Ember.js Workshop

“A framework for creating **ambitious** web applications”

- Ember.js

Why Ember.js ?

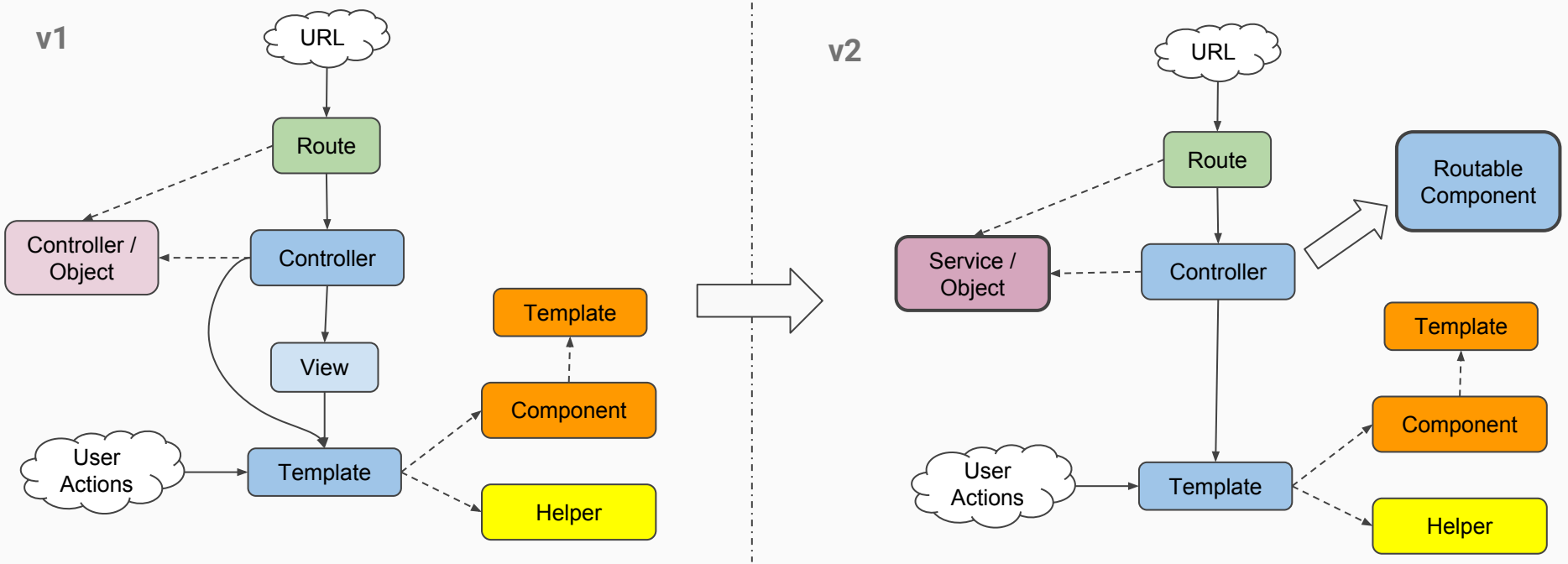
- Convention over configuration
- Best practices and design patterns
- Style
- Features
- Community
- Tools

Session 1 - Introduction to Ember.js

Ember Application Core Concepts

- **Model** - Data retrieved from backend
- **Router & Routes** - URL to system state (ui,data), management of model, sending data down, is singleton
- **Controller** - Application state, data context (v1), connection with template -> Routable Component (v2), **Services**, is singleton
- **Component** - UI modules, DOM access, sending actions up, reusable
- **Templates** - Describe the UI, are data-bound, contain other templates and components, may use **Helpers**

