get some sort of LinkedIn post back. Okay, so here it is. The creative revolution is here and it's AI powered. Gone are the days of hiring expensive designers or struggling with complex software. Today's entrepreneurs can transform ideas into a stunning visuals instantly using AI image generators. So, as you can see, we have a few emojis. We have some relevant hashtags. And then at the end, it also said this post, you know, it kind of explains why it made this post. We could easily get rid of that. If all we want is the content, we would just have to throw that in the system prompt. But now that we have the post that we want, all we have to do is send it back into our Google sheet and update that it was actually made. So, we're going to grab another sheets node. We're going to do update row and sheet. And this one's a little different. It's not just um grabbing stuff from a row. We're trying to update stuff. So, we have to say what document we want, what sheet we want. But now, it's asking us what column do we want to match on. So, basically, I'm going to choose topic. And all we have to do is go all the way back down to the sheet. We're going to choose the topic and drag it in right here. Which is basically saying, okay, when this node gets called, whenever the topic equals AI image generation, which is a variable, obviously, whatever whatever topic triggered the workflow is what's going to pop up here. We're going to update that status. So, back in the sheets, we can see that the status is currently to-do, and we need to change it to created in order for it to go green. So, I'm just going to type in created, and obviously, you have to spell this correctly the same way you have it in your Google Sheets. And then for the content, all I'm going to do is we're just going to drag in the output of the AI agent. And as you can see, it's going to be spitting out the result. And now if I hit test step and we go back into the sheet, we'll basically watch this change. Now it's created. And now we have the content of our LinkedIn post as well with some justification for why it created the post like this. And so like I said, you could basically have this be some sort of, you know, LinkedIn content making machine where every day it's going to run at 7:00 a.m. It's going to give you a post. And then what you could do also is you can automate this part of it where you're basically having it create a few new rows every day if you give it a certain sort of like general topic to create topics on and then every day you can just have more and more pumping out. So that is going to do it for our third and final workflow. Okay, so that's going to do it for this video. I hope that it was helpful. You know, obviously we connected to a ton of different credentials and a ton of different services. We even made a HTTP request to an API called Tavali. Now, if you found this helpful and you liked this sort of live step-by-step style and you're also looking to accelerate your journey with NAN and AI automations, I would definitely recommend to check out my paid community. The link for that is down in the description. Okay, so hopefully those three workflows taught you a ton about connecting to different services and setting up credentials. Now, I'm actually going to throw

in one more bonus step-by-step build, which is actually one that I shared in my paid community a while back, and I wanted to bring it to you guys now. So, definitely finish out this course, and if you're still looking for some more and you like the way I teach, then feel free to check out the paid community. The link for that's down in the description. We've got a course in there that's even more comprehensive than what you're watching right now on YouTube. We've also got a great community of people that are using Niten to build AI automations every single day. So, I'd love to see you guys in that community. But, let's move ahead and build out this bonus workflow. Hey guys. So, today I wanted to do a step by step of an invoice workflow. And this is because there's different ways to approach stuff like this, right? There's the conversation of OCR. There's a conversation of maybe extracting text from PDFs. Um, there's the conversation of if you're always getting invoices in the exact same format, you probably don't need AI because you could use like a code node to extract the different parameters and then push that through. So, that's kind of stuff we're going to talk about today. And I I haven't showed this one on YouTube. It's not like a YouTube build, but it's not an agent. It's an AI powered workflow. And I also wanted to talk about like just the foundational elements of connecting pieces, thinking about the workflow. So, what we're going to do first actually is we're going to hop into Excalar real quick. and I'm going to create a new one. And we're just going to real quickly wireframe out what we're doing. So, first thing we're going to draw out here is the trigger. So, we'll make this one yellow. We'll call this the trigger. And what this is going to be is invoice. Sorry, we're going to do new invoice. And this is going to be um Google Drive. So the Google Drive node, it's going to be triggering the workflow and it's going to be when a new invoice gets dropped into um the folder that we're watching. So that's the trigger. From there, and like I said, this is going to be a pretty simple workflow. From there, what we're going to do is basically it's going to be a PDF. So the first thing to understand is actually let me just put Google Drive over here. So the first thing to understand from here is um you know what what do the invoices look like? These are the questions that we're going to have. So the first one's what do the invoices look like? Um because that determines what happens next. So if they are PDFs that happen every single time and they're always in the same format, then next we'd want to do okay well we can just kind of extract the text from this and then we can um use a code node to extract the information we need per each parameter. Now if it is a scanned invoice where it's maybe not as we're not maybe not as able to extract text from it or like turn it into a text doc, we'll probably have to do some OCR element. Um, but if it's PDF that's generated by a computer, so we can extract the text, but they're not going to come through the same every time, which is what we have in this case. I have two example invoices. So, we know we're overall we're looking for like business name, client name, invoice number, invoice date, due date, payment method, bank details, maybe stuff like that, right? But both of these are formatted very differently. They all have the same information, but they're formatted differently. So that's why we can't that's why we want to use an AI sort of information extractor node. Um so that's one of the main questions. The other ones we'd think about would be like you know where do they go? So once we get them where do they go? Um you know the frequency of them coming in and then also like really any other action. So bas building off of where do they go? It's also like what actions will we take? So, does that mean um are we just going to throw it in a CRM or are we just going to throw it in a CRM or maybe a database or are we also going to like send them an automated follow-up based on you know the email that we extract from it and say, "Hey, we received your invoice. Thanks." So, like what does that look like? So, those are the questions we were initially going to ask. Um and then that helps us pretty much plan out the next steps. So because we figured out um basically we figured out

that we want to extract the same like x amount of information the x the same x fields. So because we found out we want to extract the same X fields but the formats may not be [Music] consistent. We will use an AI information extractor. Um that is just a long sentence. So shorten this up a little bit or sorry make it smaller a little bit. Okay so we have that. Um the invoice will be um like updated to our Google sheet which will just be like a database of invoice which I'll just call invoice database and then a follow-up email can be sent or no not a follow-up email we'll just say an email, an internal email will be sent. So, an email will be sent to the internal billing team. Okay, so this is what we've got, right? We have our questions. We've kind of answered the questions. So, now we know what the rest of the flow is going to look like. We already know this is not going to be an agent. It's going to be a workflow. So, what we're going to do is we're going to add another node right here, which is going to be, you know, PDF comes in. And what we want to do is we want to extract the text from that PDF. Um let's make this text smaller. So we're going to extract the text. And we'll do this by using a um extract text node. Okay, cool. Now once we have the text extracted, what do we need to do? We need to um just moving over these initial questions. So we have the text extracted. Extracted. What comes next? What comes next is we need to um like decide on the fields to extract. And how do we get this? We get this from our invoice database. So let's quickly set up the invoice database. I'm going to do this by opening up a Google sheet which we are just going to call the oops invoice DB. So now we need to figure out what we actually want to put into our invoice DB. So first thing we'll do is um you know we're pretending that our business is called Green Grass. So we don't need that. We don't need the business information. We really just need the client information. So invoice number will be the first thing we want. So, we're just setting up our database here. So, invoice number. From there, we want to get client name, client address, client email, client phone. So, client name, client email, [Music] oops, client name, client email, client address, and then we want client phone. Okay, so we have those five things. And let's see what else we want. probably the amount. So, we'll just do total amount due, total amount, um, and due date. Invoice date and due date. Okay. Invoice date and due date. Okay. So, we have these, what are these? Eight. Eight fields. And I'm just going to change these colors so it looks visually better for us. So, here are the fields we have and this is what we want to extract from every single invoice that we are going to receive. Cool. So, we know we have these eight things. I'm just going to actually No, we're fine. So, we have our eight fields to extract um and then they're going to be pushed to invoice DB and then we'll set up the once we have these fields we can basically um create our email. So this is going to be an AI node that's going to info extract. So it's going to extract the eight fields that we have over here. So we're going to send the data into there and it's going to extract those fields. Once we extract those fields, we don't probably need to set the data because because coming out of this will basically be those eight fields. So um you know every time what's going to happen is actually sorry let me add another node here so we can connect these. So what's going to come out of here is one item which will be the one PDF and then what's coming out of here will be eight items every time. So that's what we've got. We could also want to think about maybe if two invoices get dropped in at the same time. How do we want to handle that loop or just push through? But we won't worry about that yet. So we've got one item coming in here. the node that's extracting the info will push out the eight items and the eight items only. And then what we can do from there is update invoice DB and then from there we can also and this could be like out of here we do two things or it could be like a uh sequential if that makes sense. So, well, what else we know we need to do is we know that we also need to email billing team. And so, what I was saying there is we could either have it like this where at the same time it

branches off and it does those two things. And it really doesn't matter the order because they're both going to happen either way. So, for now, to keep the flow simple, we'll just do this or we're going to email the billing team. Um, and what's going to happen is, you know, essentially because this is internal, because this is internal, we already know like the billing email. So, you know, billing@acample.com. This is what we're going to feed in because we already know the billing email. We don't have to extract this from anywhere. Um, so we have all the info we need. We will what else do we need to feed in here? So some of these some of these fields we'll have to filter in. So some of the extracted fields because like we want to say hey you know we got this invoice on this date um to this client and it's due on this date. So, we'll have some of the extracted fields. We'll have a billing example and then potentially like potentially like previous or like an email template potentially like that's that's something we can think about or we can just prompt an agent to send off the email. So, yeah. Okay. Okay. So what we want to do here is actually this. What we need to do is the email has to be generated somewhere. So before we feed into an emailing team node and let me actually change this. So we're going to have green nodes be AI and then blue nodes are going to be not AI. So we're going to get another AI node right here which is going to be craft email. So we'll connect these pieces once again. Um, and so I hope this I hope you guys can see like this is me trying to figure out the workflow before we get into nit because then we can just plug in these pieces, right? Um, and so I didn't even think about this. I mean, obviously we would have got in there and end and realized, okay, well, we need an email to actually configure these next fields, but that's just how it works, right? So anyways, this stuff is actually hooked up to the wrong place. We need this to be hooked up over here to the craft email tool. So, email template will also be hooked up here. And then the billing example will be hooked up. This is the No, this will still go here because that's actually the email team or the email node. So, email node, send email node, which is an action and we'll be feeding in this as well as the actual email. So the email that's written by AI will be fed in. And I think that ends the process, right? So we'll just add a quick Oops. We'll just add a quick yellow note over here. And I always my my colors always change, but just trying to keep things consistent. Like in here, we're just saying, okay, the process is going to end now. Okay, so this is our workflow, right? New invoice PDF comes through. We want to extract the text. We're using an extract text node which is just going to be a static extract from PDF PDF or convert PDF to text file type of thing. We'll get one item sent to an AI node to extract the eight fields we need. The eight items will be fed into the next node which is going to update our Google sheet. Um and I'll just also signify here this is going to be a Google sheet because it's important to understand the integrations and like who's involved in each process. So this is going to be AI. This is going to be AI and that's going to be an extract node. This is going to be a Gmail node and then we have the process end. Cool. So this is our wireframe. Now we can get into naden and start building out. We can see that this is a very very sequential flow. We don't need an agent. We just need two AI nodes here. So let us get into niten and start building this thing. So um we know we know what's starting this process which is which is a trigger. So, I'm going to grab a Google Drive trigger. We're going to do um on changes to a specific file or no, no, specific folder, sorry. Changes involving a specific folder. We're going to choose our folder, which is going to be the projects folder, and we're going to be watching for a file created. So, we've got our ABC Tech Solutions. I'm going to download this as a PDF real quick. So, download as a PDF. I'm going to go to my projects folder in the drive, and I'm going to drag this guy in here. Um, there it is. Okay, so there's our PDF. We'll come in here and we'll hit fetch test event. So, we should be getting our PDF. Okay, nice. We will just make sure it's the right one. So, we we should see a ABC

Tech Solutions Invoice. Cool. So, I'm going to pin this data just so we have it here. So, just for reference, pinning data, all it does is just keeps it here. So, if we were to refresh this this page, we'll still have our pinned data, which is that PDF to play with. But if we would have not pinned it, then we would have had to fetch test event once again. So not a huge deal with something like this, but if you're maybe doing web hooks or API calls, you don't want to have to do it every time. So you can pin that data. Um or like an output of an AI node if you don't want to have to rerun the AI. But anyway, so we have our our PDF. We know next based on our wireframe. And let me just call this um invoice flow wireframe. So we know next is we need to extract text. So perfect. We'll get right into NADN. We'll click on next and we will do an extract from file. So let's see. We want to extract from PDF. And although what do we have here? We don't have any binary. So we were on the right track here, but we forgot that in order to we get the we get the PDF file ID, but we don't actually have it. So what we need to do here first is um basically download the file because we need the binary to then feed that into the extract text node. So we need the binary. So, sorry if that's like I mean really small, but basically in order to extract the text, we need to download the file first to get the binary and then we can um actually do that. So, little little thing we missed in the wireframe, but not a huge deal, right? So, we're going to extend this one off. We're going to do a Google Drive node once again, and we're going to look at download file. So, now we can say, okay, we're downloading a file. Um, we can choose from a list, but this has to be dynamic because it's going to be based on that new trigger every time. So, I'm going to do by ID. And now on the lefth hand side, we can look for the file ID. So, I'm going to switch to schema real quick. Um, so we can find the the ID of the file. We're just going to have to go through. So, we have a permissions ID right here. I don't think that's the right one. We have a spaces ID. I don't think that's the right one either. We're looking for an actual file ID. So, let's see. parents icon link, thumbnail link, and sometimes you just have to like find it and test things out, right? So, I feel like I probably have just skipped right over it. Otherwise, we'll just try out some of these other IDs. Maybe it is this one. Yeah, I think Okay, sorry. I think it is this one because we see the name is right here and the ID is right here. So, we'll try this. We're referencing that dynamically. We also see in here we could do a Google file conversion which basically says um you know if it's docs convert it to HTML if it's drawings convert it to that. If it's this convert it to that there's not a PDF one so we'll leave this off and we'll hit test step. So now we will see we got the invoice we can click view and this is exactly what we're looking at here with the invoice. So this is the correct one. Now since we have it in our binary data over here we have binary. Now we can extract it from the file. So um you know on the left is the inputs on the right is going to be our output. So we're extracting from PDF. We're looking in the input binary field called data which is right here. So I'll hit test step and now we have text. So here's the actual text right um the invoice the information we need and out of this is what we're going to pass over to extract. So let's go back to the wireframe. We have our text extracted. Now, what we want to do is extract um the specific eight fields that we need. So, hopping back into the workflow, we know that this is going to be an AI node. So, it's going to be an information extractor. We have to first of all classify we we know that one item is going in here and that's right here for us in the table, which is the actual text of the invoice. So, we can open this up and we can see this is the text of the invoice. We want to do it from attribute description. So, that's what it's looking for. So, we can add our eight attributes. So, we know there's going to be eight of them, right? So, we can create eight. But, let's just first of all go into our database to see what we want. So, the first one's invoice number. So, I'm going to copy this over here. Invoice number. And we just have to describe what that is. So, I'm just going to say the number of the invoice. And this is required. We're

going to make them all required. So, number of the invoice. Then we have the client name. Paste that in here. Um, these should all be pretty self um explanatory. So the name of the client we're going to make it required client email. So this is going to be a little bit repetitive but the email of the client and let me just quickly copy this for the next two client address. So there's client address and we're going to say the address of the client required. And then what's the last one here? Client phone. Paste that in there, which is obviously going to be the phone number of the client. And here we can say, is this going to be a string or is it going to be a number? I'm going to leave it right now as a string just because over here on the left you can see the phone. We have parenthesis in there. And maybe we want the format to come over with the parenthesis and the little hyphen. So let's leave it as a string for now. We can always test and we'll come back. But client phone, we're going to leave that. We have total amount. Same reason here. I'm going to leave this one as a string because I want to keep the dollar sign when we send it over to sheets and we'll see how it comes over. But the total amount of the invoice required. What's coming next is invoice date and due date. So, invoice date and due date, we can say these are going to be dates. So, we're changing the var the data type here. They're both required. And the date the invoice was sent. And then we're going to say the date the invoice is due. So, we're going to make sure this works. If we need to, we can get in here and make these descriptions more descriptive. But for now, we're good. We'll see if we have any options. You're an expert extraction algorithm. Only extracts relevant information from the text. If you do not know the value of the attribute to extract, you may omit the attributes value. So, we'll just leave that as is. Um, and we'll hit test step. It's going to be looking at this text. And of course, we're using AI, so we have to connect a chat model. So, this will also alter the performance. Right now, we're going to go with a Google Gemini 20 flash. See if that's powerful enough. I think it should be. And then, we're going to hit play once again. So, now it's going to be extracting information using AI. And what's great about this is that we already get everything out here in its own item. So, it's really easy to map this now into our Google sheet. So, let's make sure this is all correct. Um, invoice number. That looks good. I'm going to open up the actual one. Yep. Client name ABC. Yep. Client email finance at ABC Tech. Yep. Address and phone. We have address and phone. Perfect. We have total amount is 141 175. 14175. We have um March 8th and March 22nd. If we go back up here, March 8th, March 22nd. Perfect. So, that one extracted it. Well, and um okay, so we have one item coming out, but technically there's eight like properties in there. So, anyways, let's go back to our our uh wireframe. So, after we extracted the eight items, what do we do next? We're going to put them into our Google Sheet um database. So, what we know is we're going to grab a Google Sheets. We're going to do an append row because we're adding a row. Um, we already have a credential selected. So, hopefully we can choose our invoice database. It's just going to be the first sheet, sheet one. And now what happens is we have to map the columns. So, you can see these are dragable. We can grab each one. If I go to schema, it's a little more apparent. So, we have these eight items. And it's going to be really easy now that we use an information extractor because we can just map, you know, invoice number to invoice number, client name, client name, email, email. And it's referencing these variables because every time after we do our information extractor, they're going to be coming out as JSON.output and then invoice number. And then for client name, JSON.output client name. So we have these dynamic variables that will happen every single time. And obviously I'll show this when we do another example, but we can keep mapping everything in. And we also did it in that order. So it's really really easy to do. We're just dragging and dropping and we are finished. Cool. So if I hit test step here, this is going to give us a message that says like here are the fields

basically. So there are the fields. They're mapped correctly. Come into the sheets. We now have automatically gotten this updated in our invoice database. And um that's that. So let me just change some of these nodes. So this is going to be update database. Um this is information extractor extract from file. I'm just going to say this is download binary. So now we know what's going on in each step. And we'll go back to the wireframe real quick. What happens after we update the database? Now we need to craft the email. And this is going to be using AI. And what's going to go into this is some of the extracted fields and maybe an email template. What we're going to do more realistically is just a system prompt. So back into nitn let's add a um open AI message and model node. So what we're going to do is we're going to choose our model to talk to. In this case we'll go 40 mini. It should be powerful enough. And now we're going to set up our system prompt and our user prompt. So at this point if you don't understand the difference the system prompt is the instructions. So we're telling this node how to behave. So first I'm going to change this node name to create email because that's like obviously what's going on keeping you organized. And now how do we explain to this node what its role is? So you are an email expert. You will receive let me actually just open this up. You will receive um invoice information. Your job is to notify the internal billing team that um an invoice was received. Receive/s sent. Okay. So, honestly, I'm going to leave it at that for now. It's really simple. If we wanted to, we can get in here and change the prompting as far as like here is the format. Here is the way you should be doing it. One thing I like to do is I like to say, you know, this is like your overview. And then if we need to get more granular, we can give it different sections like output or rules or anything like that. I'm also going to say you are an email expert for green grass corp named [Music] um named um Greeny. Okay, so we have Greenie from Green Grass Corp. That's our email expert that's going to email the billing team every time this workflow happens. So that's the overview. Now in the user prompt, think of this as like when you're talking to chatbt. So obviously I had chatbt create these invoices chatgbt this when we say hello that's a user message because this is an interact like an an interaction and it's going to change every time. But behind the scenes in this chatbt openai has a system prompt in here that's basically like you're a helpful assistant. You help you know users answer questions. So this window right here that we type in is our user message and behind the scenes telling the node how to act is our system prompt. Cool. So in here I like to have dynamic information go into the user message while I like to have static information in the actual system prompt. So except for maybe the except the exception usually of like giving it the current time and day because that's an expression. So anyways, let's change this to an expression. Let's make this full screen. We are going to be giving it the invoice information that it needs to write the email because that's what it that's what it's expecting. In the system prompt, we said you will receive invoice information. So, first thing is going to be invoice number. We are going to grab invoice number and just drag it in. We're going to grab client name and just drag it in. So, it's going to dynamically get these different things every time, right? So, let's say maybe it doesn't even we don't need client email. Okay, maybe we do. We want client email. Um, so we'll give it that. But the billing team right now doesn't need the address or phone. Let's just say that. But it does want we do want them to know the total amount of that invoice. And we definitely want them to know the invoice date and the invoice due date. So we can we can now drag in these two things. So this was us just being able to customize what the AI node sees. Just keep in mind if we don't drag anything in here, even if it's all on the input, the AI node doesn't see any of it. So let's hit test step and we'll see the type of email we get. We're going to have to make some changes. I already know because you know we have to separate everything. But what it did is it created a subject which is new invoice received and

then the invoice number. Dear billing team, I hope this message finds you well. We've received an invoice that requires your attention and then it lists out some information and then it also signs off Greeny Green Grass Corp. So, first thing we want to do is um if we go back to our wireframe, what we have to send in, and we didn't document this well enough actually, but what goes into here is a um you know, in order to send an email, we need a two, we need a subject, and we need the email body. So, that those are the three things we need. the two is coming from here. So, we know that. And the subject and email are going to come from the um craft email node. So, we have the the two and then actually I'm going to move this up here. So, now we can just see where we're getting all of our pieces from. So, the two is coming from internal knowledge. This can be hardcoded, but the subject and email are going to be dynamic from the AI note. Cool. So what we want to do now is we're going to say output and we're going to tell it how to output information. So output output the following parameters separately and we're just going to say subject and email. So now it should be outputting two parameters separately, but it's not going to because even though it says here's the subject and then it gives us a subject and then it says here's the email and gives us an email, they're still in one field. Meaning if we hook up another node which would be a Gmail send email as we know which is going to be where right here. Okay, so now this is the next node. Here's the fields we need. But as you can see coming out of the create email AI node, we have this whole parameter called content which has the subject and the email. And we need to get these split up so that we can drag one into the subject and one into the two. Right? So first of all, I'm just making these expressions just so we can drag stuff in later. Um, and so that's what we need to do. And our fix there is we come into here and we just check this switch that says output content is JSON, which will then will rerun. And now we'll get subject and body. Subject and email in two different fields right here. We can see which is awesome because then we can open up our send email node. We can grab our subject. It's going to be dynamic. And we can grab our email. It's going to be dynamic. Perfect. We're going to change this to text. And we're going to add an option down here. And we're just going to say append nadn attribution and turn that off because we just don't want to see the message at the bottom that says this was sent by nadn. And if we go back to our wireframe wherever that is over here, we know that this is the email that's going to be coming through or we're going to be sending to every time because we're sending internally. So we can put that right in here, not as a variable. Every time this is going to be sending to billing@acample.com. So this really could be fixed. It doesn't have to be an expression. Cool. So we will now hit test step and we can see that we got this this email sent. So let me open up a new tab. Let me go into whoa into our Gmail. I will go to the sent items and we will see we just got this billing email. So obviously it was a fake email but this is what it looks like. We've received a new invoice from ABC Tech. Please find the details below. We got invoice number, client name, client email, total amount, total invoice date, due date. Please process these this invoice accordingly. So that's perfect. Um, we could also, if we wanted to, we could prompt it a little bit differently to say, you know, like this has been updated within the database and, um, you can check it out here. So, let's do that real quick. What we're going to do is we're going to say because we've already updated the database, I'm going to come into our Google sheet. I'm going to copy this link and I'm going to we're basically going to bake this into the email. So, I'm going to say we're going to give it a section called email. Inform the billing team of the invoice. let them know we have also updated this in the invoice database and they can view it here and we'll just give them this link to that Google doc. So every time they'll just be able to send that over. So I'm going to hit test up. We should see a new email over here which is going to include that link I hope. So there's the link. We'll run

this email tool again to send a new email. Hop back into Google Gmail. We got a new one. And now we can see we have this link. So you can view it here. We've already updated this in the invoice database. We click on the link. And now we have our database as well. So cool. Now let's say at this point we're happy with our prompting. We're happy with the email. This is done. If we go back to the wireframe, the email is the last node. So um maybe just to make it look consistent, we will just add over something here that just says nothing. And now we know that the process is done because nothing to do. So this is basically like this is what we wireframed out. So we know that we're happy with this process. We understand what's going on. Um but now let's unpin this data real quick and let's drop in another invoice to make sure that even though it's formatted differently. So this XYZ formatted differently, but the AI should still be able to extract all the information that we need. So I'm going to come in here and download this one as a PDF. We have it right there. I'm going to drop it into our Google Drive. So, we have XYZ Enterprises now. Come back into the workflow and we'll hit test fetch test event. Let's just make sure this is the right one. So, XYZ Enterprises. Nice. And I'm just going to hit test workflow and we'll watch it download, extract, get the information, update the database, create the email, send the email, and then nothing else should happen after that. So, boom, we're done. Let's click into our email. Here we have our new invoice received. So it updated differently like the subjects dynamic because it was from XYZ a different invoice number. As you remember the ABC one, it started with TH AI and this one starts with INV. So that's why the subject is different. Dear billing team, we have received a new invoice from XYZ Enterprises. Please find the details below. There's the number, the name, all this kind of information. Um the total amount was 13856. Let's go make sure that's right. Total amount 13856. Um March 8th March 22nd once again. Is that correct? March 8th March 22nd. Nice. And finance XYZ XYZ. Perfect. Okay. The invoice has been updated in the database. You can view it here. So let's click on that link. Nice. We got our information populated into the spreadsheet. As you can see, it all looks correct to me as well. Our strings are coming through nice and our dates are coming through nice. So I'm going to leave it as is. Now, keep in mind because these are technically coming through as strings, um, that's fine for phone, but Google Sheets automatically made these numbers, I believe. So, if we wanted to, we could sum these up because they're numbers. Perfect. Okay, cool. So, that's how that works, right? Um, that's the email. We wireframed it out. We tested it with two different types of invoices. They weren't consistent formatting, which means we couldn't probably have used a code node, but the AI is able to read this and extract it. As you can see right here, we got the same eight items extracted that we were looking for. So, that's perfect. Um, cool. So, going to call this one here. I will yeah I will attach the actual flow and I will attach the just a picture of this wireframe I suppose in this um post and by now you guys have already seen that I'm sure but yeah I hope this was helpful um the whole process of like the way that I approached and I know this was a 35minut build so it's not like the same as like building something more complex but as far as like a general workflow you know this is a pretty solid solid one to get started with. Um it it shows elements of using AI within a simple workflow that's going to be sequential and it shows like you know the way we have to reference our variables and how we have to drag things in and um obviously the component of like wireframing out in the beginning to understand the full flow at least 80% 85% of the full flow before you get in there. So cool. Hope you guys enjoy this one and I will see you guys in the community. Thanks. All right. All right, I hope you guys enjoyed those step-by-step builds. Hopefully, right now, you're feeling like you're in a really good spot with Naden and everything starting to piece together. This next video we're going to move into is about APIs because in order to really get more advanced with our workflows

and our AI agents, we have to understand the most important thing, which is APIs. They let our Nin workflows connect to anything that you actually want to use. So, it's really important to understand how to set up. And when you understand it, the possibilities are endless. And it's really not even that difficult. So, let's break it down. If you're building AI agents, but you don't really understand what an API is or how to use them, don't worry. You're not alone. I was in that exact same spot not too long ago. I'm not a programmer. I don't know how to code, but I've been teaching tens of thousands of people how to build real AI systems. And what changed the game for me was when I understood how to use APIs. So, in this video, I'm going to break it down as simple as possible, no technical jargon, and by the end, you'll be confidently setting up API calls within your own Agentic workflows. Let's make this easy. So the purpose of this video is to understand how to set up your own requests so you can access any API because that's where the power truly comes in. And before we get into NDN and we set up a couple live examples and I show you guys my thought process when I'm setting up these API calls. First I thought it would just be important to understand what an API really is. And APIs are so so powerful because let's say we're building agents within NADN. Basically we can only do things within NADN's environment unless we use an API to access some sort of server. So whether that's like a Gmail or a HubSpot or Air Table, whatever we want to access that's outside of Niden's own environment, we have to use an API call to do so. And so that's why at the end of this video, when you completely understand how to set up any API call you need, it's going to be a complete gamecher for your workflows and it's also going to unlock pretty much unlimited possibilities. All right, now that we understand why APIs are important, let's talk about what they actually do. So API stands for application programming interface. And at the highest level in the most simple terms, it's just a way for two systems to talk to each other. So NAND and whatever other system we want to use in our automations. So keeping it limited to us, it's our NAN AI agent and whatever we want it to interact with. Okay, so I said we're going to make this as simple as possible. So let's do it. What we have here is just a scenario where we go to a restaurant. So this is us right here. And what we do is we sit down and we look at the menu and we look at what food that the restaurant has to offer. And then when we're ready to order, we don't talk directly to the kitchen or the chefs in the kitchen. We talk to the waiter. So we'd basically look at the menu. We'd understand what we want. Then we would talk to the waiter and say, "Hey, I want the chicken parm." The waiter would then take our order and deliver it to the kitchen. And after the kitchen sees the request that we made and they understand, okay, this person wants chicken parm. I'm going to grab the chicken parm, not the salmon. And then we're basically going to feed this back down the line through the waiter all the way back to the person who ordered it in the first place. And so that's how you can see we use an HTTP request to talk to the API endpoint and receive the data that we want. And so now a little bit more of a technical example of how this works in NADN. Okay, so here is our AI agent. And when it wants to interact with the service, it first has to look at that services API documentation to see what is offered. Once we understand that, we'll read that and we'll be ready to make our request and we will make that request using an HTTP request. From there, that HTTP request will take our information and send it to the API endpoint. the endpoint will look at what we ordered and it will say, "Okay, this person wants this data. So, I'm going to go grab that. I'm going to send it back, send it back to the HTTP request." And then the HTTP request is actually what delivers us back the data that we asked and we know that it was available because we had to look at the API documentation first. So, I hope that helps. I think that looking at it visually makes a lot more sense, especially when you hear, you know, HTTP API endpoint, all this kind of stuff. But really, it's just going to be this

simple. So now let me show an example of what actually this looks like in Naden and when you would use one and when you wouldn't need to use one. So here we have two examples where we're accessing a service called open weather map which basically just lets us grab the weather data from anywhere in the world. And so on the left what we're doing is we're using open weather's native integration within nadn. And so what I mean by native integration is just that when we go into nadn and we click on the plus button to add an app and we want to see like you know the different integrations. It has air tableable it has affinity it has airtop it has all these AWS things. It has a ton of native integrations and all that a native integration is is an HTTP request but it's just like wrapped up nicely in a UI for us to basically fill in different parameters. And so once you realize that it really clears everything up because the only time you actually need to use an HTTP request is if the service you want to use is not listed in this list of all the native integrations. Anyways, let me show you what I mean by that. So like I said on the left we have Open Weather Maps native integration. So basically what we're doing here is we're sending over, okay, I'm using open weather map. I'm going to put in the latitude and the longitude of the city that I'm looking for. And as you can see over here, what we get back is Chicago as well as a bunch of information about the current weather in Chicago. And so if you were to fill this out, it's super super intuitive, right? All you do is put in the Latin long, you choose your format as far as imperial or metric, and then you get back data. And that's the exact same thing we're doing over here where we use an HTTP request to talk to Open Weather's API endpoint. And so this is just looks a little more scary and intimidating because we have to set this up ourselves. But if we zoom in, we can see it's pretty simple. We're making a git request to open weather maps URL endpoint. And then we're putting over the lat and the long, which is basically the exact same from the one on the left. And then, as you can see, we get the same information back about Chicago, and then some weather information about Chicago. And so the purpose of that was just to show you guys that these native integrations, all we're doing is we're accessing some sort of API endpoint. it just looks simpler and easier and there's a nice user interface for us rather than setting everything up manually. Okay, so hopefully that's starting to make a little more sense. Let's move down here to the way that I think about setting up these HTTP requests, which is we're basically just setting up filters and making selections. All we're doing is we're saying, okay, I want to access server X. When I access server X, I need to tell it basically, what do I want from you? So, it's the same way when you're going to order some pizza. You have to first think about which pizza shop do I want to call? And then once you call them, it's like, okay, I need to actually order something. It has to be small, medium, large. It has to be pepperoni or cheese. You have to tell it what you want and then they will send you the data back that you asked for. So when we're setting these up, we basically have like five main things to look out for. The first one you have to do every time, which is a method. And the two most common are going to be a get or a post. Typically, a get is when you're just going to access an endpoint and you don't have to send over any information. You're just going to get something back. But a post is when you're going to send over certain parameters and certain data and say okay using this information send me back what I'm asking for. The great news is and I'll show you later when we get into any end to actually do a live example. It'll always tell you to say you know is this a get or a post. Then the next thing is the endpoint. You have to tell it like which website or you know which endpoint you want to actually access which URL. From there we have three different parameters to set up. And also just realize that this one should say body parameters. But this used to be the most confusing part to me, but it's really not too bad at all. So, let's break it down. So, keep in mind when we're looking at that menu, that API

documentation, it's always going to basically tell us, okay, here are your query parameters, here are your header parameters, and here are your body parameters. So, as long as you understand how to read the documentation, you'll be just fine. But typically, the difference here is that when you're setting up query parameters, this is basically just saying a few filters. So if you search pizza into Google, it'll come google.com/arch, which would be Google's endpoint. And then we would have a question mark and then Q equals and then a bunch of different filters. So as you can see right here, the first filter is just Q equals pizza. And the Q is, you know, it stands for query parameters. And you don't even have to understand that. That's just me showing you kind of like a real example of how that works. From there, we have to set up a header parameter, which is pretty much always going to exist. And I basically just think of header parameters as, you know, authorizing myself. So, usually when you're doing some sort of API where you have to pay, you have to get a unique API key and then you'll send that key. And if you don't put your key in, then you're not going to be able to get the data back. So, like if you're ordering a pizza and you don't give them your credit card information, they're not going to send you a pizza. And usually an API key is something you want to keep secret because let's say, you know, you put 10 bucks into some sort of API that's going to create images for you. If that key gets leaked, then anyone could use that key and could go create images for themselves for free, but they'd be running down your credits. And these can come in different forms, but I just wanted to show you a really common one is, you know, you'll have your key value pairs where you'll put authorization as the name and then in the value you'll put bearer space your API key or in the name you could just put API_key and then in the value you'd put your API key. But once again, the API documentation will tell you how to configure all this. And then finally, the body parameters if you need to send something over to get something back. Let's say we're, you know, making an API call to our CRM and we want to get back information about John. We could send over something like name equals John. The server would then grab all records that have name equal John and then it would send them back to you. So those are basically like the five main things to look out for when you're reading through API documentation and setting up your HTTP request. But the beautiful thing about living in 2025 is that we now have the most beautiful thing in the world, which is a curl command. And so what a curl command is is it lets you hit copy and then you basically can just import that curl into nadn and it will pretty much set up the request for you. Then at that point it really is just like putting in your own API key and tweaking a few things if you want to. So let's take a look at this curl statement for a service called Tavi. As you can see that's the endpoint is api.tavi.com. All this basically does is it lets you search the internet. So you can see here this curl statement tells us pretty much everything we need to know to use this. So it's telling us that it's going to be a post. It's showing us the API endpoint that we're going to be accessing. It shows us how to set up our header. So that's going to be authorization and then it's going to be bearer space API token. It's basically just telling us that we're going to get this back in JSON format. And then you can see all of these different key value pairs right here in the data section. And these are basically going to be body parameters where we can say, you know, query is who is Leo Messi? So that's what we' be searching the internet for. We have topic equals general. We have search depth equals basic. So hopefully you can see all of these are just different filters where we can choose okay do we want you know do we want one max result or do we want four or do we have a time range or do we not. So this is really just at the end of the day. It's basically like ordering Door Dash because what we would have up here is you know like what actual restaurant do we want to order food from? We would put in our credit card information. We would say do we want this to be delivered to us? Do we want to pick it up?

We would basically say you know do you want a cheeseburger? No pickles. No onions. Like what are the different things that you want to flip? Do you want a side? Do you want fries? Or do you want salad? Like, what do you want? And so once you get into this mindset where all I have to do is understand this documentation and just tweak these little things to get back what I want, it makes setting up API calls so much easier. And if another thing that kind of intimidates you is the aspect of JSON, it shouldn't because all it is is a key value pair like we kind of talked about. You know, this is JSON right here and you're going to send your body parameter over as JSON and you're also going to get back JSON. So the more and more you use it, you're going to recognize like how easy it is to set up. So anyways, I hope that that made sense and broke it down pretty simply. Now that we've seen like how it all works, it's going to be really valuable to get into niten. I'm going to open up a API documentation and we're just going to set up a few requests together and we'll see how it works. Okay, so here's the example. You know, I did open weather's native integration and then also open weather as an HTTP request. And you can see it was like basically the exact same thing. Um, so let's say that what we want to do is we want to use Perplexity, which if you guys don't know what Perplexity is, it is basically, you know, kind of similar to Chatbt, but it has really good like internet search and research. So let's say we wanted to use this and hook it up to an AI agent, so it can do web search for us. But as you can see, if I type in Perplexity, there's no native integration for Perplexity. So that basically signals to us, okay, we can only access Perplexity using an HTTP request. And real quick side note, if you're ever thinking to yourself, hm, I wonder if I can have my agent interact with blank. The answer is yes, if there's API documentation. And all you have to do typically to find out if there's API documentation is just come in here and be like, you know, Gmail API documentation. And then we can see Gmail API is a restful API, which means it has an API and we can use it within our automations. Anyways, getting back to this example of setting up a perplexity HTTP request. We have our HTTP request right here and it's left completely blank. So, as you can see, we have our method, we have our endpoint, we have query, header, and body parameters, but nothing has been set up yet. So, what we need to do is we would head over to Perplexity, as you can see right here. And at the bottom, there's this little thing called API. So, I'm going to click on that. And this opens up this little page. And so, what I have here is Perplexity's API. If I click on developer docs, and then right here, I have API reference, which is integrate the API into your workflows, which is exactly what we want to do. This page is where people might get confused and it looks a little bit intimidating, but hopefully this breakdown will show you how you can understand any API doc, especially if there's a curl command. So, all I'm going to do first of all is I'm just going to right away hit copy and make sure you know you're in curl. If you're in Python, it's not the same. So, click on curl. I copied this curl command and I'm going to come back into nit hit import curl and all I have to do is paste. Click import and basically what you're going to see is this HTTP request is going to get basically populated for us. So now we have the method has been changed to post. We have the correct URL which is the API endpoint which basically is telling this node, okay, we're going to use Perplexity's API. You can see that the curl had no query parameters. So that's left off. It did turn on headers which is basically just having us put our API key in there. And then of course we have the JSON body that we need to send over. Okay. So at this point what I would do is now that we have this set up, we know we just need to put in a few things of our own. So the first thing to tackle is how do we actually authorize ourselves into Perplexity? All right. All right. So, I'm back in Perplexity. I'm going to go to my account and click on settings. And then all I'm going to do is basically I need to find where I can get my API key. So, on the lefth hand side, if I go all the way down, I can see API keys. So, I'll click

on that. And all this is going to do is this shows my API key right here. So, I'll have to click on this, hit copy, come back into NN, and then all I'm going to do is I'm going to just delete where this says token, but I'm going to make sure to leave a space after bearer and hit paste. So now this basically authorizes ourselves to use Perplexity's endpoint. And now if we look down at the body request, we can see we have this thing set up for us already. So if I hit test step, this is real quick going to make a request over. It just hit Perplexity's endpoint. And as you can see, it came back with data. And what this did is it basically searched Perplexity for how many stars are there in our galaxy. And that's where right here we can see the Milky Way galaxy, which is our galaxy, is estimated to contain between 100 billion and 400 billion stars, blah blah blah. So we know basically okay if we want to change how this endpoint works what data we're going to get back this right here is where we would change our request and if we go back into the documentation we can see what else we have to set up. So the first thing to notice is that there's a few things that are required and then some things that are not. So right here we have you know authorization that's always required. The model is always required like which perplexity model are we going to use? The messages are always required. So this is basically a mix of a system message and a user message. So here the example is be precise and concise and then the user message is how many stars are there in the galaxy. So if I came back here and I said you know um be funny in your answer. So I'm basically telling this model how to act and then instead of how many stars are there in the galaxy I'm just going to say how long do cows live? And I'll make another request off to perplexity. So you can see what comes back is this longer content. So it's not being precise and concise and it says so you're wondering how long cows live. Well, let's move into the details. So, as you can see, it's being funny. Okay, back into the API documentation. We have a few other things that we could configure, but notice how these aren't required the same way that these ones are. So, we have max tokens. We could basically put in an integer and say how many tokens do you want to use at the maximum. We could change the temperature, which is, you know, like how random the response would be. And this one says, you know, it has to be between zero and two. And as we keep scrolling down, you can see that there's a ton of other little levers that we can just tweak a little bit to change the type of response that we get back from Plexity. And so once you start to read more and more API documentation, you can understand how you're really in control of what you get back from the server. And also you can see like, you know, sometimes you have to send over booleans, which is basically just true or false. Sometimes you can only send over numbers. Sometimes you can only send over strings. And sometimes it'll tell you, you know, like what will the this value default to? and also what are the only accepted values that you actually could fill out. So for example, if we go back to this temperature setting, we can see it has to be a number and if you don't fill that out, it's going to be 0.2. But we can also see that if you do fill this out, it has to be between zero or two. Otherwise, it's not going to work. Okay, cool. So that's basically how it works. We just set up an HTTP request and we change the system prompt and we change the user prompt and that's how we can customize this thing to work for us. And that's really cool as a node because we can set up, you know, a workflow to pass over some sort of variable into this request. So it searches the web for something different every time. But now let's say we want to give our agent access to this tool and the agent will decide what am I going to search the web for based on what the human asks me. So it's pretty much the exact same process. We'll click on add a tool and we're going to add an HTTP request tool only because Plexity doesn't have a native integration. And then once again you can see we have an import curl button. So if I click on this and I just import that same curl that we did last time once again it fills out this whole

thing for us. So we have post, we have the perplexity endpoint, we have our authorization bearer, but notice we have to put in our token once again. And so a cool little hack is let's say you know you're going to use perplexity a lot. Rather than having to go grab your API key every single time, what we can do is we can just send it over right here in the authentication tab. So let me show you what I mean by that. If I click into authentication, I can click on generic credential type. And then from here, I can basically choose, okay, is this a basic O, a bearer O, all this kind of stuff. A lot of times it's just going to be header off. So that's why we know right here we can click on header off. And as you can see, we know that because we're sending this over as a header parameter and we just did this earlier and it worked. So as you can see, I have header offs already set up. I probably already have a Plexity one set up right here. But I'm just going to go ahead and create a new one with you guys to show you how this works. So I just create a new header off and all we have to do is the exact same thing that we had down in the request that we just sent over which means in the name we're just going to type in authorization with a capital A and once again we can see in the API docs this is how you do it. So authorization and then we can see that the value has to be capital B bearer space API token. So I'm just going to come into here bearer space API token and then all I have to do is you know first of all name this so we can can save it and then if I hit save now every single time we want to use perplexity's endpoint we already have our credentials saved. So that's great and then we can turn off the headers down here because we don't need to send it over twice. So now all we have to do is change this body request a little bit just to make it more dynamic. So in order to make it dynamic the first thing we have to do is change this to an expression. Now, we can see that we can basically add a variable in here. And what we can do is we can add a variable that basically just tells the AI model, the AI agent, here is where you're going to send over your internet search query. And so we already know that all that that is is the user content right here. So if I delete this and basically if I do two curly braces and then within the curly braces I do a dollar sign and I type in from, I can grab a from AI function. And this from AI function just indicates to the AI agent I need to choose something to send over here. And you guys will see an example and it will make more sense. I also did a full video breaking this down. So if you want to see that, I'll tag it right up here. Anyways, as you can see, all we have to really do is enter in a key. So I'm just going to do, you know, two quotes and within the quote, I'm going to put in search term. And so now the agent will be reading this and say, okay, whenever the user interacts with me and I know I need to search the internet, I'm just going to fill this whole thing in with the search term. So, now that that's set up, I'm just going to change this request to um actually I'm just going to call it web search to make that super intuitive for the AI agent. And now what we're going to do is we are going to talk to the agent and see if it can actually search the web. Okay, so I'm asking the AI agent to search the web for the best movies. It's going to think about it. It's going to use this tool right here and then we'll basically get to go in there and we can see what it filled in in that search term placeholder that we gave it. So, first of all, the answer it gave us was IMDb, top 250, um, Rotten Tomatoes, all this kind of stuff, right? So, that's movie information that we just got from Perplexity. And what I can do is click into the tool and we can see in the top left it filled out search term with best movies. And we can even see that in action. If we come down to the body request that it sent over and we expand this, we can see on the right hand side in the result panel, this is the JSON body that it sent over to Perplexity and it filled it in with best movies. And then of course what we got back was our content from Perplexity, which is, you know, here are some of the best movies across major platforms. All right, then real quick before we wrap up here, I just wanted to talk about some common responses that you can

