JavaScript Application Architecture with Backbone.js

JavaScript Conference 2012
Düsseldorf

Mathias Schäfer
9elements

Hello
my name is

Mathias Schäfer (molily)

molily.de

Software developer at

9elements.com

JavaScript Use Cases by Complexity

1. Unobtrusive JavaScript

Form validation, tabs, overlays, slideshows, date pickers, menus, autocompletion

2. JavaScript-driven Interfaces

Configurators, form widgets, heavy Ajax, like Facebook

3. Single Page Applications

Desktop-class applications and games, like GMail

Single-purpose Libraries vs. Full-stack Solutions

DOM scripting

Model-view-binding

HTML templating

Functional & OOP helpers and shims

Modularization & dependancy management

Application structure

Routing & History

Building & Packaging

Unit testing

Lints

Documentation

Plenty of Options

Backbone, Spine, Knockout, Angular, JavaScriptMVC

Dojo, YUI

Sproutcore, Ember, Ext JS, Qooxdoo

GWT, Cappucino

Problems We Face

There's no golden path

Few conventions and standards

Countless interpretations of traditional patterns like MVC

Reinventing the wheel

If you choose one technology stack, you're trapped

Introducing Backbone.js

Backbone.js

A simple small library (1.290 LOC) to separate business and user interface logic

Growing popularity

Quite stable

Actively developed

Free and open source

Backbone Dependencies

Underscore

as OOP and functional utility belt

jQuery, Zepto, in theory Ender…

for DOM Scripting and Ajax

_.template, Mustache, Handlebars…

for HTML templating

Backbone Classes

Backbone.Events

Backbone.Model

Backbone.Collection

Backbone.View

Backbone.History

Backbone.Router

Backbone.Events

A mixin which allows to dispatch events and register callbacks

Backbone's key feature, included by *Model*, *Collection*, *View* and *History*

Methods: *on*, *off*, *trigger*

Backbone.Model

Data storage and business logic

Key feature: the *attributes* hash

Changes on the *attributes* will fire *change* events

Backbone.Model

Models may be retrieved from and saved to a data storage

Standard sync uses RESTful HTTP

Validation constraints

Backbone.Model

```
var Car = Backbone.Model.extend();

var car = new Car({
  name: 'DeLorean DMC-12'
});

alert( car.get('name') );
```

Backbone.Collection

A list of models

Fires *add*, *remove* and *reset* events

Implements Underscore list helpers
(map, reduce, sort, filter…)

Backbone.View

A view owns a DOM element

Knows about its model or collection

Handles DOM events (user input)

Observes model events (binding)

Invokes model methods

The Render Pattern

Views typically render model data into HTML using a template engine

model attributes	*{ foo: 'Hello World.' }*
template	* {{foo}} *
output	* Hello World *

this.$el.html(this.template(this.model.toJSON()));

```
var CarView = Backbone.View.extend({
  initialize: function () {
    this.model.on('change', this.render, this);
  },
  render: function () {
    this.$el.html('Name: ' + this.model.get('name'));
  }
});
var carView = new CarView({
  model: car,
  el: $('#car')
});
carView.render();
```

Model View Binding

You need to setup binding manually.

A view might listen to model changes and then render itself from scratch or update the specific DOM.

A view might listen to user input and call model methods or dispatch events at the model.

