Contents

[Presumptions 2](#_Toc357370665)

[Technology Design 2](#_Toc357370666)

[Models & Collections 2](#_Toc357370667)

[OrderCollection 2](#_Toc357370668)

[OrderModel 2](#_Toc357370669)

[FilterModel 2](#_Toc357370670)

[BatchModel 2](#_Toc357370671)

[Views 3](#_Toc357370672)

[FilterView 3](#_Toc357370673)

[OrdersListView 3](#_Toc357370674)

[Controller 3](#_Toc357370675)

[app.js 3](#_Toc357370676)

[LESS & CSS 3](#_Toc357370677)

[Template 3](#_Toc357370678)

[Interaction Design 3](#_Toc357370679)

[Initiation process 3](#_Toc357370680)

[Event Dispatch design 4](#_Toc357370681)

[Filtering process 4](#_Toc357370682)

[Event Dispatch design 4](#_Toc357370683)

[Creating a batch process 4](#_Toc357370684)

[Event Dispatch design 4](#_Toc357370685)

# Presumptions

1. Data exchange format is JSON
2. Use REST API for AJAX call.
3. Filter items are flexible, and is fetched from backend.

# Technology Overall Design

1. MVC front end.
2. REST API, but to focus on the front end coding, I will use mock data here.
3. LESS for CSS
4. Consider unit test & template if have time
5. Frameworks includes Backbone, Jquery, Bootstrap

# MVC Design

## Models & Collections

Model & collections will:

1. Maintain the data status on the front end.
2. Maintain the REST API url.
3. Dispatch events on model status change.
4. Handle interaction with backend.
5. Validate input value.

All models & collections code are stored in js/collections/ and js/models/.

### OrderCollection

1. This class is to represent the order list on the front end.

### OrderModel

1. This class is to represent an order object on the front end.

### FilterModel

1. This class is to represent the filter object on the front end.

This class is to represent the order list on the front end. All ajax call will h

### BatchModel

This class is to represent the order list on the front end. All ajax call will h

## Views

View will:

1. Show data stored in Model or Collection
2. Update UI on Model or Collection change.
3. Handle user interaction like ‘click’.

All view codes are stored in js/views/

### FilterView

This class is to handle the FilterModel on the UI.

### OrdersListView

This class is to handel the OrderCollection on the UI

## Controller

### AppStart

Component initiation and interaction will all be handled here. In this way we can decouple the dependencies between these components, and there relationship is easier to understand.

The code will be in the app.js file

## LESS & CSS

Will use LESS to manage CSS code.

All codes are stored under css/.

## Template

Will consider using template technology to move HTML code from Javascript file.

# Interaction Design

Should provide sequence chart here, but no time.

## Fetch filter process

1. AppStart calls FilterModel.fetch().
2. FilterModel dispatch ‘change’ event after data fetched.
3. FilterView call render() on ‘change’ event

### Event Dispatch design

|  |  |  |  |
| --- | --- | --- | --- |
| **Event** | **Dispatcher** | **Listener** | **Handle** |
| Change | FilterModel | FilterView | render |

## Filtering process

1. User click “Filter Orders” on FilterView;
2. FilterView will get “query” object from FilterModel.
3. FilterView dispatch “filterOrder” event.
4. OrderCollection will fetch orders based on the “query” received with the event.

### Event Dispatch design

|  |  |  |  |
| --- | --- | --- | --- |
| **Event** | **Dispatcher** | **Listener** | **Handle** |
| filterOrder | FilterView | OrderCollection | fetch |
| invalidValue | FilterModel | FilterView | showError |

## Creating a batch process

1. User click “Create a batch” on FilterView
2. FilterView will get current data set and dispatch it with a “createBatch” event.
3. BatchModel will listen to that event and call save() to persistence this batch object to database through REST API.

### Event Dispatch design

|  |  |  |  |
| --- | --- | --- | --- |
| **Event** | **Dispatcher** | **Listener** | **Handle** |
| createBatch | FilterView | BatchModel | save |