get from your HTTP requests. So the rule of thumb to follow is if you get data back and you get a 200, you're good. Sometimes you'll get a response back, but you won't explicitly see a 200 message. But if you're getting the data back, then you're good to go. And a quick example of this is down here we have that HTTP request which we went over earlier in this video where we went to open weather maps API and you can see down here we got code 200 and there's data coming back and 200 is good that's a success code. Now if you get a request in the 400s that means that you probably set up the request wrong. So 400 bad request that could mean that your JSON's invalid. It could just mean that you have like an extra quotation mark or you have you know an extra comma something as silly as that. So let me show a quick example of that. We're going to test this workflow and what I'm doing is I'm trying to send over a query to Tavi. And you can see what we get is an error that says JSON parameter needs to be valid JSON. And this would be a 400 error. And the issue here is if we go into the JSON body that we're trying to send over, you can see in the result panel, we're trying to send over a query that has basically two sets of quotation marks. If you can see that. But here's the great news about JSON. It is so universally used and it's been around for so long that we could basically just copy the result over to chatbt. Paste it in here and say, I'm getting an error message that says JSON parameter needs to be valid JSON. What's wrong with my JSON? And as you can see, it says the issue with your JSON is the use of double quotes around the string value in this line. So now we'd be able to go fix that. And if we go back into the workflow, we take away these double quotes right here. Test the step again. Now you can see it's spinning and it's going to work. and we should get back some information about pineapples on pizza. Another common error you could run into is a 401, meaning unauthorized. This typically just means that your API key is wrong. You could also get a 403, which is forbidden. That just means that maybe your account doesn't have access to this data that you're requesting or something like that. And then another one you could get is a 404, which sometimes you'll get that if you type in a URL that doesn't exist. It just means this doesn't exist. We can't find it. And a lot of times when you're looking at the actual API documentation that you want to set up a request to. So here's an example with Tavi, it'll show you what typical responses could look like. So here's one where you know we're using Tavi to search for who is Leo Messi. This was an example we looked at earlier. And with a 200 response, we are getting back like a query, an answer results, stuff like that. We could also see we could get a 400 which would be for bad request, you know, invalid topic. We could have a 401 which means invalid API key. We could get all these other ones like 429, 432, but in general 400 is bad. And then even worse is a 500. And this just basically means something's wrong with the server. Maybe it doesn't exist anymore or there's a bug on the server side. But the good news about a 500 is it's not your fault. You didn't set up the request wrong. It just means something's wrong with the server. And it's really important to know that because if you think you did something wrong, but it's really not your fault at all, you may be banging your head against the wall for hours. So anyways, what I wanted to highlight here is there's never just like a one-sizefits-all. I know how to set up this one API call so I can just set up every single other API call the exact same way. The key is to really understand how do you read the API documentation? How do you set up your body parameters and your different header parameters? And then if you start to run into issues, the key is understanding and actually reading the error message that you're getting back and adjusting from there. All right, so that's going to do it for this video. Hopefully this has left you feeling a lot more comfortable with diving into API documentation, walking through it just step by step using those curl commands and really just understanding all I'm doing is I'm setting up filters and levers here. I don't have to get super confused. It's really not that

technical. I'm pretty much in complete control over what my API is going to send me back. The same way I'm in complete control when I'm, you know, ordering something on Door Dash or ordering something on Amazon, whatever it is. Hopefully by now the concept of APIs and HTTP requests makes a lot more sense. But really, just to drive it home, what we're going to do is hop into some actual setups in NADN of connecting to some different popular APIs and walk through a few more step by steps just to really make sure that we understand the differences that can come with different API documentation and how you read it and how you set up stuff like your credentials and the body requests. So, let's move into this next part, which I think is going to be super valuable to see different API calls in action. Okay, so in nodn when we're working with a large language model, whether that's an AI agent or just like an AI node, what happens is we can only access the information that is in the large language models training data. And a lot of times that's not going to be super up-to-date and real time. So what we want to do is access different APIs that let us search the web or do real-time search. And what we saw earlier in that third step-by-step workflow was we used a tool called Tavi and we accessed it through an HTTP request node which as you guys know looks like this. And we were able to use this to communicate with Tavali's API server. So if we ever want to access real-time information or do research on certain search terms, we have to use some sort of API to do that. So, like I said, we talked about Tavi, but in this video, I'm going to help you guys set up Perplexity, which if you don't know what it is, it's kind of like Chat GBT, but it's really, really good for web search and in-depth research. And it has that same sort of like, you know, chat interface as chatbt, but what you also have is access to the API. So, if I click on API, we can see this little screen, but what we want to go to is the developer docs. And in the developer docs, what we're looking for is the API reference. We can also click on the quick start guide right here which just shows you how you can set up your API key and get all that kind of stuff. So that's exactly what we're going to do is set up an API call to Perplexity. So I'm going to click on API reference and what we see here is the endpoint to access Perplexi's API. And so what I'm going to do is just grab this curl command from that right hand side, go back into our NEN and I'm just going to import the curl right into here. And then all we have to do from there is basically configure what we want to research and put in our own API key. So there we go. We have our node pretty much configured. And now the first thing we see we need to set up is our authorization API key. And what we could do is set this up in here as a generic credential type and save it. But right now we're just going to keep things as simple as possible where we imported the curl. And now I'm just going to show you where to plug in little things. So we have to go back over to Perplexity and we need to go get an API key. So I'm going to come over here to the left and I'm going to click on my settings. And hopefully in here we're able to find where our API key lives. Now we can see in the bottom left over here we have API keys. And what I'm going to do is come in here and just create a new secret key. And we just got a new one generated. So I'm just going to click on this button, click on copy, and all we have to do is replace the word right here that says token. So I'm just going to delete that. I'm going to make sure to leave a space after the word bearer. And I'm going to paste in my Perplexity API key. So now we should be connected. And now what we need to do is we need to set up the actual body request. So if I go back into the documentation, we can see this is basically what we're sending over. So that first thing is a model, which is the name of the model that will complete your prompt. And if we wanted to look at different models, we could click into here and look at other supported models from Perplexity. So it took us to the screen. We click on models, and we can see we have Sonar Pro or Sonar. We have Sonar Deep Research. We have some reasoning models as well. But just to keep things simple, I'm

going to stick with the default model right now, which is just sonar. Then we have an object that we're sending over, which is messages. And within the messages object, we have a few things. So first of all we're sending over content which is the contents of the message in turn of conversation. It can be in a string or an array of parts. And then we have a role which is going to be the role of the speaker in the conversation. And we have available options system user or assistant. So what you can see in our request is that we're sending over a system message as well as a user message. And the system message is basically the instructions for how this AI model on perplexity should act. And then the user message is our dynamic search query that is going to change every time. And if we go back into the documentation, we can see that there are a few other things we could add, but we don't have to. We could tell Perplexity what is the max tokens we want to use, what is the temperature we want to use. We could have it only search for things in the past week or day. So, this documentation is basically going to be all the filters and settings that you have access to in order to customize the type of results that you want to get back. But, like I said, keeping this one really simple. We just want to search the web. All I'm going to do is keep it as is. And if I disconnect this real quick and we come in and test step, it's basically going to be searching perplexity for how many stars are there in our galaxy. And then the AI model of sonar is the one that's going to grab all of these five sources and it's going to answer us. And right here it says the Milky Way galaxy, which is our home galaxy, is estimated to contain between 100 billion and 400 billion stars. This range is due to the difficulty blah blah blah blah blah. So that's basically how it was able to answer us because it used an AI model called sonar. So now if we wanted to make this search a little bit more dynamic, we could basically plug this in and you can see in here what I'm doing is I'm just setting a search term. So let's test this step. What happens is the output of this node is a research term. Then we could reference that variable of research term right in here in our actual body request to perplexity. So I would delete this fixed message which is how many stars are there in our galaxy. And all I would do is I'd drag in research term from the left, put it in between the two quotes, and now it's coming over dynamically as anthropic latest developments. And all I'd have to do now is hit test step. And we will get an answer from Perplexity about Anthropic's recent developments. There we go. It just came back. We can see there's five different sources right here. It went to Anthropic, it went to YouTube, it went to TechCrunch. And what we get is that today, May 22nd, so real time information, Claude Opus 4 was released. And that literally came out like 2 or 3 hours ago. So that's how we know this is searching the web in real time. And then all we'd have to do is have, you know, maybe an AI model is changing our search term or maybe we're pulling from a Google sheet with a bunch of different topics we need to research. But whatever it is, as long as we are passing over that variable, this actual search result from Perplexity is going to change every single time. And that's the whole point of variables, right? They they vary. They're dynamic. So, I know that one was quick, but Perplexity is a super super versatile tool and probably an API that you're going to be calling a ton of times. So, just wanted to make sure I threw that in there. So, Firecall is going to allow us to turn any website into LLM ready data in a matter of seconds. And as you can see right here, it's also open source. So, once you get over to Firecol, click on this button and you'll be able to get 500 free credits to play around with. As you can see, there's four different things we can do with Firecrawl. We can scrape, we can crawl, we can map or we can do this new extract which basically means we can give firecraw a URL and also a prompt like can you please extract the company name and the services they offer and an icebreaker out of this URL. So there's some really cool use cases that we can do with firecrawl. So in this video we're going to be mainly looking at extract, but I'm also going to

show you the difference between scrape and extract. And we're going to get into end and connect up so you can see how this works. But the playground is going to be a really good place to understand the difference between these different endpoints. All right, so for the sake of this video, this is the website we're going to be looking at. It's called quotes to scrape. And as you can see, it's got like 10 on this first page and it also has different pages of different categories of quotes. And as you can see, if we click into them, there are different quotes. So what I'm going to do is go back to the main screen and I'm going to copy the URL of this website and we're going to go into niten. We're going to open up a new node, which is going to be an HTTP request. And this is just to show you what a standard get request to a static website looks like. So we're going to paste in the URL, hit test step, and on the right hand side, we're going to get all the HTML back from the quotes to scrape website. Like I said, what we're looking at here is a nasty chunk of HTML. It's pretty hard for us to read, but basically what's going on here is this is the code that goes to the website in order to have it be styled and different fonts and different colors. So right here, what we're looking at is the entire first page of this website. So if we were to search for Harry, if I copy this, we go back into edit and we control F this. You can see there is the exact quote that has the word Harry. So everything from the website's in here, it's just wrapped up in kind of an ugly chunk of HTML. Now hopping back over to the fireall playground using the scrape endpoint, we can replace that same URL. We'll run this and it's going to output markdown formatting. So now we can see we actually have everything we're looking for with a different quotes and it's a lot more readable for a human. So that's what a web scrape is, right? We get the information back, whether that's HTML or markdown, but then we would typically feed that into some sort of LLM in order to extract the information we're looking for. In this case, we'd be looking for different quotes. But what we can do with extract is we can give it the URL and then also say, hey, get all of the quotes on here. And using this method, we can say, not just these first 10 on this page. I want you to crawl through the whole site and basically get all of these quotes, all of these quotes, all of these quotes, all of these quotes. So it's going to be really cool. So I'm going to show how this works in firecrawl and then we're going to plug it into noden. All right. So what we're doing here is we're saying extract all of the quotes and authors from this website. I gave it the website and now what it's doing is it's going to generate the different parameters that the LLM will be looking to extract out of the content of the website. Okay. So here's the run we're about to execute. We have the URL and then we have our schema for what the LLM is going to be looking for. And it's looking for text which would be the quote and it's a string. And then it's also going to be looking for the author of that quote which is also a string. And then the prompt we're feeding here to the LLM is extract all quotes and their corresponding authors from the website. So we're going to hit run and we're going to see that it's not only going to go to that first URL, it's basically going to take that main domain, which is quotes to scrape.com, and it's going to be crawling through the other sections of this website in order to come back and scrape all the quotes on there. Also, quick plug, go ahead and use code herk10 to get 10% off the first 12 months on your Firecrawl plan. Okay, so it just finished up. As you can see, we have 79 quotes. So down here we have a JSON response where it's going to be an object called quotes. And in there we have a bunch of different items which has you know text author text author text author and we have pretty much everything from that website now. Okay cool. But what we want to do is look at how we can do this in n so that we have you know a list of 20 30 40 URLs that we want to extract information from. We can just loop through and send off that automation rather than having to come in here and type that out in firecrawl. Okay. So what we're going to do is go back into edit end. And I apologize because there may be some jumping around

here, but we're basically just gonna clear out this HTTP request and grab a new one. Now, what we're going to do is we want to go into Firecrawl's documentation. So, all we have to do is import the curl command for the extract endpoint rather than trying to figure out how to fill out these different parameters. So, back in Firecrawl, once you set up your account, up in the top right, you'll see a button called docs. You want to click into there. And now, we can see a quick start guide. We have different endpoints. And what we're going to do is on the left, scroll down to features and click on extract. And this is what we're looking for. So, we've got some information here. The first thing to look at is when we're using the extract, you can extract structured data from one or multiple URLs, including wild cards. So, what we did was we didn't just scrape one single page. We basically scraped through all of the pages that had the main base domain of um quotescrape.com or something like that. And if you put a asterk after it, it's going to basically mean this is a wild card and it's going to go scrape all pages that are after it rather than just scraping this one predefined page. As you can see right here, it'll automatically crawl and parse all the URLs it can discover, then extract the requested data. And we can see that's how it worked because if we come back into the request we just made, we can see right here that it added a slash with an asterisk after quotes to scrape.com. Okay. Anyway, so what we're looking for here is this curl command. This is basically going to fill out the method, which is going to be a post request. It's going to fill out the endpoint. It'll fill out the content type, and it'll show us how to set up our authorization. And then we'll have a body request that we'll need to make some minor changes to. So in the top right I'm going to click copy and I'm going to come back into edit end. Hit import curl. Paste that in there. Hit import. And as you can see everything pretty much just got populated. So like I said the method is going to be a post. We have the endpoint already set up. And what I want to do is show you guys how to set up this authorization so that we can keep it saved forever rather than having to put it in here in the configuration panel every time. So first of all, head back over to your firecrawl. Go to API keys on the lefth hand side. And you're just going to want to copy that API key. So once you have that copied, head back into NN. And now let's look at how we actually set this up. So typically what you do is we have this as a header parameter. Not all authorizations are headers, but this one is a header. And the key or the name is authorization and the value is bearer space your API key. So what you'd typically do is just paste in your API key right there and you'd be good to go. But what we want to do is we want to save our firecrawl credential the same way you'd save, you know, a Google Sheets credential or a Slack credential. So, we're going to come into authentication, click on generic. We're going to click on generic type and choose header because we know down here it's a header off. And then you can see I have some other credentials already saved. We're going to create a new one. I'm just going to name this firecrawl to keep ourselves organized. For the name, we're going to put authorization. And for the value, we're going to type bearer with a capital B space and then paste in our API key. And we'll hit save. And this is going to be the exact same thing that we just did down below, except for now we have it saved. So, we can actually flick this field off. We don't need to send headers because we're sending them right here. And now we just need to figure out how to configure this body request. Okay, so I'm going to change this to an expression and open it up just so we can take a look at it. The first thing we notice is that by default there are three URLs in here that we would be extracting from. We don't want to do that here. So I'm going to grab everything within the array, but I'm going to keep the two quotation marks. Now all we need to do is put the URL that we're looking to extract information from in between these quotation marks. So here I just put in the quotes to scrape.com. But what we want to do if you remember is we want to put an asterisk after that so that it will go and crawl all of

the pages, not just that first page and which would only have like nine or 10 quotes. And now the rest is going to be really easy to configure because we already did this in the playground. So we know exactly what goes where. So I'm going to click back into our playground example. First thing is this is the quote that firecross sent off. So I'm going to copy that. Go back and edit in and I'm just going to replace the prompts right here. We don't want the company mission blah blah blah. We want to paste this in here and we're looking to extract all quotes and their corresponding authors from the website. And then next is basically telling the LLM, what are you pulling back? So, we just told it it's pulling back quotes and authors. So, we need to actually make the schema down here in the body request match the prompt. So, all we have to do is go back into our playground. Right here is the schema that we sent over in our example. And I'm just going to click on JSON view and I'm going to copy this entire thing which is wrapped up in curly braces. We'll come back into end and we'll start after schema colon space. Replace all this with what we just had in um fire crawl. And actually I think I've noticed the way that this copied over. It's not going to work. So let me show you guys that real quick. If we hit test step, it's going to say JSON parameter needs to be valid JSON. So what I'm going to do is I'm going to copy all of this. Now I came into chat GBT and I'm just saying fix this JSON. What it's going to do is it's going to just basically push these over. When you copy it over from Firecrol, it kind of aligns them on the left, but you don't want that. So, as you can see, it just basically pushed everything over. We'll copy this into our Nit end right there. And all it did was bump everything over once. And now we should be good to go. So, real quick before we test this out, I'm just going to call this extract. And then we'll hit test step. And we should see that it's going to be pulling. And it's going to give us a message that says um true. And it gives us an ID. And so now what we need to do next is pull this ID back to see if our request has been fulfilled yet. So I'm back in the documentation. And now we are going to look at down here asynchronous extraction and status checking. So this is how we check the status of a request. As you saw, we just made one. So here I'm going to click on copy this curl command. We're going to come back and end it in and we're going to add another HTTP request and we're going to import that in there. And you can see this one is going to be a get command. It's going to have a different endpoint. And what we need to do if you look back at the documentation is at the end of the extract slash we have to put the extract ID that we're looking to check the status of. So back in n the ID is going to be coming from the left hand side the previous node every time. So I'm just going to change the URL field to an expression. Put a backslash and then I'm going to grab the ID pull it right in there and we're good to go. Except we need to set up our credential. And this is why it's great. We already set this up as a generic as a header. And now we can just pull in easily our fire crawl off and hit test step. So what happens now is our request hasn't been done yet. So as you can see it comes back as processing and the data is an empty array. So what we're going to set up real quick is something called polling where we're basically checking in on a specific ID which is this one right here. And we're going to check and if it's if it's empty, if the data field is empty, then that means we're going to wait a certain amount of time and come back and try again. So after the request, I'm going to add a if. So, this is just basically going to help us create our filter. So, we're dragging in JSON.data, which as you can see is an empty array, and we're just going to say is empty. But one thing you have to keep in mind is this doesn't match. As you can see, we're dragging in an array, and we were trying to do a filter of a string. So, we have to go to array and then say is empty. And we'll hit test step. And this is going to say true. The data field is empty. And so, if true, what we want to do is we're going to add a wait. And this will wait for, you know, let's in in this case we'll just say five seconds. So if we hit test step, it's going to wait for five seconds.

And um I wish actually I switched the logic so that this would be on the bottom, but whatever. And then we would just drag this right back into here. And we would try it again. So now after 5 seconds had passed or however much time, we would try this again. And now we can see that we have our item back and the data field is no longer empty because we have our quotes object which has 83 quotes. So even got more than that time we did it in the playground. And I'm thinking this is just because, you know, the extract is kind of still in beta. So it may not be super consistent, but that's still way better than if we were to just do a simple getit request. And then as you can see now, if we ran this next step, this would come out. Ah, but this is interesting. So before it knows what it's pulling back, the JSON.data field is an array. And so we're able to set up is the array empty? But now it's an object. So we can't put it through the same filter because we're looking at a filter for an array. So what I'm thinking here is we could set up this continue using error output. So because this this node would error, we could hit test step and we could see now it's going to go down the false branch. And so this basically just means it's going to let us continue moving through the process. And we could do then whatever we want to do down here. Obviously this isn't perfect because I just set this up to show you guys and ran into that. But that's typically sort of the way we would think is how can we make this a little more dynamic because it has to deal with empty arrays or potentially full objects. Anyways, what I wanted to show you guys now is back in our request if we were to get rid of this asterk. What would happen? So, we're just going to run this whole process again. I'll hit test workflow. And now it's going to be sending that request only to, you know, one URL rather than the other one. Aha. And I'm glad we are doing live testing because I made the mistake of putting this in as JSON ID which doesn't exist if we're pulling from the weight node. So all we have to do in here is get rid of JSON. ID and pull in a basically a you know a node reference variable. So we're going to do two curly braces. We're going to be pulling from the extract node. And now we just want to say item.json ID and we should be good to go now. So I'm just going to refresh this and we'll completely do it again. So test workflow, we're doing the exact same thing. It's not ready yet. So, we're going to wait 5 seconds and then we're going to go check again. We hopefully should see, okay, it's not ready still. So, we're going to wait five more seconds. Come check again. And then whenever it is ready now, as you can see, it goes down this branch. And we can see that we actually get our items back. And what you see here is that this time we only got 10 quotes. Um, you know, it says nine, but computers count from zero. But we only got 10 quotes because um we didn't put an asterisk after the URL. So, Firecrawl didn't know I need to go scrape everything out of this whole base URL. I'm only going to be scraping this one specific page, which is this one right here, which does in fact only have 10 quotes. And by the way, super simple template here, but if you want to try it out and just plug in your API key and different URLs, you can grab that in the free school community. You'll hop in there, you will click on YouTube resources and click on the post associated with this video, and you'll have the JSON right there to download. Once you download that, all you have to do is import it from file right up here, and you'll have the workflow. So, there's a lot of cool use cases for firecrawl. It'd be cool to be able to pull from a a sheet, for example, of 30 or 40 or 50 URLs that we want to run through and then update based on the results. You could do some really cool stuff here, like researching a ton of companies and then having it also create some initial outreach for you. So, I hope you guys enjoyed that one. Firecrawl is a super cool tool. There's lots of functionality there and there's lots of uses of AI in Firecrawl, which is awesome. We're going to move into a different tool that you can use to scrape pretty much anything, which is called Appify, which has a ton of different actors, and you can scrape, like I said, almost anything. So, let's go into the setup video. So, Ampify is like a

marketplace for actors, which essentially let us scrape anything on the internet. As you can see right here, we're able to explore 4,500 plus pre-built actors for web scraping and automation. And it's really not that complicated. An actor is basically just a predefined script that was already built for us that we can just send off a certain request to. So, you can think of it like a virtual assistant where you're saying, "Hey, I want you to I want to use the Tik Tok virtual assistant and I want you to scrape, you know, videos that have the hashtag of AI content." Or you could use the LinkedIn job scraper and you could say, "I want to find jobs that are titled business analyst." So, there's just so many ways you could use Appify. You could get leads from Google Maps. You could get Instagram comments. You could get Facebook posts. There's just almost unlimited things you can do here. You can even tap into Apollo's database of leads and just get a ton. So today I'm just going to show you guys in NAN the easiest way to set up this Aify actor where you're going to start the actor and then you're going to just grab those results. So what you're going to want to do is head over to Aify using the link in the description and then use code 30 Nate Herk to get 30% off. Okay, like I said, what we're going to be covering today is a two-step process where you make one request to Aify to start up an actor and then you're going to wait for it to finish up and then you're just going to pull those results back in. So let me show you what that looks like. What I'm going to do is hit test workflow and this is going to start the Google Maps actor. And what we're doing here is we're asking for dentists in New York. And then if I go to my Appify console and I go over here to actors and click on the Google Maps extractor one, if I click on runs, we can see that there's one currently finishing up right now. And now that it's finished, I can go back into our workflow. I can hook it up to the get results node. Hit test step. And this is going to pull in those 50 dentists that we just scraped in New York. And you can see this contains information like their address, their website, their phone number, all this kind of stuff. So you can just basically scrape these lists of leads. So anyways, that's how this works, but let's walk through a live setup. So once you're in your Appify console, you click on the Appify store, and this is where you can see all the different actors. And let's do an example of like a social media one. So I'm going to click on this Tik Tok scraper since it's just the first one right here. And this may seem a little bit confusing, but it's not going to be too bad at all. We get to basically do all this with natural language. So let me show you guys how this works. So basically, we have this configuration panel right here. When you open up any sort of actor, they won't always all be the same, but in this one, what we have is videos with this hashtag. So, we can put something in. I put in AI content to play around with earlier. And then you can see it asks, how many videos do you want back? So, in this case, I put 10. Let's just put 25 for the sake of this demo. And then you have the option to add more settings. So, down here, we could do, you know, we could add certain profiles that we want to scrape. We could add a different search functionality. We could even have it download the videos for us. So, once you're good with this configuration, and just don't over complicate it. Think of it the same way you would like put in filters on an e-commerce website or the same way you would, you know, fill in your order when you're door dashing some food. So, now that we have this filled out the way we want it, all I'm going to do is come up to the top right and hit API and click API endpoints. The first thing we're going to do is we're going to use this endpoint called run actor. This is the one that's basically just going to send a request to Apify and start this process, but it's not going to give us the live results back. That's why the second step later is to pull the results back. What you could do is you could run the actor synchronously, meaning it's going to send it off and it's just going to spin in and it end until we're done and until it has the results. But I found this way to be more consistent. So anyways, all you have to do is click on copy and it's already going to have copied over your

appy API key. So it's really, really simple. All we're going to do here is open up a new HTTP request. I'm going to just paste in that URL that we just copied right here. And that's basically all we have to do except for we want to change this method to post because as you can see right here, it says post. And so this is basically just us putting in the actor's phone number. And so we're giving it a call. But now what we have to do is actually tell it what we want. So right here, we've already filled this out. I'm going to click on JSON and all I have to do is just copy this JSON right here. Go back into N. Flick this on to send a body and we want to send over just JSON. And then all I have to do is paste that in there. So, as you can see, what we're sending over to this Tik Tok scraper is I want AI content and I want 25 results. And then all this other stuff is false. So, I'm just going to hit test step. And so, this basically returns us with an ID and says, okay, the actor started. If we go back into here and we click on runs, we can see that this crawler is now running. and it's going to basically tell us how much it costed, how long it took, and all this kind of stuff. And now it's already done. So, what we need to do now is we need to click on API up in the top right. Click on API endpoints again and scroll all the way down to the bottom where we can see get last run data set items. So, all I need to do is hit this copy button right here. Go back into Nitn and then open up another HTTP request. And then I'm just going to paste that URL right in there once again. And I don't even have to change the method because if we go in here, we can see that this is a get. So, all I have to do is hit test step. And this is going to pull in those 25 results from our Tik Tok scrape based on the search term AI content. So, you can see right here it says 25 items. And just to show you guys that it really is 25 items, I'm just going to grab a set field. We're going to just drag in the actual text from here and hit test step. And it should Oh, we have to connect a trigger. So, I'm just going to move this trigger over here real quick. And um what you can do is because we already have our data here, I can just pin it so we don't actually have to run it again. But then I'll hit test step. And now we can see we're going to get our 25 items right here, which are all of the text content. So I think just the captions or the titles of these Tik Toks. And we have all 25 Tik Toks as you can see. So I just showed you guys the two-step method. And why I've been using it because here's an example where I did the synchronous run. So all I did was I came to the Google maps and I went to API endpoints and then I wanted to do run actor synchronously which basically means that it would run it in n and it would spin until the results were done and then it should feed back the output. So I copied that I put it into here and as you can see I just ran it with the Google maps looking for plumbers and we got nothing back. So that's why we're taking this two-step approach where as you can see here we're going to do that exact same request. We're doing a request for plumbers and we're going to fire this off. And so nothing came back in Nitn. But if we go to our actor and we go to runs, we can see right here that this was the one that we just made for plumbers. And if we click into it, we can see all the plumbers. So that's why we're taking the two-step approach. I'm going to make the exact same request here for New York plumbers. And what I'm going to do is just run this workflow. And now I wanted to talk about what we have to do because what happens is we started the actor. And as you can see, it's running right now. And then it went to grab the results, but the results aren't done yet. So that's why it comes back and says this is an item, but it's empty. So, what we want to do is we want to go to our runs and we want to see how long this is taking on average for 50 leads. As you can see, the most amount of time it's ever taken was 19 seconds. So, I'm just going to go in here and in between the start actor and grab results, I'm going to add a wait, and I'm just going to tell this thing to wait for 22 seconds just to be safe. And now, what I'm going to do is just run this thing again. It's going to start the actor. It's going to wait for 22 seconds. So, if we go back into Ampify, you can see that the actor is

once again running. After about 22 seconds, it's going to pass over and then we should get all 50 results back in our HTTP request. There we go. Just finished up. And now you can see that we have 50 items which are all of the plumbers that we got in New York. So from here, now that you have these 50 leads and remember if you want to come back into Ampify and change up your input, you can change how many places you want to extract. So if you changed this to 200 and then you clicked on JSON and you copied in that body, you would now be searching for 200 results. But anyways, that's the hard part is getting the leads into end. But now we have all this data about them and we can just, you know, do some research, send them off a email, whatever it is, we can just basically have this thing running 24/7. And if you wanted to make this workflow more advanced to handle a little bit more dynamic amount of results. What you'd want to use is a technique called polling. So basically, you'd wait, you check in, and then if the results were all done, you continue down the process. But if they weren't all done, you would basically wait again and come back. And you would just loop through this until you're confident that all of the results are done. So that's going to be it for this one. I'll have this template available in my free school community if you want to play around with it. Just remember you'll have to come in here and you'll have to switch out your own API key. And don't forget when you get to Ampify, you can use code 30 Nate Herk to get 30% off. Okay, so those were some APIs that we can use to actually scrape information. Now, what if we want to use APIs to generate some sort of content? We're going to look at an image generation API from OpenAI and we're going to look at a video generation API called Runway. So these next two workflows will explain how you set up those API calls and also how you can bake them into a workflow to be a little bit more practical. So let's take a look. So this workflow right here, all I had to do was enter in ROI on AI automation and it was able to spit out this LinkedIn post for me. And if you look at this graphic, it's insane. It looks super professional. It even has a little LinkedIn logo in the corner, but it directly calls out the actual statistics that are in the post based on the research. And for this next one, all I typed in was mental health within the workplace and it spit out this post. According to Deote Insights, organizations that support mental health can see up to 25% increase in productivity. And as you can see down here, it's just a beautiful graphic. So, a few weeks ago when Chacht came out with their image generation model, you probably saw a lot of stuff on LinkedIn like this where people were turning themselves into action figures or some stuff like this where people were turning themselves into Pixar animation style photos or whatever it is. And obviously, I had to try this out myself. And of course, this was very cool and everyone was getting really excited. But then I started to think about how could this image generation model actually be used to save time for a marketing team because this new image model is actually good at spelling and it can make words that don't look like gibberish. It opens up a world of possibilities. So here's a really quick example of me giving it a one-s sentence prompt and it spits out a poster that looks pretty solid. Of course, we were limited to having to do this in chatbt and coming in here and typing, but now the API is released, so we can start to save hours and hours of time. And so, the automation I'm going to show with you guys today is going to help you turn an idea into a fully researched LinkedIn post with a graphic as well. And of course, we're going to walk through setting up the HTTP request to OpenAI's image generation model. But what you can do is also download this entire template for free and you can use it to post on LinkedIn or you can also just kind of build on top of it to see how you can use image generation to save you hours and hours within some sort of marketing process. So this workflow right here, all I had to do was enter in ROI on AI automation and it was able to spit out this LinkedIn post for me. And if you look at this graphic, it's insane. It looks super professional. It even has a little LinkedIn logo

in the corner, but it directly calls out the actual statistics that are in the post based on the research. So 74% of organizations say their most advanced AI initiatives are meeting or exceeding ROI expectations right here. And on the other side, we can see that only 26% of companies have achieved significant AIdriven gains so far, which is right here. And I was just extremely impressed by this one. And for this next one, all I typed in was mental health within the workplace. And to spit out this post, according to Deote Insights, organizations that support mental health can see up to 25% increase in productivity. And as you can see down here, it's just a beautiful graphic. something that would probably take me 20 minutes in Canva. And if you can now push out these posts in a minute rather than 20 minutes, you can start to push out more and more throughout the day and save hours every week. And because the post is being backed by research, the graphic is being backed by the research post. You're not polluting anything into the internet. A lot of people in my comments call it AI slop. Anyways, let's do a quick live run of this workflow and then I'll walk through step by step how to set up this API call. And as always, if you want to download this workflow for free, all you have to do is join my free school community. link is down in the description and then you can search for the title of the video. You can go into YouTube resources. You need to find the post associated with this video and then when you're in there, you'll be able to download this JSON file and that is the template. So you download the JSON file. You'll go back into Nitn. You'll open up a new workflow and in the top right you'll go to import from file. Import that JSON file and then there'll be a little sticky note with a setup guide just sort of telling you what you need to plug in to get this thing to work for you. Okay, quick disclaimer though. I'm not actually going to post this to LinkedIn. you certainly could, but um I'm just going to basically send the post as well as the attachment to my email because I don't want to post on LinkedIn right now. Anyways, as you can see here, this workflow is starting with a form submission. So, if I hit test workflow, it's going to pop up with a form where we have to enter in our email for the workflow to send us the results. Topic of the post and then also I threw in here a target audience. So, you could have these posts be kind of flavored towards a specific audience if you want to. Okay, so this form is waiting for us. I put in my email. I put the topic of morning versus night people and the target audience is working adults. So, we'll hit submit, close out of here, and we'll see the LinkedIn post agent is going to start up. It's using Tavi here for research and it's going to create that post and then pass the post on to the image prompt agent. And that image prompt agent is going to read the post and basically create a prompt to feed into OpenAI's image generator. And as you can see, it's doing that right now. We're going to get that back as a base 64 string. And then we're just converting that to binary so we can actually post that on LinkedIn or send that in email as an attachment and we'll break down all these steps. But let's just wait and see what these results look like here. Okay, so all that just finished up. Let me pop over to email. So in email, we got our new LinkedIn post. Are you a morning lark or a night owl? The science of productivity. I'm not going to read through this right now exactly, but let's take a look at the image we got. When are you most productive? In the morning, plus 10% productivity or night owls thrive in flexibility. I mean, this is insane. This is a really good graphic. Okay, so now that we've seen again how good this is, let's just break down what's going on. We're going to start off with the LinkedIn post agent. All we're doing is we're feeding in two things from the form submission, which was what is the topic of the post, as well as who's the target audience. So right here, you can see morning versus night people and working adults. And then we move into the actual system prompt, which I'm not going to read through this entire thing. If you download the template, the prompt will be in there for you to look at. But basically I told it you are an AI agent specialized in creating professional educational and engaging LinkedIn posts based

on a topic provided by the user. We told it that it has a tool called Tavly that it will use to search the web and gather accurate information and that the post should be written to appeal to the provided target audience. And then basically just some more information about how to structure the post, what it should output and then an example which is basically you receive a topic. You search the web, you draft the post and you format it with source citations, clean structure, optional hashtags and a call to action at the end. And as you can see what it outputs is a super clean LinkedIn post right here. So then what we're going to do is basically we're feeding this output directly into that next agent. And by the way, they're both using chat GBT 4.1 through open router. All right, but before we look at the image prompt agent, let's just take a look at these two things down here. So the first one is the chat model that plugs into both image prompt agent and the LinkedIn post agent. So all you have to do is go to open router, get an API key, and then you can choose from all these different models. And in here, I'm using GPT4.1. And then we have the actual tool that the LinkedIn agent uses for its research which is Tavi. And what we're doing here is we're sending off a post request using an HTTP request tool to the Tavi endpoint. So this is where people typically start to feel overwhelmed when trying to set up these requests because it can be confusing when you're trying to look through that API documentation. Which is exactly why in my paid community I created a APIs and HTTP requests deep dive because truthfully you need to understand how to set up these requests because being able to connect to different APIs is where the magic really happens. So Tavi just lets your LLM connect to the web and it's really good for web search and it also gives you a thousand free searches per month. So that's the plan that I'm on. Anyways, once you're in here and you have an account and you get an API key, all I did was went to the Tavali search endpoint and you can see we have a curl statement right here where we have this endpoint. We have post as the method we have this is how we authorize ourselves and this is all going to be pretty similar to the way that we set up the actual request to OpenAI's image generation API. So, I'm not going to dive into this too much. When you download this template, all you have to do is plug in your Tavi API. But later in this video when we walk through setting up the request to OpenAI, this should make more sense. Anyways, the main thing to take away from this tool is that we're using a placeholder for the request because in the request we sent over to Tavali, we basically say, okay, here's the search query that we're going to search the internet for. And then we have all these other little settings we can tweak like the topic, how many results, how many chunks per source, all this kind of stuff. All we really want to touch right now is the query. And as you can see, I put this in curly braces, meaning it's a placeholder. I'm calling the placeholder search term. And down here, I'm defining that placeholder as what the user is searching for. So, as you can see, this data in the placeholder is going to be filled in by the model. So, based on our form submission, when we asked it to, you know, create a LinkedIn post about morning versus night people, it fills out the search term with latest research on productivity, morning people versus night people, and that's basically how it searches the internet. And then we get our results back. And now it creates a LinkedIn post that we're ready to pass off to the next agent. So the output of this one gets fed into this next one, which all it has to do is read the output. As you can see right here, we gave it the LinkedIn post, which is the full one that we just got spit out. And then our system message is basically telling it to turn that into an image prompt. This one is a little bit longer. Not too bad, though. I'm not going to read the whole thing, but essentially we're telling it that it's going to be an AI agent that transforms a LinkedIn post into a visual image prompt for a textto-image AI generation model. So, we told it to read the post, identify the message, identify the takeaways, and then create a compelling graphic prompt that can be used with a textto

image generator. We gave it some output instructions like, you know, if there's numbers, try to work those into the prompt. Um, you can use, you know, text, charts, icons, shapes, overlays, anything like that. And then the very bottom here, we just gave it sort of like an example prompt format. And you can see what it spits out is a image prompt. So it says a dynamic split screen infographic style graphic. Left side has a sunrise, it's bright yellow, and it has morning larks plus 10% productivity. And the right side is a morning night sky, cool blue gradients, a crescent moon, all this kind of stuff. And that is exactly what we saw back in here when we look at our image. And so this is just so cool to me because first of all, I think it's really cool that it can read a post and kind of use its brain to say, "Okay, this would be a good, you know, graphic to be looking at while I'm reading this post." But then on top of that, it can actually just go create that for us. So, I think this stuff is super cool. You know, I remember back in September, I was working on a project where someone wanted me to help them with LinkedIn automated posting and they wanted visual elements as well and I was like, uh, I don't know, like that might have to be a couple month away thing when we have some better models and now we're here. So, it's just super exciting to see. But anyways, now we're going to feed that output, the image prompt into the HTTP request to OpenAI. So, real quick, let's go take a look at OpenAI's documentation. So, of course, we have the GBT image API, which lets you create, edit, and transform images. You've got different styles, of course. You can do like memes with a with text. You can do creative things. You can turn other images into different images. You can do all this kind of stuff. And this is where it gets really cool, these posters and the visuals with words because that's the kind of stuff where typically AI image gen like wasn't there yet. And one thing real quick in your OpenAI account, which is different than your chatbt account, this is where you add the billing for your OpenAI API calls. You have to have your organization verified in order to actually be able to access this model through API. Right now, it took me 2 minutes. You basically just have to submit an ID and it has to verify that you're human and then you'll be verified and then you can use it. Otherwise, you're going to get an error message that looks like this that I got earlier today. But anyways, the verification process does not take too long. Anyways, then you're going to head over to the API documentation that I will have linked in the description where we can see how we can actually create an image in NAN. So, we're going to dive deeper into this documentation in the later part of this video where I'm walking through a step-by-step setup of this. But, we're using the endpoint um which is going to create an image. So, we have this URL right here. We're going to be creating a post request and then we just obviously have our things that we have to configure like the prompt in the body. We have to obviously send over some sort of API key. We have to, you know, we can choose the size. We can choose the model. All this kind of stuff. So back in NN, you can see that I'm sending a post request to that endpoint. For the headers, I set up my API key right here, but I'm going to show you guys a better way to do that in the later part of this video. And then for the body, we're saying, okay, I want to use the GBT image model. Here's the actual prompt to use for the image which we dragged in from the image prompt agent. And then finally the size we just left it as that 1024 * 1024 square image. And so this is interesting because what we get back is we get back a massive base 64 code. Like this thing is huge. I can't even scroll right now. My screen's kind of frozen. Anyways, um yeah, there it goes. It just kind of lagged. But we got back this massive file. We can see how many tokens this was. And then what we're going to do is we're going to convert that to binary data. So that's how we can actually get the file as an image. As you can see now after we turn that nasty string into a file, we have the binary image right over here. So all I did was I basically just dragged in this field right here with that nasty string. And then when you hit test step, you'll

get that binary data. And then from there, you have the binary data, you have the LinkedIn post. All you have to do is, you know, activate LinkedIn, drag it right in there. Or you can just do what I did, which is I'm sending it to myself in email. And of course, before you guys yell at me, let's just talk about how much this run costed me. So, this was 4,273 tokens. And if we look at this API and we go down to the pricing section, we can see that for image output tokens, which was generated images, it's going to be 40 bucks for a million tokens, which comes out to about 17 cents. If you can see that right here, hopefully I did the math right. But really, for the quality and kind of for the industry standard I've seen for price, that's on the cheaper end. And as you can see down here, it translates roughly to 2 cents, 7 cents, 19 cents per generated image for low, medium, blah blah blah blah blah. But anyways, now that that's out of the way, let's just set up an HTTP request to that API and generate an image. So, I'm going to add a first step. I'm just going to grab an HTTP request. So, I'm just going to head over to the actual API documentation from OpenAI on how to create an image and how to hit this endpoint. And all we're going to do is we're going to copy this curl command over here on the right. If it you're not seeing a curl command, if you're seeing Python, just change that to curl. Copy that. And then we're going to go back into Nitn. Hit import curl. Paste that in there. And then once we hit import, we're almost done. So that curl statement basically just autopop populated almost everything we need to do. Now we just have a few minor tweaks. But as you can see, it changed the method to post. It gave us the correct URL endpoint already. It has us sending a header, which is our authorization, and then it has our body parameters filled out where all we'd really have to change here is the prompt. And if we wanted to, we can customize this kind of stuff. And that's why it's going to be really helpful to be able to understand and read API documentation so you know how to customize these different requests. Basically, all of these little things here like prompt, background, model, n, output format, they're just little levers that you can pull and tweak in order to change your output. But we're not going to dive too deep into that right now. Let's just see how we can create an image. Anyways, before we grab our API key and plug that in, when you're in your OpenAI account, make sure that your organization is verified. Otherwise, you're going to get this error message and it's not going to let you access the model. Doesn't take long. Just submit an ID. And then also make sure that you have billing information set up so you can actually pay for um an image. But then you're going to go down here to API keys. You're going to create new secret key. This one's going to be called image test just for now. And then you're going to copy that API key. Now back in any then it has this already set up for us where all we need to do is delete all this. We're going to keep the space after bearer. And we can paste in our API key like that. and we're good to go. But if you want a better method to be able to save this key in Nadn so you don't have to go find it every time. What you can do is come to authentication, go to general or actually no it's generic and then you're going to choose header off and we know it's header because right here we're sending headers as a header parameter and this is where we're authorizing oursel. So we're just going to do the same up here with the header off. And then we're going to create a new one. I'm just going to call this one openai image just so we can keep ourselves organized. And then you're going to do the same thing as what we saw down in that header parameter field. Meaning the authorization is the name and then the value was bearer space API key. So that's all I'm going to do. I'm going to hit save. We are now authorized to access this endpoint. And I'm just going to turn off sending headers because we're technically sending headers right up here with our authentication. So we should be good now. Right now we'll be getting an image of a cute baby sea otter. Um, and I'm just going to say making pancakes. And we'll hit test step. And this should be running right now. Um, okay. So, bad request. Please check your