Core Concepts - Diagram



The Object Model

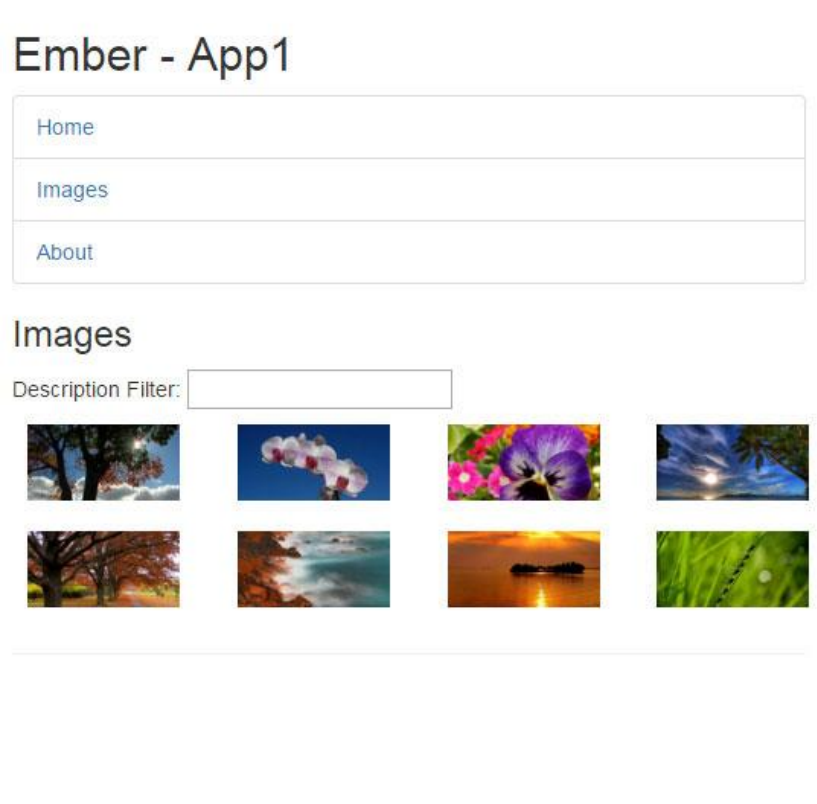
- Classes and inheritance
- Mixins
- Reopening / Overriding
- Computed properties
- Observers

em-app1

Creating a simple web app for viewing images, to introduce ember.js .

Takeaway:

Good grasp of ember app core concepts working together



em - app1

- Ember inspector
- No ember-cli yet, pure libs
- Understanding how the concepts work
- Understanding the associations of core classes

Conclusions

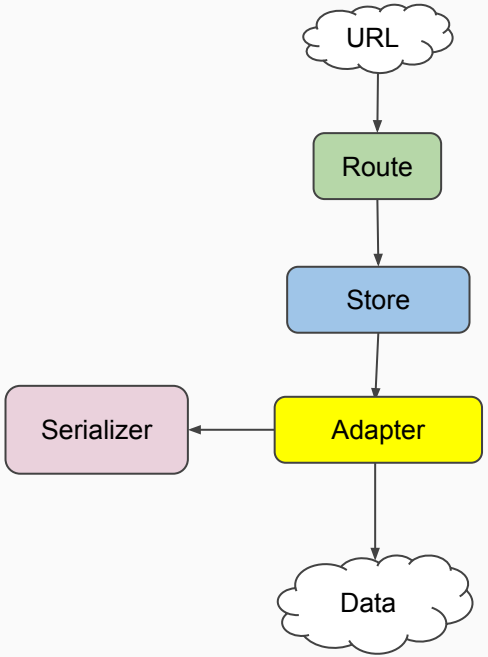
- Each url is mapped to a route
- All classes and templates are implicitly created if not specified
- Each route is associated with a template and controller (*which will become routable component*)
- Each controller of the route, provides the context to the template
- A template can be composed of components
- Each component has its own data context, so data needs to be passed down through parameters
- DOM manipulation takes place on proper lifecycle hooks

