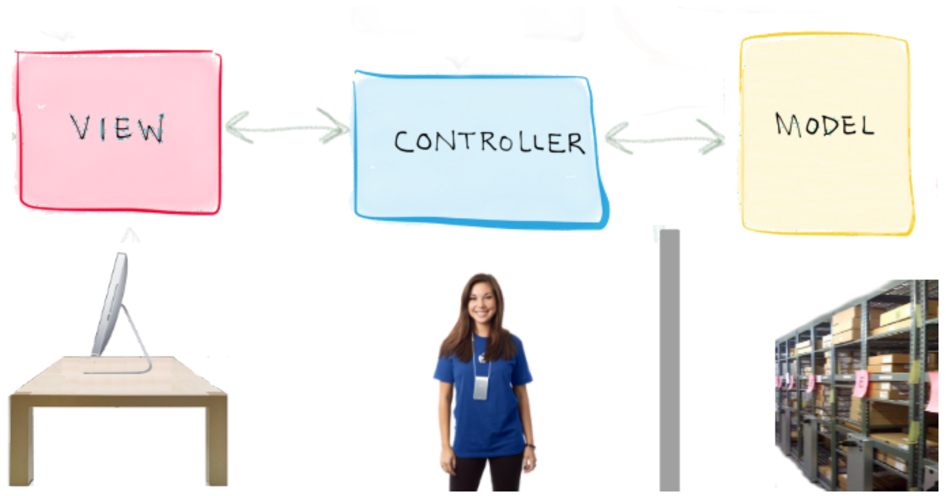
Script

Slides Link: http://mvc.tbytes.me/

Slides Script	Slides Content
Model-View-Controller, or MVC for short, is one of the most popular design patterns out there today. It has been used by generations of programmers and across many different programming languages. MVC is important to understand because it is the basic structure for many apps: whether it's a web app, mobile app, desktop program, or more. **Transition**: Before we go into all the details, let's talk a little about what MVC is.	**Model-View-Controller** (MVC) ● One of the most ubiquitous software design patterns. ● It breaks up code into three parts: model (the data), view (the look), and controller (the logic).
MVC is made up of three parts. **Models** are templates for how data will be stored. The model could be the schema for a database, or a Java class for an object. **Views** are what the users get to see and interact with. Views are created in many different ways: HTML and CSS are used in websites, XML is used to create a user interface in Android Studio, JFrame could be used to create GUIs in Java. **Controllers** hold the code that gets executed when you interact with the view, whether that code performs calculations or queries a database. **Transition**: To give you an idea of MVC in a real life setting, I will provide an analogy that compares MVC to the Apple Store.	**What is MVC?** MVC is broken into three parts: ● **Model**: Models represent the data for your application. They don't know anything about views or controllers ● **View**: Views are presented to users and are how users interact with the app. ● **Controller**: Controllers are the decision-makers. They are the glue between the model and the view.

A customer purchase at the Apple Store is a good, although slightly imperfect, analogy for MVC. When you first walk in, you're greeted with the sight of many Apple devices that look new and usable. This is the view; it's what you see and what you can interact with. When you want to buy an Apple device, you talk to an employee. The employee acts as a controller, interacting with the the objects in the back room. The back room acts a model, since the devices are organized in a specific way. The employee also handles the financial transaction before handing the device to you, just as a controller prepares the view for the user. **Transition**: Now that I have gone over what MVC is, Dania will go over when and when not to use MVC.	**Apple Store Analogy** 
MVC has many advantages as a software architectural design for an application: ● The separation of the three components of MVC (model, view, component) allows the re-use of code across applications. ● Multiple developers, as well as by separate teams, can work simultaneously on the model, views and component. In other words, this is called parallel development. ● Cohesion represents the degree of the systematic or logical connection or consistency in the logic expressed in the our code. MVC supports high cohesion, meaning that the class is well defined. ● Coupling describes the measure of the level of dependency between software modules. MVC supports low coupling, meaning that classes do not depend on each other. ● Because of the separation of responsibilities, future development or modification is easier. Modification does not affect the entire model.	**Why MVC?** There are multiple advantages to MVC: ● Separation of concerns ● Simultaneous development ● High cohesion ● Low coupling ● Ease of modification ● Code organization
● Once you get to ginormous scales MVC might not be the best choice. ● With applications that include complex states, MVC would not be ideal to use as a software architecture.	**Why not MVC** There are some drawbacks to MVC: ● May not be suitable at scale ● Limiting for apps with lots of state changes

There are really two main ways to use MVC:

- You can use a framework
- Or you can structure the code yourself

You can use JavaScript frameworks like Backbone, Angular, Ember, and the lesser known Kendo UI. Or you can use full stack frameworks like Microsoft's .NET MVC, Ruby on Rails, and Django.

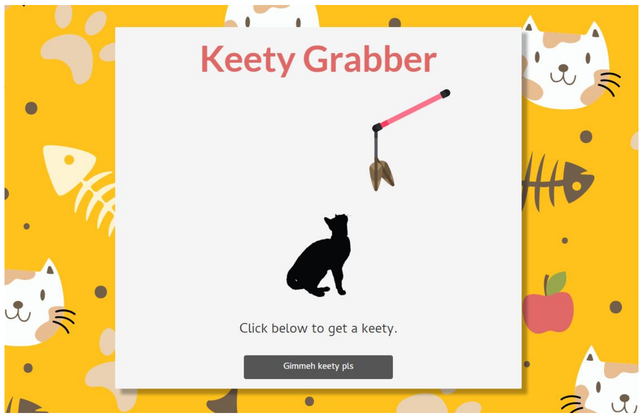
When you use a framework, you are sticking to the principle of "convention over configuration." Code is structured in a standardized way so that other developers can quickly know where to go to fix or add to your code if they have to work on it.