parameters. Invalid type for n. It expected an integer, but it got a string instead. So, if you go back to the API documentation, we can see n right here. It should be integer or null, and it's also optional. So, I'm just going to delete that. We don't really need that. And I'm going to hit test step. And while that's running real quick, we'll just go back at n. And this basically says the number of images to generate must be between 1 and 10. So that's like one of those little levers you could tweak like I was talking about if you want to customize your request. But right now by default it's only going to give us one. Looks like this HTTP request is working. So I'll check in with you guys in 20 seconds when this is done. Okay. So now that that finished up, didn't take too long. We have a few things and all we really need is this base 64. But we can see again this one costed around 17. And now we just have to turn this into binary so we can actually view an image. So I'm going to add a plus after the HTTP request. I'm just going to type in binary. And we can see convert to file, which is going to convert JSON data to binary data. And all we want to do here is move a B 64 string to file because this is a B 64 JSON. And this basically represents the image. So I'm going to drag that into there. And then when I hit test step, we should be getting a binary image output in a field called data. As you can see right here, and this should be our image of a cute sea otter making pancakes. As you can see, um it's not super realistic, and that's because the prompt didn't have any like photorealistic, hyperrealistic elements in there, but you can easily make it do so. And of course, I was playing around with this earlier, and just to show you guys, you can make some pretty cool realistic images, here was um a post I made about um if ancient Rome had access to iPhones. And obviously, this is not like a real Twitter account. Um, but this is a dinosaurs evolved into modern-day influencers. This was just for me testing like an automation using this API and auto posting, but not as practical as like these LinkedIn graphics. But if you guys want to see a video sort of like this, let me know. Or if you also want to see a more evolved version of the LinkedIn posting flow and how we can make it even more robust and even more automated, then definitely let me know about that as well. Okay. Okay. So, all I have to do in this form submission is enter in a picture of a product, enter in the product name, the product description, and my email address. And we'll send this off, and we'll see the workflow over here start to fire off. So, we're going to upload the photo. We're going to get an image prompt. We're going to download that photo. Now, we're creating a professional graphic. So, after our image has been generated, we're uploading it to a API to get a public URL so we can feed that URL of the image into Runway to generate a professional video. Now, we're going to wait 30 seconds and then we'll check in to see if the video is done. If it's not done yet, we're going to come down here and pull, wait five more seconds, and then go check in. And we're going to do this infinitely until our video is actually done. So, anyways, it just finished up. It ended up hitting this check eight times, which indicates I should probably increase the wait time over here. But anyways, let's go look at our finished products. So, we just got this new email. Here are the requested marketing materials for your toothpaste. So, first, let's look at the video cuz I think that's more exciting. So, let me open up this link. Wow, we got a 10-second video. It's spinning. It's 3D. The lighting is changing. This looks awesome. And then, of course, it also sends us that image. in case we want to use that as well. And one of the steps in the workflow is that it's going to upload your original image to your Google Drive. So here you can see this was the original and then this was the finished product. So now you guys have seen a demo. We're going to build this entire workflow step by step. So stick with me because by the end of this video, you'll have this exact system up and running. Okay. So when we're setting up a system where we're creating an image from text and then we're creating a video from that image, the two most important things are going to be that image prompt and that video prompt. So

what we're going to do is head over to my school community. The link for that will be down in the description. It's a free school community. And then what you're going to do is either search for the title of this video or click on YouTube resources and find the post associated with this video. And when you click into there, there'll be a doc that will look like this or a PDF and it will have the two prompts that you'll need in order to run the system. So head over there, get that doc, and then we can hop into the step by step. And that way we can start to build this workflow and you guys will have the prompts to plug right in. Cool. So once you have those, let's get started on the workflow. So as you guys know, a workflow always has to start with some sort of trigger. So in this case, we're going to be triggering this workflow with a form submission. So I'm just going to grab the native NAN form on new form event. So we're going to configure what this form is going to look like and what it's going to prompt a user to input. And then whenever someone actually submits a response, that's when the workflow is going to fire off. Okay. So I'm going to leave the authentication as none. The form title, I'm just putting go to market. For the form description, I'm going to say give us a product photo, title, and description, and we'll get back to you with professional marketing materials. And if you guys are interested in what I just used to dictate that text, there'll be a link for Whisper Flow down in the description. And now we need to add our form elements. So the first one is going to be not a text. We're going to have them actually submit a file. So click on file. This is going to be required. I only want them to be allowed to upload one file. So I'm going to switch off multiple files. And then for the field name, we're just going to say product photo. Okay. So now we're going to add another one, which is going to be the product title. So I'm just going to write product title. This is going to be text. For placeholder, let's just put toothpaste since that was the example. This will be a required field. So, the placeholder is just going to be the gray text that fills in the text box so people are kind of they know what to put in. Okay, we're adding another one called product description. We'll make this one required. We'll just leave the placeholder blank cuz you don't need it. And then finally, what we need to get from them is an email, but instead of doing text, we can actually make it require a valid email address. So, I'm just going to call it email and we'll just say like namele.com so they know what a valid email looks like. We'll make that required because we have to send them an email at the end with their materials. And now we should be good to go. So if I hit test step, we'll see that it's going to open up a form submission and it has everything that we just configured. And now let me put in some sample data real quick. Okay, so I put a picture of a clone bottle. The title's clone. I said the clone smells very clean and fresh and it's a very sophisticated scent because we're going to have that description be used to sort of help create that text image prompt. And then I just put my email. So I'm going to submit this form. We should see that we're going to get data back right here in our NIN, which is the binary photo. This is the product photo that I just submitted. And then we have our actual table of information like the title, the description, and the email. And so when I'm building stuff step by step, what I like to do is I get the data in here, and then I pretty much will just build node by node, testing the data all the way through, making sure that nothing's going to break when variables are being passed from left to right in this workflow. Okay, so the next thing that we need to do is we have this binary data in here and binary data is tough to reference later. So what I'm going to do is I'm just going to upload it straight to our Google Drive so we can pull that in later when we need it to actually edit that image. Okay, so that's our form trigger. That's what starts the workflow. And now what we're going to do next is we want to upload that original image to Google Drive so we can pull it in later and then use it to edit the image. So what I'm going to do is I'm going to click on the plus. I'm going to type in Google Drive. And we're going to grab a Google Drive operation. That is going to be upload

file. So, I'll click on upload file. And at this point, you need to connect your Google Drive. So, I'm not going to walk through that step by step, but I have a video right up here where I do walk through it step by step. But basically, you're just going to go to Docs. You have to open up a sort of Google Cloud profile or a console, and then you just have to connect yourself and enable the right credentials and APIs. Um, but like I said, that video will walk through it. Anyways, now what we're doing is we have to upload the binary field right here to our Google Drive. So, it's not called data. We can see over here it's called product photo. So, I'm just going to copy and paste that right there. So, it's going to be looking for that product photo. And then we have to give it a name. So, that's why we had the person submit a title. So, all I'm going to do is for the name, I'm going to make this an expression instead of fixed because this name is going to change based on the actual product coming through. I'm going to drag in the product title from the left right here. So now the the photo in Google Drive is going to be called cologne and then I'm just going to in parenthesis say original. So because this is an expression, it basically means whenever someone submits a form, whatever the title is, it's going to be title and then it's going to say original. And that's how we sort of control that to be dynamic. Anyways, then I'm just choosing what folder to go in. So in my drive, I'm going to choose it to go to a folder that I just made called um product creatives. So once we have that configured, I'm going to hit test step. We're going to wait for this to spin. it means that it's trying to upload it right now. And then once we get that success message, we'll quickly go to our Google Drive and make sure that the image is actually there. So there we go. It just came back. And now I'm going to click into Google Drive, click out of the toothpaste, and we can see we have cologne. And that is the image that we just submitted in NAN. All right. Now that we've done that, what we want to do is we want to feed the data into an AI node so that it can create a text image prompt. So I'm going to click on the plus. I'm going to grab an AI agent. And before we do anything in here, I'm first of all going to give it its brain. So, I'm going to click on the plus under chat model. I'm personally going to grab an open router chat model, which basically lets you connect to a ton of different things. Um, let me see. Open router.ai. It basically lets you connect your agents to all the different models. So, if I click on models up here, we can see that it just lets you connect to Gemini, Anthropic, OpenAI, Deepseek. It has all these models and all in one place. So, go to open router, get an API key, and then once you come back into here, all you have to do is connect your API key. And what I'm going to use here is going to be 4.1. And then I'm just going to name this so we know which one I'm using here. And then we now have our agent accessing GPT4.1. Okay. So now you're going to go to that PDF that I have in the school community and you're just going to copy this product photography prompt. Grab that. Go back to the AI agent and then you're going to click on add option. Add a system message. And then we're basically just going to I'm going to click on expression and expand this full screen so you guys can see it better. But I'm just going to paste that prompt in here. And this is going to tell the AI agent how to take what we're giving it and turn it into a text image optimized prompt for professional style, you know, studio photography. So, we're not done yet because we have to actually give it the dynamic information from our form submission every time. So, that's a user message. That's basically what it's going to look at. So, the user message is what the agent's going to look at every time. And the system message is basically like here are your instructions. So for the user message, we're not going to be using a connected chat trigger node. We're going to define below. And when we want to make sure that this changes every time, we have to make sure it's an expression. And then I'm just going to drill down over here to the form submission. And I'm going to say, okay, here's what we're going to give this agent. It's going to get the product, which the

person submitted to us in the form, and we can drag in the product, which was cologne, as you can see on the right. And then they also gave us a description. So, all I have to do now is drag in the product description. And so, now every time the agent will be looking at whatever product and description that the user submitted in order to create its prompt. So, I'm going to hit test step. We'll see right now it's using its chat model GPT4.1. And it's already created that prompt for us. So, let's just give it a quick read. Hyperrealistic photo of sophisticated cologne bottle, transparent glass, sleek minimalistic design, silver metal cap, all this. But what we have to do is we have to make sure that the image isn't being created just on this. It has to look at this, but it also has to look at the actual original image. So that's why our next step is going to be to redownload this file and then we're going to push it over to the image generation model. So at this point, you may be wondering like why are we going to upload the file if we're just going to download it again? And the reason why I had to do that is because when we get the file in the form of binary, we want to send the binary data into the HTTP request right here that actually generates the image. And we can't reference the binary way over here if it's only coming through over here. So, we upload it so that we can then download it and then send it right back in. And so, if that doesn't make sense yet, it probably will once we get over to the stage. But that's why. Anyways, next step is we're going to download that file. So, I'm going to click on this plus. We're going to be downloading it from Google Drive and we're going to be using the operation download file. So, we already should be connected because we've set up our Google credentials already. The operation is going to be download the resources a file and instead of choosing from a list, we're going to choose by ID. And all we're going to do is download that file that we previously uploaded every time. So I'm going to come over here, the Google Drive, upload photo node, drag in the ID, and now we can see that's all we have to do. If we hit test step, we'll get back that file that we originally uploaded. And we can just make sure it's the cologne bottle. Okay, but now it's time to basically use that downloaded file and the image prompt and send that over to an API that's going to create an image for us. So we're going to be using OpenAI's image generator. So here is the documentation. we have the ability to create an image or we can create an image edit which is what we want to do because we wanted to look at the photo and our request. So typically what you can do in this documentation is you can copy the curl command but this curl command is actually broken so we're not going to do that. If you copied this one up here to actually just create an image that one would work fine but there's like a bug with this one right now. So anyways I'm going to go into our n I'm going to hit the plus. I'm going to grab an HTTP request and now we're going to configure this request. So, I'm going to walk through how I'm reading the API documentation right here to set this up. I'm not going to go super super in-depth, but if you get confused along the way, then definitely check out my paid course. The link for that down in the description. I've got a full course on deep diving into APIs and HTTP requests. Anyways, the first thing we see is we're going to be making a post request to this endpoint. So, the first thing I'm going to do is copy this endpoint. We're going to paste that in. And then we're also going to make sure the method is set to post. So, the next thing that we have to do is authorize ourselves somehow. So over here I can see that we have a header and the name is going to be authorization and then the value is going to be bearer space R open AI key. So that's why I set up a header authentication already. So in authentication I went to generic and then I went to header and then you can see I have a bunch of different headers already set up. But what I did here is I chose my OpenAI one where basically all I did was I typed in here authorization and then in the value I typed in bearer space and then I pasted my API key in there. And now I have my OpenAI credential saved forever. Okay. So the first thing we have to do in our body request

over to OpenAI is we have to send over the image to edit. So that's going to be in a field called image. And then we're sending over the actual photo. So what I'm going to do is I'm going to click on send body. I'm going to use form data. And now we can set up the different names and values to send over. So the first thing is we're going to send over this image right here on the lefth hand side. And this is in a field called data. And it's binary. So, I'm going to choose instead of form data, I'm going to send over an NAN binary file. The name is going to be image because that's what it said in the documentation. And the input data field name is data. So, I'm just going to copy that, paste it in there. And this basically means, okay, we're sending over this picture. The next thing we need to send over is a prompt. So, the name of this field is going to be prompt. I'm just going to copy that, add a new parameter, and call it prompt. And then for the value, we want to send over the prompt that we had our AI agent write. So, I'm going to click into schema and I'm just going to drag over the output from the AI agent right there. And now that's an expression. So, the next thing we want to send over is what model do we want to use? Because if we don't put this in, it's going to default to dolly 2, but we want to use gpt-image- one. So, I'm going to copy GPT- image- one. We're going to come back into here, and I'm going to paste that in as the value, but then the name is model because, as you can see in here, right there, it says model. So hopefully you guys can see that when we're sending over an API call, we just have all of these different options where we can sort of tweak different settings to change the way that we get the output back. And then you have some other options, of course, like quality or size. But right now, we're just going to leave all that as default and just go with these three things to keep it simple. And I'm going to hit test step and we'll see if this is working. Okay, never mind. I got an error and I was like, okay, I think I did everything right. The reason I got the error is because I don't have any more credits. So, if you get this error, go add some credits. Okay, so added more credits. I'm going to try this again and I'll check back in. But before I do that, I wanted to say me clearly, I've been like spamming this thing with creating images cuz it's so cool. It's so fun. But everyone else in the world has also been doing that. So, if you're ever getting some sort of like errors where it's like a 500 type of error where it means like something's going on on the server side of things or you're seeing like some sort of rate limit stuff, keep in mind that there's there's a limit on how many images you can send per minute. I don't think that's been clearly defined on GPT- image-1. But also, if the OpenAI server is receiving way too many requests, that is also another reason why your request may be failing. So, just keep that in mind. Okay, so now it worked. We just got that back. But what you'll notice is we don't see an image here or like an image URL. So, what we have to do is we have this base 64 string and we have to turn that into binary data. So, what I'm going to do is after this node, I'm going to add one that says um convert to file. So we're going to convert JSON data to binary data and we're going to do B 64. So all I have to do now is show this data on the lefth hand side. Grab the base 64 string. And then when we hit test step, we should get a binary file over here, which if we click into it, this should be our professional looking photo. Wow, that looks great. It even got the wording and like the same fonts right. So that's awesome. And by the way, if we click into the results of the create image where we did the image edit, we can see the tokens. And with this model, it is basically $10 for a million input tokens and $40 for a million output tokens. So right here, you can see the difference between our input and output tokens. And this one was pretty cheap. I think it was like 5 cents. Anyways, now that we have that image right here as binary data, we need to turn that into a video using an API called Runway. And so if we go into Runway and we go first of all, let's look at the price. For a 5second video, 25 cents. For a 10-second video, 50 cents. So that's the one we're going to be doing today. But if we go to the API reference to read how

we can turn an image into a video, what we need to look at is how we actually send over that image. And what we have to do here is send over an HTTPS URL of the image. So we somehow have to get this binary data in NADN to a public image that runway can access. So the way I'm going to be doing that is with this API that's free called image BB. And um it's a free image hosting service. And what we can do is basically just use its API to send over the binary data and we'll get back a public URL. So come here, make a free account. You'll grab your API key from up top. And then we basically have here's how we set this up. So what I'm going to do is I'm going to copy the endpoint right there. We're going to go back into naden and I'm going to add an HTTP request. And let me just configure this up. We'll put it over here just to keep everything sort of square. But now what I'm going to do in here is paste that endpoint in as our URL. You can also see that it says this call can be done using post or git. But since git requests are limited by the max amount of length, you should probably do post. So I'm just going to go back in here and change this to a post. And then there are basically two things that are required. The first one is our API key. And then the second one is the actual image. Anyways, this documentation is not super intuitive. I can sort of tell that this is a query parameter because it's being attached at the end of the endpoint with a question mark and all this kind of stuff. And that's just because I've looked at tons of API documentation. So, what I'm going to do is go into nit. We're going to add a generic credential type. It's going to be a query off. Where where was query? There we go. And then you can see I've already added my image BB. But all you're going to do is you would add the name as a key. And then you would just paste in your API key. And that's it. And now we've authenticated ourselves to the service. And then what's next is we need to send over the image in a field called image. So I'm going to go back in here. I'm going to send over a body because this allows us to actually send over n binary fields. And I'm not going to do n binary. I'm going to do form data because then we can name the field we're sending over. Like I said, not going to deep dive into how that all works, but the name is going to be image and then the input data field name is going to be data because that's how it's seen over here. And this should be it. So, real quick, I'm just going to change this to get URL. And then we're going to hit test step, which is going to send over that binary data to image BB. And it hopefully should be sending us back a URL. And it sent back three of them. I'm going to be using the middle one that's just called URL because it's like the best size and everything. You can look at the other ones if you want on your end, but this one is going to load up and we should see it's the image that we got generated for us. It takes a while to load up on that first time, but as you can see now, it's a publicly accessible URL and then we can feed it into runway. So that's exactly our next step. We're going to add another request right here. It's going to be an HTTP and this one we're going to configure to hit runway. So here's a good example of we can actually use a curl command. So I'm going to click on copy over here when I'm in the runway. Generate a video from image. Come back into Naden, hit import curl, and paste that in there and hit import. And this is going to basically configure everything we need. We just have to tweak a few things. Typically, most API documentation nowadays will have a curl command. The edit image one that we set up earlier was just a little broken. Imag is just a free service, so sometimes they don't always. But let's configure this node. So, the first thing I see is we have a header off right here. And I don't want to send it like this. I want to send it up as a generic type so I can save it. Otherwise, you'd have to go get your API key every time you wanted to use Runway. So, as you can see, I've already set up my Runway API key. So, I have it plugged in, but what you would do is you'd go get your API key from Runway. And then you'd see, okay, how do we actually send over authentication? It comes through with the name authorization. And then

the header is bearer space API key. So, similar to the last one. And then that's all you would do in here when you're setting up your runway credential. Authorization bearer space my API key. And then because we have ourselves authenticated up here, we can flick off that headers. And all we have to do now is configure the actual body. Okay, so first things first, what image are we sending over to get turned into a video in that name prompt image? We're going to get rid of that value. And I'm just going to drag in the URL that we wanted that we got from earlier, which was that picture I s I showed you guys. So now runway sees that image. Next, we have the seed, which if you want to look at the documentation, you can play with it, but I'm just going to get rid of that. Then we have the model, which we're going to be using, Gen 4 Turbo. We then have the prompt text. So, this is where we're going to get rid of this. And you're going to go back to that PDF you downloaded from my free school, and you're going to paste this prompt in there. So, this prompt basically gives us that like 3D spinning effect where it just kind of does a slow pan and a slow rotate. And that's what I was looking for. If you're wanting some other type of video, then you can tweak that prompt, of course. For the duration, if you look in the documentation, it'll say the duration only basically allows five or 10. So, I'm just going to change this one to 10. And then the last one was ratio. And I'm just going to make the square. So here are the accepted ratio values. I'm going to copy 960 by 960. And we're just going to paste that in right there. And actually before we hit test step, I've realized that we're missing something here. So back in the documentation, we can see that there's one thing up here which is required, which is a header. X-runway-version. And then we need to set the value to this. So I'm going to copy the header. And we have to enable headers. I I deleted it earlier, but we're going to enable that. So we have the version. And then I'm just going to go copy the value that it needs to be set to and we'll paste that in there as the value. Otherwise, this would not have worked. Okay, so that should be configured. But before we test it out, I want to show you guys how I set up the polling flow like this that you saw in the demo. So what we're going to do here is we need to go see like, okay, once we send over our request right here to get a video from our image, it's going to return an ID and that doesn't mean anything to us. So what we have to do is get our task. So that is the basically we send over the ID that it gives us and then it'll come back and say like the status equals pending or running or we'll say completed. So what I'm going to do is copy this curl command for getting task details. We're going to hook it up to this node as an HTTP request. We're going to import that curl. Now that's pretty much set up. We have our authorization which I'm going to delete that because as you know we just configured that earlier as a header off. So, I'm just going to come in here and grab my Runway API key. There it is. I couldn't find it for some reason. Um, we have the version set up. And now all we have to do is drag in the actual ID from the previous one. So, real quick, I'm just going to make this an expression. Delete ID. And now we're pretty much set up. So, first of all, I'm going to test this one, which is going to send off that request to runway and say, "Hey, here's our image. Here's the prompt. Make a video out of it." And as you can see, we got back an ID. Now I'm going to use this next node and I'm going to drag in that ID from earlier. And now it's saying, okay, we're going to check in on the status of this specific task. And if I hit test step, what we're going to see is that it's not yet finished. So it's going to come back and say, okay, status of this run or status of this task is running. So that's why what I'm going to do is add an if. And this if is going to be saying, okay, does this status field right here, does that equal running in all caps? Because that's what it equals right now. If yes, what we're going to do is we are going to basically wait for a certain amount of time. So here's the true branch. I'm going to wait and let's just say it's 5 seconds. So I'll just call this five seconds. I'm going to wait for 5 seconds and then I'm going to come back here and try again. So as you saw in

the demo, it basically tried again like seven or eight times. And this just ensures that it's never going to move on until we actually have a finished photo. So what you could also do is basically say does status equal completed or whatever it means when it completes. That's another way to do it. You just have to be careful to make sure that whatever you're setting here as the check is always 100% going to work. And then what you do is you would continue the rest of the logic down this path once that check has been complete. And then of course you probably don't want to have this check like 10 times every single time. So what you would do is you'd add a weight step here. And once you know about how long it takes, you'd add this here. So last time I had it at 30 seconds and it waited like eight times. So let's just say I'm going to wait 60 seconds here. So then when this flow actually runs, it'll wait for a minute, check. If it's still not done, it'll continuously loop through here and wait 5 seconds every time until we're done. Okay, there we go. So now status is succeeded. And what I'm going to do is just view this video real quick. Hopefully this one came out nicely. Let's take a look. Wow, this is awesome. Super clean. It's rotating really slowly. It's a full 10-second video. You can tell it's like a 3D image. This is awesome. Okay, cool. So now if we test this if branch, we'll see that it's going to go down the other one which is the false branch because it's actually completed. And now we can with confidence shoot off the email with our materials. So I'm going to grab a Gmail node. I'm going to click send a message. And we are going to have this configured hopefully because you've already set up your Google stuff. And now who do we send this to? We're going to go grab that email from the original form submission which is all the way down here. We're going to make the subject, which I'm just going to say marketing materials, and then a colon. And we'll just drag in the actual title of the product, which in here was cologne. I'm changing the email type to text just because I want to. Um, we're going to make the body an expression. And we're just going to say like, hey, here is your photo. And obviously this can be customized however you want. But for the photo, what we have to do is grab that public URL that we generated earlier. So right here there is the photo URL. Here is your video. And for the video, we're going to drag in the URL we just got from the output of that um runway get task check. So there is the video URL. And then I'm just going to say cheers. Last thing I want to do is down here append edit an attribution and turn that off. This just ensures that the email doesn't say this email was sent by NAN. And now if we hit test step right here, this is pretty much the end of the process. And we can go ahead and check. Uh-oh. Okay, so not authorized. Let me fix that real quick. Okay, so I just switched my credential because I was using one that had expired. So now this should go through and we'll go take a look at the email. Okay, so did something wrong. I can already tell what happened is this is supposed to be an expression and dynamically come through as the title of the product, but we accidentally somehow left off a curly brace. So, if I come back into here and and add one more curly brace right here to the description or sorry, the subject now, we should be good. I'll hit test step again. And now we'll go take a look at that email. Okay, there we go. Now, we have the cologne and we have our photo and our video. So, let's click into the video real quick. I'm just so amazed. This is this is just so much fun. It look the the lighting and the the reflections. It's it's all just perfect. And then we'll click into the photo just in case we want to see the actual image. And there it is. This also looks awesome. All right, so that's going to do it for today's video. I hope you guys enjoyed this style of walking step by step through some of the API calls and sort of my thought process as to how I set up this workflow. Okay, at this point I think you guys probably have a really good understanding of how these AI workflows actually function and you're probably getting a little bit antsy and want to build an actual AI agent. Now, so we're about to get into building your first AI agent step by step. But before that, just wanted to drive home the

concept of AI workflows versus AI agents one more time and the benefits of using workflows. But of course, there are scenarios where you do need to use an agent. So, let's break it down real quick. Everyone is talking about AI agents right now, but the truth is most people are using them completely wrong and admittedly myself included. It's such a buzzword right now and it's really cool in n to visually see your agents think about which tools they have and which ones to call. So, a lot of people are just kind of forcing AI agents into processes where you don't really need it. But in reality, a simple AI workflow is not only going to be easier to build, it's going to be more cost- effective and also more reliable in the long run. If you guys don't know me, my name's Nate. And for a while now, I've been running an agency where we deliver AI solutions to clients. And I've also been teaching people from any background how to build out these things practically and apply them to their business through deep dive courses as well as live calls. So, if that sounds interesting to you, definitely check out the community with the link in the description. But let's get into the video. So, we're going to get into Naden and I'm going to show you guys some mistakes of when I've built agents when I should have been building AI workflows. But before that, I just wanted to lay out the foundations here. So, we all know what chatbt is. At its core, it's a large language model that we talk to with an input and then it basically just gives us an output. So, if we wanted to leverage chatbt to help us write a blog post, we would ask it to write a blog post about a certain topic. It would do that and then it would give us the output which we would then just copy and paste somewhere else. And then came the birth of AI agents, which is when we actually were able to give tools to our LLM so that they could not only just generate content for us, but they could actually go post it or go do whatever we wanted to do with it. AI agents are great and there's definitely a time and a place for them because they have different tools and basically the agent will use its brain to understand, okay, I have these three tools based on what the user is asking me. Do I call this one and then do I output or do I call this one then this one or do I need to call all three simultaneously? It has that option and it has the variability there. So, this is going to be a non-deterministic workflow. But the reality is most of the processes that we're trying to enhance for our clients are pretty deterministic workflows that we can build out with something more linear where we still have the same tools. We're still using AI, but we have everything going step one, step two, step three, step four, step five, step six, which is going to reduce the variability there. It's going to be very deterministic and it's going to help us with a lot of things. So stick with me because I'm going to show you guys an AI agent video that I made on YouTube a few months back and I started re-evaluating it. Like why would I ever build out the system like that? It's so inefficient. So I'll show you guys that in a sec. But real quick, let's talk about the pros of AI workflows over AI agents. And I narrowed it down to four main points. The first one is reliability and consistency. One of the most important concepts of building an effective AI agent is the system prompt because it has to understand what its tools are, when to use each one, and what the end goal is. and it's on its own to figure out which ones do I need to call in order to provide a good output. But with a workflow, we're basically keeping it on track and there's no way that the process can sort of deviate from the guardrails that we've set up because it has to happen in order and it can't really go anywhere else. So this makes systems more reliable because there's never going to be a transfer of data between workflows where things may get messed up or incorrect mappings being sent across, you know, agent to a different agent or agent to tool. We're just basically able to go through the process linearly. So the next one is going to be cost efficiency. When we're using an agent and it has different tools, every time it hits a tool, it's going to go back to its brain. It's going to rerun through its system prompt and it's going to think about what is my next step here. And every time you're

accessing that AI agent's brain, it costs you money. So if we're able to eliminate that aspect of decision-m and just say, okay, you you finished step two, now you have to go on to step three. There's no decision to be made. We don't have to make that extra API call to think about what comes next, and we're saving money. Number three is easier debugging and maintenance. When we have an AI workflow, we can see exactly which node errors. We can see exactly what mappings are incorrect and what happened here. Whereas with an AI agent workflow, it's a little bit tougher because there's a lot of manipulating the system prompt and messing with different tool configurations. And like I said, there's data flowing between agent to tool or between agent to subworkflow. And that's where a lot of things can happen that you don't really have full visibility into. And then the final one is scalability. kind of backpacks right off of number three. But if you wanted to add more nodes and more functionality to a workflow, it's as simple as, you know, plugging in a few more blocks here and there or adding on to the back. But when you want to increase the functionality of an AI agent, you're probably going to have to give it more tools. And when you give it more tools, you're going to have to refine and add more lines to the system prompt, which could work great initially, but then previous functionality, the first couple tools you added, those might stop working or those may become less consistent. So basically, the more control that we have over the entire workflow, the better. AI is great. There are times when we need to make decisions and we need that little bit of flexibility. But if a decision doesn't have to be made, why would we leave that up to the AI to hallucinate 5 or 10% of the time when we could basically say, "Hey, this is going to be 100% consistent." Anyways, I've made a video that talks a little bit more about this stuff, as well as other things I've learned over the first 6 months of building agents. If you want to watch that, I'll link it up here. But let's hop into n and take a look at some real examples. Okay, so the first example I want to share with you guys is a typical sort of rag agent. And for some reason it always seems like the element of rag has to be associated with an agent, but it really doesn't. So what we have is a workflow where we're putting a document from Google Drive into Pine Cone. We have a customer support agent and then we have a customer support AI workflow. And both of the blue box and the green box, they do the exact same thing, but this one's going to be more efficient and we also have more control. So let's break this down. Also, if you want to download this template to play around with, you can get it for free if you go to my free school community. The link for that's down in the description as well. You'll come into here, click on YouTube resources, and click on the post associated with this video. And then the workflow will be right here for you to download. Okay, so anyways, here is the document that we're going to be looking at. It has policy and FAQ information. We've already put it into Pine Cone. As you can see, it's created eight vectors. And now what we're going to do is we're going to fire off an email to the customer support agent to see how it handles it. Okay, so we just sent off, do you offer price matching or bulk discounts? We'll come back into the workflow, hit run, and we should see the customer support agent is hitting the vector database, and it's also hitting its reply email tool. But what you'll notice is that it hit its brain. So, Google Gemini 2.0 Flash in this case, not a huge deal because it's free. But if you were using something else, it's going to have hit that API three different times, which would be three separate costs. So, let's check and see if it did this correctly. So, in our email, we got the reply, "We do not offer price matching currently, but we do run promotions and discounts regularly. Yes, bulk orders may qualify for a discount. Please contact our sales team at salestechhaven.com for inquiries. So, let's go validate that that's correct. So, in the FAQ section of this doc, we have that they don't offer price matching, but they do run promotions and discounts regularly. And then for bulk discounts, um you have to hit up the sales team. So, it answered correctly.

Okay. So, now we're going to run the customer support AI workflow down here. It's going to grab the email. It's going to search Pine Cone. It's going to write the email. I'll explain what's going on here in a sec. And then it responds to the customer. So, there's four steps here. It's going to be an email trigger. It's going to search the knowledge base. It's going to write the email and then respond to the customer in an email. So, why would we leave that up to the agent to decide what it needs to do if it's always going to happen in those four steps every time? All right, here's the email we just got in reply. As you can see, this is the one that the agent wrote, and this one looks a lot better. Hello, thank you for reaching out to us. In response to your inquiry, we currently do not offer price matching. However, we do regularly run promotions and discounts, so be sure to keep an eye out for those. That's accurate. Regarding bulk discounts, yes, they may indeed qualify for a discount. So reach out to our sales team. If you have any other questions, please feel free to reach out. Best regards, Mr. Helpful, TechHaven. And obviously, I told it to sign off like that. So, now that we've seen that, let's actually break down what's going on. So, it's the same trigger. You know, we're getting an email, and as you can see, we can find the text of the email right here, which was, "Do you guys offer price matching or bulk discounts?" We're feeding that into a pine cone node. So, if you guys didn't know, you don't even need these to be only tools. You can have them just be nodes. where we're searching for the prompts that is, do you guys offer price matching or bulk discounts? And maybe you might want an AI step between the trigger and the search to maybe like formulate a query out of the email if the email is pretty long. But in this case, that's all we did. And now we can see we got those four vectors back, same way we would have with the agent. But what's cool is we have a lot more control over it. So as you can see, we have a vector and then we have a score, which basically ranks how relevant it the vector was to the query that we sent off. And so we have some pretty low ones over here, but what we can do is say, okay, we only want to keep if the score is greater than 04. So it's only going to be keeping these two, as you can see, and it's getting rid of these two that aren't super relevant. And this is something that's a lot easier to control in this linear flow compared to having the agent try to filter through vector results up here. Anyways, then we're just aggregating however many results it pulls back. if it's four, if it's three, or if it's just one, it's still just going to aggregate them together so that we can feed it into our OpenAI node that's going to write the email. So basically, in the user prompt, we said, "Okay, here's the customer inquiry. Here's the original email, and here's the relevant knowledge that we found. All you have to do now is write an email." And so by giving this AI node just one specific goal, it's going to be more quality and consistent with its outputs rather than we gave the agent multiple jobs. It had to not only write the email, but it also had to figure out how to search through information and figure out what the next step was. So this node, it only has to focus on one thing. It has the knowledge handed to it on a silver platter to write the email with. And basically, we said, you're Mr. Helpful, a customer support rep for Tech Haven. Your job is to respond to incoming customer emails with accurate information from the knowledge base. You must only answer using relevant knowledge provided to you. Don't make anything up. We gave it the tone and then we said only output the body in a clean format. it outputs that body and then all it had to do is map in the correct message ID and the correct message content. Simple as that. So, I hope this makes sense. Obviously, it's a lot cooler to watch the agent do something like that up here, but this is basically the exact same flow and I would argue that it's going to be a lot better, more consistent, and cheaper. Okay, so now to show an example where I released this as a YouTube video and a couple weeks later I was like, why did I do it like that? So, what we have here is a technical analyst. And so basically we're talking to it through Telegram and it has one tool which is

basically going to get a chart image and then it's going to analyze the chart image and then it sends it back to us in Telegram. And this is the workflow that it's actually calling right here where we're making an HTTP request to chart- image. We're getting the chart, downloading it, analyzing the image, sending it back, and then responding back to the agent. So there's basically like two transfers of data here that we don't need because as you can see down here, we have the exact same process as one simple AI workflow. So there's going to be much much less room for error here. But first of all, let's demo how this works and then we'll demo the actual AI workflow. Okay, so it should be listening to us now. I'm going to ask it to analyze Microsoft. And as you can see, it's now hitting that tool. We won't see this workflow actually in real time just because it's like calling a different execution, but this is the workflow that it's calling down here. I can actually just it's basically calling this right here. Um, so what it's going to do is it's going to send us an image and then a second or two later it's going to send us an actual analysis. So there is Microsoft's stock chart and now it's creating that analysis as you can see right up here and then it's going to send us that analysis. We just got it. So if you want to see the full video that I made on YouTube, I'll I'll tag it right up here. But not going to dive too much into what's actually happening. I just want to prove that we can do the exact same thing down here with a simple workflow. Although right here, I did evolve this workflow a little bit. So it's it's not only looking at NASDAQ, but it can also choose different exchanges and feed that into the API call. But anyways, let's make this trigger down here active and let's just show off that we can do the exact same thing with the workflow and it's going to be better. So, test workflow. This should be listening to us. Now, I'm just going to ask it to um we'll do a different one. Analyze uh Bank of America. So, now it's getting it. It is going to be downloading the chart. Actually, want to open up Telegram so we can see downloading the chart, analyzing the image. It's going to send us that image and then pretty much immediately after it should be able to send us that analysis. So we don't have that awkward 2 to 5 second wait. Obviously we're waiting here. But as soon as this is done, we should get the both the image and the text simultaneously. There you go. And so you can see the results are basically the same. But this one is just going to be more consistent. There's no transfer of data between workflow. There's no need to hit an AI model to decide what tool I need to use. It is just going to be one seamless flow. You can al also get this workflow in the free school community if you want to play around with it. Just wanted to throw that out there. Anyways, that's going to wrap us up here. I just wanted to close off with this isn't me bashing on AI agents. Well, I guess a little bit it was. AI agents are super powerful. They're super cool. It's really important to learn prompt engineering and giving them different tools, but it's just about understanding, am I forcing an agent into something that doesn't need it? Am I exposing myself to the risk of lower quality outputs, less consistency, more difficult time scaling this thing? Things along those lines. And so that's why I think it's super important to get into something like Excal wireframe out the solution that you're looking to build. Understand what are all the steps here. What are the different API calls or different people involved? What could happen here? Is this deterministic or is there an aspect of decision-m and variability here? Essentially, is every flow going to be the same or not the same? Cool. So now that we have that whole concept out of the way, I think it's really important to understand that so that when you're planning out what type of system you're going to build, you're actually doing it the right way from the start. But now that we understand that, let's finally set up our first AI agent together. Let's move into that video. All right, so at this point you guys are familiar with Naden. You've built a few AI workflows and now it's time to actually build an AI agent, which gets even cooler. So before we actually hop into there and do that, just want to do a quick refresher on this little diagram we talked about

at the beginning of this video, which is the anatomy of an AI agent. So we have our input, we have our actual AI agent, and then we have an output. The AI agent is connected to different tools, and that's how it actually takes action. And in order to understand which tools do I need to use, it will look at its brain and its instructions. The brain comes in the form of a large language model, which in this video, we'll be using open router to connect to as many different ones as we want. and you guys have already set up your open router credentials. Then we also have access to memory which I will show you guys how we're going to set up in nadn. Then finally it uses its instructions in order to understand what to do and that is in the form of a system prompt which we will also see in naden. So all of these elements that we've talked about will directly translate to something in nen and I will show you guys and call out exactly where these are so there's no confusion. So we're going to hop in nitn and you guys know that a new workflow always starts with a trigger. So, I'm going to hit tab and I'm going to type in a chat trigger because we want to just basically be able to talk to our AI agent right here in the native Nadm chat. So, there is our trigger and what I'm going to do is click the plus and add an AI agent right after this trigger so we can actually talk to it. And so, this is what it looks like. You know, we have our AI agent right here, but I'm going to click into it so we can just talk about the difference between a user message up here and a system message that we can add down here. So going back to the example with chatbt and with our diagram when we're talking to chat gbt in our browser every single time we type and say something to chatbt that is a user message because that message coming in is dynamic every time. So you can see right here the source for the prompt that the AI agent will be listening for as if it was chatbt is the connected chat trigger node. So we're set up right here and the agent will be reading that every time. If we were feeding in information to this agent that wasn't coming from the chat message trigger, we'd have to change that. But right now, we're good. And if we go back to our diagram, this is basically the input that we're feeding into the AI agent. So, as you can see, input goes into the agent. And that's exactly what we have right here. Input going into the agent. And then we have the system prompt. So, I'm going to click back into the agent. And we can see right here, we have a system message, which is just telling this AI agent, you are a helpful assistant. So, right now, we're just going to leave it as that. And back in our diagram that is right here, its instructions, which is called a system prompt. So the next thing we can see that we need is we need to give our AI agent a brain, which will be a large language model and also memory. So I'm going to flick back into N. And you can see we have two options right here. The first one is chat model. So I'm first of all just going to click on the plus for chat model. I'm going to choose open router. And we've already connected to open router. And now I just get to choose from all of these different chat models to use. So I'm just going to go ahead and choose a GBT 4.1 Mini. And I'm just going to rename this node GPT 4.1 mini just so we know which one we're using. Cool. So now we have our input, our AI agent, and a brain. But let's give it some memory real quick, which is as simple as just clicking the plus under memory. And I'm just going to for now choose simple memory, which stores it in and it end. There's no credentials required. And as you can see, the session ID is looking for the connected chat trigger node. because we're using the connected chat trigger node, we don't have to change anything. We are good to go. So, this is basically the core part of the agent, right? So, what I can do is I can actually talk to this thing. So, I can say, "Hey," and we'll see what it says back. It's going to use its memory. It's going to um use its brain to actually answer us. And it says, "Hello, how can I assist you?" I can say, "My name is Nate. I am 23 years old." And now what I'm going to basically test is that it's storing all of this as memory and it's going to know that. So now it says, "Nice to meet you, Nate. How can I help you?" Now I'm going to ask you, you know,

what's my name and how old am I? So we'll send that off. And now it's going to be able to answer us. Your name is Nate and you are 23 years old. How can I assist you further? So first of all, the reason it's being so helpful is because its system message says you're a helpful assistant. The next piece would be it's using its brain to answer us and it's using its memory to make sure it's not forgetting stuff about our current conversation. So those are the three parts right there. Input, AI agent, brain, and instructions. And now it's time to add the tools. So in this example, we're going to build a super simple personal assistant AI agent that can do three things. It's going to be able to look in our contact database in order to grab contact information. with that contact information. It's going to be able to send an email and it's going to be able to create a calendar event. So, first thing we're going to do is we're going to set up our contact database. And what I'm going to do for that is just I have this Google sheet. Really simple. It just says name and email. This could be maybe you have your contacts in Google contacts. You could connect that or an Air Table base or whatever you want. This is just the actual tool, the actual integration that we want to make to our AI agent. So, what I'm going to do is throw in a few rows of example names and emails in here. Okay. So, we're just going to stick with these three. We've got Michael Scott, Ryan Reynolds, and Oprah Winfrey. And now, what we're going to be able to do is have our AI agent look at this contact database whenever we ask it to send an email to someone or make a calendar event with someone. If I go back and add it in, the first thing we have to do is add a tool to actually access this Google sheet. So, I'm going to click on tool. I'm going to type in Google sheet. It's as simple as that. And you can see we have a Google Sheets tool. So, I'm going to click on that. And now we have to set up our credential. You guys have already connected to Google Sheets in the previous workflow, so it shouldn't be too difficult. So choose your credential. And then the first thing is a tool description. What we're going to do is we are going to just set this automatically. And this basically describes to the AI agent what does this tool do. So we could set it manually and describe ourselves, but if you just set it automatically, the AI is going to be pretty good at understanding what it needs to do with this tool. The next thing is a resource. So what are we actually looking for? We're looking for a sheet within a document, not an entire document itself. Then the operation is we want to just get rows. So I'm going to leave it all as that. And then what we need to do is actually choose our document and then the sheet within that document that we want to look at. So for document, I'm going to choose contacts. And for sheet, there's only one. I'm just going to choose sheet one. And then the last thing I want to do is just give this actual tool a pretty intuitive name. So I'm just going to call this contacts database. There you go. So now it should be super clear to this AI agent when to use this tool. We may have to do some system prompting actually to say like, hey, here are the different tools you have. But for now, we're just going to test it out and see if it works. So what I'm going to do is open up the chat and just ask it, can you please get Oprah Winfreyy's contact information. There we go. We'll send that off and we will watch it basically think. And then there we go. Boom. It hit the Google Sheet tool that we wanted it to. And if I open up the chat, it says Oprah Winfreyy's contact information is email opra winfrey.com. If we go into the base, we can see that is exactly what we put for her contact information. Okay, so we've confirmed that the agent knows how to use this tool and that it can properly access Google Sheets. The next step now is to add another tool to be able to send emails. So, I'm going to move this thing over. I'm going to add another tool and I'm just going to search for Gmail and click on Gmail tool. Once again, we've already covered credentials. So, hopefully you guys are already logged in there. And then what we need to do is just configure the rest of the tool. So tool description set automatically resource message operation send and then we have to fill out the two the

subject the email type and the message. What we're able to do with our AI agents and tools is something super super cool. We can let our AI agent decide how to fill out these three fields that will be dynamic. And all I have to do is click on this button right here to the right that says let the model define this parameter. So I'm going to click on that button. And now we can see that it says defined automatically by the model. So basically if I said hey can you send an email to Oprah Winfrey saying this um and this it would then interpret our message our user input and it would then fill out who's this going to who's the subject and who's the email. So I'll show you guys an example of that. It's super cool. So I'm just going to click on this button for subject and also this button for message. And now we can see the actual AI use its brain to fill out these three fields. And then also I'm just going to change the email type to text because I like it how it comes through as text. So real quick, just want to change this name to send email. And all we have to do now is we're going to chat with our agent and see if it's able to send that email. All right. So I'm sending off this message that asks to send an email to Oprah asking how she's doing and if she has plans this weekend. And what happened is it went straight to the send email tool. And the reason it did that is because in its memory, it remembered that it already knows Oprah Winfreyy's contact information. So if I open chat, it says the email's been sent asking how she's doing and if she has plans this weekend. Is there anything else that you would like to do? So real quick before we go see if the email actually did get sent, I'm going to click into the tool. And what we can see is on this left hand side, we can see exactly how it chose to fill out these three fields. So for the two, it put oprafree.com, which is correct. For the subject, it put checking in. And for the message, it put hi Oprah. I hope this weekend finds you well. How are you doing? Do you have any plans? Best regards, Nate. And another thing that's really cool is the only reason that it signed off right here as best regards Nate is because once again, it used its memory and it remembers that our name is Nate. That's how it filled out those fields. Let me go over to my email and we'll take a look. So, in our sent, we have the checking in subject. We have the message that we just read in and it in. And then we have this little thing at the bottom that says this email was automatically sent by NADN. We can easily turn that off if we go into NADN. Open up the tool. We add an option at the bottom that says append naden attribution. And then we just turn off the append naden attribution. And as you can see if we click on add options, there are other things that we can do as well. Like we can reply to the sender only. We can add a sender name. We can add attachments. All this other stuff. But at a high level and real quick setup, that is the send email tool. And keep in mind, we still haven't given our agent any sort of system prompt besides saying you're a helpful assistant. So, super cool stuff. All right, cool. And now for the last tool, what we want to do is add a create calendar event. So, I'm going to search calendar and grab a Google calendar node. We already should be set up. Or if you're not, actually, all you have to do is just create new credential and sign in real quick because you already went and created your whole Google Cloud thing. We're going to leave the description as automatic. The resource is an event. The operation is create. The calendar is going to be one that we choose from our account. And now we have a few things that we want to fill out for this tool. So basically, it's asking what time is the event going to start and what time is the event going to end. So real quick, I'm just going to do the same thing. I'm going to let the model decide based on the way that we interact with it with our input. And then real quick, I just want to add one more field, which is going to be a summary. And basically whatever gets filled in right here for summary is what's going to show up as the name of the event in Google calendar. But once again we're going to let the model automatically define this field. So let's call this node create event. And actually one more thing I forgot to do is we want to add an attendee. So we can actually let