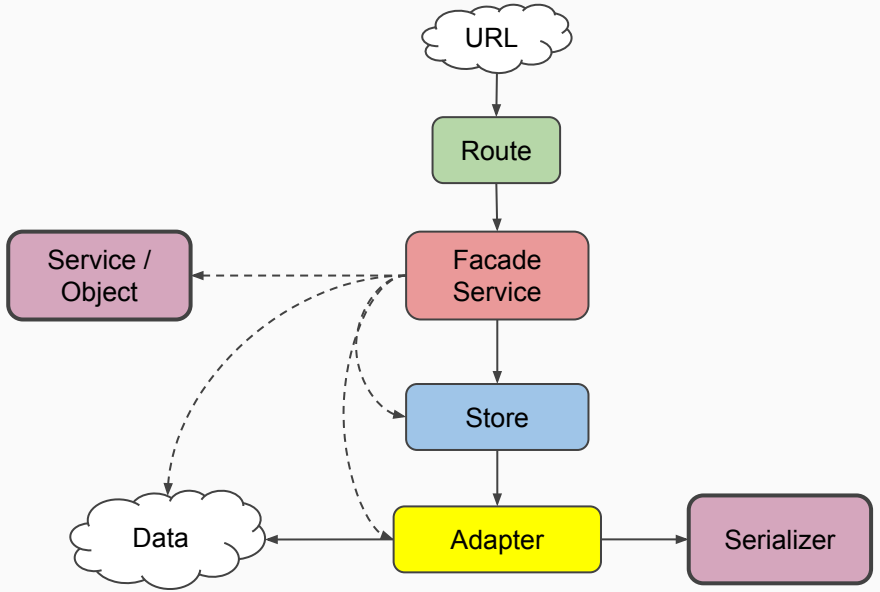
Best Practices

- **Computed properties** that only use properties of the same class, then they probably concern/**belong to the same class** and should be defined within it.
- **Properties** that are not primitives should be **initialized** for every class instantiation, otherwise they will be **shared** by all instances.
- To **recalculate** computed properties on demand call ***notifyPropertyChange*** function of Ember.Observable class.
- To **always recalculate** a computed property use ***volatile*** function in definition.
- Use **Ember.get** and **Ember.set** when not certain if it is an Ember object or not.
- **Mixins** for **composition** of functionality, **extend** for **inheritance** i.e. if it is of this kind.
- Use the **on** function if sequence of calls is not relevant, override the function if sequence is an issue and for better **performance**. But in the latter case always call **this._super.apply(this,arguments)** or in es6 **this._super(...arguments)** .
- If the **promise returned in model** function of a route takes **too long** and the system hangs, take advantage of the **loading state action or loading route**. Also possible to complete the model preparation in **setupController** function.