You don't have to use a framework though. You can structure your code in a MVC way on your own too. That's what we will show you in our demo.

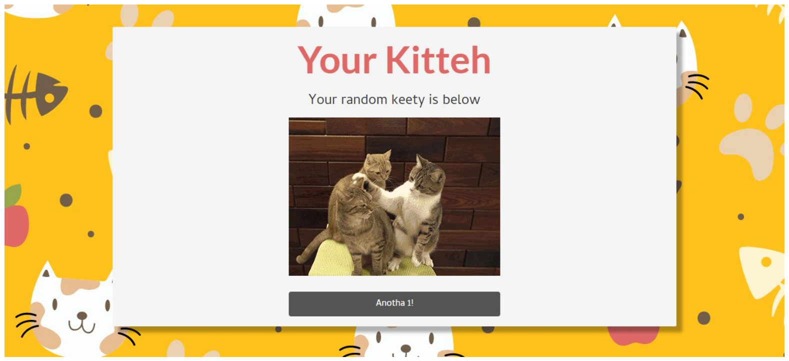
We wanted to show you how you can structure your code into an MVC way by showing you a quick demo.

Before we dive into into the code, let's take a look at the mini web app we made to showcase MVC. I like cats so we created a web app that basically gets us a bunch of cats. What we are looking at right now is the view for the application. When you visit a specific page (url of the app), the controller decides which view is rendered. The jade files are compiled into HTML/CSS and rendered. If you click the button, it takes you to the next page which is the heart of our application.

If you keep clicking the button, random pictures and gifs of cats appear. You will notice some of the pictures have random text at the bottom that are different ways of saying hello. That is coming from our database (the model) of our application.


Keety Grabber

Click below to get a keety.


Your Kitteh

Your random keety is below

You can find our mini MVC application's code on GitHub: https://github.com/terabytes/keety.

There are three parts to MVC: **model** (the data), **view** (the look), and **controller** (the logic).

Before we dive into the code, let's take a look at the structure of our application. `helloPhrases.js` is the model for our app. `Kitteh.jade` is the view. `Main.js` is the controller.

We can look at the `helloPhrases.js` file to take a look at our data model. Our app is a MEAN stack application so it uses Mongodb. We use Mongoose (an object data modeling language) to structure our data. Schemas in mongoose provide a way to organize Mongodb data, which is normally unstructured. This lets us manage our data better.

`Kitteh.jade` is where our view code sits. Jade is a templating engine for HTML/CSS that allows you use JavaScript in the view. If you look at the code, you can see familiar HTML tags like the `h1` tag or `p` tag even though it uses different syntax. You can see where the `img` is included. The value for `#{link}` comes from the controller file.

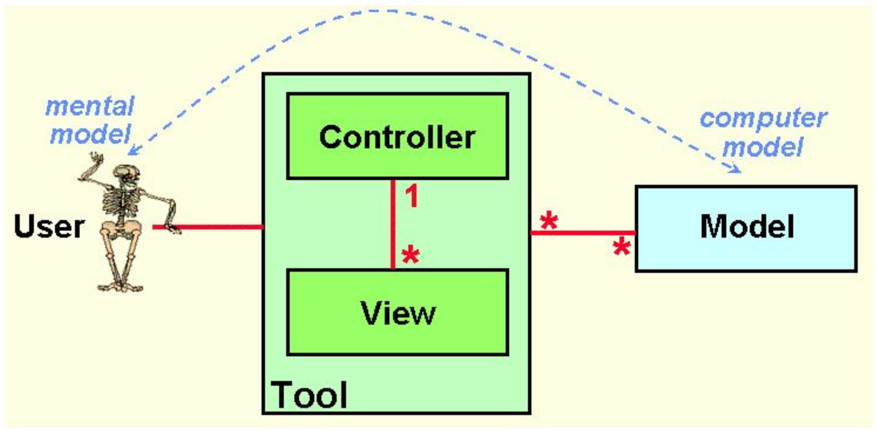
`Main.js` is the controller, or application logic, for our app. When you're at the kitteh view page (where the gifs show up), the first controller it hits is `loadKitty.` It makes a RESTful call to the database using the API we made. That calls the `renderKitteh` function which prepares stuff for the kitteh view (like getting the url ready, adding the phrase to the url), then actually renders (or shows) it.

```
├── app_api
|   ├── controllers
|   ├── models
|   |   └── helloPhrases.js     # Schema for db
|   └── routes
├── app_server
|   ├── controllers
|   |   └── main.js             # App logic for ctrlr
|   ├── routes
|   └── views
|       └── kitteh.jade         # Templates for view
├── node_modules
├── public
├── app.js
├── package.json
└── README.md
```

MVC was originally created by some Norwegian guy to help end users mess with an app in a more intuitive way. It organizes your code and separates the presentation from the application logic and from the data. If you want to level up as a developer and be prepared for working on industry code, be sure to learn MVC.

Conclusion

- The original concept of MVC was intended to help an end user manipulate an underlying computer system in a more intuitive way.

	 mental model computer model User Controller 1 * View Tool * * Model
Here are the sources that we used for this presentation. Thank you all for listening, and we hope that you've enjoyed this presentation.	**Sources** ● Simple Explanation of MVC ● Coding Horror ● Chrome Developer Docs ● Trygve Reenskaug ● Flux vs. MVC ● Free Code Camp