the agent add someone to an event as well. So that is the new tool. We're going to hit save. And remember no system prompts. Let's see if we can create a calendar event with Michael Scott. All right. All right. So, we're asking for dinner with Michael at 6 p.m. What's going to happen is it probably Okay, so we're going to have to do some prompting because we don't know Michael Scott's contact information yet, but it went ahead and tried to create that email. So, it said that it created the event and let's click into the tool and see what happened. So, it tried to send the event invite to michael.scottample.com. So, it completely made that up because in our contacts base, Michael Scott's email is mikegreatcott.com. So, it got that wrong. That's the first thing it got wrong. The second thing it got wrong was the actual start and end date. So, yes, it made the event for 6 p.m., but it made it for 6 p.m. on April 27th, 2024, which was over a year ago. So, we can fix this by using the system prompt. So, what I'm going to do real quick is go into the system prompt, and I'm just going to make it just an expression and open it up full screen real quick. What I'm going to say next is you must always look in the contacts database before doing something like creating an event or sending an email. You need the person's email address in order to do one of those actions. Okay, so that's a really simple thing we can add. And then also what I want to tell it is what is today's current date and time? So that if I say create an event for tomorrow or create an event for today, it actually gets the date right. So, I'm just going to say here is the current date slashtime. And all I have to do to give it access to the current date and time is do two curly braces. And then right here you can see dollar sign now which says a date time representing the current moment. So if I click on that on the right hand side in the result panel you can see it's going to show the current date and time. So we're happy with that. Our system prompt has been a little bit upgraded and now we're going to just try that exact same query again and we'll see what happens. So, I'm going to click on this little repost message button. Send it again. And hopefully now, there we go. It hits the contact database to get Michael Scott's email. And then it creates the calendar event with Michael Scott. So, down here, it says, I've created a calendar event for dinner with Michael Scott tonight at 6. If you need any more assistance, feel free to ask. So, if I go to my calendar, we can see we have a 2-hour long dinner with Michael Scott. If I click onto it, we can see that the guest that was invited was mikegreatscott.com, which is exactly what we see in our contact database. And so, you may have noticed it made this event for 2 hours because we didn't specify. If I said, "Hey, create a 15-minute event," it would have only made it 15 minutes. So, what I'm going to do real quick is a loaded prompt. Okay, so fingers crossed. We're saying, "Please invite Ryan Reynolds to a party tonight that's only 30 minutes long at 8 p.m. and send him an email to confirm." So, what happened here? It went to go create an event and send an email, but it didn't get Ryan Reynolds email first. So, if we click into this, we can see that it sent an email to ryan.rrensacample.com. That's not right. And it went to create an event at ryan.rerensacample.com. And that's not right either. But the good news is if we go to calendar, we can see that it did get the party right as far as it's 8 p.m. and only 15 minutes. So, because it didn't take the right action, it's not that big of a deal. We know now that we have to go and refine the system prompt. So to do that, I'm going to open up the agent. I'm going to click into the system prompt. And we are going to fix some stuff up. Okay. So I added two sentences that say, "Never make up someone's email address. You must look in the contact database tool." So as you guys can see, this is pretty natural language. We're just instructing someone how to do something as if we were teaching an intern. Okay. So what I'm going to do real quick is clear this memory. So I'm just going to reset the session. And now we're starting from a clean slate. And I'm going to ask that exact same query to do that multi-step thing with Ryan Reynolds. All right. Take two. We're inviting Ryan Reynolds to

a party at 9:00 p.m. There we go. It's hitting the contacts database. And now it's going to hit the create event and the send email tool at the same time. Boom. I've scheduled a 30-minute party tonight at 9:00 p.m. and invited Ryan Reynolds. So, let's go to our calendar. We have a 9 p.m. party for 30 minutes long, and it is ryanpool.com, which is exactly what we see in our contacts database. And then, if we go to our email, we can see now that we have a party invitation for tonight to ryanpool.com. But what you'll notice is now it didn't sign off as Nate because I cleared that memory. So this would be a super simple fix. We would just want to go to the system prompt and say, "Hey, when you're sending emails, make sure you sign off as Nate." So that's going to be it for your first AI agent build. This one is very simple, but also hopefully really opens your eyes to how easy it is to plug in these different tools. And it's really just about your configurations and your system prompts because system prompting is a really important skill and it's something that you kind of have to just try out a lot. You have to get a lot of reps and it's a very iterative process. But anyways, congratulations. You just built your first AI agent in probably less than 20 minutes and now add on a few more tools. Play around with a few more parameters and just see how this kind of stuff works. In this section, what I'm going to talk about is dynamic memory for your AI agents. So if you remember, we had just set up this agent and we were using simple memory and this was basically helping us keep conversation history. But what we didn't yet talk about was the session ID and what that exactly means. So basically think of a session ID as some sort of unique identifier that identifies each separate conversation. So, if I'm talking to you, person A, and you ask me something, I'm gonna go look at conversations from our conversation, person A and Nate, and then I can read that for context and then respond to you. But if person B talks to me, I'm going to go look at my conversation history with person B before I respond to them. And that way, I keep two people and two conversations completely separate. So, that's what a session ID is. So, if we were having some sort of AI agent that was being triggered by an email, we would basically want to set the session ID as the email address coming in because then we know that the agent's going to be uniquely responding to whoever actually sent that email that triggered it. So, just to demonstrate how that works, what I'm going to do is just manipulate the session ID a little bit. So, I'm going to come into here and I'm going to instead of using the chat trigger node for the session ID, I'm going to just define it below. And I'm just going to do that exact example that I just talked to you guys about with person A and person B. So I'm just going to put a lowercase A in there as the session ID key. So once I save that, what I'm going to do is just say hi. Now it's going to respond to me. It's going to update the conversation history and say hi. I'm going to say my name is um Bruce. I don't know why I thought of Bruce, but my name's Bruce. And now it says nice to meet you Bruce. How can I assist you? Now what I'm going to do is I'm going to change the session ID to B. We'll hit save. And I'm just going to say what's my name? What's my name? And it's going to say I don't have access to your name directly. If you'd like, you can provide your name or any other details you want me to know. How can I assist you today? So person A is Bruce. Person B is no name. And what I'm going to do is go back to putting the key as A. Hit save. And now if I say, "What is my name?" with a misspelled my, it's going to say, "Hey, Bruce." There we go. Your name is Bruce. How can I assist you further? And so that's just a really quick demo of how you're able to sort of actually have dynamic um conversations with multiple users in one single agent flow because you can make this field dynamic. So, what I'm going to do to show you guys a practical use of this, let's say you're wanting to connect your agent to Slack or to Telegram or to WhatsApp or to Gmail. You want the memory to be dynamic and you want it to be unique for each person that's interacting with it. So, what I have here is a Gmail trigger.

I'm going to hit test workflow, which should just pull in an email. So, when we open up this email, we can see like the actual body of the email. We can see, you know, like history. We can see a thread ID, all this kind of stuff. But what I want to look at is who is the email from? Because then if I feed this into the AI agent and first of all we would have to change the actual um user message. So we are no longer talking to our agent with the connected chat trigger node, right? We're connecting to it with Gmail. So I'm going to click to find below. The user message is basically going to be whatever you want the agent to look at. So don't even think about end right now. If you had an agent to help you with your emails, what would you want it to read? You'd want it to read maybe a combination of the subject and the body. So that's exactly what I'm going to do. I'm just going to type in subject. Okay, here's the subject down here. And I'm going to drag that right in there. And then I'm just going to say body. And then I would drag in the actual body snippet. And it's a snippet right now because in the actual Gmail trigger, we have this flicked on as simplified. If we turn that off, it would give us not a snippet. It would give us a full email body. But for right now, for simplicity, we'll leave it simplified. But now you can see that's what the agent's going to be reading every time, not the connected chat trigger node. And before we hit test step, what we want to do is we want to make the sender of this email also the session key for the simple memory. So we're going to define below and what I'm going to do is find the from field which is right here and drag that in. So now whenever we get a new email, we're going to be looking at conversation history from whoever sent that email to trigger this whole workflow. So I'll hit save and basically what I'm going to do is just run the agent. And what it's going to do is update the memory. It's going to be looking at the correct thing and it's taking some action for us. So, we'll take a look at what it does. But basically, it said the invitation email for the party tonight has been sent to Ryan. If you need any further assistance, please let me know. And the reason why it did that is because the actual user message basically was saying we're inviting Ryan to a party. So, hopefully that clears up some stuff about dynamic um user messages and dynamic memory. And now you're on your way to building some pretty cool Jetic workflows. And something important to touch on real quick is with the memory within the actual node. What you'll notice is that there is a context window length parameter. And this says how many past interactions the model receives as context. So this is definitely more of the short-term memory because it's only going to be looking at the past five interactions before it crafts its response. And this is not just with a simple memory node. What we have here is if I delete this connection and click on memory, you can see there are other types of memory we can use for our AI agents. Let's say for example we're doing Postgress which later in this course you'll see how to set this up. But in Postgress you can see that there's also a context window length. So just to show you guys an example of like what that actually looks like. What we're going to do is just connect back to here. I'm going to drag in our chat message trigger which means I'm going to have to change the input of the AI agent. So we're going to get rid of this whole um defined below with the subject and body. We're going to drag in the connected chat trigger node. Go ahead and give this another save. And now I'm just going to come into the chat and say, "Hello, Mr. Agent. What is going on here? We have the memory is messed up." So remember, I just changed the session ID from our chat trigger to the Gmail trigger um the address, the email address of whoever just sent us the email. So I'm going to have to go change that again. I'm just going to simply choose connected chat trigger node. And now it's referencing the correct session ID. Our variable is green. We're good to go. We'll try this again. Hello, Mr. Agent. It's going to talk to us. So, just save that as memory. My name is Nate. Okay. Nice to meet you, Nate. How can I assist you? My favorite color is blue. And I'm going to say, you know, tell me about myself. Okay.

So, it's using all that memory, right? We basically saw a demo of this, but it basically says, other than your name and your favorite color is blue, what else is there about you? So if I go into the agent and I click over here into the agent logs, we can see the basically the order of operations that the agent took in order to answer us. So the first thing that it does is it uses its simple memory. And that's where you can see down here, these are basically the past interactions that we've had, which was um hello Mr. Agent, my name is Nate, my favorite color is blue. And this would basically cap out at five interactions. So that's all we're basically saying in this context window length right here. So, just wanted to throw that out there real quick. This is not going to be absolutely unlimited memory to remember everything that you've ever said to your agent. We would have to set that up in a different way. All right, so you've got your agent up and running. You have your simple memory set up, but something that I alluded to in that video was setting up memory outside of NADN, which could be something like Postgress. So in this next one, we're going to walk through the full setup of creating a superbase account, connecting your Postgress and your Superbase so that you can have your short-term memory with Postgress and then you can also connect a vector database with Superbase. So let's get started. So today I'm going to be showing you guys how to connect Postgress SQL and Superbase to Nadin. So what I'm going to be doing today is walking through signing up for an account, creating a project, and then connecting them both to NADN so you guys can follow every step of the way. But real quick, Postgress is an open- source relational database management system that you're able to use plugins like PG vector if you want vector similarity search. In this case, we're just going to be using Postgress as the memory for our agent. And then Superbase is a backend as a service that's kind of built on top of Postgress. And in today's example, we're going to be using that as the vector database. But don't want to waste any time. Here we are in Naden. And what we know we're going to do here for our agent is give it memory with Postgress and access to a vector database in Superbase. So for memory, I'm going to click on this plus and click on Postgress chat memory. And then we'll set up this credential. And then over here we want to click on the plus for tool. We'll grab a superbase vector store node and then this is where we'll hook up our superbase credential. So whenever we need to connect to these thirdparty services what we have to do is come into the node go to our credential and then we want to create a new one. And then we have all the stuff to configure like our host our username our password our port all this kind of stuff. So we have to hop into superbase first create account create a new project and then we'll be able to access all this information to plug in. So here we are in Superbase. I'm going to be creating a new account like I said just so we can walk through all of this step by step for you guys. So, first thing you want to do is sign up for a new account. So, I just got my confirmation email. So, I'm going to go ahead and confirm. Once you do that, it's going to have you create a new organization. And then within that, we create a new project. So, I'm just going to leave everything as is for now. It's going to be personal. It's going to be free. And I'll hit create organization. And then from here, we are creating a new project. So, I'm going to leave everything once again as is. This is the organization we're creating the project in. Here's the project name. And then you need to create a password. And you're going to have to remember this password to hook up to our Subabase node later. So, I've entered my password. I'm going to copy this because like I said, you want to save this so you can enter it later. And then we'll click create new project. This is going to be launching up our project. And this may take a few minutes. So, um, just have to be patient here. As you can see, we're in the screen. It's going to say setting up project. So, we pretty much are just going to wait until our project's been set up. So, while this is happening, we can see that there's already some stuff that may look a little confusing.

We've got project API keys with a service ro secret. We have configuration with a different URL and some sort of JWT secret. So, I'm going to show you guys how you need to access what it is and plug it into the right places in Naden. But, as you can see, we got launched to a different screen. The project status is still being launched. So, just going to wait for it to be complete. So, everything just got set up. We're now good to connect to NAN. And what you want to do is typically you'd come down to project settings and you click on database. And this is where everything would be to connect. But it says connection string has moved. So, as you can see, there's a little button up here called connect. So, we're going to click on this. And now, this is where we're grabbing the information that we need for Postgress. So this is where it gets a little confusing because there's a lot of stuff that we need for Postgress. We need to get a host, a username, our password from earlier when we set up the project, and then a port. So all we're looking for are those four things, but we need to find them in here. So what I'm going to do is change the type to Postgress SQL. And then I'm going to go down to the transaction pooler, and this is where we're going to find the things that we need. The first thing that we're looking for is the host, which if you set it up just like me, it's going to be after the -h. So it's going to be AWS, and then we have our region.pool.subase.com. So we're going to grab that, copy it, and then we're going to paste that into the host section right there. So that's what it should look like for host. Now we have a database and a username to set up. So if we go back into that superbase page, we can see we have a D and a U. So the D is going to stay as Postgress, but for user, we're going to grab everything after the U, which is going to be postgress.com, and then these um different characters. So I'm going to paste that in here under the user. And for the password, this is where you're going to paste in the password that you use to set up your Subbase project. And then finally at the bottom, we're looking for a port, which is by default 5342. But in this case, we're going to grab the port from the transaction pooler right here, which is following the lowercase P. So we have 6543. I'm going to copy that, paste that into here as the port. And then we'll hit save. And we'll see if we got connection tested successfully. There we go. We got green. And then I'm just going to rename this so I can keep it organized. So there we go. We've connected to Postgress as our chat memory. We can see that it is going to be using the connected chat trigger node. That's how it's going to be using the key to store this information. and it's going to be storing it in a table in Subabase called Naden chat histories. So real quick, I'm going to talk to the agent. I'm just going to disconnect the subbase so we don't get any errors. So now when I send off hello AI agent, it's going to respond to us with something like hey, how can I help you today? Hello, how can I assist you? And now you can see that there were two things stored in our Postgress chat memory. So we'll switch over to superbase. And now we're going to come up here in the left and go to table editor. We can see we have a new table that we just created called NAN chat histories. And then we have two messages in here. So the first one as you can see was a human type and the content was hello AI agent which is what we said to the AI agent and then the second one was a type AI and this is the AI's response to us. So it said hello how can I assist you today. So this is where all of your chats are going to be stored based on the session ID and just once again this session ID is coming from the connected chat trigger node. So it's just coming from this node right here. As you can see, there's the session ID that matches the one in our our chat memory table. And that is how it's using it to store sort of like the unique chat conversations. Cool. Now that we have Postgress chat memory set up, let's hook up our Superbase vector store. So, we're going to drag it in. And then now we need to go up here and connect our credentials. So, I'm going to create new credential. And we can see that we need two things, a host and a service role secret. And the host is not going to be the same one as the host that we used to

set up our Postgress. So let's hop into Superbase and grab this information. So back in Superbase, we're going to go down to the settings. We're going to click on data API and then we have our project URL and then we have our service ro secret. So this is all we're using for URL. We're going to copy this, go back to Subase, and then we'll paste this in as our host. As you can see, it's supposed to be HTTPS um and then your Superbase account. So we'll paste that in and you can see that's what we have.co. Also, keep in mind this is because I launched up an organization and a project in Superbase's cloud. If you were to self-host this, it would be a little different because you'd have to access your local host. And then of course, we need our service ro secret. So back in Superbase, I'm going to reveal, copy, and then paste it into an end. So let me do that real quick. And as you can see, I got that huge token. Just paste it in. So what I'm going to do now is save it. Hopefully it goes green. There we go. We have connection tested successfully. And then once again, just going to rename this. The next step from here would be to create our Superbase vector store within the platform that we can actually push documents into. So you're going to click on docs right here. You are going to go to the quick start for setting up your vector store and then all you have to do right here is copy this command. So in the top right, copy this script. Come back into Subabase. You'll come on the lefth hand side to SQL editor. You'll paste that command in here. You don't change anything at all. You'll just hit run. And then you could should see down here success. No rows returned. And then in the table editor, we'll have a new table over here called documents. So this is where when we're actually vectorizing our data, it's going to go into this table. Okay. Okay. So, I'm just going to do a real quick example of putting a Google doc into our Subbase vector database just to show you guys that everything's connected the way it should be and working as it should. So, I'm going to grab a Google Drive node right here. I'm going to click download file. I'm going to select a file to download which in this case I'm just going to grab body shop services terms and conditions and then hit test step. And we'll see the binary data which is a doc file over here. And now we have that information. And what we want to do with it is add it to superbase superbase vector store. So, I'm going to type in superbase. We'll see vector store. The operation is going to be add documents to vector store. And then we have to choose the right credential because we have to choose the table to put it in. So this is in this case we already made a table. As you can see in our superbase it's called documents. So back in here I'm going to choose the credential I just made. I'm going to choose insert documents and I'm going to choose the table to insert it to not the N chat histories. We want to insert this to documents because this one is set up for vectorization. From there I have to choose our document loader as well as our embeddings. So I'm not really going to dive into exactly what this all means right now. If you're kind of confused and you're wanting a deeper dive on rag and building agents, definitely check out my paid community. We've got different deep dive topics about all this kind of stuff. But I'm just going to set this up real quick so we can see the actual example. I'm just choosing the binary data to load in here. I'm choosing the embedding and I'm choosing our text splitter which is going to be recursive. And so now all I have to do here is hit run. It's going to be taking that binary data of that body shop file. It split it up. And as you can see there's three items. So if we go back into our Superbase vector store and we hit refresh, we now see three items in our vector database and we have the different content and all this information here like the standard oil change, the synthetic oil change is coming from our body shop document that I have right here that we put in there just to validate the rag. And we know that this is a vector database store rather than a relational one because we can see we have our vector embedding over here which is all the dimensions. And then we have our metadata. So we have stuff like the source and um the blob type, all this kind of

stuff. And this is where we could also go ahead and add more metadata if we wanted to. Anyways, now that we have vectors in our documents table, we can hook up the actual agent to the correct table. So in here, what I'm going to call this is um body shop. For the description, I'm going to say use this to get information about the body shop. And then from the table name, we have to choose the correct table, of course. So we know that we just put all this into something called documents. So I'm going to choose documents. And finally, we just have to choose our embeddings, of course, so that it can embed the query and pull stuff back accurately. And that's pretty much it. We have our AI agent set up. So, let's go ahead and do a test and see what we get back. So, I'm going to go ahead and say what brake services are offered at the body shop. It's going to update the Postgress memory. So, now we'll be able to see that query. It hit the Superbase vector store in order to retrieve that information and then create an augmented generated answer for us. And now we have the body shop offers the following brake services. 120 per axle for replacement, 150 per axle for rotor replacement, and then full brake inspection is 30 bucks. So, if we click back into our document, we can see that that's exactly what it just pulled. And then, if we go into our vector database within Subase, we can find that information in here. But then we can also click on NAN chat history, and we can see we have two more chats. So, the first one was a human, which is what we said. What brake services are offered at the body shop? And then the second one was a AI content, which is the body shop offers the following brake services, blah blah blah. And this is exactly what it just responded to us with within NADN down here as you can see. And so keep in mind this AI agent has zero prompting. We didn't even open up the system message. All that's in here is you are a helpful assistant. But if you are setting this up, what you want to do is you know explain its role and you want to tell it you know you have access to a vector database. It is called X. It has information about X Y and Z and you should use it when a client asks about X Y and Z. Anyways that's going to be it for this one. Subase and Postgress are super super powerful tools to use to connect up as a database for your agents, whether it's going to be relational or vector databases and you've got lots of options with, you know, self-hosting and some good options for security and scalability there. Now that you guys have built an agent and you see the way that an agent is able to understand what tools it has and which ones it needs to use, what's really really cool and powerful about NAND is that we can have a tool for an AI agent be a custom workflow that we built out in Nadn or we can build out a custom agent in Naden and then give our main agent access to call on that lower agent. So what I'm about to share with you guys next is an architecture you can use when you're building multi- aent systems. It's basically called having an orchestrator agent and sub agents or parent agents and child agents. So, let's dive into it. I think you guys will think it's pretty cool. So, a multi- aent system is one where we have multiple autonomous AI agents working together in order to get the job done and they're able to talk to each other and they're able to use the tools that they have access to. What we're going to be talking about today is a type of multi- aent system called the orchestrator architecture. And basically what that means that we have one agent up here. I call it the parent agent and then I call these child agents. But we have an orchestrator agent that's able to call on different sub aents. And the best way to think about it is this agent's only goal is to understand the intent of the user. Whether that's through Telegram or through email, whatever it is, understanding that intent and then understanding, okay, I have access to these four agents and here is what each one is good at. Which one or which ones do I need to call in order to actually achieve the end goal? So, in this case, if I'm saying to the agent, can you please write me a quick blog post about dogs and send that to Dexter Morgan, and can you also create a dinner event for tonight at 6 p.m. with Michael Scott? And

thank you. Cool. So, this is a pretty loaded task, right? And can you imagine if this one agent had access to all of these like 15 or however many tools and it had to do all of that itself, it would be pretty overwhelmed and it wouldn't be able to do it very accurately. So, what you can see here is it is able to just understand, okay, I have these four agents. They each have a different role. Which ones do I need to call? And you can see what it's doing is it called the contact agent to get the contact information. Right now, it's calling the content creator agent. And now that that's finished up, it's probably going to call the calendar agent to make that event. And then it's going to call the email agent in order to actually send that blog that we had the content creator agent make. And then you can see it also called this little tool down here called Think. If you want to see a full video where I broke down what that does, you can watch it right up here. But we just got a response back from the orchestrator agent. So, let's see what it said. All right, so it said, "The blog post about dogs has been sent to Dexter Morgan. A dinner event for tonight at 6 p.m. with Michael Scott has been created. And if you need anything else, let me know." And just to verify that that actually went through, you can see we have a new event for dinner at 6 p.m. with Michael Scott. And then in our email and our scent, we can see that we have a full blog post sent to Dexter Morgan. And you can see that we also have a link right here that we can click into, which means that the content creator agent was able to do some research, find this URL, create the blog post, and send that back to the orchestrator agent. And then the orchestrator agent remembered, okay, so I need to send a blog post to Dexter Morgan. I've got his email from the contact agent. I have the blog post from the content creator agent. Now all I have to do is pass it over to the email agent to take care of the rest. So yes, it's important to think about the tools because if this main agent had access to all those tools, it would be pretty overwhelming. But also think about the prompts. So, in this ultimate assistant prompt, it's pretty short, right? All I had to say was, "You're the ultimate assistant. Your job is to send the user's query to the correct tool. You should never be writing emails or ever creating summaries or doing anything. You just need to delegate the task." And then what we did is we said, "Okay, you have these six tools. Here's what they're called. Here's when you use them." And it's just super super clear and concise. There's almost no room for ambiguity. We gave it a few rules, an example output, and basically that's it. And now it's able to interpret any query we might have, even if it's a loaded query. As you can see, in this case, it had to call all four agents, but it still got it right. And then when it sends over something to like the email agent, for example, we're able to give this specific agent a very, very specific system prompt because we only have to tell it about you only have access to these email tools. And this is just going back to the whole thing about specialization. It's not confusing. It knows exactly what it needs to do. Same thing with these other agents. You know, the calendar agent, of course, has its own prompts with its own set of calendar tools. The contact agent has its own prompt with its own set of contact tools. And then of course we have the content creator agent which has to know how to not only do research using its tavly tool but it also has to format the blog post with you know proper HTML. As you can see here there was like a title there were headings there were you know inline links all that kind of stuff. And so because we have all of this specialized can you imagine if we had all of that system prompt thrown into this one agent and gave it access to all the tools just wouldn't be good. And if you're still not convinced, think about the fact that for each of these different tasks, because we know what each agent is doing, we're able to give it a very specific chat model because, you know, like for something like content creation, I like to use cloud 3.7, but I wouldn't want to use something as expensive as cloud 3.7 just to get contacts or to add contacts to my contact database. So that's why I went with Flash here. And then for these ones, I'm using 4.1 Mini. So you're able

to have a lot more control over exactly how you want your agents to run. And so I pretty much think I hit on a lot of that, but you know, benefits of multi-agent system, more reusable components. So now that we have built out, you know, an email agent, whenever I'm building another agent ever, and I realize, okay, maybe it would be nice for this agent to have a couple email functions. Boom, I just give it access to the email agent because we've already built it and this email agent can be called on by as many different workflows as we want. And when we're talking about reusable components, that doesn't have to just mean these agents are reusable. It could also be workflows that are reusable. So, for example, if I go to this AI marketing team video, if you haven't watched it, I'll leave a link right up here. These tools down here, none of these are agents. They're all just workflows. So, for example, if I click into the video workflow, you can see that it's sending data over to this workflow. And even though it's not an agent, it still is going to do everything it needs to do and then send data back to that main agent. Similarly, with this create image tool, if I was to click into it real quick, you can see that this is not an agent, but what it's going to do is it's going to take information from that orchestrator agent and do a very specific function. That way, this main agent up here, all it has to do is understand, I have these different tools, which one do I need to use. So, reusable components and also we're going to have model flexibility, different models for different agents. We're going to have easier debugging and maintenance because like I said with the whole prompting thing, if you tried to give that main agent access to 25 tools and in the prompt you have to say here's when you use all 25 tools and it wasn't working, you wouldn't know where to start. You would feel really overwhelmed as to like how do I even fix this prompt. So by splitting things up into small small tasks and specialized areas, it's going to make it so much easier. Exactly like I just covered point number four, clear prompts logic and better testability. And finally, it's a foundation for multi-turn agents or agent memory. Just because we're sending data from main agent to sub agent doesn't mean we're losing that context of like we're talking to Nate right now or we're talking to Dave right now. We can still have that memory pass between workflows. So things get really really powerful and it's just pretty cool. Okay, so we've seen a demo. I think you guys understand the benefits here. Just one thing I wanted to throw out before we get into like a live build of a multi- aent system is just because this is cool and there's benefits doesn't mean it's always the right thing to do. So if you're forcing a multi-agent orchestrator framework into a process that could be a simple single agent or a simple AI workflow, all you're going to be doing is you're going to be increasing the latency. You're going to be increasing the cost because you're making more API calls and you're probably going to be increasing the amount of error just because kind of the golden rule is you want to eliminate as much data transfer between workflows as you can because that's where you can run into like some issues. But of course there are times when you do need dedicated agents for certain functions. So, let's get into a new workflow and build a really simple example of an orchestrator agent that's able to call on a sub agent. All right. So, what we're going to be doing here is we're going to build an orchestrator agent. So, I'm going to hit tab. I'm going to type in AI agent and we're going to pull this guy in. And we're just going to be talking to this guy using that little chat window down here for now. So, first thing we need to do as always is connect a brain. I'm going to go ahead and grab an open router. And we're just going to throw in a 4.1 mini. And I'll just change this name real quick so we can see what we're using. And from here, we're basically just going to connect to a subworkflow. And then we'll go build out that actual subworkflow agent. So the way we do it is we click on this plus under tool. And what we want to do is call nen workflow tool because you can see it says uses another nen workflow as a tool. Allows packaging any naden node as a tool. So it's super cool.

That's how we can send data to these like custom things that we built out. As you saw earlier when I showed that little example of the marketing team agent, that's how we can do it. So I'm going to click on this. And basically when you click on this, there's a few things to configure. The first one is a description of when do you use this tool. You'll kind of tell the agent that here and you'll also be able to tell a little bit in a system prompt, but you have to tell it when to use this tool. And then the next thing is actually linking the tool. So you can see we can choose from a list of our different workflows in NAN. You can see I have a ton of different workflows here, but all you have to do is you have to choose the one that you want this orchestrator agent to send data to. And one thing I want to call attention to here is this text box which says the tool will call the workflow you defined below and it will look in the last node for the response. The workflow needs to start with an execute workflow trigger. So what does this mean? Let's just go build another workflow and we will see exactly what it means. So I'm going to open up a new workflow which is going to be our sub agent. So, I'm going to hit tab to open up the nodes. And it's obviously prompting us to choose a trigger. And we're going to choose this one down here that says when executed by another workflow, runs the flow when called by the execute workflow node from a different tool. So, basically, the only thing that can access this node and send data to this node is one of these bad boys right here. So, these two things are basically just connected and data is going to be sending between them. And what's interesting about this node is you can have a couple ways that you accept data. So, by default, I usually just put it on accept all data. And this will put things into a field right here called query. But if you wanted to, you could also have it send over specific fields. So, if you wanted to only get like, you know, a phone number and you wanted to get a name and you wanted to get an email and you wanted those all to already be in three separate fields, that's how you could do that. And a practical example of that would be in my marketing team right here in the create image. You can see that I'm sending over an image title, an image prompt, and a chat ID. And that's another good example of being able to send, you know, like memory over because I have a chat ID coming from here, which is memory to the agent right here. But then I can also send that chat ID to the next workflow if we need memory to be accessed down here as well. So in this case, just to start off, we're not going to be sending over specified fields. We're just going to do accept all data and let us connect an AI agent to this guy. So I'm going to type in AI agent. We'll pull this in. The first thing we need to do is we need to change this because we're not going to be talking through the connected chat trigger node as we know because we have this trigger right here. So what we're going to do is save this workflow. So now it should actually register an end that we have this workflow. I'm going to go back in here and we're just going to connect it. So we know that it's called subwork sub aent. So grab that right there. And now you can see it says the sub workflow is set up to receive all input data. Without specific inputs, the agent will not be able to pass data to this tool. you can define the specific inputs in the trigger. So that's exactly what I just showed you guys with changing that right there. So what I want to do is show how data gets here so we can actually map it so the agent can read it. So what we need to do before we can actually test it out is we need to make sure that this orchestrator agent understands what this tool will do and when to use it. So let's just say that this one's going to be an email agent. First thing I'm going to do is just intuitively name this thing email agent. I'm then going to type in the description call this tool to take any email actions. So now it should basically, you know, signal to this guy whenever I see any sort of query come in that has to do with email. I'm just going to pass that query right off to this tool. So as you can see, I'm not even going to add a system message to this AI agent yet. We're just going to see if we can understand. And I'm going to come in here and

say, "Please send an email to Nate asking him how he's doing." So, we fire that off and hopefully it's going to call this tool and then we'll be able to go in there and see the query that we got. The reason that this errored is because we haven't mapped anything. So, what I'm going to do is click on the tool. I'm going to click on view subexecution. So, we can pop open like the exact error that just happened. And we can see exactly what happened is that this came through in a field called query. But the main agent is not looking for a field called query. It's looking for a field called chat input. So I'm just going to click on debug and editor so we can actually pull this in. Now all I have to do is come in here, change this to define below, and then just drag in the actual query. And now we know that this sub agent is always going to receive the orchestrator agents message. But what you'll notice here is that the orchestrator agent sent over a message that says, "Hi Nate, just wanted to check in and see how you're doing. Hope all is well." So there's a mistake here because this main agent ended up basically like creating an email and sending it over. All we wanted to do is basically just pass the message along. So what I would do here is come into the system prompt and I'm just going to say overview. You are an orchestrator agent. Your only job is to delegate the task to the correct tool. No need to write emails or create summaries. There we go. So just with a very simple line, that's all we're going to do. And before we shoot that off, I'm just going to go back into the sub workflow and we have to give this thing an actual brain. so that it can process messages. We're just going to go with a 4.1 mini once again. Save that. So, it actually reflects on this main agent. And now, let's try to send off this exact same query. And we'll see what it does this time. So, it's calling the email agent tool. It shouldn't error because we we we fixed it. But, as you can see now, it just called that tool twice. So, we have to understand why did it just call the sub agent twice. First thing I'm going to do is click into the main agent and I'm going to click on logs. And we can see exactly what it did. So when it called the email agent once again it sent over a subject which is checking in and an actual email body. So we have to fix the prompting there right? But then the output which is basically what that sub workflow sent back said here's a slightly polished version of your message for a warm and clear tone blah blah blah. And then for some reason it went and called that email agent again. So now it says please send the following email and it sends it over again. And then the sub agent says I can't send emails directly but here's the email you can send. So, they're both in this weird loop of thinking they are creating them an email, but not actually being able to send them. So, let's take a look and see how we can fix that. All right, so back in the sub workflow, what we want to do now is actually let this agent have the ability to send emails. Otherwise, they're just going to keep doing that endless loop. So, I'm going to add a tool and type in Gmail. We're going to change this to a send message operation. I'm just going to rename this send email. And we're just going to have the two be defined by the model. We're going to have the subject be defined by the model. and we're going to have the message be defined by the model. And all this means is that ideally, you know, this query is going to say, hey, send an email to nateample.com asking what's up. The agent would then interpret that and it would fill out, okay, who is it going to? What's the subject and what's the message? It would basically just create it all itself using AI. And the last thing I'm going to do is just turn off the Nent attribution right there. And now let's give it another shot. And keep in mind, there's no system prompt in this actual agent. And I actually want to show you guys a cool tip. So when you're building these multi- aent systems and you're doing things like sending data between flows, if you don't want to always go back to the main agent to test out like how this one's working, what you can do is come into here and we can just edit this query and just like set some mock data as if the main agent was sending over some stuff. So I'm going to say like we're pretending the orchestrator agent

sent over to the sub workflow. send an email to nate@example.com asking what's up and we'll just get rid of that R. And then now you can see that's the query. That's exactly what this agent's going to be looking at right here. And if we hit play above this AI agent, we'll see that hopefully it's going to call that send email tool and we'll see what it did. So it just finished up. We'll click into the tool to see what it did. And as you can see, it sent it to Nate example. It made the subject checking in and then the message was, "Hey Nate, just wanted to check in and see what's up. best your name. So, my thought process right now is like, let's get everything working the way we want it with this agent before we go back to that orchestrator agent and fix the prompting there. So, one thing I don't like is that it's signing off with best your name. So, we have a few options here. We could do that in the system prompt, but same thing with like um specialization. If this tool is specialized in sending emails, we might as well instruct it how to send emails in this tool. So for the message, I'm going to add a description and I'm going to say always sign off emails as Bob. And that really should do it. So because we have this mock data right here, I don't have to go and, you know, send another message. I can just test it out again and see what it's going to do. So it's going to call the send email tool. It's going to make that message. And now we will go ahead and look and see if it's signed off in a better way. Right here, we can see now it's signing off best, Bob. So, let's just say right now we're happy with the way that our sub agent's working. We can go ahead and come back into the main agent and test it out again. All right. So, I'm just going to shoot off that same message again that says, "Send an email to Nate asking him how he's doing." And this will be interesting. We'll see what it sends over. It was one run and it says, "Could you please provide Nate's email address so I can send the message?" So, what happened here was the subexecution realized we don't have Nate's email address. And that's why it basically responded back to this main agent and said, "I need that if I need to send the message." So if I click on subexecution, we will see exactly what it did and why it did that and it probably didn't even call that send email tool. Yeah. So it actually failed and it failed because it tried to fill out the two as Nate and it realized that's not like a valid email address. So then because this sub agent responds with could you please provide Nate's email address so I can send the message. That's exactly what the main agent saw right here in the response from this agent tool. So that's how they're able to talk to each other, go back and forth, and then you can see that the orchestrator agent prompted us to actually provide Nate's email address. So now we're going to try, please send an email to nativeample.com asking him how the project is coming along. We'll shoot that off and everything should go through this time and it should basically say, oh, which project are you referring to? This will help me provide you with the most accurate and relevant update. So once again, the sub agent is like, okay, I don't have enough information to send off that message, so I'm going to respond back to that orchestrator agent. And just because we actually need one to get through, let me shoot off one more example. Okay, hopefully this one's specific enough. We have an email address. We have a specified name of a project. And we should see that hopefully it's going to send this email this time. Okay, there we go. The email asking Nate how Project Pan is coming along. It's been sent. Anything else you need? So, at this point, it would be okay. Which other agents could I add to the system to make it a bit easier on myself? The first thing naturally to do would be I need to add some sort of contact agent. Or maybe I realize that I don't need a full agent for that. Maybe that needs to just be one tool. So basically what I would do then is I'd add a tool right here. I would grab an air table because that's where my contact information lives. And all I want to do is go to contacts and choose contacts. And now I just need to change this to search. So now this tool's only job is to return all of the contacts in my contact database. I'm just going to come in here and call

this contacts. And now keep in mind once again there's still nothing in the system prompt about here are the tools you have and here's what you do. I just want to show you guys how intelligent these models can be before you even prompt them. And then once you get in there and say, "Okay, now you have access to these seven agents. Here's what each of them are good at, it gets even cooler." So, let's try one more thing and see if it can use the combination of contact database and email agent. Okay, so I'm going to fire this off. Send an email to Dexter Morgan asking him if he wants to get lunch. You can see that right away it used the contacts database, pulled back Dexter Morgan's email address, and now we can see that it sent that email address over to the email agent, and now we have all of these different data transfers talking to each other, and hopefully it sent the email. All right, so here's that email. Hi, Dexter. Would you like to get lunch sometime soon? Best Bob. The formatting is a little off. We can fix that within the the tool for the email agent. But let's see if we sent that to the right email, which is dextermiami.com. If we go into our contacts database, we can see right here we have dextermorggan dextermiami.com. And like I showed you guys earlier, what you want to do is get pretty good at reading these agent logs. So you can see how your agents are thinking and what data they're sending between workflows. And if we go to the logs here, we can see first of all, it used its GPT4.1 mini model brain to understand what to do. It understood, okay, I need to go to the contacts table. So I got my contact information. Then I need to call the email agent. And what I sent over to the email agent was send an email to dextermiami.com asking him if he wants to get lunch. And that was perfect. All right. All right, so that's going to do it for this one. Hopefully this opened your eyes to the possibilities of these multi- aent systems in N&N and also hopefully it taught you some stuff because I know all of this stuff is like really buzzwordy sometimes with all these agents agents agents but there are use cases where it really is the best path but it's all about like understanding what is the end goal and how do I want to evolve this workflow and then deciding like what's the best architecture or system to use. So that was one type of architecture for a multi-agent system called the orchestrator architecture. But that's not the only way to have multiple agents within a workflow or within a system. So in this next section, I'm going to break down a few other architectures that you can use so that you can understand what's possible and which one fits your use case best. So let's dive right in. So in my ultimate assistant video, we utilize an agentic framework called parent agent. So as you can see, we have a parent agent right here, which is the ultimate assistant that's able to send tasks to its four child agents down here, which are different workflows that we built out within NAND. If you haven't seen that video, I'll tag it right up here. But how it works is that the ultimate assistant could get a query from the human and decide that it needs to send that query to the email agent, which looks like this. And then the email agent will be able to use its tools in Gmail and actually take action. From there, it responds to the parent agent and then the parent agent is able to take that response back from this child agent and then respond to us in Telegram. So, it's a super cool system. It allows us to delegate tasks and also these agents can be activated in any specific order. It doesn't always have to be the same. But is this framework always the most effective? No. So today I'm going to be going over four different agentic frameworks that you can use in your end workflows. The first one we're going to be talking about is prompt chaining. The second one is routing. The third one is parallelization. And the fourth one is evaluator optimizer. So we're going to break down how they all work, what they're good at. But make sure you stick around to the end because this one, the evaluator optimizer, is the one that's got me most excited. So before we get into this first framework, if you want to download these four templates for free so you can follow along, you can do so by joining my free school community. You'll come in here, click on

YouTube resources, click on the post associated with this video, and then you'll have the workflow right here to download. So, the link for that's down in the description. There's also going to be a link for my paid community, which is if you're looking for a more hands-on approach to learning NAND. We've got a great community of members who are also dedicated to learning NAN, sharing resources, sharing challenges, sharing projects, stuff like that. We've got a classroom section where we're going over different deep dive topics like building agents, vector databases, APIs, and HTTP requests. And I also just launched a new course where I'm doing the step-by-step tutorials of all the videos that I've shown on YouTube. And finally, we've got five live calls per week to make sure that you're getting questions answered, never getting stuck, and also networking with individuals in the space. We've also got guest speakers coming in in February, which is super exciting. So, I'd love to see you guys in these calls. Anyways, back to the video here. The first one we're going to be talking about is prompt chaining. So, as you can see, the way this works, we have three agents here, and what we're doing is we're passing the output of an agent directly as the input into the next agent, and so on so forth. So, here are the main benefits of this type of workflow. It's going to lead to improved accuracy and quality because each step focuses on a specific task which will help reduce errors and hallucinations. Greater control over each step. We can refine the system prompt of the outline writer and then we can refine the prompt of the evaluator. So we can really tweak what's going on and how data is being transferred. Specialization is going to lead to more effective agents. So as you can see in this example, we're having one agent write the outline. One of them evaluates the outline and makes suggestions. And then finally, we pass that revised outline to the blog writer who's in charge of actually writing the blog. So this is going to lead to a much more cohesive, thought through actual blog in the end compared to if we would just fed in all of this system prompt into one agent. And then finally, with this type of framework, we've got easier debugging and optimization because it's linear. We can see where things are going wrong. Finally, it's going to be more scalable and reusable as we're able to plug in different agents wherever we need them. Okay, so what we have to do here is we're just going to enter in a keyword, a topic for this um blog. So, I'm just going to enter in coffee, and we'll see that the agents start going to work. So, the first one is an outline writer. Um, one thing that's also really cool about this framework and some of the other ones we're going to cover is that because we're splitting up the different tasks, we're able to utilize different large language models. So, as you can see, the outline writer, we gave 20 Flash because it's it's free. Um, it's it's powerful, but not super powerful, and we just need a brief outline to be written here. And then we can pass this on to the next one that uses 40 Mini. It's a little more powerful, a little more expensive, but still not too bad. and we want this more powerful chat model to be doing the evaluating and refining of the outline. And then finally, for the actual blog writing content, we want to use something like Claw 3.5 or even Deepseek R1 because it's going to be more powerful and it's going to take that revised outline and then structure a really nice blog post for us. So that's just part of the specialization. Not only can we split up the tasks, but we can plug and play different chat models where we need to rather than feeding everything through one, you know, one Deep Seeker, one blog writer at the very beginning. So, this one's finishing up here. It's about to get pushed into a Google doc where we'll be able to go over there and take a look at the blog that it got for us about coffee. So, looks like it just finished up. Here we go. Detailed blog post based on option one, a comprehensive guide to coffee. Here's our title. Um, we have a rich history of coffee from bean to cup. We have um different methods. We have different coffee varieties. We have all this kind of stuff, health benefits and risks. Um, and as you can see, this pretty much was a four-page blog.

