Single Page Application

Nag

single page app

Why do we want to write single page apps?

The main reason is that they allow us to offer a native-app-like experience to the user

single page app

Redraw any part of the UI without requiring a server roundtrip to retrieve HTML

single page app

> “ separating the data from the presentation of data by having a model layer that handles data and a view layer that reads from the models”
>
> — Mixu's single page app book

single page app

Hard part is maintaining the code

To write maintainable code, we need to keep things simple

It is easy to add complexity (coupling with dependencies) in order to solve a worthless problem; and it is easy to solve a problem in a way that doesn't reduce complexity

structuring - SPA

Modern web app is structured from three different perspectives

- Architecture

- Asset packaging

- Run-time state

Architecture

What (conceptual) parts does our app consist of?

How do the different parts communicate with each other?

How do they depend on each other?

Asset packaging

How is our app structured into files and files into logical modules?

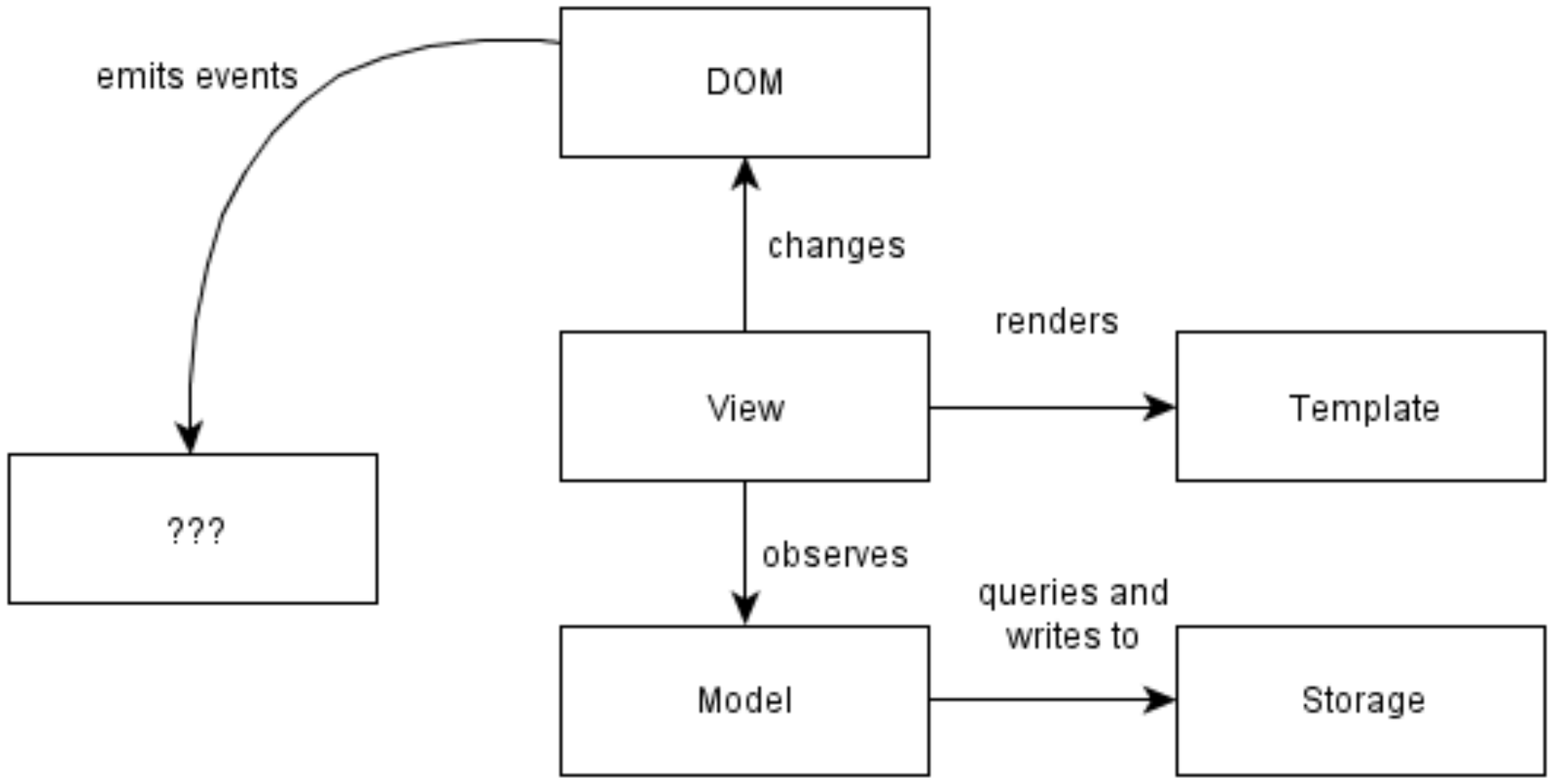
How are these modules built and loaded into the browser?

How can the modules be loaded for unit testing?

Run-time state

When loaded into the browser, what parts of the app are in memory?

How do we perform transitions between states and gain visibility into the current state for troubleshooting?



A modern web application architecture

A modern web application architecture

More specifically:

— *Write-only DOM* : No state / data is read from the DOM.

— *Models as the single source of truth*

— *Views observe model changes*

— *Minimizing DOM dependent-code*

Controllers ?

Single page apps have more complex state transitions than a server-side app:

- there are DOM events that cause small state changes in views
- there are model events when model values are changed
- there are application state changes that cause views to be swapped
- there are global state changes, like going offline in a real time app
- there are delayed results from AJAX that get returned at some point from backend operations

Asset packaging

Messy and random (no modules)

Every piece of code is made global by default

Names are global

Fully traversable namespaces

Load order matters, because anything can overwrite or change anything else

Implicit dependencies on anything global

Files and modules do not have any meaningful connection

Only runnable in a browser because dependencies are not isolated

Asset packaging..

Packages and modules (modular)

Packages expose a single public interface

Names are local to the package

Implementation details are inaccessible outside the package

Load order does not matter thanks to packaging

Explicitly declared dependencies

Each file exposes one module

Runnable from the command line with a headless browser

Run-time state

Run time state refers to what the app looks like when it is running in your browser

 - things like what variables contain what information and what steps are involved in moving from one activity (e.g. page) to another.

Run-time state

There are three interesting relationships here

- URL < - > state

- Definition < - > initialization

- HTML elements < - > view objects and HTML events < - > view changes

SPA

- three perspectives -
  - one from the point of view of the architect
  - one from the view of the filesystem
  - and finally one from the perspective of the browser.