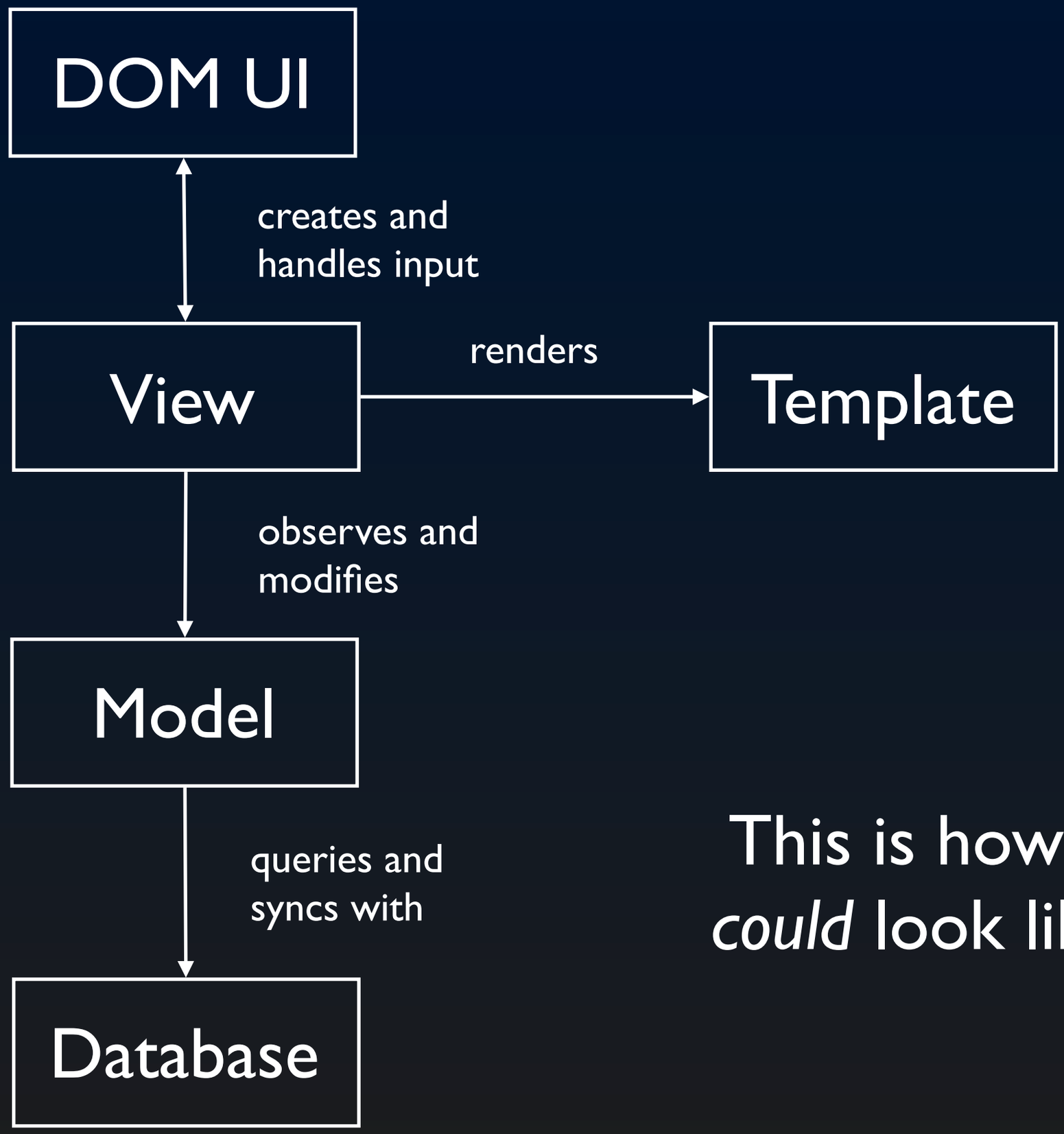
Backbone.Router and Backbone.History

A *Router* maps URIs to its methods

History is the actual workhorse, observes URI changes and fires callbacks

Hash URIs (*location.hash*, *hashchange*) or HTML5 History (*pushState*, *popstate*)

Routers usually create models and views



This is how it
could look like.

BACKBONE.JS

Backbone.js gives structure to web applications by providing **models** with key-value binding and custom events, **collections** with a rich API of enumerable functions, **views** with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.

Text

That's it (add routing).

And *that's all*.

Application Architecture

on top of Backbone.js

Lowering Expectations

Backbone is minimalistic by design and not a full-fledged solution.

Backbone provides no top-level patterns to structure an application.

Not MVC, MVP or MVVM.

“There’s More Than One Way To Do It” vs. “Convention Over Configuration”

True Story

Y **Hacker News** new | comments | ask | jobs | submit login

▲ koko775 5 days ago | link | parent

I'm trying to learn Backbone, I really am. But there's so many tutorials suggesting everything that I don't have a clue how I should structure a project and what to think about as a codebase grows.

Granted, I'm fairly new to Javascript, but - no offense meant - I can't get excited about node.js. Backbone is the first thing about Javascript that really makes me want to write it. And I know enough to know what I lack is primarily practical experience, but every step I take with Backbone feels like a misstep, because it seems like there's no one right answer by design, so I'm paralyzed.

What to do?

http://news.ycombinator.com/item?id=3532542

What is an Application?

An application has numerous screens with specific transitions between them.

A screen typically consists of multiple views.

Modules depend on each other and communicate with each other.

A lot of async I/O happens.

The “Todo List Example” is not such an app.

Backbone as a Basis

If you're planning an application,
Backbone is just the beginning.

Build yourself an abstraction layer,
but don't reinvent the wheel.

Standing on the Shoulders of Github

Thorax
https://github.com/walmartlabs/thorax

Marionette
https://github.com/derickbailey/backbone.marionette

Backbone Cellar
https://github.com/ccoenraets/backbone-cellar

Layoutmanager
https://github.com/tbranyen/backbone.layoutmanager

Aura
https://github.com/addyosmani/backbone-aura

Chaplin

https://github.com/moviepilot/chaplin

Meet Chaplin

Derived from Moviepilot.com,
a real-world single-page application

An example architecture,
not a ready-to-use library

How to DRY, enforce conventions, and write readable code?

Decide how create objects, fetch data, render views, subscribe to events etc.

Extend the core classes of Backbone (*Model*, *Collection*, *View*)

CoffeeScript class hierarchies with *super* calls as well as object composition

CollectionView for rendering collections

How to build modules with loose coupling for a scalable architecture?

Module encapsulation and dependency management via RequireJS (AMD)

Share information using a Mediator object

Cross-module communication using the Publish/Subscribe pattern

How to bundle the code for a specific screen (models, collections, views)?

Backbone.Router maps URLs to its own methods

Better separate routing and the code which creates the models and views

Introduce *Controllers* and reinvent the *Router*

A controller represents a screen of the application

How to manage top-level state?

ApplicationController for core models and views

ApplicationView as dispatcher and controller manager

Creates and removes controllers, tracks the current state

Router – *ApplicationView* – Controllers

How to implement user authentication?

SessionController for user management

Creates the login dialogs

Pub/Sub-driven login process:
!login, *login*, *!logout*, *logout* events

Client-side login with OAuth providers like Facebook, Google or Twitter

How to boost performance and prevent memory leaks?

Strict memory management and standardized object disposal

All controllers, models, collections, views implement a *dispose* destructor

Create core classes and an abstration layer which allow for automatic disposal

How to handle asynchronous dependencies?

Backbone's own event handling

Publish/Subscribe

Mixin jQuery Deferreds into models

Function wrappers and accumulators
(*deferMethods*, *deferMethodsUntilLogin*, *wrapAccumulators*…)

Questions? Ideas? Opinions?

I'm molily on Twitter & Github

mathias.schaefer@9elements.com

contact@9elements.com