We've got a conclusion at the end. Anyways, let's dive into what's going on here. So, the concept is passing the output into the input and then taking that output and passing it into the next input. So, here we have here's the topic to write a blog about which all it got here was the word coffee. That's what we typed in. The system message is that you are an expert outline writer. Your job is to generate a structured outline for a blog post with section titles and key points. So, here's the first draft at the outline using 20 flash. Then, we pass that into an outline evaluator that's using for Mini. We said here's the outline. We gave it the outline of course and then the system message is you're an expert blog evaluator. Your job is to revise this outline and make sure it hits these four criteria which are engaging introduction, clear section breakdown, logical flow, and then a conclusion. So we told it to only output the revised outline. So now we have a new outline over here. And finally, we're sending that into a Claude 3.5 blog writer where we gave it the revised outline and just said, "You're an expert blog writer. Generate a detailed blog post using this outline with well ststructured paragraphs and engaging content." So that's how this works. You can see it will be even more powerful once we hook up, you know, like some internet search functionality and if we added like an editor at the end before it actually pushed it into the the Google doc or whatever it is. But that's how this framework works. But let's move into aic framework number two. Now we're going to talk about the routing framework. In this case, we have an initial LLM call right here to classify incoming emails. And based on that classification, it's going to route it up as high priority, customer support, promotion, their finance, and billing. And as you can see, there's different actions that are going to take place. We have different agents depending on what type of message comes through. So the first agent, which is the text classifier here, basically just has to decide, okay, which agent do I need to send this email off to? Anyways, why would you want to use routing? Because you're going to have an optimized response handling. So as we can see in this case, we're able to set up different personas for each of our agents here rather than having one general AI response agent. Then this can be more scalable and modular. It's going to be faster and more efficient. And then you can also introduce human escalation for critical issues like we do up here with our high priority agent. And finally, it's just going to be a better user experience for for your team and also your customers. So I hit test step. What we're getting here is an email that I just sent to myself that says, "Hey, I need help logging into my account. Can you help me?" So this email classifier is going to label this as customer support. As soon as we hit play, it's going to send it down the customer support branch right here. As you can see, we got one new item. What's going on in this step is that we're just labeling it in our Gmail as a customer support email. And then finally, we're going to fire it off to the customer support agent. In this case, this one is trained on customer support activities. Um, this is where you could hook up a customer support database if you needed. And then what it's going to do is it's going to create an email draft for us in reply to the email that we got. So, let's go take a look at that. So, here's the email we got. Hey, I need help logging into my account. As you can see, our agent was able to label it as customer support. And then finally, it created this email, which was, "Hey, Nate, thanks for reaching out. I'd be happy to assist you with logging into your account. Please provide me with some more details um about the issue you're experiencing, blah blah blah." And then this one signs off, "Best regards, Kelly, because she's the customer support rep." Okay, let's take a look at a different example. Um, we'll pull in the trigger again and this time we're going to be getting a different email. So, as you can see, this one says, "Nate, this is urgent. We need your outline tomorrow or you're fired." So, hopefully this one gets labeled as high priority. It's going to go up here to the high priority branch. Once again, we're going to label that email as high priority. But instead of activating

an email draft reply tool, this one has access to a Telegram tool. So, what it's going to do is text us immediately and say, "Hey, this is the email you got. You need to take care of this right away." Um, and obviously the logic you can choose of what you want to happen based on what route it is, but let's see. We just got telegram message, urgent email from Nate Hkelman stating that an outline is needed by tomorrow or there will be serious consequences, potential termination. So that way it notifies us right away. We're able to get into our email manually, you know, get get caught up on the thread and then respond how we need to. And so pretty much the same thing for the other two. Promotional email will get labeled as promotion. We come in here and see that we are able to set a different persona for the pro promotion agent, which is you're in charge of promotional opportunities. Your job is to respond to inquiries in a friendly, professional manner and use this email to send reply to customer. Always sign off as Meredith from ABC Corp. So, each agent has a different sort of persona that it's able to respond to. In finance agent, we have we have this agent signing off as a as Angela from ABC Corp. Um, anyways, what I did here was I hooked them all up to the same chat model and I hooked them all up to the same tool because they're all going to be sending an email draft here. As you can see, we're using from AAI to determine the subject, the message, and the thread ID, which it's going to pull in from the actual Gmail trigger, or sorry, the Gmail trigger is not using from AAI. We're we're mapping in the Gmail trigger because every time an email comes through, it can just look at that um email in order to determine the thread ID for sending out an email. But you don't have to connect them up to the same tool. I just did it this way because then I only had to create one tool. Same thing with the different chat models based on the, you know, importance of what's going through each route. You could switch out the chat models. We could have even used a cheaper, easier one for the classification if we wanted to, but in this case, I just hooked them all up to a 40 mini chat model. Anyways, this was just a really simple example of routing. You could have 10 different routes, you could have just two different routes, but the idea is that you're using one agent up front to determine which way to send off the data. Moving on to the third framework, we've got parallelization. What we're going to do here is be using three different agents, and then we're going to merge their outputs, aggregate them together, and then feed them all into a final agent to sort of, you know, throw it all into one response. So what this is going to do is give us faster analysis rather than processing everything linearly. So in this case we're going to be sending in some input and then we're going to have one agent analyze the emotion behind it, one agent do the intent behind it, and then one agent analyze any bias rather than doing it one by one. They're all going to be working simultaneously and then throwing their outputs together. So it can decrease the latency there. They're going to be specialized, which means we could have specialized system prompts like we do here. We also could do specialized um large language models again where we could plug in different different models if we wanted to maybe feed through the same prompt use cloud up here, OpenAI down here and then you know DeepSeek down here and then combine them together to make sure we're getting the best thoughtout answer. Um comprehensive review and then more scalability as well. But how this one's going to work is we're putting in an initial message which is I don't trust the mainstream media anymore. They always push a specific agenda and ignore real issues. People need to wake up and stop believing everything they see on the news. So, we're having an emotion agent, first of all, analyze the emotional emotional tone, categorize it as positive, neutral, negative, or mixed with a brief explanation. The intent agent is going to analyze the intent behind this text, and then finally, the bias agent is going to analyze this text for any potential bias. So, we'll hit this off. Um, we're going to get those three separate analysises um or analysis, and then we're going to

be sending that into a final agent that's going to basically combine all those outputs and then write a little bit of report based on our input. So, as you can see, right now, it's waiting here for um the input from the bias agent. Once that happens, it's going to get aggregated, and now it's being sent into the final agent, and then we'll take a look at um the report that we got in our Google doc. Okay, just finished up. Let's hop over to Docs. We'll see we got an emotional tone, intent, and bias analysis report. Overview is that um the incoming text has strong negative sentiment towards mainstream media. Yep. Emotional tone is negative sentiment. Intent is persuasive goal. Um, the bias analysis has political bias, generalization, emotional language, lack of evidence. Um, it's got recommendations for how we can make this text more neutral, revised message, and then let's just read off the conclusion. The analysis highlights a significant level of negativity and bias in the original message directed towards mainstream media. By implementing the suggested recommendations, the author can promote a more balanced and credible perspective that encourages critical assessment of media consumption, blah blah blah. So, as you can see, that's going to be a much better, you know, comprehensive analysis than if we would have just fed the initial input into an agent and said, "Hey, can you analyze this text for emotion, intent, and bias?" But now, we got that split up, merged together, put into the final one for, you know, a comprehensive review and an output. And it's going to turn the the, you know, data in into data out process. It's going to be a lot more efficient. Finally, the one that gets me the most excited, um, the evaluator optimizer framework, where we're going to have an evaluator agent decide if what's passing through is good or not. If it's good, we're fine, but if it's not, it's going to get optimized and then sent back to the evaluator for more evaluation. And this is going to be an endless loop until the evaluator agent says, "Okay, finally, it's good enough. We'll send it off." So, if you watch my human in the loop video, it's going to be just like that where we were providing feedback and we were the ones basically deciding if it was good to go or not. But in this case, we have an agent that does that. So it's going to be optimizing all your workflows on the back end without you being in the loop. So obviously the benefits here are that it's going to ensure high quality outputs. It's going to reduce errors and manual review. It's going to be flexible and scalable. And then it's going to optimize the AI's performance because it's sort of an iterative approach that um you know focuses on continuous improvement from these AI generated responses. So what we're doing here is we have a biography agent. What we told this agent to do is um basically write a biography. You're an expert biography writer. You'll receive information about a person. Your job is to create an entire profile using the information they give you. And I told it you're allowed to be creative. From there, we're setting the bio. And we're just doing this here so that we can continue to feed this back over and over. That way, if we have five revisions, it'll still get passed every time. The most recent version to the agent and also the most recent version when it's approved will get pushed up here to the Google doc. Then we have the evaluator agent. What we told this agent to do is um evaluate the biography. Your job is to provide feedback. We gave a criteria. So, make sure that it includes a quote from the person. Make sure it's light and humorous and make sure it has no emojis. Only need to output the feedback. If the biography is finished and all criteria are met, then all you need to output is finished. So, then we have a check to say, okay, does the output from the evaluator agent say finished or is it feedback? If it's feedback, it's going to go to the optimizer agent and continue on this loop until it says finished. Once it finally says finished, as you can see, we set JSON.output, output which is the output from the evaluator agent equals finished. When that happens, it'll go up here and then we'll see it in our Google doc. But then what we have in the actual optimizer agent is we're giving it the biography and this is where we're referencing the set

field where we earlier right here where we set the bio. This way the optimizer agent's always getting the most updated version of the bio. And then we're also going to get the feedback. So this is going to be the output from the evaluator agent because if it does go down this path, the evaluator agent, it means that it output feedback rather than saying finished. So it's getting feedback, it's getting the biography, and then we're saying you're an expert reviser. Your job is to take the biography and optimize it based on the feedback. So it gets all it needs in the user message, and then it outputs us a better optimized version of that biography. Okay, so let's do an example real quick. Um, if you remember in the biography agent, well, all we have to do is give it a, you know, some information about a person to write a biography on. So, I'm going to come in here and I'm just going to say Jim 42 um, lives by the ocean. Okay, so that's all we're going to put in. We'll see that it's writing a brief biography right now. And then we're going to see it get evaluated. We're going to see if it, you know, met those criteria. If it doesn't, it's going to get sent to the optimizer agent. the optimizer agent is going to get um basically the criteria it needs to hit as well as the original biography. So here's the evaluator agent. Look at that. It decides that it wasn't good enough. Now it's being sent to the optimizer agent who is going to optimize the bio, send it back and then hopefully on the second run it'll go up and get published in the docs. If it's not good enough yet, then it will come back to the agent and it will optimize it once again. But I think that this agent will do a good job. There we go. We can see it just got pushed up into the doc. So let's take a look at our Google doc. Here's a biography for Jim Thompson. He lives in California. He's 43. Um, ocean enthusiast, passion, adventure, a profound respect for nature. It talks about his early life, and obviously he's making all this up. Talks about his education, talks about his career, talks about his personal life. Here we have a quote from Jim, which is, "I swear the fish are just as curious about me as I am about them." We've even got another quote. Um, a few dad jokes along the way. Why did the fish blush? Because it saw the ocean's bottom. So, not sure I completely get that one. Oh, no. I get that one. Um anyways then hobbies, philosophy, legacy and a conclusion. So this is you know a pretty optimized blog post. It meets all the criteria that we had put into our agents as far as you know this is what you need to evaluate for. It's very light. There's no emojis. Threw some jokes in there and then it has some quotes from Jim as well. So as you can see all we put in was Jim 43 lives by the ocean and we got a whole basically a story written about this guy. And once again just like all of these frameworks pretty much you have the flexibility here to change out your model wherever you want. So let's say we don't really mind up front. we could use something really cheap and quick and then maybe for the actual optimizer agent we want to plug in something a little more um you know with reasoning aspect like deepse R1 potentially. Anyways, that's all I've got for you guys today. Hope this one was helpful. Hope this one, you know, sparked some ideas for next time you're going into edit end to build an agentic workflow. Maybe looking at I could actually have structured my workflow in this framework and it would have been a little more efficient than the current way I'm doing it. Like I said, these four templates will be in the free school community if you want to download them and just play around with them to understand what's going on, understand, you know, when to use each framework, stuff like that. All right, so we understand a lot of the components that actually go into building an effective agent or an effective agent system, but we haven't really yet spent a lot of time on prompting, which is like 80% of an agent. It's so so important. So, in this next section, we're going to talk about my methodology when it comes to prompting a tools agent, and we're going to do a quick little live prompting session near the end. So, if that sounds good to you, let's get started. Building AI agents and hooking up different tools is fun and all, but the quality and consistency of the performance of your

agents directly ties back to the quality of the system prompt that you put in there. Anyways, today what we're going to be talking about is what actually goes into creating an effective prompt so that your agents perform as you want them to. I'm going to be going over the most important thing that I've learned while building out agents and prompting them that I don't think a lot of people are doing. So, let's not waste any time and get straight into this one. All right, so I've got a document here. If you want to download this one to follow along or just have it for later, you can do so by joining my free school community. The link for that's down in the description. You'll just click on YouTube resources and find the post associated with this video and you'll be able to download the PDF right there. Anyways, what we're looking at today is how we can master reactive prompting for AI agents in NAD. And the objective of this document here is to understand what prompting is, why it matters, develop a structured approach to reactive prompting when building out AI agents, and then learn about the essential prompt components. So, let's get straight into it and start off with just a brief introduction. What is prompting? Make sure you stick around for this one because once we get through this doc, we're going to hop into NN and do some live prompting examples. So, within our agents, we're giving them a system prompt. And this is basically just coding them on how to act. But don't be scared of the word code because we're just using natural language instead of something like Python or JavaScript. A good system prompt is going to ensure that your agent is behaving in a very clear, very specific, and a very repeatable way. So, instead of us programming some sort of Python agent, what we're doing is we're just typing in, "You're an email agent. Your job is to assist the user by using your tools to take the correct action." Exactly as if we were instructing an intern. And why does prompting matter? I'm sure by now you guys already have a good reason in your head of why prompting matters, and it's pretty intuitive, but let's think about it like this as well. Agents are meant to be running autonomously, and they don't allow that back and forth interaction like chatbt. Now, yes, there can be some human in the loop within your sort of agentic workflows, but ideally you put in an input, it triggers the automation, triggers the agent to do something, and then we're getting an output. Unlike chatbt where you ask it to help you write an email, and you can say, "Hey, make that shorter," or you can say, "Make it more professional." We don't have that um luxury here. We just need to trust that it's going to work consistently and high quality. So, our goal as prompters is to get the prompts right the first time so that the agent functions correctly every single time it's triggered. So, the key rule here is to keep the prompts clear, simple, and actionable. You don't want to leave any room for misinterpretation. Um, and also, less is more. Sometimes I'll see people just throw in a novel, and that's just obviously going to be more expensive for you, and also just more room to confuse the agent. So, less is more. So, now let's get into the biggest lesson that I've learned while prompting AI agents, which is prompting needs to be done reactively. I see way too many people doing this proactively, throwing in a huge system message, and then just testing things out. This is just not the way to go. So let's dive into what that actually means to be prompting reactively. First of all, what is proactive prompting? This is just writing a long detailed prompt up front after you have all your tools configured and all of the sort of, you know, standard operating procedures configured and then you start testing it out. The problem here is that you don't know all the possible edge cases and errors in advance and debugging is going to be a lot more difficult because if something breaks, you don't know which part of the prompt is causing the issue. You may try to fix something in there and then the issue originally you were having is fixed, but now you cause a new issue and it's just going to be really messy as you continue to add more and more and you end up just confusing both yourself and the agent. Now, reactive prompting on the other hand is just

starting with absolutely nothing and adding a tool, testing it out, and then slowly adding sentence by sentence. And as you've seen in some of my demos, we're able to get like six tools hooked up, have no prompt in there, and the agent's still working pretty well. At that point, we're able to start adding more lines to make the system more robust. But the benefits here of reactive prompting are pretty clear. The first one is easier debugging. You know exactly what broke the agent. Whether that's I added this sentence and then the automation broke. All I have to do is take out that sentence or I added this tool and I didn't prompt the tool yet. So that's what caused the automation to break. So I'm just going to add a sentence in right here about the tool. This is also going to lead to more efficient testing because you can see exactly what happens before you hard prompt in fixes. And essentially, you know, I'll talk about hard prompting more later, but essentially what it is is um you're basically seeing an error and then you're hard prompting in the error within the system prompt and saying, "Hey, like you just did this. That was wrong. Don't do that again." And we can only do that reactively because we don't know how the agent's going to react before we test it out. Finally, we have the benefit that it prevents over complicated prompts that are hard to modify later. If you have a whole novel in there and you're getting errors, you're not going to know where to start. You're going to be overwhelmed. So, taking it step by step, starting with nothing and adding on things slowly is the way to go. And so, if it still isn't clicking yet, let's look at a real world example. Let's say you're teaching a kid to ride a bike. If you took a proactive approach, you'd be trying to correct the child's behavior before you know what he or she is going to do. So, if you're telling the kid to keep your back straight, lean forward, you know, don't tilt a certain way, that's going to be confusing because now the kid is trying to adjust to all these things you've said and it doesn't even know what it was going to do, what he or she was going to do in the in the beginning. But if you're taking a reactive approach and obviously maybe this wasn't the best example cuz you don't want your kid to fall, but you let them ride, you see what they're doing, you know, if they're leaning too much to the left, you're going to say, "Okay, well, maybe you need to lean a little more to the right to center yourself up." um and only correct what they actually need to have corrected. This is going to be more effective, fewer unnecessary instructions, and just more simple and less overwhelming. So, the moral of the story here is to start small, observe errors, and fix one problem at a time. So, let's take a look at some examples of reactive prompting that I've done in my ultimate assistant workflow. As you can see right here, I'm sure you guys have seen that video by now. If you haven't, I'll link it right up here. But, I did a ton of reactive prompting in here because I have one main agent calling four different agents. And then within those sub agents, they all have different tools that they need to call. So this was very very reactive when I was prompting this workflow or this system of agents. I started with no persistent prompt at all. I just connected a tool and I tested it out to see what happened. So an an example would be I hooked up an email agent, but I didn't give it in any instructions and I running the AI to see if it will call the tool automatically. A lot of times it will and then it only comes to when you add another different agent that you need to prompt in, hey, these are the two agents you have. Here's when to use each one. So anyways, adding prompts based on errors. Here I have my system prompts. So if you guys want to pause it and read through, you can take a look. But you can see it's very very simple. I've got one example. I've got basically one brief rule and then I just have all the tools it has and when to use them. And it's very very concise and not overwhelming. And so what I want you guys to pay attention to real quick is in the overview right here. I said, you know, you're the ultimate personal assistant. Your job is to send the user's query to the correct tool. That's all I had at first. And then I was getting this error where I was saying, "Hey, write an email to Bob." And

what was happening is it wasn't sending that query to the email tool, which is supposed to do. It itself was trying to write an email even though it has no tool to write an email. So then I reactively came in here and said, "You should never be writing emails or creating event summaries. You just need to call the correct tool." And that's not something I could have proactively put in there because I didn't really expect the agent to be doing that. So I saw the error and then I basically hardcoded in what it should not be doing and what it should be doing. So another cool example of hard coding stuff in is using examples. You know, we all understand that examples are going to help the agent understand what it needs to do based on certain inputs and how to use different tools. And so right here you can see I added this example, but we'll also look at it down here because I basically copied it in. What happened was the AI failed in a very specific scenario. So I added a concrete example where I gave it an input, I showed the actions it should take, and then I gave it the output. So in this case, what happened was I asked it to write an email to Bob and it tried to send an email or try it tried to hit the send email agent, but it didn't actually have Bob's email address. So the email didn't get sent. So what I did here was I put in the input, which was send an email to Bob asking him what time he wants to leave. I then showed the two actions it needs to take. The first one was use the contact agent to get Bob's email. Send this email address to the email agent tool. And then the second action is use the email agent to send the email. And then finally, the output that we want the personal assistant to say back to the human is, "The email has been sent to Bob. Anything else I can help you with?" The idea here is you don't need to put examples in there that are pretty intuitive and that the agent's going to get right already. You only want to put in examples where you're noticing common themes of the agents failing to do this every time. I may as well hardcode in this example input and output and tool calls. So, step four is to debug one error at a time. always change one thing and one thing only at a time so you know exactly what you changed that broke the automation. Too too often I'll see people just get rid of an entire section and then start running things and now it's like okay well we're back at square one because we don't know exactly what happened. So you want to get to the point where you're adding one sentence, you're hitting run and it's either fixing it or it's not fixing it and then you know exactly what to do. You know exactly what broke or fixed your automation. And so one thing honestly I want to admit here is I created that system prompt generator on my free school community. Um, and really the idea there was just to help you with the formatting because I don't really use that thing anymore because the fact that doing that is very proactive in the sense that we're dropping in a sort of a query into chat GBT, the custom GPT I built, it's giving us a system prompt and then we're putting that whole thing in the agent and then just running it and testing it. And in that case, you don't know exactly what you should change to fix little issues. So, just wanted to throw that out there. I don't really use that system prompt generator anymore. I now always like handcraft my prompts. Anyways, from there, what you want to do is scale up slowly. So once you confirm that the agent is consistently working with its first tool and its first rule in its prompt, then you can slowly add more tools and more prompt rules. So here's an example. You'll add a tool. You'll add a sentence in the prompt about the tool. Test out a few scenarios. If it's working well, you can then add another tool and keep testing out and slowly adding pieces. But if it's not, then obviously you'll just hard prompt in the changes of what it's doing wrong and how to fix that. From there, you'll just test out a few more scenarios. Um, and then you can just kind of rinse and repeat until you have all the functionality that you're looking for. All right, now let's look at the core components of an effective prompt. Each agent you design should follow a structured prompt to ensure clarity, consistency, and efficiency. Now, there's a ton of different types of prompting you can do

based on the role of agent. Ultimately, they're going to fall under one of these three buckets, which is toolbased prompting, conversational prompting, or like content creation type prompting, and then categorization/ealuation prompting. And the reason I wanted to highlight that is because obviously if we're creating like a content creation agent, we're not going to say what tools it has if it has no tools. But um yeah, I just wanted to throw that out there. And another thing to keep in mind is I really like using markdown formatting for my prompts. As you can see these examples, we've got like different headers with pound signs and we're able to specify like different sections. We can use bolded lists. We can use numbered lists. I've seen some people talk about using XML for prompting. I'm not a huge fan of it because um as far as human readability, I think markdown just makes a lot more sense. So that's what I do. Anyways, now let's talk about the main sections that I include in my prompts. The first one is always a background. So whether this is a role or a purpose or a context, I typically call it something like an overview. But anyways, just giving it some sort of background that defines who the agent is, what its overall goal is. And this really sets the foundation of, you know, sort of identifying their persona, their behavior. And if you don't have the section, the agent is kind of going to lack direction and it's going to generate really generic or unfocused outputs. So set its role and this could be really simple. You can kind of follow this template of you are a blank agent designed to do blank. Your goal is blank. So you are a travel planning AI assistant that helps users plan their vacations. Your goal is to pro provide detailed personalized travel itineraries based on the user's input. Then we have tools. This is obviously super super important when we're doing sort of non-deterministic agent workflows where they're going to have a bunch of different tools and they have to use their brain, their chat model to understand which tool does what and when to use each one. So, this section tells the agent what tools it has access to and when to use them. It ensures the AI selects the right tool for the right task. And a well structured tools section prevents confusion and obviously makes AI more efficient. So, here's an example of what it could look like. We have like the markdown header of tools and then we have like a numbered list. We're also showing that the tools are in bold. This doesn't have to be the way you do it, but sometimes I like to show them in bold. Um, and it's you can see it's really simple. It's it's not too much. It's not overwhelming. It's not too um you know, it's just very clear. Google search, use this tool when the user asks for real-time information. Email sender, use this tool when the user wants to send a message. Super simple. And what else you can do is you can define when to use each tool. So right here we say we have a contact database. Use this tool to get contact information. You must use this before using the email generator tool because otherwise it won't know who to send the email to. So you can actually define these little rules. Keep it very clear within the actual tool layer of the prompt. And then we have instructions. I usually call them rules as you can see. Um, you could maybe even call it like a standard operating procedure. But what this does, it outlines specific rules for the agent to follow. It dictates the order of operations at a high level. Just keep in mind, you don't want to say do this in this order every time because then it's like, why are you even using an agent? The whole point of an agent is that it's, you know, it's taking an input and something happens in this black box where it's calling different tools. It may call this one twice. It may call this one three times. It may call them none at all. Um, the idea is that it's variable. It's not deterministic. So, if you're saying do it and this every time, then you should just be using a sequential workflow. It shouldn't even be an agent. But obviously, the rules section helps prevent misunderstandings. So, here's like a high level instruction, right? You're greeting the user politely. If the user provides incomplete information, you ask follow-up questions. Use the available tools only when necessary. Structure your response in clear, concise

sentences. So, this isn't saying like you do this in this order every time. It's just saying when this happens, do this. If this happens, do that. So, here's an example for AI task manager. When a task is added, you confirm with the user. If a deadline is missing, ask the user to specify one. If a task priority is high, send a notification. Store all tasks in the task management system. So, it's very clear, too. Um, we don't need all these extra filler words because remember, the AI can understand what you're saying as long as it has like the actual context words that have meaning. You don't need all these little fillers. Um, you don't need these long sentences. So, moving on to examples, which you know, sample inputs and outputs and also actions within those between the inputs and outputs. But this helps the AI understand expectations by showing real examples. And these are the things that I love to hard code in there, hard prompt in there. Because like I said, there's no point in showing an example if the AI was already going to get that input and output right every time. You just want to see what it's messing up on and then put an example in and show it how to fix itself. So more clear guidance and it's going to give you more accurate and consistent outputs. Here's an example where we get the input that says, can you generate a trip plan for Paris for 5 days? The action you're going to take is first call the trip planner tool to get X, Y, and Z. Then you're going to take another action which is calling the email tool to send the itinerary. And then finally, the output should look something like this. Here's a 5-day Paris itinerary. Day 1, day 2, day 3, day 4, day 5. And then I typically end my prompts with like a final notes or important reminders section, which just has like some miscellaneous but important reminders. It could be current date and time, it could be rate limits, it could be um something as simple as like don't put any emojis in the output. Um, and sometimes why I do this is because something can get lost within your prompt. And sometimes like I I've thrown the today's date up top, but then it only actually realizes it when it's in the bottom. So playing around with the actual like location of your things can be sometimes help it out. Um, and so having a final notes section at the bottom, not with too many notes, but just some quick things to remember like always format responses as markdown. Here's today's date. If unsure about an answer, say I don't have that information. So just little miscellaneous things like that. Now, I wanted to quickly talk about some honorable mentions because like I said earlier, the prompt sections and components varies based on the actual type of agent you're building. So, in the case of like a content creator agent that has no tools, um you wouldn't give it a tool section, but you may want to give it an output section. So, here's an output section that I had recently done for my voice travel agent. Um, which if you want to see that video, I'll drop a link right here. But what I did was I just basically included rules for the output because the output was very specific with HTML format and it had to be very structured and I wanted horizontal lines. So I created a whole section dedicated towards output format as you can see. And because I used three pound signs for these subsections, the agent was able to understand that all this rolled up into the format of the output section right here. So anyways, I said the email should be structured as HTML that will be sent through email. Use headers to separate each section. Add a horizontal line to each section. Um, I said what it what should be in the subject. I said what should be in the introduction section. I said how you should list these departure dates, return dates, flights for the flight section. Um, here's something where I basically gave it like the HTML image tag and I showed how to put the image in there. I showed to make I said like make a inline image rather than um an attachment. I said to have each resort with a clickable link. I also was able to adjust the actual width percentage of the image by specifying that here in the prompt. Um, so yeah, this was just getting really detailed about the way we want the actual format to be structured. You can see here we have activities that I actually misspelled in my agent, but it didn't matter.

Um, and then finally just a sign off. And then just some final additional honorable mentions, something like memory and context management, um, some reasoning, some error handling, but typically I think that these can be just kind of one or two sentences that can usually go in like the rules or instructions section, but it depends on the use case, like I said. So, if it needs to be pretty robust, then creating an individual section at the bottom called memory or error handling could be worth it. It just depends on, like I said, the actual use case and the goal of the agent. Okay, cool. So, now that we've got through that document, let's hop into Nitn and we'll just do some really quick examples of some reactive live prompting. Okay, so I'm going to hit tab. I'm going to type in AI agent. We're going to grab one and we're going to be communicating with it through this connected chat trigger node. Now, I'm going to add a chat model real quick just so we can get set up up and running. We have our 40 Mini. We're good to go. And just a reminder, there is zero assistant prompt in here. All it is is that you are a helpful assistant. So, what's the first thing to do is we want to add a tool. Test it out. So, I'm going to add a um Google calendar tool. I'm just going to obviously select my calendar to pull from. I'm going to, you know, fill in those parameters using the model by clicking that button. And I'm just going to say this one's called create event. So, we have create event. And so, now we're going to do our test and see if the tool is working properly. I'm going to say create an event for tonight at 700 p.m. So send this off. We should see the agents able to understand to use this create event tool because it's using an automatic description. But now we see an issue. It created the start time for October 12th, 2023 and the end time for also October 12th, 2023. So this is our first instance of reactive prompting. It's calling the tool correctly. So we don't really need to prompt in like the actual tool name yet. Um it's probably best practice just to just to do so. But first, I'm just going to give an overview and say you are a calendar. Actually, no. I'm just going to say you are a helpful assistant because that's all it is right now. And we don't know what else we're adding into this guy. But now we'll just say tools is create event just so it's aware. Use this to create an event. And then we want to say final notes. um here is the current date and time because that's where it messed up is because it didn't know the current date and time even though it was able to call the correct tool. So now we'll just send this same thing off again and that should have fixed it. We reactively fixed the error and um we're just making sure that it is working as it should now. Okay, there we go. It just hit the tool and it says the event has been created for tonight at 7 p.m. And if I click into my calendar, you can see right there we have the event that was just created. So cool. Now that's working. What we're going to do now is add another tool. So, we'll drag this one over here. And let's say we want to do a send email tool. We're going to send a message. We're going to change the name to send email. And just so you guys are aware like how it's able to know right here, tool description, we're setting automatically. If we set manually, we would just say, you know, use this tool to send an email. But we can just keep it simple. Leave it as set automatic. I'm going to turn on to subject and message as defined by the model. And that's going to be it. So now we just want to test this thing again before we add any prompts. We'll say send an email to bobacample.com asking what's up. We'll send this off. Hopefully it's hitting the right tool. So we should see there we go. It hit the send email tool and the email got sent. We can come in here and check everything was sent correctly. Although what we noticed is it's signing off as best placeholder your name and we don't want to do that. So let's come in here and let's add a tool section for this tool and we'll tell it how to how to act. So send email. That's another tool it has. And we're going to say use this to send an email. Then we're going to say sign off emails as Frank. Okay. So that's reactively fixing an error we saw. I'm just now going to send off that same query. We already know that it knows how to call the tool. So it's going to do

that once again. There we go. We see the email was sent. And now we have a sign off as Frank. So that's two problems we've seen. And then we've added one super short line into the system prompt and fixed those problems. Now let's do something else. Let's say in Gmail we want to be able to label an email. And in order to label an email, as you can see, add label to a message, we need a message ID and we need a label name or an ID for that label. And this is we could choose from a list, but more realistically, we want the label ID to be pulled in dynamically. So if we need to get these two things, what we have to do is first get emails and also get labels. So first I'm going to do get many. I'm going to say this is using, you know, we're we're calling this tool get get emails. And then we don't want to return all. We want to do a limit. And we also want to choose from a sender. So we'll have this also be dynamically chosen. So cool. We don't have a system prompt in here about this tool, but we're just going to say get my last email from Nate Herkman. So we'll send that off. It should be hitting the get emails tool, filling in Nate Herkman as the sender. And now we can see that we just got this email with a subject hello. We have the message ID right here. So that's perfect. And now what we need to do is we need to create a tool to get the label ID. So I'm going to come in here and I'm going to say um get many and we're going to go to label. We're going to do um actually we'll just return all. That works. There's not too many labels in there. Um and we have to name this tool of course. So we're going to call this get labels. So once again there's no tools in or no prompt in here about these two tools at all. and we're gonna say get my email labels. We'll see if it hits the right tool. There we go. It did. And it is going to basically just tell us, you know, here they are. So, here are our different labels. Um, and here are the ones that we created. So, promotion, customer support, high priority, finance, and billing. Cool. So, now we can try to actually label an email. So, that email that we just got from um from Nate Hkelman that said hello, let's try to label that one. So, I'm going to add another Gmail tool, and this one's going to be add a label to a message. And we need the message ID and the label ID. So, I'm just going to fill these in with the model parameter, and I'm going to call this tool add label. So, there's no prompting for these three tools right here, but we're going to try it out anyway and see what happens. So, add a promotion label to my last email from Nate Herklman. Send that off. See what happens? It's getting emails. It tried to add a label before. So, now we're kind of We got in that weird loop. As you can see, it tried to add a label before it got labels. So, it didn't know what to do, right? Um, we'll click into here. We'll see that I don't really exactly know what happened. Category promotions. Looking in my inbox, anything sent from Nate Hkelman, we have the email right here, but it wasn't accurately labeled. So, let's go back into our agent and prompt this thing a little bit better to understand how to use these tools. So, I'm going to basically go into the tools section here and I'm going to tell it about some more tools that it has. So, get emails, right? This one was it was already working properly and we're just saying use this to get emails. Now, we have to add get labels. We're just saying use this to get labels. Um, and we know that we want it to use this before actually trying to add a label, but we're not going to add that yet. We're going to see if it can work with a more minimalistic prompt. And then finally, I'm going to say add labels. And this one is use this tool to add a label to an email. Okay. So now that we just have very basic tool descriptions in here, we don't actually say like when to use it or how. So I'm going to try this exact same thing again. Add a promotion label to my last email from Nate HKman. Once again, it tried to use ad label before and it tried to just call it twice as you can see. So not working. So back in email, I just refreshed and you can see the email is still not labeled correctly. So, let's do some more reactive prompting. What we're going to do now is just say in order to add labels, so in the description of the ad label tool, I'm going to say you must first use get emails to get the message ID. And

actually, I want to make sure that it knows that this is a tool. So, what I'm going to do is I'm going to put it in in a quote, and I'm going to make it the exact same capitalization as we defined over here. So you must first use get emails to get the message ID of the email to label. Then you must use get labels to get the label ID of the email to label. Okay. So we added in this one line. So if it's still not working, we know that this line wasn't enough. I'm going to hit save and I'm going to try the exact same thing again. Add a promotion label to my last email. So it's getting now it's getting labels and now it still had an error with adding labels. So, we'll take a look in here. Um, it said that it did it successfully, but obviously didn't. It filled in label 127 blah blah blah. So, I think the message ID is correct, but the label ID is not. So, what I'm going to try now is reactively prompting in here. I'm going to say the label ID of the email to label. We'll try that. We'll see if that fixes it. It may not. We'll have to keep going. So, now we'll see. It's going to at least it fixed the order, right? So, it's getting emails and getting labels first. And now look at that. We successfully got a labeled email. As you can see, we have our um maybe we didn't. We'll have to go into Gmail and actually check. Okay, never mind. We did. As you can see, we got the promotion email for this one from Nate Hookman that says hello. And um yeah, that's just going to be a really cool simple example of how we sort of take on the process of running into errors, adding lines, and being able to know exactly what caused what. So, I know the video was kind of simple and I went through it pretty fast, but I think that it's going to be a good lesson to look back on as far as the mindset you have and approaching reactively prompting and adding different tools and testing things because at the end of the day, building agents is a super super testheavy iterative, you know, refining process of build, test, change, build, test, change, all that kind of stuff. All right, so these next sections are a little bit more miscellaneous, but cool little tips that you can play around with with your AI agents. We're going to be talking about output parsing, human in the loop, error workflows, and having an agent have a dynamic brain. So, let's get into it. All right, so output parsing. Let's talk about what it actually means and why you need to use it. So, just to show you guys what we're working with, I'm just going to come in here real quick and ask our agent to create an email for us. And when it does this, the idea is that it's going to create a subject and a body so that we could drag this into a Gmail node. So actually before I ask it to do that, let's just say we're we're dragging in a Gmail node and we want to have this guy send an email. We're if I can find this node which is right up here. Okay, send a message. Now what you can see is that we have different fields that we need to configure the two, the subject and the message. So ideally when we're asking the agent to create an email, it will be able to output those three different things. So let me just show an example of that. Please send an email to nateample.com asking what's up. We need the to the subject and the email. Okay, so ideally we wouldn't say that every time because um we would have that in the system prompt. The issue is this workflow doesn't let you run if the node is errored and it's errored because we didn't fill out stuff. So I'm just going to resend this message. But let me show you guys exactly why we need to use an output parser. So it outputs the two to the subject and the message. And if we actually click into it though, the issue is that it comes through all in one single, you know, item called output. And so you can see we have the two, the subject, and the message. And now if I went into here to actually map these variables, I couldn't have them separated or I would need to separate them in another step because I want to drag in the dynamic variable, but I can only reference all of it at once. So that's not good. That's not what we want. That's why we need to connect an output parser. So I'm going to click into here. And right here there's an option that says require specific output format. And I'm going to turn that on. What that does is it just gave us another option to our AI agent. So typically we basically right here have chat model, memory,

and tool. But now we have another one called output parser. So this is awesome. I'm going to click onto the output parser. And you can see that we have basically three options. 99.9% of the time you are just going to be using a structured output parser which means you're able to give your agent basically a defined JSON schema and it will always output stuff in that schema if you need to have it kind of be a little bit automatically fixed with AI like I said I almost never have to use this but that's what you would use the autofixing output parser for so if I click on the structured output parser what happens is right now we see a JSON example so if we were to talk to our agent and say hey can you um you know tell me some information about California. It would output the state in one string item called state and then would also output an array of cities LA, San Francisco, San Diego. So what we want to do is we want to quickly define to our AI agent how to output information and we know that we wanted to output based on this node. We need a two, we need a subject, and we need a message. So don't worry, you're not going to have to write any JSON yourself. I'm going to go to chatgbt and say, "Help me write a JSON example for a structured output parser in NADN. I need the AI agent to output a two field, a subject field, and a body field." We'll just go ahead and send this off. And as you guys know, all LLMs are trained really well on JSON. It's going to know exactly what I'm asking for here. And all I'm going to have to do is copy this and paste that in. So once this finishes up, it's very simple. Two, subject body. And it's being a little extra right now and giving me a whole example body, but I just have to copy that. I have to go into here and just replace that JSON example. Super simple. And now hopefully I don't even have to prompt this guy at all. And we'll give it a try. But if it's not working, what we would do is we would prompt in here and say, "Hey, here is basically how we want you to output stuff. Here's your job." All that kind of stuff, right? But let me just resend this message. We'll take a look. We'll see that it called its output parser because this is green. And now let's activate the Gmail node and click in. Perfect. So what we see on this left hand side now is we have a two, we have a subject, and we have a body, which makes this so much easier to actually map out over here and drag in. So in different agents in this course, you're going to see me using different structured output parsers, whether that is to get to subject and body, whether that is to create different stories, stuff like that. Let me just show one more quick example of like a different way you could use this. I'm going to delete this if I can actually delete it. And we are going to just change up the structure output parser. So let's say we want an AI agent to create a story for us. So I'm going to just talk to this guy again. Help me write a different JSON example where I want to have the agent output a title of the story, an array of characters, and then three different scenes. Okay. So we'll send that off and see what it does. And just keep in mind it's creating this JSON basically a template that's telling your agent how to output information. So we would basically say, "Hey, create me a story about um a forest." And it would output a title, three characters, and three different scenes as you can see here. So we'll copy this. We'll paste this into here. And once again, I'm not even going to prompt the agent. And let's see how it does. Please create me a story about an airplane. Okay, we'll go ahead and take a look at what this is going to do. This one's going to spin a little bit longer. Oh, wow. Didn't even take too long. So, it called the structured output parser. And now, let's click into the agent and see how it output. Perfect. So, we have the title is the adventure of Skyward the airplane. We have four characters, Skyward, Captain Jane, Navigator Max, and ground engineer Leo. And then you can see we have four different scenes that each come with a scene number and a description. So if we wanted to, we could have this be like, you know, maybe we want an image prompt for each of these scenes. So we can feed that into an image generation model and we would just have to go into that chatbt and say, "Hey, for each scene, add another field called image

prompt." And it would just basically take care of it. So just wanted to show you how this works, how easy it is to set up these different JSON structured output parsers and why it's actually valuable to do within. So hopefully that opened your eyes a little bit. Appreciate your time. Okay, our workflow is actively listening for us in Telegram and I'm going to ask it to make an expost about coffee at night. So, as you can see, this first agent is going to search the internet using Tavi and create that initial X post for us. Now, we just got a message back in our Telegram that says, "Hey, is this post good to go?" Drinking coffee at night can disrupt your sleep since caffeine stays in your system for hours, often leading to poorer sleep quality. So, what I'm going to do is click on respond. And this gives us the ability to give our agent feedback on the post that it initially created. So, here is that response window and I'm going to provide some feedback. So, I'm telling the agent to add at the end of the tweet unless it's decaf. And as soon as I hit submit, we're going to see this go down the path. It's going to get classified as a denial message. And now the revision agent just made those changes and we have another message in our telegram with a new X post. So now, as you can see, we have a new post. I'm going to click on respond and open up that window. And what we can see here is now we have the changes made that we requested. At the end, it says unless it's decaf. So now all we have to do is respond good to go. And as soon as we submit this, it's going to go up down the approval route and it's going to get submitted and posted to X. So, here we go. Let's see that in action. I'll hit submit and then we're going to watch it get posted onto X. And let's go check and make sure it's there. So, here's my beautiful X profile. And as you can see, I was playing around with some tweets earlier. But right here, we can see drinking coffee at night can disrupt your sleep. We have the most recent version because it says unless it's decaf. And then we can also click into the actual blog that Tavi found to pull this information from. So, now that we've seen this workflow in action, let's break it down. So the secret that we're going to be talking about today is the aspect of human in the loop, which basically just means somewhere along the process of the workflow. In this case, it's happening right here. The workflow is going to pause and wait for some sort of feedback from us. That way, we know before anything is sent out to a client or posted on social media, we've basically said that we 100% agree that this is good to go. And if the initial message is not good to go, we have the ability to have this unlimited revision loop where it's going to revise the output over and over until we finally agree that it's good to go. So, we have everything color coded and we're going to break it down as simple as possible. But before we do that here, I just wanted to do a real quick walkthrough of a more simple human in the loop because what's going on up here is it's just going to say, "Do you like this?" Yes or no compared to down here where we actually give textbased feedback. So, we'll break them both down, but let's start up here real quick. And by the way, if you want to download the template for free and play around with either of these flows, you can get that in my free school community. The link for that will be down in the description. And when it comes to human in the loop in Naden, if you click on the plus, you can see down here, human in the loop, wait for approval or human input before continuing. You click on it, you can see there's a few options, and they all just use the operation called send and wait for response. So obviously there's all these different integrations, and I'm sure more will even start to roll out, but in this example, we're just using Telegram. Okay, so taking a look at this more simple workflow, we're going to send off the message, make an expost about AI voice agents. What's happening is the exact same thing as the demo where this agent is going to search the web and then it's going to create that initial content for us. And now we've hit that human in the loop step. As you can see, it's spinning here purple because it's waiting for our approval. So in our Telegram, we see the post. It asks us if this is good to go. And let's just

say that we don't like this one, and we're going to hit decline. So when I hit decline, it goes down this decision point where it basically says, you know, did the human approve? Yes or no. If yes, we'll post it to X. If no, it's going to send us a denial message, which basically just says post was denied. Please submit another request. And so that's really cool because it gives us the ability to say, okay, do we like this? Yes. And it will get posted. Otherwise, just do nothing with it. But what if we actually want to give it feedback so that it can take this post? We can give it a little bit of criticism and then it will make another one for us and it just stays in that loop rather than having to start from square one. So that's exactly what I did down here with the human and loop 2.0 know where we're able to give textbased feedback instead of just saying yes or no. So now we're going to break down what's going on within every single step here. So what I'm going to do is I'm going to click on executions. I'm going to go to the one that we did in the live demo and bring that into the workflow so we can look at it. So what we're going to do is just do another live run and walk through step by step the actual process of this workflow. So I'm going to hit test workflow. I'm going to pull up Telegram and then I'm going to ask it to make us an expost. Okay, so I'm about to fire off make me an expost about crocodiles. So sent that off. This expost agent is using its GPT41 model as well as Tavly search to do research, create that post, and now we have the human in the loop waiting for us. So before we go look at that, let's break down what's going on up front. So the first phase is the initial content. This means that we have a telegram trigger, and that's how we're communicating with this workflow. And then it gets fed into the first agent here, which is the expost agent. Let's click into the expost agent and just kind of break down what's going on here. here. So, the first thing to notice is that we're looking for some sort of prompt. The agent needs some sort of user message that it's going to look at. In this case, we're not doing the connected chat trigger node. We're looking within our Telegram node because that's where the text is actually coming through. So, on this lefth hand side, we can see all I basically did was right here is the text that we typed in, make me an expost about crocodiles. And all I did was I dragged this right into here as the user message. And that is what the agent is actually looking at in order to take action. And then the other thing we did was gave the agent a system message which basically defines its behavior. And so here's what we have. The overview is you are an AI agent responsible for creating expost based on a user's request. Your instructions are to always use the Tavly search tool to find accurate information. Write an informative engaging tweet, include a brief reference to the source directly in the tweet and only output the tweet. We listed its tool which it only has one called tavly search and we told it to use this for real-time web search and then just gave it an example basically saying okay here's an input you may get here's the action you will take and then here's the output that we want you to output and then we just gave them final notes and I know I may be read through this pretty quick but keep in mind you can download the template for free and the prompt will be in there and then what you could do is you can click on the logs for an agent and you can basically look at its behavior so we can see that it used its chat model GBT4.1 read through the system prompt decided said, "Okay, I need to go use tably search." So, here's how it searched for crocodile information. And then it used its model again to actually create that short tweet right here. And then we'll just take a quick look at what's going on within the actual Tavi search tool here. So, if you download this template, all you'll have to do is plug in your own credential. Everything else should be set up for you. But, let me just break it down real quick. So, if you go to tavly.com and create an account, you can get a,000 free searches per month. So, that's the kind of plan I'm on. But anyways, here is the documentation. You can see right here we have the Tavi search endpoint which is right here. All we have to do is authorize ourselves. So we'll have an

authorization as a header parameter and then we'll do bearer space our API token. So that's how you'll set up your own credential. And then all I did was I copied this data field into the HTTP request. And this is where you can do some configuration. You can look through the docs to see how you want to make this request. But all I wanted to do here was just change the search query. So back in end you can see in my body request I I changed the query by using a placeholder. Right here it says use a placeholder for any data to be filled in by the model. So I changed the query to a placeholder called search term. And then down here I defined the search term placeholder as what the user is searching for. So what this means is the agent is going to interpret our query that we sent in telegram. It's then going to use this tavly tool and basically use its brain to figure out what should I search for. And in this case on the lefth hand side you can see that it filled out the search term with latest news or facts about crocodiles. And then we get back our response with information and a URL. And then it uses all of this in order to actually create that post. Okay. So, here's where it may seem like it's going to get a little tricky, but it's not too bad. Just bear with me. And I wanted to do some color coding here so we could all sort of stay on the same page. So, what we're doing now is we're setting the post. And this is super important because we need to be able to reference the post later in the workflow. whether that's when we're actually sending it over to X or when we're making a revision and we need the revision agent to look at the original post as well as the feedback from the human. So in the set node, all we're doing is we're basically setting a field called post and we're dragging in a variable called JSON.output. And this just means that it's going to be grabbing the output from this agent or the revision agent no matter what. As you can see, it's looped back into this set because if we're defining a variable using dollar sign JSON, it means that we're going to be looking for whatever node immediately finished right before this one. And so that's why we have to keep this one kind of flexible because we want to make sure that at the end of the day, if we made five or six or seven revisions, that only the most recent version will actually be posted on X. So then we move into the human in the loop phase of this workflow. And as you can see, it's still spinning. It's been spinning this whole time while we've been talking, but it's waiting for our response. So anyways, it's a send and wait for a response operation. As you can see right here, the chat ID is coming from our Telegram trigger. So if I scroll down in the Telegram trigger on the lefth hand side, you can see that I have a chat ID right here. And all I did was I dragged this in right here. Basically just meaning, okay, whoever communicates with this workflow, we need to send and get feedback from that person. So that's how we can make this dynamic. And then I just made my message basically say, hey, is this good to go? And then I'm dragging in the post that we set earlier. So this is another reason why it's important is because we want to request feedback on the most recent version as well, not the first one we made. And then like I mentioned within all of these human in the loop nodes, you have a few options. So you can do free text, which is what we're doing here. Earlier what we did was approval, which is basically you can say, hey, is there an approve button? Is there an approve and a denial button? How do you want to set that up? But this is why we're doing free text because it allows for us to actually give feedback, not just say yes or no. Cool. So what we're going to do now is actually give our feedback. So, I'm going to come into here. We have our post about crocodiles. So, I'm going to hit respond and it's going to open up this new page. And so, yes, it's a little annoying that this form has to pop up in the browser rather than natively in Telegram or whatever, you know, Slack, Gmail, wherever you're doing the human in the loop, but I'm sure that'll be a fix that'll come soon. But, it's just right now, I think it's coming through a web hook. So, they just kind of have to do it like this. Anyways, let's say that we want to provide some feedback and say make this shorter. So, I'm going to say make this shorter.