Session 2 - A form component with ember-cli and ember-data

Ember Data with Facade Service



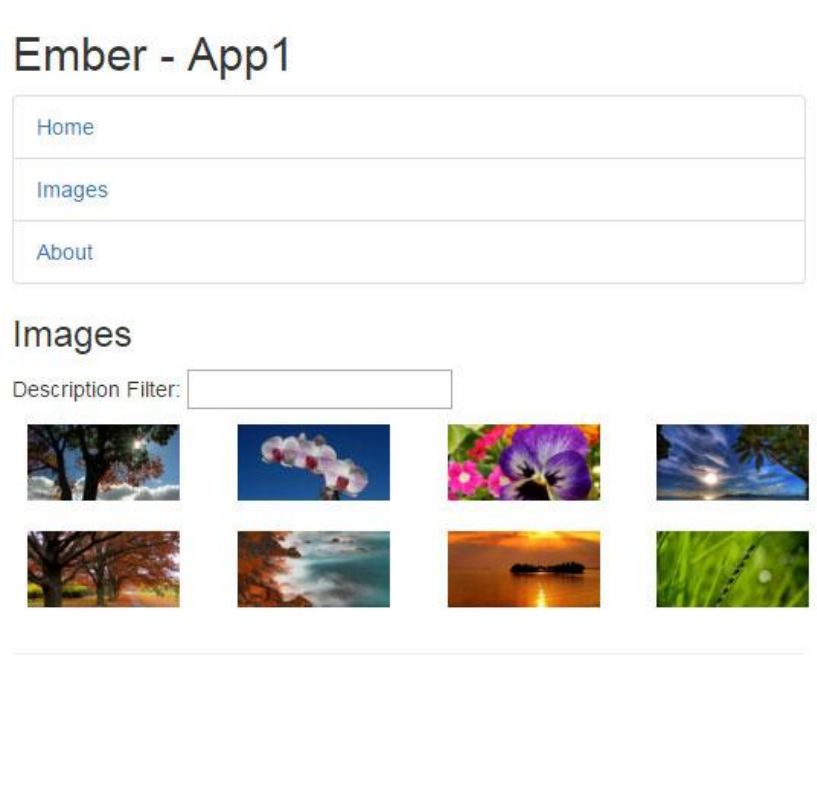


em-app1-cli

Introduction to ember-cli, while migrating em-app1 web application.

Takeaway:

Understanding of ember-cli commands and structure

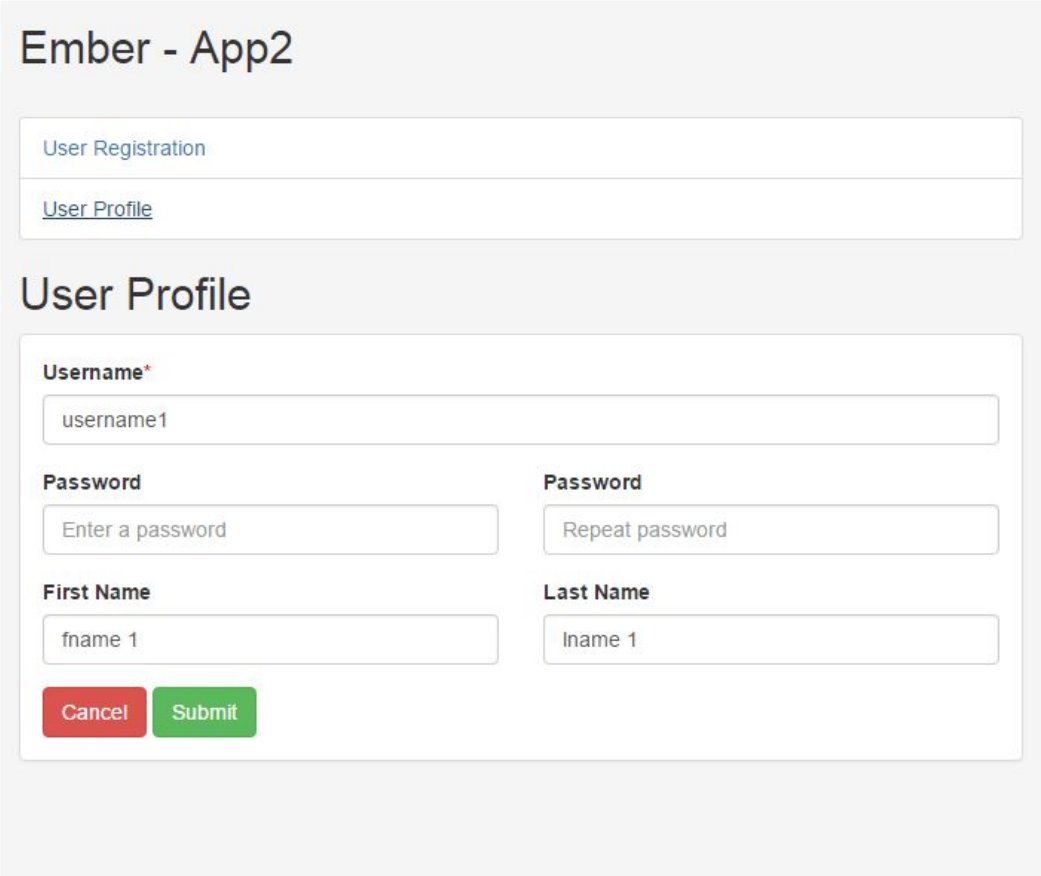


em-app2

Using ember-cli and ember-data to create a reusable form component through a simple web app.

