Architektur Client:

Für die Strukturierung der Client-Programmierung, von Personal Canban, mit JavaScript haben wir uns für das MVC-Architekturmust entschieden.
Für die Implementierung dieses Musters wollen wir das *Ember* (http://emberjs.com) Framework nutzen.

Zum Grundverständnis einer Ember Web Applikation ist es wichtig ihren URL Aufbau zu verstehen.
So eine URL könnte folgendermaßen aussehen:

http://www.emberapp.de/#post/123
|--1--|---------- 2 ----------|----- 3 -----|

1. Schema
2. Host
3. Fragment

Das Fragment wird in den meisten Fällen zum Verweis auf Anker in HTML-Datein verwendet, was zu keinem Server Request führt, solange der *REST* der URL sich nicht verändert. Dies nutzt *Ember* damit bei einem Klick auf einen Link die Seite nicht neu geladen wird, sondern einfach das Fragment ausgelesen werden kann um darauf hin den neuen Seiteninhalt per JavaScript nachzuladen.

Das **MVC** Muster wird in *Ember wie folgt implementiert:*

Model (*Modell*):

EmberDataModels sind JavaScript Objekte mit Eigenschaften und Funktionen – welche in diesem Zusammenhang für berechnete Eigenschaften verwendet werden.
In unserem Fall werden diese die Tabellen unserer Datenbank widerspiegeln.
Außerdem enthalten sie Funktionen welche ihr Speichern oder Löschen ermöglicht.
Dafür greifen sie auf den *EmberDataStorage* zu, welcher für das Puffern und Persistieren der Modell Daten zuständig ist. Er aktualisiert dann den Puffer und greift auf den für die Applikation bestimmten *DataStorageAdapter* – in unserem Fall der *RESTAdapter*, bzw. der LocalStorageAdapter – zu, um die Daten zu persistieren. Der *EmberDataStorage* ermöglicht das Erstellen, Lesen, Aktualisieren und Löschen von Modell Einträgen.

View (*Präsentation*):

Die Implementierung der View findet bei *Ember* mit Hilfe der Template-Engine *Handlebars* (http://handlebarsjs.com/) statt.
Handelbars ist zum einen zuständig für die Templates, d. H. die Grundgerüste, der einzelnen Seiten der Applikation und zum anderen für die der einzelnen Komponenten (z . B. das Hauptmenü), welche auf mehreren verschiedenen Seiten wiederverwendet werden können.
Durch *Handlebars* ist es möglich auf von *Ember* übergebene Modell Daten oder Controller Funktionen zu zugreifen und somit die Ausgabe dynamisch zu gestalten.
Außerdem ist es auch möglich z. B. die Ausgabe von Inhalten an Bedingungen zu knüpfen oder über eine Menge von Modell Einträgen zu iterieren.

Für das Design der Seiten werden wir *LESS* (http://lesscss.org/) verwenden. Dabei handelt es sich um eine Erweiterung des *CSS* Standards. Es ist sehr gut dafür geeignet die sonst häufigeren Redundanzen in *CSS* Dateien zu vermindern (*don't repeat yourself*). Durch *LESS* ist es z. B. möglich Variablen zu definieren oder auch Regeln zu verschachteln, jedoch muss es vor der Verwendung in reines *CSS* Kompiliert werden.

Controller (*Steuerung)*:

Ein wichtiger Teil der Steuerung ist der *EmberRouter*. Durch ihn werden der Name, der Pfad und die möglichen URL-Parameter der einzelnen *EmberRoutes* bestimmt.
Diese sind dann dafür zuständig der Anfrage den richtigen *EmberController*, das *EmberDataModel* und das *Handlebars-Template* zu zuordnen.

EmberController sind dafür zuständig den Templates zusätzlichen Funktionen zur Verfügung zu stellen, wie z. B. die Filterung aller zur Verfügung stehenden Model Einträge anhand einer bestimmten Eigenschaft.
Außerdem können in dem Controller auch Formulare ausgewertet und Informationen per AJAX Request mit dem Server ausgetauscht werden.
EmberData stellt außerdem einen *RESTAdapter* bereit mit dem es möglich ist, über eine entsprechende *RESTAPI*, mit dem Server zu kommunizieren und so Model Daten abzufragen oder dem Server Änderungen mitzuteilen.

Durch die Trennung in die drei Bereiche im MVC-Architekturmuster herrscht eine klare Trennung der Verantwortlichkeiten (s*eparation of concerns*).

In der Standard Implementierung müssen alle Handlebars-Templates in eine Datei geschrieben werden. Damit wir dieses umgehen können haben wir uns dafür entschlossen *Grunt* (http://gruntjs.com/) zu verwenden. *Grunt* ist ein *Nodejs* (http://nodejs.org/) plugin, welches uns erlaubt die Templates in einzelne Dateien zu splitten, da es die einzelnen Templates mithilfe einer *Handelbars* Erweiterung in eine Datei kompiliert.
Weitere Vorteile welche wir durch *Grunt* erhalten sind folgende:

- “watch-Task“: Automatische Ausführung aller Aufgaben sobald eine “beobachtete“ Datei verändert wurde
- automatische *LESS* Kompilierung
- Minifizierung von *JavaScript* und *CSS* Dateien für Release Versionen
- jshint hinweise in der Windows Konsole