And as I submit it, you're going to see it go to this decision point and then it's going to move either up or down. And this is pretty clearly a denial message. So we'll watch it get denied and go down to the revision agent as you can see. And just like that that quickly, we already have another one to look at. So before we look at it and give feedback, let's just look at what's actually going on within this decision point. So in any automation, you get to a point where you have to make a decision. And what's really cool about AI automation is now we can use AI to make a decision that typically a computer couldn't because a typical decision would be like is this number greater than 10 or less than 10. But now it can read this text that we submitted. Make this shorter and it can say okay is this approved or declined. And basically I just gave it some short definitions of like what an approval message might look like and what a denial message might look like. And you can look through that if you download the template. But as you can see here it pushed this message down the declined branch because we asked it to make a revision. And so it goes down the denial branch which leads into the revision agent. And this one's really really simple. All we did here was we gave it two things as the user message. We said here's the post to revise. So as you can see it's this is the initial post that the first agent made for us. And then here is the human feedback. So it's going to look at this. It's going to look at this and then it's going to make those changes because all we said in the system prompt was you're an expert Twitter writer. Your job is to take an incoming post and revise it based on the feedback that the human submitted. And as you can see here is the output. it made the tweet a lot shorter. And that's the beauty of using the set node is because now we loop that back in. The most recent version has been submitted to us for feedback. So, let's open that up real quick in our Telegram. And now you can see that the shorter tweet has been submitted to us. And it's asking for a response. So, at this point, let's say we're good to go with this tweet. I'm going to click respond. Open up this tab. Recent crocodile attacks in Indonesia. Highlight the need for caution in their habitats. Stay safe. We've got a few emojis. And I'm just going to say, let's just say send it off because it can interpret multiple ways of saying like yes, it's good to go. So, as soon as I hit submit, we're going to watch it go through the decision point and then post on our X. So, you see that right here, text classifier, and now it has been posted to X. And I just gave our X account a refresh. You can see that we have that short tweet about recent crocodile attacks. Okay, so now that we've seen another example of a live run through, a little more detailed, let me talk about why I made the color coding like this. So the set note here, its job is basically just I'm going to be grabbing the most recent version of the post because then I can feed it into the human in the loop. I can then feed that into the revision if we need to make another revision because you want to be able to make revisions on top of revisions. You don't want to be only making revisions on the first one. Otherwise, you're going to be like, what's the point? And then also, of course, you want to post the most recent version, not the original one, because again, what's the point? So in here, you can see there's two runs. The first one was the first initial content creation and then the second one was the revised one. Similarly, if we click into the next node which was request feedback, the first time we said make this shorter and then the second time we said send it off. And then if we go into the next node which was the text classifier, we can see the first time it got denied because we said make this shorter and the second time it said send it off and it got approved. And that's basically the flow of you know initial creation. We're setting the most recent version. We're getting feedback. We're making a decision using AI. And as you can tell for the text classifier, I'm using 2.0 Flash rather than GPT 4.1. And then of course, if it's approved, it gets posted. If it's not, it makes revisions. And like I said, this is unlimited revisions. And it's revisions on top of revisions. So when it comes to Human in the

Loop, you can do it in more than just Telegram, too. So if you click on the plus, you can see right here, Human in the Loop, wait for approval or human input before continuing. We've got Discord, Gmail, Chat, Outlook, Telegram, Slack. We have a lot of stuff you can do. However, so far with my experience, it's been limited to one workflow. And what do I mean by that? It's kind of tough to do this when you're actually giving an agent a tool that's supposed to be waiting for human approval. So, let me show you what I mean by that. Okay, so here's an agent where I tried to do a human in the loop tool because we have the send and wait message operation as a tool for an agent. So, let me show you what goes on here. We'll hit test workflow. Okay, so I'm going to send off get approval for this message. Hey John, just wanted to see if you had the meeting minutes. And you're going to watch that it's going to call the get approval tool, but here's the issue. So, it's waiting for a response, right? And we have the ability to respond, but the waiting is happening at the agent level. It really should be waiting down here for the tool because, as you saw in the previous example, the response from this should be the actual feedback from the human. And we haven't submitted that yet. And right now, the response from this tool is literally just the message. So, what you'll see here is if I go back into Telegram and I click on respond and we open up this tab, it basically just says no action required. And if I go back into the workflow, you can see it's still spinning here and there's no way for this to give another output. So, it just doesn't really work. And so, what I was thinking was, okay, why don't I just make another workflow where I just use the actual node like we saw on the previous one. That should work fine because then it should just spin down here on the tool level until it's ready. And so, let me show you what happens if we do that. Okay, so like I said, I built a custom tool down here which is called get approval. It would be sending the data to this workflow, it would send off an approval message using the send and wait, and it doesn't really work. I even tried adding a wait here. But what happens typically is when you use a workflow to call another one, it's going to be waiting and looking in the last node of that workflow for the response, but it doesn't yet work yet with these operations. And I'll show you guys why, and I'm sure NNN will fix this soon, but it's just not there yet. So, I'm just going to send off the exact same query, get approval for this message. We'll see it call the tool and basically as you can see it finished up instantly and now it's waiting here and we already did get a message back in telegram which basically said ready to go hey John just wanted to see if you had the meeting minutes and it gives us the option to approve or deny and if we click into the subworkflow this one that it actually sent data to. We can see that the execution is waiting. So this workflow is properly working because it's waiting here for human approval. But if we go back into the main flow it's waiting here at the agent level rather than waiting here. So, there's no way for the agent to actually get our live feedback and use that to take action how it needs to. So, I just wanted to show you guys that I had been experimenting with this as a tool. It's not there yet, but I'm sure it will be here soon. And when it is, you can bet that I'll have a video out about it. Today, I'm going to be showing you guys how you can set up an error workflow in NAN so that you can log all of your errors as well as get notified every time one of your active workflows fails. The cool part is all we have to do is set up one error workflow and then we can link that one to all of our different active workflows. So I think you'll be pretty shocked how quick and easy this is to get set up. So let's get into the video. All right, so here's the workflow that we're going to be using today as our test workflow that we're going to purposely make error and then we're going to capture those errors in a different one and feed that into a Google sheet template as well as some sort of Slack or email notification. And if you haven't seen my recent video on using this new think tool in edit then I'll tag it right up here. Anyways, in order for a workflow to trigger an error workflow, it has to be active. So, first things first, I'm going to make this

workflow active. There we go. This one has been activated. And now, what I'm going to do is go back out to my NAD. We're going to create a new workflow. And this is going to be our error logger workflow. Okay. So, you guys are going to be pretty surprised by how simple this workflow is going to be. I'm going to add a first step. And I'm going to type an error. And as you can see, there's an error trigger, which says triggers the workflow when another workflow has an error. So, we're going to bring this into the workflow. We don't have to do anything to configure it. You can see that what we could do is we could fetch a test event just to see what information could come back. But what we're going to do is just trigger a live one because we're going to get a lot more information than what we're seeing right here. So, quickly pay attention to the fact that I named this workflow error logger. I'm going to go back into my ultimate assistant active workflow. Up in the top right, I'm going to click on these three dots, go down to settings, and then right here, there's a setting called error workflow, which as you can see, a second workflow to run if the current one fails. The second workflow should always start with an error trigger. And as you saw, we just set that up. So, all I have to do is choose a workflow. I'm going to type an error and we called it error logger. So I'm going to choose that one, hit save. And now these two workflows are basically linked so that if this workflow ever has an error that stops the workflow, it's going to be captured in our second one over here with the information. So let's see a quick example of that. Okay, so this workflow is active. It has a telegram trigger as you can see. So I'm going to drag in my telegram and I'm just going to say, "Hey." And what's going to happen is obviously we're going to get a response back because this workflow is active and it says, "How can I assist you today?" Now, what I'm going to do is I'm just going to get rid of the chat model. So, this agent essentially has no brain. I'm going to hit save. We're going to open up Telegram again, and we're going to say, "Hey." And now, we should see that we're not going to get any response back in Telegram. If we go into the executions of this ultimate assistant, you can see that we just got an error right now. And that was when we just sent off the query that said, "Hey," and it errored because the chat model wasn't connected. So, if we hop into our error logger workflow and click on the executions, we should see that we just had a new execution. And if we click into it, we'll see all the information that came through. So what it's going to tell us is the ID of the execution, the URL of the workflow, the name of the workflow, and then we'll also see what node errored and the error message. So here under the object node, we can see different parameters. We can see what's kind of how the node's configured. We can see the prompt even. But what we're interested in is down here we have the name, which is ultimate assistant. And then we have the message, which was a chat model sub node must be connected and enabled. So anyways, we have our sample data. I'm going to hit copy to editor, which just brings in that execution into here so we can play with it. And now what I want to do is map up the logic of first of all logging it in a Google sheet. So here's the Google sheet template I'm going to be using. We're going to be putting in a timestamp, a workflow name, the URL of the workflow, the node that errored, and the error message. If you guys want to get this template, you can do so by joining my free school community. The link for that's down in the description. Once you join the community, all you have to do is search for the title of the video up top, or you can click on YouTube resources, and you'll find the post. And then in the post is where you'll see the link to the Google sheet template. Anyways, now that we have this set up, all we have to do is go back into our error logger. We're going to add a new node after the trigger. And I'm going to grab a sheets node. What we want to do is append a row in sheets. Um I'm just going to call this log error. Make sure I choose the right credential. And then I'm going to choose the sheet which is called error logs. And so now we can see we have the values we need to send over to these

different columns. So for time stamp, all I'm going to do is I'm actually going to make an expression and I'm just going to do dollar sign now. And this is basically just going to send over to Google Sheets the current time whenever this workflow gets triggered. And if you don't like the way this is coming through, you can play around with format after dollar sign now. And then you'll be able to configure it a little bit more. And you can also ask chat to help you out with this JavaScript function. And feel free to copy this if you want. I'm pulling in the full year, month, day, and then the time. Okay, cool. Then we're just pretty much going to drag and drop the other information we need. So the first thing is the workflow name. And to get to that, I'm going to close out of the execution. and then we'll see the workflow and we can pull in the name right there which is ultimate personal assistant. For the URL, I'm going to open back up execution and grab the URL from here. For node, all I have to do is look within the node object. We're going to scroll down until we see the name of the node which is right here, the ultimate assistant. And then finally, the error message, which should be right under that name right here. Drag that in, which says a chat model sub node must be connected and enabled. So now that we're good to go here, I'm going to test step and then we'll check our Google sheet and make sure that that stuff comes through correctly. And as you can see, it just got populated and we have the URL right here, which if we clicked into, it would take us to that main ultimate personal assistant workflow. As you can see, when this loads up, and it takes us to the execution that actually failed as well, so we could sort of debug. So now we have an error trigger that will update a Google sheet and log the information. But maybe we also want to get notified when there's an error. So I'm just going to drag this off right below and I'm going to grab the Slack node and I'm going to choose to send a message right here. and then we'll just configure what we want to send over. Okay, so we're going to be sending a message to a channel. I'm going to choose the channel all awesome AI stuff. And then we just need to configure what the actual message is going to say. So I'm going to change this to an expression, make this full screen and let's fill this out. So pretty much just customize this however you want. Let's say I want to start off with workflow error and we will put the name of the workflow. So I'll just close out of here, throw that in there. So now it's going to come through saying workflow error ultimate personal assistant. And then I'm going to say like what node errored at what time and what the error message was. So first let's grab the name of the node. Um if I just have to scroll down to name right here. So ultimate assistant errored at and I'm going to do that same dollar sign now function. So it says ultimate assistant errored at 2025417. And then I'm just going to go down and say the error message was and we're just going to drag in the error message. There we go. And then finally, we'll just provide a link to the workflow. So see this execution here. And then we'll drag in the link, which is all the way up top. And we should be good to go. And then if you want to make sure you're not sending over a little message at the bottom that says this was sent from Nad, you're going to add an option. You're going to click on include link to workflow. And then you're going to turn that off. And now we hit test up. And we hop into Slack. And we can see we got workflow error ultimate personal assistant. We have all this information. We can click into this link. and we don't have this little message here that says automated with this NN workflow. So maybe you could just set up a channel dedicated towards error logging, whatever it is. Okay, so let's save this real quick and um let's just do another sort of like example. Um one thing to keep in mind is there's a difference between the workflow actually erroring out and going red and just something not working correctly. And I'll show you exactly what I meant by that. So this example actually triggered the error workflow because the execution on this side is red and it shows an error. But what happens is, for example, with our Tavly tool right here, I have no authentication pulled up. So

this tool is not going to work. But if I come into Telegram and say search the web for Apples, it's going to work. This this workflow is going to go green even though this tool is not going to work. And we'll see exactly why. So as you can see, it says I'm currently unable to search the web due to a connection error. So if we go into the execution, we can see that this thing went green even though it didn't work the way we wanted to. But what happened is the tool came back and it was green and it basically just didn't work because our authentication wasn't correct. And then you can even see in the think node, it basically said the web search function is encountering an authentication error. I need to let the user know the search isn't currently available and offer alternative ways to help blah blah blah. But all of these nodes actually went green and we're fine. So this example did not trigger the error logger. As you can see, if we check here, there's nothing. We check in Slack, there's nothing. So what we can do is we'll actually make something error. So I'll go into this memory and it's going to be looking for a session ID within the telegram trigger and I'm just going to add an extra d. So this variable is not going to work. It's probably going to error out and now we'll actually see something happen with our error workflow. So I'm just going to say hey and we will watch basically nothing will come back here. Okay, that confused me but I realized I didn't save the workflow. So now that we've saved it, this is not going to work. So let's once again say hey. And we should see that nothing's going to come back over here. Um, I believe if we go into our error logger, we should see something pop through. We just got that row and you can see the node changed, the error message changed, all that kind of stuff. And then in Slack, we got another workflow error at a new time and it was a different node. And finally, we can just come into our error logger workflow and click on executions. And we'll see the newest run was the one that we just saw in our logs and in our Slack, which was the memory node that aired. As you can see right here, simple memory. And so really the question then becomes, okay, well what happens if this workflow in itself errors too. I really don't foresee that happening unless you're doing some sort of crazy AI logic over here. But it really needs to just be as simple as you're mapping variables from here somewhere else. So you really shouldn't see any issues. Maybe an authentication issue, but I don't know. Maybe if this if this workflow is erring for you a ton, you probably are just doing something wrong. Anyways, that's going to do it for this one. I know it was a quicker one, but hopefully if you didn't know about this, it's something that you'll implement and it will be helpful. If you've ever wondered which AI model to use for your agents and you're tired of wasting credits or overpaying for basic tasks, then this video is for you because today I'm going to be showing you a system where the AI agent picks its brain dynamically based on the task. This is not only going to save you money, but it's going to boost performance and we're also getting full visibility into the models that it's choosing based on the input and we'll see the output. That way, all we have to do is come back over here, update the prompt, and continue to optimize the workflow over time. As you can see, we're talking to this agent in Slack. So, what I'm going to do is say, "Hey, tell me a joke." You can see my failed attempts over there. And it's going to get this message. As you can see, it's picking a model, and then it's going to answer us in Slack, as well as log the output. So, we can see we just got the response, why don't scientists trust Adams? Because they make up everything. And if I go to our model log, we can see we just got the input, we got the output, and then we got the model which was chosen, which in this case was Google Gemini's 2.0 Flash. And the reason it chose Flash is because this was a simple input with a very simple output and it wanted to choose a free model so we're not wasting credits for no reason. All right, let's try something else. I'm going to ask it to create a calendar event at 1 p.m. today for lunch. Once this workflow fires off, it's going to choose the model. As you can see, it's sending that over to the dynamic agent to

create that calendar event. It's going to log that output and then send us a message in Slack. So, there we go. I just have created the calendar event for lunch at 1 p.m. today. If you need anything else, just let me know. We click into the calendar real quick. There is our launch event at one. And if we go to our log, we can see that this time it used OpenAI's GBT 4.1 mini. All right, we'll just do one more and then we'll break it down. So, I'm going to ask it to do some research on AI voice agents and create a blog post. Here we go. It chose a model. It's going to hit Tavi to do some web research. It's going to create us a blog post, log the output, and send it to us in Slack. So, I'll check in when that's done. All right, so it just finished up and as you can see, it called the Tavly tool four times. So, it did some in-depth research. It logged the output and we just got our blog back in Slack as you can see. Wow. It is pretty thorough. It talks about AI voice agents, the rise of voice agents. Um there's key trends like emotionally intelligent interactions, advanced NLP, real-time multilingual support, all this kind of stuff. Um that's the whole blog, right? It ends with a conclusion. And if you're wondering what model it used for this task, let's go look at our log. We can see that it ended up using Claude 3.7 sonnet. And like I said, it knew it had to do research. So it hit the tabletly tool four different times. The first time it searched for AI voice agents trends, then it searched for case studies, then it searched for growth statistics, and then it searched for ethical considerations. So, it made us a pretty like holistic blog. Anyways, now that you've seen a quick demo of how this works, let's break down how I set this up. So, the first things first, we're talking to it in Slack and we're getting a response back in Slack. And as you can see, if I scroll up here, I had a a few fails at the beginning when I was setting up this trigger. So, if you're trying to get it set up in Slack, um it can be a little bit frustrating, but I have a video right up here where I walk through exactly how to do that. Anyways, the key here is that we're using Open Router as the chat model. So, if you've never used Open Router, it's basically a chat model that you can connect to and it basically will let you route to any model that you want. So, as you can see, there's 300 plus models that you can access through Open Router. So, the idea here is that we have the first agent, which is using a free model like Gemini 2.0 Flash. we have this one choosing which model to use based on the input. And then whatever this model chooses, we're using down here dynamically for the second agent to actually use in order to use its tools or produce some sort of output for us. And just so you can see what that looks like, if I come in here, you can see we're using a variable. But if I got rid of that and we change this to fixed, you can see that we have all of these models within our open router dynamic brain to choose from. But what we do is instead of just choosing from one of these models, we're basically just pulling the output from the model selector agent right into here. And that's the one that it uses to process the next steps. Cool. So let's first take a look at the model selector. What happens in here is we're feeding in the actual text that we sent over in Slack. So that's pretty simple. We're just sending over the message. And then in the system message here, this is where we actually can configure the different models that the AI agent has access to. So I said, "You're an agent responsible for selecting the most suitable large language model to handle a given user request. Choose only one model from the list below based strictly on each model's strengths." So we told it to analyze the request and then return only the name of the model. We gave it four models. Obviously, you could give it more if you wanted to. And down here, available models and strengths. We gave it four models and we basically defined what each one's good at. You could give it more than four if you wanted to, but just for this sake of the demo, I only gave it four. And then we basically said, return only one of the following strings. And as you can see in this example, it returned anthropic claude 3.7 sonnet. And so one quick thing to note here is when you use Gemini 2.0 flash, for some reason it likes to output a new line after a lot of

these strings. So all I had to do later is I clean up this new line and I'll show you exactly what I mean by that. But now we have the output of our model and then we move on to the actual Smartyp Pants agent. So in this one, we're giving it the same user message as the previous agent where we're just basically coming to our Slack trigger and we're dragging in the text from Slack. And what I wanted to show you guys is that here we have a system message and all I gave it was the current date and time. So I didn't tell it anything about using Tavi for web search. I didn't tell it how to use its calendar tools. This is just going to show you that it's choosing a model intelligent enough to understand the tools that it has and how to use them. And then of course the actual dynamic brain part. We looked at this a little bit, but basically all I did is I pulled in the output of the previous agent, the model selector agent. And then, like I said, we had to just trim up the end because if you just dragged this in and Open Router was trying to reference a model that had a new line character after it, it would basically just fail and say this model isn't available. So, I trimmed up the end and that's why. And you can see in my Open Router account, if I go to my activity, we can see which models we've used and how much they've costed. So, anyways, Gemini 2.0 Flash is a free model, but if we use it through open router, they have to take a little bit of a, you know, they got to get some kickback there. So, it's not exactly free, but it's really, really cheap. But the idea here is, you know, Claude 3.7 sonnet is more expensive and we don't need to use it all the time, but if we want our agent to have the capability of using Claude at some point, then we probably would just have to plug in Claude. But now, if you use this method, if you want to talk to the agent just about some general things or looking up something on your calendar or sending an email, you don't have to use Claude and waste these credits. You could go ahead and use a free model like 2.0 Flash or still a very powerful cheap model like GPT 4.1 Mini. And that's not to say that 2.0 Flash isn't super powerful. It's just more of a lightweight model. It's very cheap. Anyways, that's just another cool thing about Open Router. That's why I've gotten in the habit of using it because we can see the tokens, the cost, and the breakdown of different models we've used. From there, we're feeding the output into a Google sheet template, which by the way, you can download this workflow as well as these other ones down here that we'll look at in a sec. You can download all this for free by joining my Free School community. All you have to do is go to YouTube resources or search for the title of this video and when you click on the post associated with this video, you'll have the JSON which is the end workflow to download as well as you'll see this Google sheet template somewhere in that post so that you can just basically copy it over and then you can plug everything into your environment. Anyways, just logging the output of course and we're sending over a timestamp. So I just said, you know, whatever time this actually runs, you're going to send that over the input. So the Slack message that triggered this workflow. The output, I'm basically just bringing the output from the Smartyp Pants agent right here. And then the model is the output from the model selector agent. And then all that's left to do is send the response back to the human in Slack where we connected to that same channel and we're just sending the output from the agent. So hopefully this is just going to open your eyes to how you can set up a system so that your actual main agent is dynamically picking a brain to optimize your cost and performance. And in a space like AI where new models are coming out all the time, it's important to be able to test out different ones for their outputs and see like what's going on here, but also to be able to compare them. So, two quick tools I'll show you guys. This first one is Vellum, which is an LLM leaderboard. You can look at like reasoning, math, coding, tool use. You have all this stuff. You can compare models right here where you can select them and look at their differences. And then also down here is model comparison with um all these different statistics you can look at. You can look at context,

window, cost, and speed. So, this is a good website to look at, but just keep in mind it may not always be completely up to date. Right here, it was updated on April 17th, and today is the 30th, so doesn't have like the 4.1 models. Anyways, another one you could look at is this LM Arena. So, I'll leave the link for this one also down in the description. You can basically compare different models by chatting with them like side by side or direct. People give ratings and then you can look at the leaderboard for like an overview or for text or for vision or for whatever it is. just another good tool to sort of compare some models. Anyways, we'll just do one more quick before we go on to the example down below. Um because we haven't used the reasoning model yet and those are obviously more expensive. So, I'm asking you a riddle. I said you have three boxes. One has apples, one has only oranges, and one has a mix of both. They're all incorrectly labeled and you can pick one fruit from the box without looking. How can you label all boxes correctly? So, let's see what it does. Hopefully, it's using the reasoning model. Okay, so it responded with a succinct way to see it is to pick one piece of fruit from the box labeled apples and oranges. Since that label is wrong, the box must actually contain only apples or only oranges. Whatever fruit you draw tells you which single fruit box that really is. Once you know which box is purely apples or purely oranges, you can use the fact that all labels are incorrect to deduce the proper labels for the remaining two boxes. And obviously, I had chatbt sort of give me that riddle and that's basically the answer it gave back. So, real quick, let's go into our log and we'll see which model it used. And it used OpenAI's 01 reasoning model. And of course, we can just verify that by looking right here. And we can see it is OpenAI 01. So, one thing I wanted to throw out there real quick is that Open Router does have sort of like an auto option. You can see right here, Open Router/auto, but it's not going to give you as much control over which models you can choose from, and it may not be as costefficient as being able to define here are the four models you have, and here's when to use each one. So, just to show you guys like what that would do if I said, "Hey," it's going to use its model and it's going to pick one based on the input. And here you can see that it used GPT4 mini. And then if I go ahead and send in that same riddle that I sent in earlier, remember earlier it chose the reasoning model, but now it's going to choose probably not the reasoning model. So anyways, looks like it got the riddle right. And we can see that the model that it chose here was just GPT40. So I guess the argument is yes, this is cheaper than using 01. So if you want to just test out your workflows by using the auto function, go for it. But if you do want more control over which models to use, when to use each one, and you want to get some higher outputs in certain scenarios, then you want to take probably the more custom route. Anyways, just thought I'd drop that in there. But let's get back to the video. All right, so now that you've seen how this agent can choose between all those four models, let's look at like a different type of example here. Okay, so down here we have a rag agent. And this is a really good use case in my mind because sometimes you're going to be chatting with a knowledge base and it could be a really simple query like, can you just remind me what our shipping policy is? Or something like that. But if you wanted to have like a comparison and like a deep lookup for something in the knowledge base, you'd probably want more of a, you know, a more intelligent model. So we're doing a very similar thing here, right? This agent is choosing the model with a free model and then it's going to feed in that selection to the dynamic brain for the rag agent to do its lookup. And um what I did down here is I just put a very simple flow if you wanted to download a file into Superbase just so you can test out this Superbase rag agent up here. But let's chat with this thing real quick. Okay, so here is my policy and FAQ document, right? And then I have my Subbase table where I have these four vectors in the documents table. So what we're going to do is query this agent for stuff that's in that policy and FAQ

document. And we're going to see which model it uses based on how complex the query is. So if I go ahead and fire off what is our shipping policy, we'll see that the model selector is going to choose a model, send it over, and now the agent is querying Superbase and it's going to respond with here's Tech Haven's shipping policy. Orders are processed within 1 to two days. standard shipping takes 3 to seven business days, blah blah blah. And if we compare that with the actual documentation, you can see that that is exactly what it should have responded with. And you'll also notice that in this example, we we're not logging the outputs just because I wanted to show a simple setup. But we can see the model that it chose right here was GPT 4.1 mini. And if we look in this actual agent, you can see that we only gave it two options, which was GPT 4.1 mini and anthropic cloud 3.5 sonnet, just because of course I just wanted to show a simple example. But you could up this to multiple models if you'd like. And just to show that this is working dynamically, I'm going to say what's the difference between our privacy policy and our payment policy. And what happens if someone wants to cancel their order or return an item? So, we'll see. Hopefully, it's choosing the cloud model because this is a little bit more complex. Um, it just searched the vector database. We'll see if it has to go back again or if it's writing an answer. It looks like it's writing an answer right now. And we'll see if this is accurate. So, privacy versus payment. We have privacy focuses on data protection. payment covers accepted payment methods. Um what happens if someone wants to cancel the order? We have order cancellation can be cancelled within 12 hours. And we have a refund policy as well. And if we go in here, we could validate that all this information is on here. And we can see this is how you cancel. And then this is how you refund. Oh yeah, right here. Visit our returns and refund page. And we'll see what it says is that here is our return and refund policy. And all this information matches exactly what it says down here. Okay. So those are the two flows I wanted to share with you guys today. Really, I just hope that this is going to open your eyes to the fact that you can have models be dynamic based on the input, which really in the long run will save you a lot of tokens for your different chat models. All right, so now we're going to move on to web hooks, which I remember seemed really intimidating as well, just like APIs and HTTP requests, but they're really even simpler. So, we're going to dive into what exactly they are and show an example of how that works in NN. And then I'm going to show you guys two workflows that are triggered by NAN web hooks and then we send data back to that web hook. So, don't worry, you guys will see exactly what I'm talking about. Let's get into it. Okay, web hooks. So, I remember when I first learned about APIs and HTTP requests, and then I was like, what in the world is a web hook? They're pretty much the same thing except for, think about it like this. The web hook, rather than us sending off data somewhere or like sending off an API call, we are the one that's waiting for an API call. We're just waiting and listening for data. So let me show you an example of what that actually looks like. So here you can see we have a web hook trigger and a web hook is always going to come in the form of a trigger because essentially our end workflow is waiting for data to be sent to it. Whether that's like a form is submitted and now the web hook gets it or whatever that is we are waiting for the data here. So when I click into the web hook what we see is we have a URL and this is basically the URL that wherever we're sending data from is going to send data to. So later in this course, I'm going to show you an example where I do like an 11labs voice agent. And so our end web hook URL right here, that's where 11 Labs is sending data to. Or I'll show you an example with lovable where I build a little app and then our app is sending data to this URL. So that's how it works, right? Important things to remember is you still have to set up your method. So if you are setting up some sort of request on a different service and you're sending data to this web hook, it's probably going to be a post. So, you'll

want to change that and make sure that these actually align. Anyways, what we're going to do is we're going to change this to a post because I I just know it's going to be post. And this is our web hook URL, which is a test URL. So, I'm going to click on the button to copy it. And what I'm going to do is take this and I'm going to go into Postman, which is just kind of like an API platform that I use to show some demos of how we can send these requests. So, I'm going to click on send an API request in here. This basically just lets us test out and see if our web hook's working. So what I'm going to do is I'm going to change the request to post. I'm going to enter the web hook URL from my NAND web hook. Okay. And now basically what we have is the ability to send over certain information. So I'm just going to go to body. I'm going to send over form data. And now you can see just like JSON, it's key value pairs. So I'm just going to send over a field called text. And the actual value is going to be hello. Okay. So that's it. I'm going to click on send. But what happens is we're going to get a request back which is a 404 error. It says the web hook is not registered and basically there's a hint that says click the test workflow button. So because this workflow I'm sorry because the web hook is supposed to be listening right now it's not listening. And the reason it's not listening is because we are in an act inactive workflow like we've talked about before and we haven't clicked listen for test event. So if I click listen now you can see it is listening for this URL. Okay. So, I go back into Postman. I hit send. And now it's going to say workflow was started. I come back into here. We can see that the node has executed. And what we have is our text that we entered right there in the body, which was text equals hello. We also get all this other random stuff that we don't really need. Um, I don't even know what this stuff really stands for. We can see our host was our our nit cloud account. All this kind of stuff. Um, but that's not super important for us right now, right? I just wanted to show that that's how it works. So, they're both configured as post. Our postman is sending data to this address and we saw that it worked right. So what comes next is the fact that we can respond to this web hook. So right now we're set to respond immediately which would be sending data back right away. But let's say later you'll see an example in 11 labs and my end workflow we where we want to grab data do something with it and send something back. And how that would work is we would change this method to using respond to web hook node and then we basically like add like maybe an AI agent right here and then after the AI agent we would basically just go respond to web hook and as you can see it just returns data to that same web hook address. So that would just be sending data back to lovable 11s postman wherever we actually got the initial request from we would do something with that data and send it back to that web hook. So really, it's the exact same way that we think about APIs except for we are now the API server rather than the person who's sending the request. So if you think about the example with um being at a restaurant, you know, like we would look at the menu, we would send off a request through the waiter to the kitchen and then the kitchen would get that request via a web hook. They would create the food and then they would send it back to the actual person via responding to web hook. So that's all it is. is we are just on the other side of the table now. And then one last important thing to know about web hooks in NN is ideally you have to make it active. So we would click on active but what it's going to tell us is that you can now make calls to your production web hook URL no longer the test URL. So if you remember in here we have test and production and what I copied into Postman was the test. So if I go back into Postman and I hit send, nothing happens, right? But if I now go to production URL, copy the production, go into Postman and paste in the production URL, hit send. There we go. Workflow was started because that's the actual active workflow production URL. So I can click on executions and we can see this is the newest one we just got when we made our workflow active and it did in fact start. And

just to prove to you that that's how it works, let me change the body which now instead of saying hello, it's going to say active run. Oops. And we'll send that off. Go into edit end and we'll basically just watch a new execution will pop up because it's actively listening. I'm not doing anything. There you go. You see it pop up. And if we click into the actual web hook data, we will see right here the text says active run. Cool. So that's web hooks. Hopefully that clears up any confusion that you may have had regarding web hooks. But let's dive into some examples in NAND where we have a web hook trigger and then we're also configuring that other third party app that's going to send data to our web hook and trigger the workflow. Hey there, I'm Archer. What can I do for you? Hey Archer, can you help me send an email? Of course, I'd be delighted to help. Who's the email for and what's it about? Just give me the details and I'll get it sent out. Awesome. Yeah, I just wanted to send one to Phil. I wanted to ask about an update for the project that we discussed last week. Got it. So, you want to send an email to Phil asking about a project update? No problem. I'll handle that. Just hang tight while I send this off. All done. It's been sent to Phil. He'll be getting that email any minute. Anything else I can help you with? Yeah, could you actually go ahead and help me send another one? Of course. I'm always happy to lend a hand, or rather an email address. Who is this email for and what's it about? Just give me the details and I'll whip it up. Just another one to Phil. I forgot to mention that I wanted to see if he wants to get lunch tomorrow. All right, another email to Phil. This time to ask about lunch tomorrow. I'm on it. Just a sec. Gotcha. The email is on its way to Phil. He'll be sure to see your invitation for lunch tomorrow. Let me know if you need anything else. No, that's going to be it. Thanks, Archer. You're very welcome. Always happy to help, especially when it comes to emails. happy to handle any other messages you need to send in the future. Have a great day. Okay, so that was a quick demo of the super cool conversational voice agent that we built using 11 Labs and Nad. His name's Archer and he helps us send emails. As you can tell, he was very enthusiastic about sending emails. So, here's that first email that we asked Archer to send to Phil. Hi, Phil. I hope this message finds you well. I'm writing to inquire about the project update we discussed last week. Could you please share the latest developments at your earliest convenience? Looking forward to your response. Best Nate. And then we asked Archer to send another email just asking if he wants to get lunch tomorrow to Phil. So, hi Phil. I was wondering if you're available for lunch tomorrow. Let me know what works for you. Best Nate. So, now that we've seen a quick demo, we heard the voice. We've seen the emails actually come through. We're going to hop back into Nad and we're going to explain what's going on here so that you guys can get this sort of system up and running for yourselves. Okay, so there are a few things that I want to break down here. First of all, just within NAN, whenever you're building an AI agent, as you guys should know, there's going to be an input and then that input is going to be fed into the agent. The agent's going to use its system prompt and its brain to understand what tools it needs to hit. It's going to use those tools to take action and then there's going to be some sort of output. So, in the past when we've done tutorials on personal assistants, email agents, whatever it was, rag agents, usually that the input that we've been using has been something like Telegram or Gmail or even just the NAN chat trigger. Pretty much all we're switching out here for the input and the output is 11 Labs. So, we're going to be getting a post request from 11 Labs, which is going to send over the body parameters like who the email is going to, um, what the message is going to say, stuff like that. And then the agent once it actually does that, it's going to respond using this respond to web hook node. So, we'll get into 11 Labs and I'll show you guys how I prompted the agent and everything like that in 11 Labs. But first, let's take a quick look at what's going on in the super simple agent setup here in NN. So, these are tools that

I've used multiple times on videos on my channel. The first one is contact data. So, it's just a simple Google sheet. This is what it looks like. Here's Phil's information with the correct Gmail that we were having information sent to. And then I just put other ones in here just to sort of dummy data. But all we're doing is we're hooking up the tool um Google Sheets. It's going to be reading get rows sheet within the document. We link the document. That's pretty much all we had to do. Um and then we just called it contact data so that when we're prompting the agent, it knows when to use this tool, what it has. And then the actual tool that sends emails is the send email tool. So in here we're connecting a Gmail tool. Um this one is you know we're using all the from AI functions which makes it really really easy. Um we're sending a message of course and so the from AI function basically takes the query coming in from the agent and understands um dynamically the AI is looking for okay what's the email address based on the user's message. Okay we grab the email address we're going to put it in the two parameter. How can we make a subject out of this message? We'll put it here. And then how can we actually construct an email body and we put it there. So that's all that's going on here. Here we've got our tools. We've obviously got a chat model. In this case, we're just using um GPT40. And then we have the actual what's taking place within the agent. So obviously there's an input coming in. So that's where we define this information. Input agent output and then the actual system message for the agent. So the system message is a little bit different than the user agent. The system message is defining the role. This is your job as an agent. This is what you should be doing. These are the tools you have. And then the user message is like each execution, each each run, each time that we interact with the agent through 11 Labs, it's going to be a different user message coming in, but the system message is always going to remain the same as it's the prompt for the AI agents behavior. Anyways, let's take a look at the prompt that we have here. First, the overview is that you are an AI agent responsible for drafting and sending professional emails based on the user's instructions. You have access to two tools, contact data to find email addresses and send email to compose and send emails. Your objective is to identify the recipient's contact information, draft a professional email and sign off as Nate before sending. The tools you have obviously uh contact data. It retrieves email addresses based on the name. So we have an example input John Doe. Example output an email address and then send email. Sends an email with a subject and a body. The example input here is an email address. Um subject and a body with example email subject body. Um so that's what we have for the system message. And then for the um user message, as you can see, we're basically just saying um okay, so the email is going to be for this person and the email content is going to be this. So in this case, this execution it was the email's for Phil and the email content is asking about lunch tomorrow. So that's all that we're being fed in from 11 Labs. And then the agent takes that information to grab the contact information and then it uses its AI brain to make the email message. Finally, it basically just responds to the web hook with um the email to Phil regarding lunch tomorrow has been successfully sent and then 11 Labs captures that response back and then it can respond to us with gotcha. We were able to send that off for you. Is there anything else you need? So, that's pretty much all that's going on here. Um if you see in the actual web hook, what we're getting here is, you know, there's different things coming back. We have different little technical parameters, all this kind of stuff. all that we want to configure and I'll show you guys how we configure this in 11 Labs is the the JSON body request that's being sent over. So we're in table format. If we went to JSON, we could see down here we're looking at body. In the body, we set up two fields to send over from 11 Labs to Nidend using that post request web hook. The first field that we set up was two. And as you can see, that's when the 11 Labs model based on what we say

figures out who the email is going to and puts that there and then figures out what's the email content. What do you want me to say in this email? And then throws that in here. So, um, that's how that's going to work as far as setting up the actual web hook node right here. Um, we have a we wanted to switch this to a post method because 11 Labs is sending us information. Um, we have a test URL and a production URL. The test one we use for now and we have to manually have naden listen for a test event. Um I will show an example of what happens if we don't actually do this later in the video. But when you push the app into production, you make the workflow active, you would want to put this web hook in 11 Labs as the production URL rather than the test URL so that you can make sure that the stuff's actually coming over. We put our path as end just to clean up this URL. All that it does is changes the URL. Um and then authentication we put none. And then finally for response instead of doing immediately or wait when last node finishes we want to do using respond to web hook node. That way we get the information the agent takes place and then responds and then all we have here is respond to web hook. So it's very simple as you can see it's only you know really four nodes you know the email the brain um and then the two tools and the web hooks. So um hopefully that all made sense. We are going to hop into 11 labs and start playing around with this stuff. Also, quick side note, if you want to hop into this workflow, check out the prompts, play around with how I configured things. Um, you'll be able to download this workflow for free in the free school community. Link for that will be down in the description. You'll just come into here, you'll click on YouTube resources, you will click on the post associated with this video, and then you're able to download the workflow right here. Once you download the workflow, you can import it from file, and then you will have this exact canvas pop up on your screen. Then, if you're looking to take your skills with Naden a little bit farther, feel free to check out my paid community. The link for that will also be down in the description. Great community in here. A lot of people obviously are learning NAN and um asking questions, sharing builds, sharing resources. Got a great classroom section going over, you know, client builds and some deep dive topics as well as five live calls per week. So, you can always make sure you're getting your questions answered. Okay. Anyways, back to the video. So, in 11 Labs, this is the email agent. This is just the test environment where we're going to be talking to it to try things out. So, we'll go back and we'll see how we actually configured this agent. And if you're wondering why I named him Marcher, it's just because his actual voice is Archer. So, um, that wasn't my creativity there. Anyways, once we are in the configuration section of the actual agent, we need to set up a few things. So, first is the first message. Um, we pretty much just when we click on call the agent, it's going to say, "Hey there, I'm Marcher. What can I do for you?" Otherwise, um, if we leave this blank, then we will be the ones to start the conversation. But from there, you will set up a system prompt. So in here, this is a prompt I have is you are a friendly and funny personal assistant who loves helping the user with tasks in an upbeat and approachable way. Your role is to assist the user with sending emails. When the user provides details like who the email is for and what's it about, you will pass that information to the NAN tool and wait for its response. I'll show you guys in a sec how we configure the NAN tool and how all that works. But anyways, once you get confirmation from NAN that the email was sent, cheerfully let the user know it's done and ask if there's anything else you can help with. Keep your tone light, friendly, and witty while remaining efficient and clear in your responses. So, as you can see in the system prompt, I didn't even really put in anything about the way it should be conversating as far as like sounding natural and using filler words and um and sometimes I do that to make it sound more natural, but this voice I found just sounded pretty good just as is. Then, we're setting up the large language model. Um, right