Takeaway:

A reusable edit form component



Best Practices

- **Business logic** should be placed in reusable classes i.e. **Services** or custom Object implementations.
- Place the code relate to the **management of data model** i.e. preparation and CRUD actions, in corresponding **routes**.
- **Abstract away** the implementation details of data model handling with a **facade pattern**.
- Follow **Data Down Actions Up** approach.
- Prefer **block parameters** when using components.
- It is possible to have a **route** with params and without params, but define the one without the params last since the **last definition has precedence** over the others.

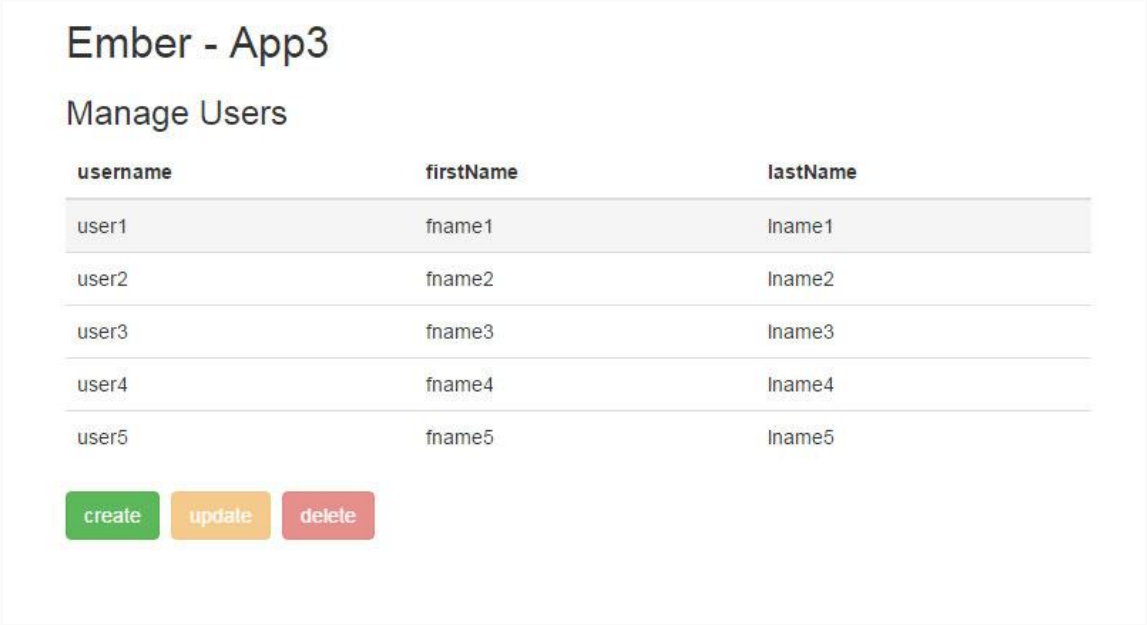
Session 3 - Creating a table component with records of data

em-app3

Using ember-cli and ember-data to create a reusable table to browse records of data. Use the form component of em-app2 to complete the CRUD functionality.

Takeaway:

A reusable browse table component



Conclusions

Approach overview of building a use case,

- Break storyboards/mockups into routes
- Create static markup for the templates
- Prepare model
- Place dynamic data into templates
- Break templates into components

Best Practices

- If a functionality can be implemented either with a **computed property** or an **observer** function, prefer the **computed property** to control.
- When using **observer** functions be extra **careful** of code execution and performance.
- If it is possible to specify an **action** in a nested or **parent route**, choose the parent route as it adds flexibility eg switching the nested route, common to other nested routes if needed.
- Make **reusable components**.
- Group **injections** and **overrides** of base classes **in initializers**.

Ember.js is the right tool, use it wisely…

Thanks!

email:
cmelas@thinkful.com

source code:
https://github.com/m3lc/thinkful-courses-ember

slides:
https://goo.gl/wa9lFP