now we're using Gemini 1.5 Flash just because it says it's the fastest. You have other things you can use here, but I'm just sticking with this one. And so this is what it uses to extract information pretty much out of the conversation to pass it to NAND or figure out how it's going to respond to you. That's what's going on here. And then with temperature, um, I talked about I like to put a little bit higher, especially for some fun use cases like this. Um, basically this is just the randomness and creativity of the responses generated so that it's always going to be a little different and it's going to be a little fun um, the higher you put it. But if you wanted it to be more consistent and you had like, you know, you were trying to get some sort of information back um right the way you want it, then you would probably want to lower this a little bit. Um and then you have stuff like knowledge base. So if this was maybe like um a customer support, you'd be able to put some knowledge base in there. Or if you watch my previous voice video about um sort of doing voice rag, you could still do the sending it to NADN, hitting a vector database from Naden and then getting the response back. But anyways, um, in this case, this is where we set up the tool that we were able to call up here as you saw in the system prompt. So the tool NAN, this is where you're putting the web hook from your the web hook URL from NAN. That's where you're putting that right here. As you can see, um, web hook- test NAN. The method is going to be a post. So we can send information from 11 Labs to NAN. And we just named it Naden to make the system prompt make more sense for um just just for me when I was creating this. It makes sense to send something to the tool called Naden. Anyways, as you can see, the description is use this tool to take action upon the user's request. And so we can pretty much just leave it as that. We don't have any headers or authorization going in here, but we do need to send over body parameters. Um otherwise, if we didn't have this, nothing would be sent over to Naden at all. So the description of the body parameters is in a friendly way ask the user to provide the name of the recipient and what the email is about unless they already provided that information. So the LLM is understanding when it's conversating with the human it needs to extract the name of this person to send an an email to and then actually what is the email going to have. As you can see we didn't have to put in like oh what's the subject of the email because our AI agent in N8N can use its brain to figure out a subject. So we have two parameters that we're setting over. If you remember when we were in here, we got the two parameters from the body right here, which was two and email content. And so that's as you can see what we set up here. So the first one was a string. It was called two and this is who the email is going to. And then the second one was also a string. It's called email content. And this is what is the email about. So we hit save changes. Now we have a tool that gets called whenever um the voice agent realizes it has information to send over to N to send an email. and it has the right parameters to do so. So now we have this voice agent pretty much set up here with its tools and its system prompt. We're going to test the agent. And remember earlier I talked about the fact that right now we're testing. So we have to have the web hook actively listening. Once this is in production, it's going to be listening all the time. But right now we have to manually hit test event before we call it. Otherwise, it's not going to work. So I'm going to talk to Archer, ask him to send an email. It's going to go through and then we're going to ask him to send another email where we don't hit this web hook test event and then we'll see what he says. You're probably just going to say he couldn't access the database or he couldn't find the contact information. Something like that. Hey there, I'm Archer. What can I do for you? Hey Archer, can you help me send an email? Of course. I'd love to help. Who's the email for and what's it about? Tell me the details and I'll whip it up in a jiffy. Sweet. Um, it's going to be for Phil. Okay, so the email is for Phil. What's it about? I'm ready to get this email sent out. Yeah, just let him know that I'm caught in traffic and I'm

going to be late to the meeting. Got it. So, you want to let Phil know you're running late because of traffic. I'll get that email sent out right away. All done. I just sent the email to Phil letting him know you're running late. Anything else I can help you with? Yeah, I'm actually going to need to send another one. No problem. Who's this one for and what's it about? I'm ready to send email number two. Yeah. Can you let Phil know that um I actually just drove my car off a bridge? Oh my goodness. I can't send an email like that. Maybe you should call Phil instead. Is there someone else you want to send an email to? Um, no. Can you send an email to Phil? I'm so sorry, but I can't send an email like that. It sounds like you're in a difficult situation. Is there someone else you can reach out to for help? Okay, didn't expect that. So, I guess the model doesn't want to say something like that. We'll try again. Um, workflow is not active. So, let's see. Hey there, I'm Archer. What can I do for you? Hey, Archer, can you send an email? Of course, I'd be happy to help you with that. Who is the email for and what is it about? I'll get it sent out in a jify. Just wanted to touch base with Phil about um the meeting later today. Okay. So, you want to send an email to Phil about the meeting later today. Got it. I'll get this done right away. Anything else I can help with today? Oops. Looks like there was an error sending that email. It's possible there's a typo in the email address or something is wrong on my end. Let me try again. Can you please double check the email address? Okay. Anyway, so that's how that works. As you can see, the second time the web hook wasn't active, so wasn't able to send that um that email for us because it pretty much endnot passed through. So, that's going to be it for this one. I hope that everything made sense. Um it's just really cool how easy it basically is to switch out an input and you can have the agent function the same. Obviously, a few things would change as you start to add more tools. Your user message would have to be tweaked a little bit. You'd have to tweak the actual system prompts a little bit. But, as you can see in this one, kept it very, very simple. Basically, just told it its role. Gave it the the two tools and how to use them. And as you can see, um it was pretty seamless as far as being able to have the agent fill in things, make the messages, and then send them off pretty easily. Today, we're going to be talking about how you can build anything with Lovable and NAN. So we're going to be doing a live build of spinning up a web page with lovable and then also building the backend on nit. But first of all, I wanted to go over high level what this architecture looks like. So right here is lovable. This is what we're starting off with. And this is where we're going to be creating the interface that the user is interacting with. What we do here is we type in a prompt in natural language and Lovable basically spins up that app in seconds. And then we're able to talk back and forth and have it make minor fixes for us. So what we can do is when the user inputs information into our lovable website it can send that data to nadn the nadn workflow that we're going to set up can you know use an agent to take action in something like gmail or slack air tableable or quickbooks and then naden can send the data back to lovable and display it to the user and this is really just the tip of the iceberg there's also some really cool integrations with lovable and superbase or stripe or resend so there's a lot of ways you can really use lovable to develop a full web app and so while we're talking high We just wanted to show you an example flow of what this naden could look like where we're capturing the information the user is sending from lovable via web hook. We're feeding that to a large language model to create some sort of content for us and then we're sending that back and it will be displayed in the Lovable web app. So let's head over to Lovable and get started. So if you've never used Lovable before, don't worry. I'm going to show you guys how simple it is. You can also sign up using the link in the description for double the credits. Okay, so this is all I'm going to start with just to show you guys how simple this is. I said, "Help me create a web app called Get Me Out of This, where a user can submit a problem

they're having." Then I said to use this image as design inspiration. So, I Googled landing page design inspiration, and I'm just going to take a quick screenshot of this landing page, copy that, and then paste it into Lovable. And then we'll fire this off. Cool. So, I just sent that off. And on the right hand side, we're seeing it's going to spin up a preview. So, this is where we'll see the actual web app that it's created and get to interact with it. Right now, it's going to come through and start creating some code. And then on the lefth hand side is where we're going to have that back and forth chat window to talk to lovable in order to make changes for us. So right now, as you can see, it's going to be creating some of this code. We don't need to worry about this. Let's go into nit real quick and get this workflow ready to send data to. Okay, so here we are in Nen. If you also haven't used this app before, there'll be a link for it down in the description. It's basically just going to be a workflow builder and you can get a free trial just to get started. So you can see I have different workflows here. We're going to come in and create a new one. And what I'm going to do is we're gonna add a first step that's basically saying, okay, what actually triggers this workflow. So I'm gonna grab a web hook. And so all a web hook is is, you know, it looks like this. And this is basically just a trigger that's going to be actively listening for something to send data to it. And and data is received at this URL. And so right now there's a test URL and there's a production URL. Don't worry about that. We're going to click on this URL to copy it to our clipboard. And basically we can give this to Lovable and say, "Okay, whenever a user puts in a problem they're having, you're going to send the data to this web hook." Cool. So hopping over to Lovable. As you can see, it's still coding away and looks like it's finishing up right now. And it's saying, "I've created a modern problem-solving web app with a hero section, submission form, and feature section in blue color." Um, looks like there's an error. So all we have to do is click on try to fix, and it should go back in there and continue to spin up some more code. Okay, so now it looks like it finished that up. And as you can see, we have the website filled up. And so it created all of this with just uh an image as inspiration as well as just me telling it one sentence, help me create a web app called get me out of this where a user can submit a problem they're having. So hopefully this should already open your eyes to how powerful this is. But let's say for the sake of this demo, we don't want all this. We just kind of want one simple landing page where they send a problem in. So all I'd have to do is on this lefth hand side, scroll down here and say make this page more simple. We only need one field which is what problem can we help with? So we'll just send that off. Very simple query as if we were just kind of talking to a developer who was building this website for us and we'll see it modify the code and then we'll see what happens. So down here you can see it's modifying the code and now we'll see what happens. It's just one interface right here. So it's created like a title. It has these different buttons and we could easily say like, okay, when someone clicks on the home button, take them here. Or when someone clicks on the contact button, take them here. And so there's all this different stuff we can do, but for the sake of this video, we're just going to be worrying about this interface right here. And just to give it some more personality, what we could do is add in a logo. So I can go to Google and search for a thumbs up logo PNG. And then I can say add this logo in the top left. So I'll just paste in that image. We'll fire this off to lovable. And it should put that either right up here or right up here. We'll see what it does. But either way, if it's not where we like it, we can just tell it where to put it. Cool. So, as you can see, now we have that logo right up there. And let's say we didn't like this, all we'd have to do is come up to a previous version, hit on these three dots, and hit restore. And then it would just basically remove those changes it just made. Okay. So, let's test out the functionality over here. Let's say a problem is we want to get out of Oh, looks like the font is coming through white. So, we need to make sure this is changed. And boom, we

just told it to change the text to black and now it's black and we can see it. So anyways, I want to say get me out of a boring meeting. So we'll hit get me out of this and we'll see what happens. It says submitting and nothing really happens. Even though it told us, you know, we'll get back to you soon. Nothing really happened. So, what we want to do is we want to make sure that it knows when we hit this button, it's going to send that data to our Naden web hook. So, we've already copied that web hook to our clipboard, but I'm just going to go back into Naden. We have the web hook. We'll click on this right here back into lovable. Basically just saying when I click get me out of this, so this button right here, send the data to this web hook. And also, what we want to do is say as a post request because it's going to be sending data. So, we're going to send that off. And while it's making that change to the code, real quick, we want to go into edit end and make sure that our method for this web hook is indeed post. So I don't want to dive into too much what that means really, but Lovable is going to be sending a post request to our web hook. Meaning there's going to be stuff within this web hook like body parameters and different things. And so if this wasn't configured as a post request, it might not work. So you'll see once we actually get the data and we catch it in any of them. But anyways, now when the users click on get me out of this, the form will send the problem description to your web hook via a post request. So let's test it out. So we're going to say I forgot to prepare a brief for my meeting. We're going to go back and end it in real quick and make sure we hit listen for test event. So now our web hook is actively listening back in lovable. We'll click get me out of this and we will see what happens. We can come and end it in and we can now see we got this information. So here's the body I was talking about where we're capturing a problem which is I forgot to prepare a brief for my meeting. So, we now know that Lovable is able to send data to NAND. And now it's on us to configure what we want to happen in NAND so we can send the data back to Lovable. Cool. So, what I'm going to do is I'm going to click on the plus that's coming off of the web hook. And I'm going to grab an AI agent. What this is going to do is allow us to connect to a different chat model and then the agent's going to be able to take this problem and produce a response. And I'm going to walk through the step by step, but if you don't really want to worry about this and you just want to worry about the lovable side of things, you can download the finished template from my free school community. I'll link that down in the description. That way, you can just plug in this workflow and just give lovable your noden web hook and you'll be set up. But anyways, if you join the free school community, you'll click on YouTube resources, click on the post associated with this video, and you'll be able to download the JSON right here. And then when you have that JSON, you can come into Nadn, open up a new workflow, click these three dots on the top, and then click import from file. And when you open that up, it'll just have the finished workflow for you right here. But anyways, what I'm going to do is click into the AI agent. And the first thing is we have to configure what information the agent is going to actually read. So first of all, we're going to set up that as a user prompt. We're going to change this from connected chat trigger node to define below because we don't have a connected chat trigger node. We're using a web hook as we all know. So we're going to click on define below and we are basically just going to scroll down within the web hook node where the actual data we want to look at is which is just the problem that was submitted by the user. So down here in the body we have a problem and we can just drag that right in there and that's basically all we have to do. And maybe we just want to define to the agent what it's looking at. So we'll just say like the problem and then we'll put a colon. So now you can see in the result panel this is what the agent will be looking at. And next we need to give it a system message to understand what it's doing. So, I'm going to click on add option, open up a system message, and I am going to

basically tell it what to do. So, here's a system message that I came up with just for a demo. You're an AI excuse generator. Your job is to create clever, creative, and context appropriate excuses that someone could use to avoid or get out of a situation. And then we told it to only return the excuse and also to add a touch of humor to the excuses. So, now before we can actually run this to see how it's working, we need to connect its brain, which is going to be an AI chat model. So, what I'm going to do is I'm going to click on this plus under chat model. For this demo, we'll do an OpenAI chat model. And you have to connect a credential if you haven't done so already. So, you would basically come into here, click create new credential, and you would just have to insert your API key. So, you can just Google OpenAI API. You'll click on API platform. You can log in, and once you're logged in, you just have to go to your dashboard, and then on the left, you'll have an API key section. All you'll have to do is create a new key. We can call this one um test lovable. And then when you create that, you just copy this value. Go back into Nitn. Paste that right here. And then when you hit save, you are now connected to OpenAI's API. And we can finally run this agent real quick. If I come in here and hit test step, we will see that it's going to create an excuse for I forgot to prepare a brief for my meeting, which is sorry, I was too busy trying to bond with my coffee machine. Turns out it doesn't have a prepare briefs setting. So basically what we have is we're capturing the problem that a user had. We're using an AI agent to create a excuse. And then we need to send the data back to Lovable. So all we have to do here is add the plus coming off of the agent. We're going to call this a respond to web hook node. And we're just going to respond with the first incoming item, which is going to be the actual response from the agent. But all we have to do also to configure this is back in the web hook node, there's a section right here that says respond, instead of responding immediately, we want to respond using the respond to web hook node. So now it will be looking over here, and that's how it's going to send data back to lovable. So this is pretty much configured the way we need it, but we have to configure Lovable now to wait for this response. Okay. So what I'm telling Lovable is when the data gets sent to the web hook, we wait for the response from the web hook, then output that in a field that says here is your excuse. So we'll send this off to Lovable and see what it comes up with. Okay, so now it said that I've added a new section that displays here is your excuse along with the response message from the web hook when it's received. So let's test it out. First, I'm going to go back and edit in and we're going to hit test workflow. So the web hook is now listening for us. So we'll come into our lovable web app and say I want to skip a boring meeting. We'll hit get me out of this. So now that data should be captured in Naden. It's running. And now the output is I just realized my pet goldfish has a lifealtering decision to make regarding his tank decorations and I simply cannot miss this important family meeting. So it doesn't look great, but it worked. And if we go into edit end, we can see that this run did indeed finish up. And the output over here was I just realized my pet goldfish has a lifealtering decision blah blah blah. So basically what what's happening is the web hook is returning JSON which is coming through in a field called output and then we have our actual response which is exactly what lovable sent through. So it's not very pretty and we can basically just tell it to clean that up. So what I just did is I said only return the output fields value from the web hook response not the raw JSON. So we wanted to just output this right here which is the actual excuse. And so some of you guys may not even have had this problem pop up. I did a demo of this earlier just for testing and I basically walked through these same steps and this wasn't happening. But you know sometimes it happens. Anyways, now it says the form only displays the value from the output field. So let's give it another try. So back in we're going to hit test workflow. So it's listening for us in lovable. We're going to give it a problem. So I'm saying I overslept and I'm running late. I'm going to click get me out

of this. And we'll see the workflow just finished up. And now we have the response in a clean format which is I accidentally hit the snooze button until it filed for a restraining order against me for harassment. Okay. So now that we know that the functionality within N is working. It's sending data back. We want to customize our actual interface a little bit. So the first thing I want to do just for fun is create a level system. So every time someone submits a problem, they're going to get a point. And if they get five points, they'll level up. If they get 20 total points, they'll level up again. Okay. So I just sent off create a dynamic level system. Every time a user submits a problem, they get a point. Everyone starts at level one and after five points, they reach level two. Then after 50 more points, they reach level three. And obviously, we'd have to bake in the rest of the the levels and how many points you need. But this is just to show you that this is going to increase every time that we submit a problem. And also, you'd want to have some sort of element where people actually log in and get authenticated. And you can store that data in Superbase or in um you know, Firebase, whatever it is, so that everyone's levels are being saved and it's specific to that person. Okay, so looks like it just created a level system. It's reloading up our preview so we can see what that looks like now. Um, looks like there may have been an error, but now, as you can see right here, we have a level system. So, let's give it another try. I'm going to go into Nitn. We're going to hit test workflow. So, it's listening once again, and we're going to describe a problem. So, I'm saying my boss is mean. I don't want to talk to him. We're going to hit submit. The NN workflow is running right now on the back end. And we just got a message back, which is, I'd love to chat, but I've got a hot date with my couch and binge watching the entire season of Awkward Bosses. And you can see that we got a point. So, four more points to unlock level two. But before we continue to throw more prompts so that we get up to level two, let's add in one more cool functionality. Okay, so I'm just firing off this message that says add a drop down after what problem can we help with that gives the user the option to pick a tone for the response. So the options can be realistic, funny, ridiculous, or outrageous. And this data of course should be passed along in that web hook to NADN because then we can tell the agent to say okay here's the problem and here's the tone of an excuse the user is requesting and now it can make a request or a response for us. So looks like it's creating that change right now. So now we can see our dropown menu that has realistic, funny, ridiculous, and outrageous. As you can see before you click on it, it's maybe not super clear that this is actually a drop down. So let's make that more clear. And what I'm going to do is I'm going to take a screenshot of this section right here. I'm going to copy this and I'm just going to paste it in here and say make this more clear that it is a drop-own selection and we'll see what it does here. Okay, perfect. So, it just added a little arrow as well as a placeholder text. So, that's way more clear. And now what we want to do is test this out. Okay, so now to test this out, we're going to hit test workflow. But just keep in mind that this agent isn't yet configured to also look at the tone. So this tone won't be accounted for yet. But what we're going to do is we have I overslept and the response is going to be funny. We'll hit generate me a or sorry get me out of this. So we have a response and our level went up. We got another point. But if we go into Nit, we can see that it didn't actually account for the tone yet. So all we have to do is in the actual user message, we're basically just going to open this up and also add a tone. And we can scroll all the way down here and we can grab the tone from the body request. And now it's getting the problem as well as the tone. And now in the system prompt, which is basically just defining to the agent its role. We have to tell it how to account for different tones. Okay, so here's what I came up with. I gave it some more instructions and I said, "You're going to receive a problem as well as a tone. And here are the possible tones, which are realistic, funny, ridiculous, and outrageous." And I

kind of said what that means. And then I said, "Your excuse should be one to three sentences long, and match the selected tone." So that's all we're going to do. We're going to hit save. Okay. So now that it's looking at everything, we're going to hit test workflow. The web hook's listening. We'll come back into here and we're going to submit. I broke my friend's iPhone and the response tone should be outrageous. So, we're going to send that off. And it's loading because our end workflow is triggering. And now we just got it. We also got a message that says we earned a point. So, right here, we now only need two more for level two. But the excuse is I was trying to summon a unicorn with my telekinetic powers and accidentally transformed your iPhone into a rogue toaster that launched itself off the counter. I swear it was just trying to toast a bagel. Okay, so obviously that's pretty outrageous and that's how we know it's working. So, I'm sure you guys are wondering what would you want to do if you didn't want to come in here and every single time make this thing, you know, test workflow. What you would do is you'd switch this to an active workflow. Now, basically, we're not going to see the executions live anymore with all these green outlines. But what's happening now is it's using the production URL. So, we're going to have to copy the production URL, come back into Lovable, and just basically say I switched the URL or sorry, let's call I switched the web hook to this. And we'll paste that in there, and it should just change the data. The logic should be all the exact same because we've already built that into this app, but we're just going to switch the web hook. So, now we don't have to go click test workflow every time in NAN. And super excited. We have two more problems to submit and then we'll be level two. So now it says the web hook URL has been updated. So let's test it out. As you can see in here, we have an active workflow. We're not hitting test workflow. We're going to come in here and submit a new problem. So we are going to say um I want to take four weeks off work, but my boss won't let me. We are going to make the response tone. Let's just do a realistic one. And we'll click get me out of this. It's now calling that workflow that's active and it's listening. So we got a point. We got our response which is I've been dealing with some unforeseen family matters that need my attention. I believe taking 4 weeks off will help me address them properly. I plan this time to use this time to ensure everything is in order so I can return more focused and productive. I would definitely say that that's realistic. What we can do is come into NAN. We can click up here on our executions and we can see what just happened. So this is our most recent execution and if we click into here it should have been getting the problem which was I want to take four weeks off work and the tone which was realistic. Cool. Cool. So, now that we know that our active new web hook is working, let's just do one more query and let's earn our level two status. I'm also curious to see, you know, we haven't worked in any logic of what happens when you hit level two. Maybe there's some confetti. Maybe it's just a little notification. We're about to find out. Okay, so I said I got invited on a camping trip, but I hate nature. We're going to go with ridiculous and we're going to send this off. See what we get and see what level two looks like. Okay, so nothing crazy. We could have worked in like, hey, you know, make some confetti pop up. All we do is we get promoted to level two up here. But, you know, as you can see, the bar was dynamic. It moved and it did promote us to level two. But the excuse is, I'd love to join, but unfortunately, I just installed a new home system that detects the presence of grass, trees, and anything remotely outdoorsy. If I go camping, my house might launch an automated rescue mission to drag me back indoors. So, that's pretty ridiculous. And also, by the way, up in the preview, you can make it mobile. So, we can see what this would look like on mobile. Obviously, it's not completely optimized yet, so we'd have to work on that. But that's the ability to do both desktop and mobile. And then when you're finally good with your app, up in the top right, we can hit publish, which is just going to

show us that we can connect it to a custom domain or we can publish it at this domain that is made for us right here. Anyways, that is going to be it for today's video. This is really just the tip of the iceberg with, you know, nodn already has basically unlimited capabilities. But when you connect that to a custom front end when you don't have to have any sort of coding knowledge, as you can see, all of these prompts that I use in here was just me talking to it as if I was talking to a developer. And it's really, really cool how quick we spun this up. All right, hopefully you guys thought that was cool. I think that 11 Labs is awesome and it's cool to integrate agents with that as well as lovable or bolt. or these other vibe coding apps that let you build things. That would have taken so much longer and you would have kind of had to know how to code. So, really cool. So, we're nearing the end of the course, but it would be pretty shameful if I didn't at least cover what MCP servers are because they're only going to get more commonly used as we evolve through the space. So, we're going to talk about MCP servers. We're going to break it down as simple as possible. And then I'm going to do a live setup where I'm self-hosting NADN in front of you guys step by step. and then I'm going to connect to a community node in NN that lets us access some MCP servers. So, let's get into it. Okay, so model context protocol. I swear the past week it's been the only thing I've seen in YouTube comments, YouTube videos, Twitter, LinkedIn, it's just all over. And I don't know about you guys, but when I first started reading about this kind of stuff, I was kind of intimidated by it. I didn't completely understand what was going on. It was very techy and, you know, kind of abstract. I also felt like I was getting different information based on every source. So, we're going to break it down as simple as possible how it makes AI agents more intelligent. Okay, so we're going to start with the basics here. Let's just pretend we're going back to Chad GBT, which is, you know, a large language model. What we have is an input on the left. We're able to ask it a question. You know, help me write this email, tell me a joke, whatever it is. We feed in an input. The LM thinks about it and provides some sort of answer to us as an output. And that's really all that happens. The next evolution was when we started to give LLM tools and that's when we got AI agents because now we could ask it to do something like write an email but rather than just writing the email and giving it back to us it could call a tool to actually write that email and then it would tell us there you go the job's done. And so this really started to expand the capabilities of these LLM because they could actually take action on our behalf rather than just sort of assisting us and getting us 70% of the way there. And so before we start talking about MCP servers and how that enhances our agents abilities, we need to talk about how these tools work and sort of the limitations of them. Okay, so sticking with that email example, let's pretend that this is an email agent that's helping us take action in email. What it's going to do is each tool has a very specific function. So this first tool over here, you can see this one is going to label emails. The second tool in the middle is going to get emails and then this third one on the right is going to send emails. So if you watched my ultimate assistant video, if you haven't, I'll tag it right up here. What happened was we had a main agent and then it was calling on a separate agent that was an email agent. And as you can see here was all of its different tools and each one had one very specific function that it could do. And it was basically just up to the email agent right here to decide which one to use based on the incoming query. And so the reason that these tools aren't super flexible is because within each of these configurations, we basically have to hardcode in what is the operation I'm doing here and what's the resource. And then we can feed in some dynamic things like different message ids or label ids. Over here you know the operation is get the resources message. So that won't change. And then over here the operation is that we're sending a message. And so this was really cool because agents were able to use their brains whatever large language model we had plugged into them to

understand which tool do I need to use. And it still works pretty well. But when it comes to being able to scale this up and you want to interact with multiple different things, not just Gmail and Google Calendar, you also want to interact with a CRM and different databases, that's where it starts to get a little confusing. So now we start to interact with something called an MCP server. And it's basically just going to be a layer between your agent and between the tools that you want to hit, which would be right here. And so when the agent sends a request to the specific MCP server, in this case, let's pretend it's notion, it's going to get more information back than hey, what tools do I have? And what's the functionality here? It's also going to get information about like what are the resources there, what are the schemas there, what are the prompts there, and it uses all of that to understand how to actually take the action that we asked back here in the whole input that triggered the workflow. when it comes to different services talking to each other. So in this case Nadn and notion there's been you know a standard in the way that we send data across and we get data back which has been the rest APIs and these standards are really important because we have to understand how can we actually format our data and send it over and know that it's going to be received in the way that we intend it to be. And so that's exactly what was going on back up here where every time that we wanted to interact with a specific tool we were hitting a specific endpoint. So the endpoint for labeling emails was different for the endpoint for sending emails. And besides just those endpoints or functions being different, there was also different things that we had to configure within each tool call. So over here you can see what we had to do was in order to send an email, we have to give it who it's going to, what the subject is, the email type, and the message, which is different from the information we need to send to this tool, which is what's the message ID you want to label, and what is the label name or ID to give to that message. By going through the MCP server, it's basically going to be a universal translator that takes the information from the LLM and it enriches that with all of the information that we need in order to hit the right tool with the right schema, fill in the right parameters, access the right resources, all that kind of stuff. The reason I put notion here for an example of an MCP server is because within your notion, you'll have multiple different databases and within those databases, you're going to have tons of different columns and then all of those, you know, are going to have different pages. So, being able to have the MCP server translate back to your agent, here are all of the databases you have. Here is the schema or the different fields or columns that are in each of your databases. Um, and also here are the actions you can take. Now, using that information, what do you want to do? Real quick, hopping back to the example of the ultimate assistant. What we have up here is the main agent and then it had four child workflows, child agents that it could hit that had specializations in certain areas. So the Gmail agent, which we talked about right down here, the Google calendar agent, the contact agent, which was Air Table, and then the content creator agent. So all that this agent had to do was understand, okay, based on what's coming in, based on the request from the human, which of these different tools do I actually access? And we can honestly kind of think of these as MCP servers. Because once the query gets passed off to the Gmail agent, the Gmail agent down here is the one that understands here are the tools I have, here are the different like, you know, parameters I need to fill out. I'm going to take care of it and then we're going to respond back to the main agent. This system made things a little more dynamic and flexible because then we didn't have to have the ultimate assistant hooked up to like 40 different tools, you know, all the combinations of all of these. And it made its job a little more easier by just delegating the work to different MCP servers. And obviously, these aren't MCP servers, but it's kind of the same concept. The difference here is that let's say all

of a sudden Gmail adds more functionality. We would have to come in here and add more tools in this case. But what's going on with the MCP servers is whatever MCP server that you're accessing, it's on them to continuously keep that server updated so that people can always access it and do what they need to do. By this point, it should be starting to click, but maybe it's not 100% clear. So, we're going to look at an actual example of like what this really looks like in action. But before we do, just want to cover one thing, which is, you know, the agent sending over a request to a server. the server translates it in order to get all this information and get the tool calls, all that kind of stuff. Um, and what's going on here is called MCP protocol. So, we have the client, which is just the interface that we're using. In this case, it's NN. It could be your claude or your, you know, coding window, whatever it is. And then we're sending over something to the MCP server, and that's called MCP protocol. Also, one thing to keep in mind here that I'm not going to dive into, but if you were to create your own MCP server and it had access to all of your own resources, your schemas, your tools, all that kind of stuff, you just got to be careful there. There's some security concerns because if anyone was getting into that server, they could basically ask for anything back. So, that's something that was brought up in my paid community. We were having a great discussion about MCP and stuff like that. So, just keep it in mind. So, let's look more at an example in Naden once again. So coming down here, let's pretend that we have this beautiful air table agent that we built out in NAN. As you can see, it has these um seven different tools, which is get record, update record, get bases, create record, search record, delete record, and get bases schema. The reason we needed all of these different tools is because, as you know, they each have different operations inside of them, and then they each have different parameters to be filled out. So the agent takes care of all of that. But this could be a lot more lean of a system if we were able to access Air Table's MCP server as you see what we're doing right here because this is able to list all the tools that we have available in Air Table. So here you can see I asked the Air Table agent what actions do I have? It then listed these 13 different actions that we have which are actually more than the seven we had built out here. And we can see we have list records, search records, and then 11 more. And this is actually just the agent telling us the human what we have access to. But what the actual agent would look at in order to use the tool is a list of the tools where it would be here's the name, here's the description of when you use this tool, and then here's the schema of what you need to send over to this tool. Because when we're listing records, we have to send over different information like the base ID, the table ID, max records, how we want to filter, which is different than if we want to list tables because we need a base ID and a detail label. So all of this information coming back from the MCP server tells the agent how it needs to fill out all of these parameters that we were talking about earlier where it's like send email. You have different things than you need to fill out for labeling emails. So once the agent gets this information back from the MCP server, it's going to say okay well I know that I need to use the search records tool because the user asked me to search for records with the name Bob in it. So I have this schema that I need to use and I'm going to use my air tableable execute tool in order to do so. And basically what it's going to do is going to choose which tool it needs based on the information it was fed previously. So in this case the air table execute tool would search records and it would do it by filling in this schema of information that we need to pass over to air tableable. So now I hope you can see how basically what's going on in this tool is all 13 tools wrapped up into one and then what's going on here is just feeding all the information we need in order to make the correct decision. So, this is the workflow we were looking at for the demo. We're not going to dive into this one because it's just a lot to look at. I just wanted to put a ton of MCP servers in one agent and see that even if we had

no system prompt, if it could understand which one to use and then still understand how to call its tools. So, I just thought that was a cool experiment. Obviously, what's next is I'm going to try to build some sort of huge, you know, personal type assistant with a ton of MCP servers. But for now, let's just kind of break it down as simple as possible by looking at an individual MCP agent. And so I I don't know why I called it an MCP agent. In this case, it's just kind of like a firecrawl agent with access to firecraws MCP server. So yeah. Okay. So taking a look at firecraw agent, we're going to ask what tools do you have? It's hitting the firecrawl actions right now in order to pull back all of the resources. And as you can see, it's going to come back and say, hey, we have these, you know, nine actions you can take. I don't know if it's nine, but it's going to be something like that. It was nine. So as you can see, we have access to scrape, map, crawl, batch scrape, all this other stuff. And what's really cool is that if we click into here, we can see that we have a description for when to use each tool and what you actually need to send over. So if we want to scrape, it's a different schema to fill out than if we want to do something like extract. So let's try actually asking it to do something. So let's say um extract the rewards program name from um chipotle.com. So we'll see what it does here. Obviously, it's going to do the same thing, listing its actions, and then it should be using the firecrawl extract method. So, we'll see what comes back out of that tool. Okay, it went green. Hopefully, we actually got a response. It's hitting it again. So, we'll see what happened. We'll dive into the logs after this. Okay, so on the third run, it finally says the rewards program is called Chipotle Rewards. So, let's take a look at run one. It used firecrawl extract and it basically filled in the prompt extract the name of the rewards program. It put it in as a string. We got a request failed with status code 400. So, not sure what happened there. Run two, it did a firecross scrape. We also got a status code 400. And then run three, what it did was a firecall scrape once again, and it was able to scrape the entire thing. And then it used its brain to figure out what the rewards program was called. Taking a quick look at the firewall documentation, we can see that a 400 error code means that the parameters weren't filled out correctly. So what happened here was basically it just didn't fill these out exactly correctly the schema of like the prompt and everything to send over. And so really these kind of issues just come down to a matter of you know making the tool parameters more robust and also more prompting within the actual firecrawl agent itself. But it's pretty cool that it was able to understand okay this didn't work. Let me just try some other things. Okay, so real quick just wanted to say if you want to hop into Nit and test out these MCP nodes, you're going to have to self-host your environment because you need to use the community nodes and you can only access those if you are self-hosted. Today we're going to be going through the full setup of connecting MCP servers to NN. I'm going to walk through how you self-host your NN. I'm going to walk through how you can install the community node and then how to actually set up the community node. The best part is you don't have to open up a terminal or a shell and type in any install commands. All we have to do is connect to the servers through NIND. So, if that sounds like something that interests you, let's dive into it. Make sure you guys stick around for the end of this one because we're going to talk about the current limitations of these MCP nodes. We're going to talk about some problems you may face that no one else is talking about and really what does it mean to actually be able to scale these agents with MCP servers. Now, there are a ton of different platforms that you can use to host NN. The reason I'm using Alstio is because it's going to be really simple for deploying and managing open- source software. Especially with something like NN, you can pretty much deploy it in just a few clicks and it's going to take care of installation, configuration, security, backups, updates, all this kind of stuff. So, there's no need for you to have that DevOps knowledge because I certainly don't. So, that's why

we're going with Alstio. It's also SOCK 2 and GDPR compliant. So, that's important. Anyways, like I said, we're going to be going through the full process. So, I'm going to click on free trial. I'm going to sign up with a new account. So, I'm going to log out and sign up as a new user. Okay. Now that I entered that, we're already good to go. The first thing I'm going to do is set up a payment method so that we can actually spin up a service. So, I went down to my account in the lefth hand side and then I clicked on payment options and I'm going to add a card real quick. Now that that's been added, it's going to take a few minutes for our account to actually be approved. So, I'm just going to wait for that. You can see we have a support ticket that got opened up, which is just waiting for account activation. Also, here's the approval email I got. Just keep in mind it says it'll be activated in a few minutes during business hours, but if you're doing this at night or on the weekends, it may take a little longer. Okay, there we go. We are now activated. So, I'm going to come up here to services and I'm going to add a new service. What we're going to do is just type in nadn and it's going to be a really quick oneclick install. Basically, I'm going to just be deploying on htzner as a cloud provider. I'm going to switch to my region and then you have different options for service plans. So, these options obviously have different numbers of CPUs, different amount of RAM and storage. I'm going to start right now on just the medium. I would keep in mind that MCP servers can be kind of resource intensive. So, if you are running multiple of them and your environment is crashing, then you're probably just going to want to come in here and upgrade your service plan. So, we can see down here, here is the estimated hourly price. Here's the plan we're going with. And I'm going to go ahead and move forward. Now, we're going to set up the name. So, this will pop up as your domain for your NAND environment. Then, I went ahead and called this Nad- demo. What you can do here is you can add more volume. So, if you wanted to, you could increase the amount of storage. And as you can see down here, it's going to increase your hourly price. I'm not going to do that, but you do have that option. And then of course you have some advanced configuration for software updates and system updates. Once again, I'm just going to leave that as is. And then you can also choose the level of support that you need. You can scan through your different options here. Obviously, you'll have a different price associated with it. But on the default plan, I'm just going to continue with level one support. And now I'm going to click on create service. Okay. So, I don't have enough credits to actually deploy this service. So, I'm going to have to go add some credits in my account. So, back in the account, I went to add credits. And now that I have a card, I can actually add some credits. So, I'm going to agree to the terms and add funds. Payment successful. Nice. We have some money to play with. Down here, we can see 10 credits. This is also where we'll see how much we're spending per hour once we have this service up and running. Unfortunately, we have to do that all again. So, let me get back to the screen we were just on. Okay, now we're back here. I'm going to click create service. We're deploying your service. Please wait. And this is basically just going to take a few minutes to spin up. Okay, so now what we see is the service is currently running. We can click into the service and we should be able to get a link that's going to take us to our NN instance. So, here's what it looks like. We can see it's running. and we have all these different tabs and all these different things to look through. We're going to keep it simple today and not really dive into it. But what we're going to do is come down here to our network and this is our actual domain to go to. So if I select all of this and I just click go to this app, it's going to spin up our NN environment and because this is the first time we're visiting it. We just have to do the setup. Okay. So once we have that configured, going to hit next. We have to do some fun little onboarding where it's going to ask us some questions right here. So then when you're done with that, you just got to click get started and

you now have this option to get some paid features for free. I'm going to hit skip and we're good to go. We are in NAN. So what's next is we need to install a community node. So if you come down here to your settings and you click on settings, um you can see you're on the community plan. We can go all the way down here to community nodes. And now we have to install a community node. So in the description we have the GitHub repository for this NAD nodes MCP that was made by Nerding. And you can see there's some information on how to actually install this. But all we have to do is basically just copy this line right here. I'm just going to copy NAN- Nodes MCP. Click on install community node. Put that in there. Hit understand. And so the reason you can only do this on self-hosted is because these nodes are not a native verified node from Naden. So it's just like, you know, we're downloading it from a public source, at least the code. And then we hit install. Package installed. We can now see we have one community node, which is the MCP. Cool. So I'm going to leave my settings and I'm going to open up a new workflow. And we're just going to hit tab to see if we have the actual node. So if I type in MCP, we can see that we have MCP client and we have this little block, which just means that it is part of the community node. So, I'm going to click into here, and we can see we have some different options. We can execute a tool. We can get a prompt template. We can list available prompts, list available resources, list available tools, and read a resource. Right now, let's go with list available tools. Um, the main one we'll be looking at is listing tools and then executing tools. So, quick plug for the school community. If you're looking for a more hands-on learning experience, as well as wanting to connect with over 700 members who are also dedicated to this rapidly evolving space, then definitely give this community a look. We have great discussions, great guest speakers as you can see. We also have a classroom section with stuff like building agents, vector databases, APIs and HTTP request, step-by-step builds. All the live calls are recorded, all this kind of stuff. So, if this sounds interesting to you, then I'd love to see you in a live call. Anyways, let's get back to the video. So, obviously, we have the operation, but we haven't set up a credential yet. So now what you're going to do is go to a different link in the description which is the GitHub repository for different MCP servers and we can pretty much connect to any of these like I said without having to run any code in our terminal and install some stuff at least because we're hosting in the cloud. If we're hosting locally it may be a little different. Okay, so I've seen a ton of tutorials go over like Brave Search or Firecrawl. Um so let's try to do something a little more fun. I think first we'll start off with Airbnb because this one is going to be free. You don't even have to go get an API key. So that's really cool. So, I'm going to click into this Airbnb MCP server. There's a bunch of stuff going on here. And if you understand GitHub and repositories and some code, you can look through like the Docker file and everything, which is pretty cool. But for us non techies, all we have to do is come down here. It's going to tell us what tools are available. But we just need to look at how to actually install this. And so, all we're looking for is the MPX type installer. And so after my testing, I tried this one first, but it wouldn't let us execute the tool because we need to use this thing that is ignore robots text, which just basically lets us actually access the platform. So you can see here we have a command, which is npx, and then we have an array of arguments, which is -y, this open B&B thing, and then also the ignore robots text. So first of all, I'm just going to grab the command, which is npx. Copy that. Go back and edit in, and we're going to create a new credential. This one's going to be for Airbnb. So I'm just going to name this so we have it kept. And then we're just going to paste the command right into there, mpx. Now we can see we have arguments to fill out. So I'm going to go back into that documentation. We can see the arguments are -ash-y. And then the next one is the open B&B. And then the next one is ignore robots text. So we're going to put them in one by one.

So first is the dashy. Now I'm going to go back and grab the at@ openb put a space and then paste it in there. And then I'm going to put another space. And then we're going to grab this last part which is the ignore robots txt. So once we paste that in there, we can basically just hit save. As you can see, we've connected successfully. Um the credential is now in our space and we didn't have to type in anything in a terminal. And now if we hit test step, we should be able to pull in the tools that this MCP server gives us access to. So it's as easy as that. As you can see, there are two tools. The first one is Airbnb search. Here's when you use it, and then here's the schema to send over to that tool. And then the second one is Airbnb listing details. Here's when you want to use that, and then here's the schema that you would send over. And now from here, which is really cool, we can click on another node, which is going to be an MCP client once again. And this time, we want to execute a tool. We already have our credential set up. We just did that together. And now all we have to do is configure the tool name and the tool parameters. So just as a quick demo that this actually works. The tool name, we're going to drag in Airbnb search, as you can see. And then for the parameter, we can see these are the different things that we need to fill out. And so all I'm going to do is just send over a location. So I obviously hardcoded in location equals Los Angeles. That's all we're going to try. And now we're going to hit test step and we should see that we're getting Airbnbs back that are in Los Angeles. There we go. We have um ton of different items here. So, let's actually take a look at this listing. So, if I just copy this into the browser, we should see an Airbnb arts district guest house. This is in Culver City, California. And obviously, we could make our search more refined if we were able to also put in like a check-in and checkout date, how many adults, how many children, how many pets. We could specify the price, all this kind of stuff. Okay, cool. So that was an example of how we search through an MCP server to get the tools and then how we can actually execute upon that tool. But now if we want to give our agent access to an MCP server, what we would do is obviously we're going to add an AI agent. We are first of all going to come down here, give it a chat input so we can actually talk to the agent. So we'll add that right here. And now we obviously need to connect a chat model so that the agent has a brain and then give it the MCP tools. So first of all, just to connect a chat model, I'm just going to grab an open AI. I'm sorry for being boring, but all we have to do is create a credential. So, if you go to your OpenAI account and grab an API key. So, here's my account. As you can see, I have a ton of different keys, but I'm just going to create a new one. This is going to be MCP test and then all we have to do is copy that key. Come back and end it in and we're going to paste that right in here. So, there's our key. Hit save. We'll go green. We're good to go. We're connected to OpenAI. And now we can choose our model. So, for Mini is going to work just fine here. Now, to add a tool once again, we're going to add the MCP client tool right here. And let's just do Airbnb one more time. So, we're connected to Airbnb list tools and I'm just going to say what tools do I have and what's going to happen is it errored because the NAND nodes MCP tool is not recognized yet even though the MCP nodes are. So, we have to go back into Alstio real quick and change one thing. So, coming back into the GitHub repository for the NN MCP node, we can see it gives us some installation information, right? But if we go all the way down to how to use it as a tool, um if I can scroll all the way down here. So here is an example of using it as a tool. You have to set up the environment variable within your hosting environment. So whether it's Allesio or Render or Digital Ocean or wherever you're doing it, it'll be a little different, but you just have to navigate down to where you have environment variables. We have to set nad community package_allow tool usage. We have to set that to equal true. So I'm going to come back into our Alstio service. And right here we have the software which is NAN version latest. And what we can do is we

can you know restart, view app logs. We can change the version here or we can update the config which if we open this up it may look a little intimidating but all we're looking for is right here we have environment and we can see we have like different stuff with our Postgress with our web hook tunnel URLs all this kind of stuff and so at the bottom I'm just going to add a new line and I'm just going to paste in that command we had which was nadn community packages allow and then instead of an equal I'm going to put a colon and now we have that nadn community packages allow is set to true and I'm just adding a space after the colon so now it's link and all we're just going to do is hit update and restart. And so this is going to respin up our instance. Okay, so it looks like we are now finished up. I'm going to go ahead and close out of this. We can see that our instance is running. So now I'm going to come back into here and I actually refresh this. So our agent's gone. So let me get him back real quick. All right, so we have our agent back. We're going to go ahead and add that MCP tool once again. Right here we are going to have our credential already set up. The operation is list tools. And now let's try one more time asking it what tools do you have? And it knows to use this node because it's the operation here is list tools. So it's going to be pretty intelligent about it. Now it's able to actually call that tool because we set up that environment variable. So let's see what Airbnb responds with as far as what tools it actually can use. Cool. So I have access to the following tools. Airbnb search and listing details. Now let's add the actual tool that's going to execute on that tool. So Airbnb um once again we have a credential already set up. The operation we're going to choose execute tool instead. And now we have to set up what is going on within this tool. So the idea here is that when the client responds with okay I have Airbnb search and I have Airbnb listing details the agent will then figure out based on what we asked which one do I use and the agent has to pass that over to this next one which is actually going to execute. So what we want to do here is the tool name cannot be fixed because we want to make this dynamic. So, I'm going to change this to an expression and I'm going to use the handy from AI function here, which is basically we're just going to tell the AI agent, okay? You know, based on what's going on, you choose which tool to use and you're going to put that in here. So, I'm going to put in quotes tool and then I'm going to just define what that means. And in quotes after a comma, I'm going to say the tool selected. So, we'll just leave it as simple as that. And then what's really cool is for the tool parameters, this is going to change based on the actual tool selected because there's different schemas or parameters that you can send over to the different tools. So we're going to start off by just hitting this button, which lets the model define this parameter. It's going to get back what not only what tool am I using, but what schema do I need to send over. So it should be intelligent enough to figure it out for simple queries. So let's change this name to Airbnb execute. I'm going to change this other one to Airbnb tools and then we'll have the agent try to figure out what's going on. And just a reminder, there's no system prompt in here. It literally just says your helpful assistant. So, we'll see how intelligent this stuff is. Okay, so I'm asking it to search for Airbnbs in Chicago for four adults. Let's try that off. We should obviously be using the Airbnb search tool. And then we want to see if it can fill out the parameters with a location, but also how many adults are going to be there because earlier all we did was location. So, we got a successful response already back. Once this finishes up, we should see potentially a few links down here that actually link to places. So, here we go. Um, luxury designer penthouse Gold Coast. It's apartment. It has three bedrooms, eight beds. So, that definitely fits four guests. And you can also see it's going to give us the price per night as well as, you know, the rating and just some other information. So, let's click into this one real quick and we'll take a look. Make sure it actually is in Chicago and it has all the stuff. This one does have 10 guests. So, awesome. And we can see we got five total listings.

So without having to configure, you know, here's the API documentation and here's how we set up our HTTP request, we're already able to do some pretty cool Airbnb searches. So let's take a look in the Airbnb execute tool. We can see that what it sent over was a location as well as a number of adults, which is absolutely perfect. The model was able to determine how to format that and send it over as JSON. And then we got back our actual search results. And now we're going to do something where you actually do need an API key because most of these you are going to need an API key. So we're going to go ahead and do Brave search because you can search the web um using Brave Search API. So we're going to click into this and all we have to do is once again we can see the tools here but we want to scroll down and see how you actually configure it. So the first step is to go to Brave Search and get an API key. You can click on this link right here and you'll be able to sign in and get 2,000 free queries and then you'll grab your API key. So I'm going to log in real quick. So it may send you a code to your email to verify it. You'll just put in the code, of course, and then we're here. As you can see, I've only done one request so far. I'm going to click on API keys on this lefth hand side, and we're just going to copy this token, and then we can put it into our configuration. So, let's walk through how we're going to do that. So, I'm going to come in here and add a new tool. We're going to add another MCP client tool, and we're going to create a new credential because we're no longer connecting to Airbnb's server. We're connecting to Brave Search Server. So, create new credential. Let me just name this one real quick so we don't get confused. And then of course we have to set up our command, our arguments, and our environments. And this is where we're going to put our actual API key. Okay, so first things first, the command. Coming back into the Brave Search MCP server documentation, we can see that we can either do Docker, but what we're doing every time we're connecting to this in NN is going to be MPX. So our command once again is MPX. Copy that, paste it into the command. And now let's go back and get our arguments, which is always going to start off with -ashy. Then after that, put a space. We're going to connect to this MCP server, which is the Brave Search. And then you can see that's it. In the Airbnb one, we had to add the robots text. In this one, we didn't. So, everyone is going to configure a little bit differently, but all you have to do is just read through the command, the arguments, and then the environment variables. And in this case, unlike the Airbnb one, we actually do need an API key. So, what we're going to do is we're going to put in all caps brave_api_key. So, in the environment variables, I'm going to change this to an expression just so we can actually see. Brave API_key. And then I'm going to put an equals and then it says to put your actual API key. So that's where we're going to paste in the API key from Brave Search. Okay. So I put in my API key. Obviously I'm going to remove that after this video gets uploaded. But now we'll hit save and we'll make sure that we're good to go. Cool. And now we're going to actually test this out. So I'm going to call this Brave Search Tools. Um and then before we add the actual execute tool, I'm just going to ask and make sure it works. So what Brave Search tools do you have? And it knows of course to hit the brave search because we gave it a good name. And it should be pulling back with its different functions which I believe there are two. Okay. So we have Brave web search and we have Brave local search. We also have, you know, of course the description of when to use each one and the actual schemas to send over. So let's add a tool and make sure that it's working. We're going to click on the plus. We're going to add an MCP client tool. We already have our Brave credential connected. We're going to change the operation to execute tool. And once again, we're going to fill in the tool name and the parameters. So for the tool name, same exact thing. We're going to do from AI. And once again, this is just telling the AI what to fill in here. So we're going to call it tool. We're going to give it a very brief description of the tool

selected. And then we are just going to enable the tool parameters to be filled out by the model automatically. Final thing is just to call this Brave search execute. Cool. There we go. So now we have um two functions for Airbnb, two for Brave search, and let's make sure that the agent can actually distinguish between which one to use. So I'm going to say search the web for information about AI agents. So we'll send that off. Looks like it's going straight to the Brave Search execute. So we may have to get into the system prompt and tweak it a little bit. Now it's going back to the Brave Search tools to understand, okay, what actions can I take? And now it's going back to the Brave Search execute tool. And hopefully this time it'll get it right. So, it looks like it's going to compile an answer right now based on its search result and then we'll see exactly what happened. There we go. So, we have looks like Oh, wow. It gave us nine different articles. Um, what are AI agents by IBM? We can click into here to read more. So, this takes us straight to IBM's article about AI agents. We have one also from AWS. We can click into there. There's Amazon. And let's go all the way to the bottom. We also have one on agents from Cloudflare. So, let's click into here. And we can see it took us exactly to the right place. So super cool. We didn't have to configure any sort of API documentation. As you can see in Brave Search, if we wanted to connect to this a different way, we would have had to copy this curl command, statically set up the different headers and the parameters. But now with this server, we can just hit it right away. So let's take a look in the agent logs, though, because we want to see what happened. So the first time it tried to go straight to the execute tool and as you can see it filled in the parameters incorrectly as well as the actual tool name because it didn't have the information from the server. Then it realized okay I need to go here first so that I can find out what I can do. I tried to use a tool called web search as you can see earlier web search. But what I needed to do was use a tool called brave web search. So now on the second try back to the tool it got it right and it said brave web search. It also filled out some other information like how many articles are we looking for and what's the offset. So if we were to come back in here and say get me one article on dogs. Let's see what it would do. So hopefully it's going to fill in the count as one. Once again it went straight to the tool and it may I was going to say if we had memory in the agent it probably would have worked because it would have seen that it used brave web search previously but there's no memory here. So, it did the exact same pattern and we would basically just have to prompt in this agent, hey, search the MCP server to get the tools before you try to execute a tool. But now we can see it found one article. It's called it's just Wikipedia. So, we can click in here and see it's dog on Wikipedia. But if we click into the actual Brave search execute tool, we can see that what it filled out for the query was dogs and it also knew to make the count one rather than last time it was 10. Okay. Okay, so something I want you guys to keep in mind is when you're connecting to different MCP servers, the setup will always be the same where you'll look in the GitHub repository, you'll look at the command, which will be npx, you'll look at the arguments, which will be -ashy, space, the name of the server, and then sometimes there'll be more. And then after that, you'll do your environment variable, which is going to be a credential, some sort of API key. So here, what we did was we asked Air Table to list its actions. And in this case, as you can see, it has 13 different actions. And within each action, there's going to be different parameters to send over. So, when you start to scale up to some of these MCP servers that have more actions and more parameters, you're going to have to be a little more specific with your prompting. As you can see in this agent, there's no prompting going on. It's just your helpful assistant. And what I'm going to try is in my Air Table, I have a base called contacts, a table called leads, and then we have this one record. So, let's try to ask it to get that record. Okay. So, I'm asking it to get the records in my Air Table base called contacts, in

my table called leads. Okay, so we got the error receive tool input did not match expected schema. And so this really is because what has to happen here is kind of complex. It has to first of all go get the bases to grab the base ID and then it has to go grab the tables in that base to get the table ID and then it has to formulate that over in a response over here. As you can see, if the operation was to list records, it has to fill out the base ID and the table ID in order to actually get those records back. So that's why it's having trouble with that. And so a great example of that is within my email agent for my ultimate assistant. In order to do something like label emails, we have to send over the message ID of the email that we want to label. And we have to send over the ID of the label to actually add to that message. And in order to do those two things, we first have to get all emails to get the message ID. And then we have to get labels to get the label ID. So it's a multi-step process. And that's why this agent with minimal prompting and not a super robust parameter in here. It's literally just defining by the model, it's a little bit tough. But if I said something like get my air table bases, we'll see if it can handle that because that's more of a one-step function. And it looks like it's having trouble because if we click into this actions, we can see that the operation of listing bases sends over an empty array. So it's having trouble being able to like send that data over. Okay, so I'm curious though. I went into my Air Table and I grabbed a base ID. Now I'm going to ask what tables are in this air table base ID and I gave it the base ID directly so it won't have to do that list basis function. And now we can see it actually is able to call the tool hopefully. So it's green and it probably put in that base ID and we'll be able to see what it's doing here. But this just shows you there are still obviously some limitations and I'm hoping that Nad will make a native you know MCP server node. But look what it was able to do now is it has here are the tables within the air table base ID that we provided and these are the four tables and this is correct. And so now I'm asking it what records are in the air table base ID of this one and the table ID of this one. And it should be able to actually use its list records tool now in order to fill in those different parameters. And hopefully we can see our record back which should be Robert California. So we got a successful tool execute as you can see. Let's wait for this to pop back into the agent and then respond to us. So now we have our actual correct record. Robert California Saber custom AI solution all this kind of stuff. And as you can see, that's exactly what we're looking at within our actual Air Table base. And so, I just thought that that would be important to show off here how this is like really cool, but it's not fully there yet. So, I definitely think it will get there, especially if we get some more native integrations with Naden. But, I thought that that would be a good demo to show the way that it needs to fill in these parameters in order to get records. And, you know, this type of example applies to tons of different things that you'll do within MCP servers. So, there's one more thing I want to show you guys real quick, just so you will not be banging your head against the wall the way I was a couple days ago when I was trying to set up Perplexity. So, because you have all these different servers to choose from, you may just trust that they're all going to be the exact same and they're going to work the same. So, when I went to set up the Plexity ask MCP server, I was pretty excited. Command was mpx. I put in my arguments. I put in my environment variable, which was my perplexity API key. And you can see I set this up exactly as it should be. My API keys in there. I triple checked to make sure it was correct. And then when I went to test step, basically what happened was couldn't connect to the MCP server. Connection closed. And so after digging into what this meant, because I set up all these other ones, as you can see in here, I did these and I have more that I've connected to. The reason why this one isn't working, I imagine, is because on the server side of things, on Perplexity side of things, it's either going undergoing maintenance or it's not fully published yet. And it's not anything wrong with the way that

you're deploying it. So, I just wanted to throw that out there because there may be some other ones in this big list that are not fully there yet. So, if you are experiencing that error and you know that you're filling out that, you know, MPX and the arguments and the environment variable correct, then that's probably why don't spend all day on it. Just wanted to throw that out there because, you know, I had I had a moment the other day. Well, it's been a fun journey. I appreciate you guys spending all this time with me. We've got one more section to close off on and this is going to be kind of just the biggest lessons that I had learned over the first six months of me building AI agents as a non-programmer. Let's go. Because everyone's talking about this kind of stuff, there's a lot of hype and there's a lot of noise to cut through. So, the first thing I want to do is talk about the hard truths about AI agents and then I'll get into the seven lessons that I've learned over the past six months building these things. So, the first one is that most AI agent demos online are just that, they're demos. So, the kind of stuff that you're going to see on LinkedIn, blog posts, YouTube, admittedly, my own videos as well, these are not going to be productionready builds or productionready templates that you could immediately start to plug into your own business or try to sell to other businesses. You'll see all sorts of cool use cases like web researchers, salespeople, travel agents. Just for some context, these are screenshots of some of the videos I've made on YouTube. This one is like a content creator. This one is a human and loop calendar agent. We've got a technical analyst. We have a personal assistant with all its agents over here. stuff like that. But the reality is that all of these pretty much are just going to be, you know, proof of concepts. They're meant to open everyone's eyes to what this kind of stuff looks like visually, how you can spin this kind of stuff up, the fundamentals that go into building these workflows. And at least me personally, my biggest motivation in making these videos is to show you guys how you can actually start to build some really cool stuff with zero programming background. And so why do I give all those templates away for free? It's because I want you guys to download them, hit run, see the data flow through and understand what's going on within each node rather than being able to sell that or use it directly in your business because everyone has different integrations. Everyone's going to have different system prompting and different little tweaks that they need for an automation to be actually high value for them. Besides that, a lot of this is meant to be within a testing environment, but if you push it into production and you expose it to all the different edge cases and tons of different users, things are going to come through differently and the automation is going to break. And what you need to think about is even these massive companies in the space like Apple, Google, Amazon, they're also having issues with AI reliability like what we saw with Apple intelligence having to be delayed. So if a company like this with a massive amount of resources is struggling with some of these productionready deployments, then it's kind of unrealistic to think that a beginner or non-programmer in these tools can spin up something in a few days that would be fully production ready. And by that I just mean like the stuff you see online. You could easily get into nodn, build something, test it, and get it really robust in order to sell it. That's not what I'm saying at all. Just kind of the stuff you see online isn't there yet. Now, the second thing is being able to understand the difference between AI agents and AI workflows. And it's one of those buzzwords that everyone's kind of calling everything an agent when in reality that's not the truth. So, a lot of times people are calling things AI agents, even if they're just sort of like an AI powered workflow. Now, what's an AI powered workflow? Well, as you can see right down here, this is one that I had built out. And this is an AI powered workflow because it's very sequential. As you can see, the data moves from here to here to here to here to here to here. And it goes down that process every time. Even though there are some elements in here using AI like

this basic chain and this email writing agent. Now, this has a fixed amount of steps and it flows in this path every single time. Whereas something over here like an AI agent, it has different tools that it's able to call and based on the input, we're not sure if it's going to call each one once or it's going to call this one four times or if it's going to call this one and then this one. So that's more of a non-deterministic workflow. And that's when you need to use something like an AI agent. The difference here is that it's choosing its own steps. The process is not predefined, meaning every time we throw an input, we're not sure what's going to happen and what we're going to get back. And then the agent also loops, calls its tools, it observes what happens, and then it reloops and thinks about it again until it realizes, okay, based on the input, I've done my job. Now I'm going to spit something out. And so here's just a different visualization of, you know, an AI agent with an input, the agent has decision, and then there's an output or this AI workflow where we have an input, tool one, LLM call, tool two, tool three, output where it's going to happen in that process every single time. And the truth is that most problems don't require true AI agents. they can simply be solved with building a workflow that is enhanced with AI. And a common mistake, and I think it's just because of all the hype around AI agents, is that people are opting straight away to set up an agent. Like in this example right here, let's say the input is a form trigger where we're getting a form response. We're using this tool to clean up the data. We're using this LLM call. So it's an AI enhanced workflow to actually write a personalized email. We're using this to update the CRM and then we're using this to send the email and then we get the output back as the human. We could also set this up as a AI agent where we're getting the form response. We're sending this agent the information and it can choose, okay, first I'm going to clean the data and then I'm going to come back here and think about it and then I'm going to update the CRM and then I'm going to create an email and then I'm going to send the email and then I'm going to output and respond to the human and tell it that, you know, we we did that job for you. But because this process is pretty linear, it's going to be a lot more consistent if we do a workflow. It's going to be easier to debug. Whereas over here, the agent may mess up some tool calls and do things in the wrong order. So it's better to just structure it out like that. And so if we start approaching using these no code tools to build AI workflows first, then we can start to scale up to agents once we need more dynamic decision-m and tool calling. Okay, but that's enough of the harsh truths. Let's get into the seven most important lessons I've learned over the six months of building AI agents as a non-programmer. So the first one is to build workflows first. And notice I don't even say AI workflows here, I say workflows. So, over the past six months of building out these systems and hopping on discovery calls with clients where I'm trying to help them implement AI into their business processes, we always start by, you know, having them explain to me some of their pain points and we talk through processes that are repetitive and processes that are a big time suck. And a lot of times they'll come in, you know, thinking they need an AI agent or two. And when we really start to break down this process, I realize this doesn't need to be an agent. This could be an AI workflow. And then we break down the process even more and I'm like, we don't even need AI here. We just need rule-based automation and we're going to send data from A to B and just do some manipulation in the middle. So let's look at this flowchart for example. Here we have a form submission. We're going to store data. We're going to route it based on if it's sales, support, or general. We'll have that ticket or notification. Send an automated acknowledgement. And then we'll end the process. So this could be a case where we don't even need AI. If we're having the forms come through and there's already predefined these three types which are either sales, support, or general, that's a really easy rules-based automation. Meaning, does inquiry type equal sales? If yes,

we'll go this way and so on and so forth. Now, maybe there's AI we need over here to actually send that auto acknowledgement or it could be as simple as an automated message that we're able to define based on the inquiry type. Now, if this the form submission is just a a block of text and we need an AI to read it, understand it and decide if it's sales, support, or general, then we would need AI right here. And that's where we would have to assess what the data looks like coming in and then what we need to do with the data. So, it's always important to think about, do we even need AI here? Because a lot of times when we're trying to cut off some of that lowhanging fruit, when we realize that we're doing some of this stuff too manually, we don't even need AI yet. We're just going to create a few workflow automations and then we can start getting more advanced with adding AI in certain steps. So hopefully this graphic adds a little more color here. On the left we're looking at a rule-based sort of filter and on the right we're looking at an AI powered filter. So let's take a look at the left one first. We have incoming data. So let's just say we're routing data off based on if someone's budget is greater than 10 or less than 10. Hopefully it's greater than 10. Um so the filter here is is X greater than 10? If yes, we'll send it up towards tool one. If no, we're going to send it down towards tool two. And those are the only two options because those are the only two buckets that a number can fit into. Unless I guess it's exactly equal to 10. I probably should have made this sign a greater than or equal to, but anyways, you guys get the point. Now, over here, if we're looking at an AI powered sort of filter right here, we're using a large language model to evaluate the incoming data, answer some sort of question, and then route it off based on criteria. So incoming data we have to look at or sorry not we the AI is looking at what type of email this is because this uses some element of reasoning or logic or decision-m something that actually needs to be able to read the context and understand the meaning of what's coming through in order to make that decision. This is where before AI we would have to have a human in the loop. We'd have to have a human look at this data and analyze which way it's going to go rather than being able to write some sort of code or filter to do so because it's more than just like what words exist. It's actually like when these words come together in sentences and paragraphs, what does it mean? So AI is able to read that and understand it and now it can decide if it's a complaint, if it's billing or if it's promotion and then based on what type it is, we'll send it off to a different tool to take the next action. So the big takeaway here is to find the simplest approach first. You may not even need an agent at all. So why would you add more complexity if you don't have to? And also if you start to learn the fundamentals of workflow automation, data flow, logic, creative problem solving, all that kind of stuff, it's going to make it so much easier when you decide to scale up and start building out these multi-aggentic systems as far as, you know, sending data between workflows and understanding routing. Your life's going to be a lot easier. So only use AI where it actually is going to provide value. And also using AI and hitting an LLM isn't free typically. And I mean if you're self-hosting, but anyways, it's not free. So why would you want to spend that extra money in your workflow if you don't have to? You can scale up when you need the system to decide the steps on its own, when you need it to handle more complex multi-step reasoning, and when you needed to control usage dynamically. And I highlighted those three words because that's very like human sounding, right? Decide, reason, dynamic. Okay, moving on to lesson number two. This is to wireframe before you actually get in there and start building. One of the biggest mistakes that I made early on and that I see a ton of people making early on is jumping straight into their builder, whatever it is, and trying to get the idea in their head onto a canvas without mapping it out at all. And this causes a lot of problems. So, the three main ones here are you you start to create these messy, over complicated workflows because you haven't thought out the whole process yet.

You're going to get confused over where you actually need AI and where you don't. and you may end up spending hours and hours debugging, trying to revise um all this kind of stuff because you didn't consider either a certain integration up front or a certain functionality up front or you didn't realize that this could be broken down into different workflows and it would make the whole thing more efficient. I can't tell you how many times when I started off building these kind of things that I got almost near the end and I realized I could have done this with like 20 less nodes or I could have done this in two workflows and made it a lot simpler. So, I end up just deleting everything and restarting. So what we're looking at right here are a different Excalibraw wireframes that I had done. As you can see, I kind of do them differently each time. There's not really a, you know, defined way that you need to do this correctly or correct color coding or shapes. The idea here is just to get your thoughts from your head onto a paper or onto a screen and map it out before you get into your workflow builder because then in the process of mapping things out, you're going to understand, okay, there may be some complexities here or I need all of this functionality here that I didn't think of before. And this isn't really to say that there's one correct way to wireframe. As you can see, sometimes I do it differently. Um, there's not like a designated schema or color type or shape type that you should be using. Whatever makes sense to you really. But the idea here is even if you don't want to wireframe and visually map stuff out, it's just about planning before you actually start building. So, how can you break this whole project? You know, a lot of people ask me, I have an input and I know what that looks like and I know what I want to get over here, but in between I have no idea what that looks like. So, how can we break this whole project into workflows? And each workflow is going to have like individual tasks within that workflow. So, breaking it down to as many small tasks as possible makes it a lot more easy to handle. Makes it a lot less overwhelming than looking at the entire thing at once and thinking, how do I get from A to Z? And so, what that looks like to either wireframe or to just write down the steps is you want to think about what triggers this workflow. How does this process start? And what does the data look like coming in that triggers it? From there, how does the data move? Where does it go? Am I able to send it down one path? Do I have to send it off different ways based on some conditional logic? Do I need some aspect of AI to take decisions based on the different types of data coming through? You know, what actions have to be taken? Where do we need rag or API calls involved? Where do we need to go out somewhere to get more external data to enrich the context going through to the next LLM? What integrations are involved? So, if you ask yourself these kind of questions while you're writing down the steps or while you're wireframing out the skeleton of the build, you are going to answer so many more questions, especially if it comes to, you know, you're trying to work with a client and you're trying to understand the scope of work and understand what the solution is going to look like. If you wireframe it out, you're going to have questions for them that they might have not have thought of either, rather than you guys agree on a scope of work and you start building this thing out and then all of a sudden there's all these complexities. Maybe you priced way too low. Maybe you don't know the functionality. And the idea here is just to completely align on what you're trying to build and what the client wants or what you're trying to build and what you're actually going to do in your canvas. So there's multiple use cases, but the idea here is that it's just going to be so so helpful. And because you're able to break down every single step and every task involved, you'll have a super clear idea on if it's going to be an agent or if it's going to be a workflow because you'll see if the stuff happens in the same order or if there's an aspect of decision-m involved. So, when I'm approaching a client build or an internal automation that I'm trying to build for myself, there is no way that more than half my time is spent in the builder. pretty much upfront I'm

doing all of the wireframing and understanding what this is going to look like because the goal here is that we're basically creating a step-by-step instruction manual of how to put the pieces together. You should think of it as if you're putting together a Lego set. So, you would never grab all the pieces from your bag of Legos, rip it open, and just start putting them together and trying to figure out where what goes where. You're always going to have right next to you that manual where you're looking at like basically the step-by-step instructions and flipping through. So, that's what I do with my two monitors. On the left, I have my wireframe. On the right, I have my NADN and I'm just looking back and forth and connecting the pieces where I know the integrations are supposed to be. You need a clear plan. Otherwise, you're not going to know how everything fits together. It's like you were trying to, you know, build a 500 piece puzzle, but you're not allowed to look at the actual picture of a completed puzzle, and you're kind of blindly trying to put them together. You can do it. It can work, but it's going to take a lot longer. Moving on to number three, we have context is everything. The AI is only going to be as good as the information that you provide it. It is really cool. The tech has come so far. These AI models are super super intelligent, but they're pre-trained. So, they can't just figure things out, especially if they're operating within a specific domain where there's, you know, industry jargon or your specific business processes. It needs your subject matter expertise in order to actually be effective. It doesn't think like we do. It doesn't have past experiences or intuition, at least right away. We can give it stuff like that. It only works with the data it's given. So, garbage in equals garbage out. So, what happens if you don't provide high quality context? Hallucination. The AI is going to start to make up stuff. Tool misuse. It's not going to use the tools correctly and it's going to fail to achieve your tasks that you need it to do. And then vague responses. If it doesn't have clear direction and a clear sight of like what is the goal? What am I trying to do here? It is just not going to be useful. It's going to be generic and it's going to sound very obviously like it came from a chat GBT. So, a perfect example here is the salesperson analogy. Let's say you hire a superstar salesman who is amazing, great sales technique. He understands how to build rapport, closing techniques, communication skills, just like maybe you're taking a GPT40 model out of the box and you're plugging it into your agent. Now, no matter how good that model is or the salesperson is, there are going to be no closed sales without the subject matter expertise, the business process knowledge, you know, understanding the pricing, the features, the examples, all that kind of stuff. So, the question becomes, how do you actually provide your AI agents with better context? And there are three main ways here. The first one is within your agent you have a system prompt. So this is kind of like the fine-tuning of the model where we're training it on this is your role. This is how you should behave. This is what you're supposed to do. Then we have the sort of memory of the agent which is more of the short-term memory we're referring to right here where it can understand like the past 10 interactions it had with the user based on the input stuff like that. And then the final aspect which is very very powerful is the rag aspect where it's able to go retrieve information that it doesn't currently have but it's able to understand what do I need to go get and where can I go get it. So it can either hit different APIs to get real-time data or it can hit its knowledge base that hopefully is syncing dynamically and is updated. So either way it's reaching outside of its system prompt to get more information from these external sources. So anyways preloaded knowledge. This is basically where you tell the agent its job, its description, its role. As if on day one of a summer internship, you told the intern, "Okay, this is what you're going to do all summer." You would define its job responsibilities. You would give key facts about your business, and you would give it rules and guidelines to follow. And then we move on to the user specific context, which is just sort of its memory based on the

person it's interacting with. So, this reminds the AI what the customer has already asked, previous troubleshooting steps that have been taken, maybe information about the customer. And without this user context, specific memory, the AI is going to ask the same questions over and over. It's going to forget what's already been conversated about, and it'll probably just annoy the end user with repetitive information and not very tailored information. So we're able to store these past interactions so that the AI can see it before it responds and before it takes action so that it's actually more seamless like a human conversation. It's more natural and efficient. And then we have the aspect of the real-time context. This is because there's some information that's just too dynamically changing or too large to fit within the actual system prompt of the agent. So instead of relying on this predefined knowledge, we can retrieve this context dynamically. So maybe it's as simple as we're asking the agent what the weather is. So, it hits that weather API in order to go access real-time current information about the weather. It pulls it back and then it responds to us. Or it could be, you know, we're asking about product information within a database. So, it could go hit that knowledge base what that has all of our product information and it will search through it, look for what it needs, and then pull it back and then respond to us with it. So, that's the aspect of Rag and it's super super powerful. Okay. And this is a great segue from Rag. Now, we're talking about vector databases and when not to use a vector database. So, I think something similar happened here with vector databases as the same way it happened with AI agents is that it was just some cool magic buzzword and it sounded like almost too good to be true. So, everyone just started overusing them and overusing the term. And that's definitely something that I have to admit that I fell victim to because when I first started building this stuff, I was taking all types of data, no matter what it was, and I was just chucking it into a vector database and chatting with it. And because you know 70% of the time I was getting the right answers. I was like this is so cool because it's that you know as you can see based on this illustration it is sort of like that multi-dimensional data representation. It's a multi-dimensional space where the data points that you were storing are stored as these little vectors these little dots everywhere. And they're not just placed in there. They're placed in there intelligently because the actual context of the chunk that you're putting into the vector database it's placed based on its meaning. So, it's embedded based on all these numerical representations of data. As you can see, like right up here, this is what the sort of um embedding dimensions look like. And each point has meaning. And so, it's placed somewhere where other things are placed that are similar. They're placed near them. So, over here we have like, you know, animals, cat, dog, wolf, those are placed similarly. We have like fruits over here, but also like tech stuff because Google's here and Apple, which isn't the fruit, but it's also the tech brand. So, you know, it also kind of shifts as as you embed more vectors in there. So, it's just like multi-changing. It's very intelligent and the agent's able to scan everything and grab back all the chunks that are relevant really quickly. And like I said, it's just kind of one of those buzzwords that super cool. However, even though it sounds cool, after building these systems for a while, I learned that vector databases are not always necessary for most business automation needs. If your data is structured and it needs exact retrieval, which a lot of times company data is very structured and you do need exact retrieval, a relational database is going to be much better for that use case. And you know, just because it's a buzzword, that's exactly what it is, a buzz word. So that doesn't always mean it's the best tool for the job. So because in college I studied business analytics, I've had a little bit of a background with like databases, relational databases, and analytics. Um, but if you don't really understand the difference between structured and unstructured data and what a relational database is, we'll go over it real quick. Structured data is basically

anything that can fit into rows and columns because it has an organized sort of predictable schema. So in this example, we're looking at customer data and we have two different tables and this is relational data because over here we have a customer ID column. So customer ID 101 is Alice and we have Alice's email right here. Customer ID 102 is Bob. We have Bob's email and then we have a different table that is able to relate back to this customer lookup table because we match on the fields customer ID. Anyways, this is an order table it looks like. So we have order one by customer ID 101 and the product was a laptop. And we may think okay well we're looking at order one. Who was that? We can relate it back to this table based on the customer ID and then we can look up who that user was. So there's a lot of use cases out there. When I said, you know, a lot of business data is going to be structured like user profiles, sales records, you know, invoice details, all this kind of stuff. You know, even if it's not a relational aspect of linking two tables together, if it's structured data, which is going to be, you know, a lot of chart stuff, number stuff, um, Excel sheets, Google Sheets, all that kind of stuff, right? And if it's structured data, it's going to be a lot more efficient to query it using SQL rather than trying to vectorize it and put it into a vector database for semantic search. So we said as a non-programmer, if you're, you know, I'm sure you've been hearing SQL quering and maybe you don't understand exactly what it is. This is what it is, right? So we're almost kind of using natural language to extract information, but we could have, you know, half a million records in a table. And so it's just a quicker way to actually filter through that stuff to get what we need. So in this case, let's say the SQL query we're doing is based on the user question of can you check the status of my order for a wireless mouse placed on January 10th. On the left, we have an orders table. And this is the information we need. These are the fields, but there may be 500,000 records. So we have to filter through it really quickly. And how we would do this is we would say, okay, first we're going to do a select statement, which just means, okay, we just want to see order ID, order date, order status, because those are the only columns we care about. We want to grab it from the orders table. So, this table and then now we set up our filters. So, we're just looking for only rows where product name equals wireless mouse because that's the product she bought. And then um and the order date is January 10, 2024. So, we're just saying whenever these two conditions are met, that's when we want to grab those records and actually look at them. So, that's an example of like what a SQL query is doing. And then on the other side of things, we have unstructured data, which is usually the best use case for unstructured data going into a vector database, based on my experience, is just vectorizing a ton of text. So big walls of text, chunking them up, throwing them into a vector database, and they're placed, you know, based on the meaning of those chunks, and then can be grabbed back semantically, intelligently by the agent. But anyways, this is a quick visualization that I made right over here. Let's say we have um a ton tons of PDFs, and they're just basically policy information. We take that text, we chunk it up. So, we're just splitting it based on the characters within the actual content. And then each chunk becomes a ve a vector, which is just one of these dots in this threedimensional space. And they're placed in different areas, like I said, based on the actual meaning of these chunks. So, super cool stuff, right? So then when the agent wants to, you know, look in the vector database to pull some stuff back, it basically makes a query and vectorizes that query because it will be placed near other things that are related and then it will grab like everything that's near it and that's how it pulls back if we're doing like a nearest neighbor search. But don't want to get too technical here. I wanted to show an example of like why that's beneficial. So on the left we have product information about blankets and on the right we also have product information about blankets and we just decided on the right it's a vector database on the left it's a relational database and so let's

say we hooked this up to you know a customer chatbot on a website and the customer asked I'm looking for blankets that are fuzzy now if it was a relational database the agent would be looking through and querying for you know where the description contains the word fuzzy or Maybe material is contains the word fuzzy. And because there's no instances of the word fuzzy right here, we may get nothing back. But on the other side of things, when we have the vector database, because each of these vectors are placed based on the meaning of their description and their material, the agent will be able to figure out, okay, if I go over here and I pull back these vectors, these are probably fuzzy because I understand that it's cozy fleece or it's um, you know, handwoven cotton. So that's like why there's some extra benefits there because maybe it's not a word for word match, but the agent can still intelligently pull back stuff that's similar based on the actual context of the chunks and the meaning. Okay, moving on to number five. Why prompting is critical for AI agents. Um, we already talked about it a little bit, I guess, in the context is everything section because prompting is giving it more context, but this should be a whole lesson in itself because it is truly an art. And you have to find that fine line between you don't want to over prompt it and you want to minimize your token usage, but you also want to give it enough information. But, um, when people think of prompting, they think of chatgbt, as you can see right here, where you have the luxury of talking to chat, it's going to send you something back. You can tell it, hey, make that shorter, or hey, make it more professional. It'll send it back and you can keep going back and forth and making adjustments until you're happy with it and then you can finally accept the output. But when we're dealing with AI agents and we're trying to make these systems autonomous, we only have one shot at it. So, we're going to put in a system prompts right here that the agent will be able to look at every time there's like an input and we have to trust that the output and the actions taken before the output are going to be high quality. And so, like I said, this is a super interesting topic and if you want to see a video where I did more of a deep dive on it, you can check it out. I'll tag it right here. Um, where I talked about like this lesson, but the biggest thing I learned building these agents over the past six months was reactive prompting is way better than proactive prompting. Admittedly, when I started prompting, I did it all wrong. I was lazy and I would just like grab a custom GPT that I saw someone use on YouTube for, you know, a prompt generator that generates the most optimized prompts for your AI agents. I think that that's honestly a bunch of garbage. I even have created my own AI agent system prompt architect and I posted it in my community and people are using it, but I wouldn't recommend to use it to be honest. Um, nowadays I think that the best practice is to write all of your prompts from scratch by hand from the beginning and start with nothing. So, that's what I meant by saying reactive prompting. Because if you're grabbing a whole, you know, let's say you have 200 lines of prompts and you throw it in here into your system prompt and then you just start testing your agent, you don't know what's going on and why the agent's behaving as it is. You could have an issue pop up and you add a different line in the system prompt and the issue that you originally were having is fixed, but now a new issues popped up and you're just going to be banging your head against the wall trying to debug this thing by taking out lines, testing, adding lines, testing. it's just going to be such a painful process when in reality what you should do is reactive prompt. So start with nothing in the system prompt. Give your agent a tool and then test it. Throw in a couple queries and see if you're liking what's coming back. You're going to observe that behavior and then you have the ability to correct the system prompt reactively. So based on what you saw, you can add in a line and say, "Hey, don't do that." Or, you know, this worked. Let's add another tool and add another prompt now or another line in the prompt. Because what we know right now is that it's working based on

what we have. That way if we do add a line and then we test and then we observe the behavior and we see that it broke, we know exactly what broke this automation and we can pinpoint it rather than if we threw in a whole pre-generated system prompt. So that's the main reason why I don't do that anymore. And then it's just that loop of test, reprompt, test again, reprompt. Um, and what's super cool about this is because you can basically hard prompt your agent with things in the system prompt because you're able to show it examples of, you know, hey, I just asked you this and you took these steps. That was wrong. Don't do that again. This is what you should have done. And basically, if you give it that example within the system prompt, you're training this thing to not behave like that. And you're only improving the consistency of your agent's performance. So the the key elements of a strong AI agent prompt and this isn't like every single time. These are the five things you should have because every agent's different. For example, if you're creating a context creation agent, you wouldn't need a tool section really if it's not if it doesn't have any tools. You' just be prompting it about its output and about its role. But anyways, the first one that we're talking about is role. This is sort of just like telling the AI who it is. So this could be as simple as like you're a legal assistant specializing in corporate law. Your job is to summarize contracts in simple terms and flag risky clauses. Something like that. It gives the AI clear purpose and it helps the model understand the tone and the scope of its job. Without this, the AI is not going to know how to frame responses and you're going to get either very random outputs or you're going to get very vague outputs that are very clearly generated by AI. Then of course you have the context which is going to help the agent understand, you know, what is actually coming in every time because essentially you're going to have different inputs every time even though the system prompt is the same. So saying like this is what you're going to receive, this is what you're going to do with it, um this is your end goal. So that helps tailor the whole process and make it more seamless as well. That's one common mistake I actually see with people's prompting when they start is they forget to define what are you going to be getting every time? Because the agency, they're going to be getting a ton of different emails or maybe a ton of different articles, but it needs to know, okay, this information that you're throwing at me, what is it? Why am I getting it? Then of course the tool instructions. So when you're building a tools agent, this is the most important thing in my mind. Yes, it's good to add rules and show like when you use each thing, but having an actual section for your tools is going to increase the consistency a lot. At least that's what I found because this tells the AI exactly what tools are available, when to use them, how they work. Um, and this is really going to ensure correct tool usage rather than the AI trying to go off of like these sort of guidelines because it's a nondeterministic workflow and um trying to trying to guess of which one will do what and um yeah, have a tool section and define your tools. Then you've got your rules and constraints and this is going to help prevent hallucination. It's going to help the agent stick to sort of like a standard operating procedure. Now, you just have to be careful here because you don't want to say something like do all of these in this order every time because then it's like why are you even using an agent? You should just be using a workflow, right? But anyways, just setting some foundational like if X do Y, if Z do A, like that sort of thing. And then finally, examples, which I think are super super important, but I would never just put these in here blind. I would only use examples to directly counter and directly uh correct something that's happened. So what I alluded to earlier with the hard prompting. So let's say you give the agent an input. It calls tool one and then it gives you an output that's just incorrect, completely incorrect. You'd want to give it in the example, you could show, okay, here's the input I just gave you. Now here's what you should have done. Call tool two and then call tool three and then give me the

output. And then it knows like, okay, that's what I did. This is what I should have done. So if I ever get an input similar to this, I can just call these two tools because I know that's an example of like how it should behave. So hard prompting is really really going to come in handy and not just in the examples but also just with the rest of your system prompt. All right, moving on to number six. We have scaling agents can be a nightmare. And this is all part of like one of the hard truths I talked about earlier where a lot of the stuff you see online is a great proof of concept, a great MVP, but if you were to try to push this into production in your own business, you're going to notice it's not there yet. Because when you first build out these AI agents, everything can seem to work fine. It's a demo. It's cool. It really opens your eyes to, you know, the capabilities, but it hasn't usually gone under that rigorous testing and evaluation and setting up these guard rails um and all of that, you know, continuous monitoring that you need to do to evaluate its performance before you can push it out to all, you know, 100 users that you want to eventually push it out to. You know, on a single user level, if you have a few hallucinations every once in a while, it's not a huge deal. But as you scale the use case, you're just going to be scaling hallucinations and scaling problems and scaling all these failures. So that's where it gets tricky. You can start to get retrieval issues as your database grows. It's going to be harder for your agent to cut through the noise and grab what it needs. So you're going to get more, you know, inaccuracies. You're going to have some different performance bottlenecks and the agents, you know, latency is going to increase. you're going to start to get inconsistent outputs and you're going to experience all those edge cases that you hadn't thought of when you were the only one testing because, you know, there's just an infinite amount of scenarios that an agent could be exposed to. So, a good little lesson learned here would be to scale vertically before you start to try to scale horizontally, which um we'll break that down. And I made this little visualization so we can see what that means. So, let's say we want to help this business with their customer support, sales, inventory, and HR management. Rather than building out little building blocks of tools within each of these processes, let's try to perfect one area first vertically and then we'll start to move across the organization and look at doing other things and scaling to more users. So, because we can focus on this one area, we can set up a knowledge base and set up like the data sources and build that automated pipeline. We can set up how this kind of stuff gets organized with our sub workflows. We can set up, you know, an actual agent that's going to have different tool calls. And within this process, what we have over here are evaluations, monitoring performance, and then setting up those guard rails because we're testing throughout this, you know, ecosystem vertically and and getting exposure to all these different edge cases before we try to move into other, you know, areas where we need to basically start this whole process again. We want to have done the endto-end system first, understand you know the problems that may arise, how to make it robust and how to evaluate and you know iterate over it before we start making more automations. And so like I alluded to earlier, if you try to start scaling horizontally too quick before you've done all these testing, you are going to notice that hallucinations are going to increase. Your retrieval quality is going to drop as more users users come in. The agent's handling more memory. it's handling more um more knowledge in its database to try to cut through your response times are going to slow and then you're just going to get more inconsistent results. And so you can do things like, you know, setting strict retrieval rules and guard rails. You could do stuff like segmenting your data into different vector databases based on the context or different name spaces or different, you know, relational databases. You could use stuff like um asynchronous processing or caching in order to improve that response time. And then you could also look at doing stuff like only you know um having an agent evaluate and making

sure that the confidence on these responses are above a certain threshold otherwise we'll you know request human help and not actually respond ourselves or the agent wouldn't respond itself. So the seventh one is that no code tools like nadn have their limits. They're super great. They're really empowering non-developers to get in here and spin up some really cool automations. And it's really like the barrier to entry is so low. you can learn how to build this stuff really quickly, which is why I think it's awesome and you know, it's the main tool that I use um personally and also within my agency. But when you start to get into what we just talked about with lesson number six, scaling and making these things really robust and production ready, you may notice some limits with no code tools like NE. Now, the reason I got into building stuff like this is because, you know, obviously non-programmer and it has a really nice drag and drop interface where you can build these workflows very visually without writing scripts. So, Nen is, you know, basically open source because you can self-host it. The code is accessible. Um, and it's built on top of Langchain, which is just like a basically a language that helps connect to different things and create these like agents. Um, and because of that, it's just wrapped up really pretty for us in a graphical user interface where we can interact with it in that drag and drop way without having to get in there and hands- on keyboard write code. And it has a ton of pre-built integrations as you can see right here. Connect anything to everything. Um, I think there's like a thousand integrations right here. And all of these different integrations are API calls. They're just wrapped up once again in a pretty way for us in a user interface. And like I talked about earlier, when I started, I didn't even know what an API was. So the barrier entry was super low. I was able to configure this stuff easily. And besides these built-in integrations, they have these really simple tool calls. So development is really fast with building workflows compared to traditional coding. um the modularity aspect because you can basically build out a workflow, you can save that as a tool and then you can call that tool from any other workflow you want. So once you build out a function once, you've basically got it there forever and it can be reused which is really cool. And then my favorite aspect about n and the visual interface is the visual debugging because rather than having 300 lines of code and you know you're getting errors in line 12 and line 45, you're going to see it's going to be red or it's going to be green and you know if it's green you're good and if you see red there's an error. So you know exactly usually you know exactly where the problem's coming from and you're able to get in there look at the execution data and get to the bottom of it pretty quick. But overall these no code platforms are great. They allow us to connect APIs. We can connect to pretty much anything because we have an HTTP request within NADN. Um they're going to be really really good for rulebased decision-m like super solid. Um if we're creating workflows that's just going to do some data manipulation and transferring data around you know your typical ETL based on the structured logic super robust. you can make some really really awesome basic AI powered workflows where you're integrating different LLMs. You've got all the different chat models basically that you can connect to um for you know different classification or content generation or outreach anything like that. Um your multi-agentic workflows because like I said earlier you have the aspect of creating different tools um as workflows as well as creating agents as workflows that you can call on from multiple agents. So you can really get into some cool multi-agentic inception thing going on with with agents calling agents calling agents. um and passing data between different workflows and then just the orchestration of AI services, coordinating multiple AI tools within a single process. So that's the kind of stuff that NN is going to be super super good at. And now the hidden limitations of these noode AI workflow/ aent builders. Let's get into it. Now, in my opinion, this stuff really just comes down to when we're trying to get into like enterprise solutions at scale

with a ton of users and a ton of authentication and a ton of data. Because if you're building out your own internal automations, you're going to be solid. Like there's not going to be limitations. If you're building out, you know, proof of concepts and MVPs, um, YouTube videos, creating content, like you can do it all, I would say. But when you need to start processing, you know, massive data sets that are going to scale to thousands or millions of users, your performance can slow down or even fail. And that's maybe where you'd want to rely on some custom code backend to actually spin up these more robust functionalities. In these agentic systems, tool calling is really, really critical. The agent needs to be able to decide which one to use and do it efficiently. And like I talked about, Nen is built on top of lang chain. It provides a structured way to call AI models and APIs, but it lacks some of that flexibility of writing that really custom code within there for complex decision-m. And then when it comes to authentication at scale, it can struggle with like secure large scale authentication and data access control. Obviously, you can hook up to external systems to help with some of that processing, but when it comes to maybe like handling OOTH tokens and all these encrypted credentials and session management, not that it's not doable with NN, um it just seems like getting in there with some custom code, it could be quicker and more robust. And also, that's coming from someone who doesn't actually do that myself. Um, just some stuff I've heard and you know, with what's going on within the agency. Now ultimately it seems like if you are delivering this stuff at scale for some big clients um the best approach is going to be a mix a hybrid of no code and custom code because you can use NN to spin up stuff really quick. You've got that modularity you can orchestrate automate you know connect to anything you need but then working in every once in a while some custom Python script for some of that complex you know large scale processing and data handling. And when you combine these two together, you're going to be able to spin some stuff up a lot quicker, and that's going to be pretty robust and powerful. Thanks so much for making it all the way to the end of this course. I know it's pretty massive, but I wanted to pack it with a ton of value, and hopefully you guys did find it valuable as well, and you feel a lot more comfortable right now building AI workflows and AI agents than when you started this course. If you enjoyed or if you learned something new, please give it a like and a subscribe. It definitely helps me out a ton. Um, like I said, super happy that you made it to the end of the course. And if you did and if you appreciate my teaching style or you want to even go more in depth than this course right here, then definitely check out my paid community. The link for that is down in the description. It's a community full of people who are learning how to build and are building a automations using naden and a lot of them are coming from no code backgrounds as well. So great place to get some questions answered, brainstorm people, collaborate on projects, stuff like that. And we also have five live calls per week. So, make sure you jump in there and meet other people in the space as well as make sure you're not getting stuck and you can get your questions answered on a call. Like I said, would be great to see you in the community. But anyways, thanks so much for making it to the very end of this huge course. Appreciate you guys and I will see you in the next video. Thanks so much everyone.