Trello Clone

How I built one

https://github.com/handofthecode/trello_clone

Data Modeling

-Trello is an application that allows users to create and manage lists as well as nested cards which can be moved between lists, edited, and reorganized.

-After considering the pros and cons, I decided to use embedded objects instead of object references to organize the lists and cards. Here are some of the factors that I considered when weighing the decision...

Reference vs Embedding

Reference	Embedding
-Great at modelling many-to-many relationships and perfectly capable of one-to-many relationships.	-Great at modeling one-to-many relationships.
-Doesn't exist in an ordered state. Needs to be sorted by a property.	-Nested objects exist in an ordered state within an array and can be easily re-ordered.
-All objects (lists or cards) can be stored in just two collections. This is good for querying and server processing.	-The visual model matches the data model. But difficult to access nested models directly, without also a reference to the collection.

This feature of embedded objects is what ultimately led to my decision to use this data structure for my Trello Clone App.

Sample Data

```
{"listSerial":2,
"cardSerial":3,
"lists":[
            {
              "title":"listTitle",
              "Id":1,
              "cards":[
                              {
                                "title":"cardtitle",
                                "Id":1
                              },
                              {
                                "title":"cardTitle2",
                                "id":2,
                                “description: “abcdefg”
                              }
                      ]
            }
      ]
}
```

Embedding with Backbone.js

It turns out that Backbone does not support painless nested collections. There are libraries such as backbone-nested that make the process more streamlined but I opted to just use native backbone, so I would be forced to work around its limitations. Would I come to regret this decision? Stay tuned...

toJSON

One crucial thing I had to figure out was to reimplement the “toJSON” method of the list models to include the card collection. Backbone’s “toJSON” method clones just the model’s attributes and reconstructs a simple object out of those properties. Here is my solution, which takes the “cards” collection and adds it to the object manually In order to preserve the card data in the returned object.

```
toJSON: function() {
  var json = _.clone(this.attributes);
  json.cards = this.cards.toJSON();
  return json;
}
```

Early on, I actually tried to maintain a cards collection within the models attributes as well as the cards property on the object at all times. I eventually realized this was efficient or easy to maintain, and so I found this much more elegant solution, creating the simple object, only when necessary.

Code Structure

$$\begin{tikzcd}
 & \text{AppView} \arrow[d] \arrow[dr] & \\
 & \text{BoardView} \arrow[dl] \arrow[d] & \text{SearchView} \\
\text{ModalView} & \text{ListsView} &
\end{tikzcd}$$

Views

The appView instantiates the collection and passes it to the search and board views beneath it. Rather than have one controller, I found it less convoluted for views to just trigger events on the collection that they all have a references to. Within the BoardView, I only used one view (listsView) to manage the lists and cards. As such, listsView ended up being pretty big. Continuing on this project, I would seriously consider making a cardsView.

I had initially used just one render method to populate the lists and then the cards within each list. Now that I have append methods and render methods and templates for both lists and cards, adding a separate cardsView would make a lot of sense for managing the code going forward.

Triggering Events

Another limitation with using nested collections with Backbone is that changes on the cards property collection or it’s cards won’t automatically cause the outer collection to fire an appropriate event. I would have had to either listen to each of my “cards” collections or make those nested collections aware of the parent collection to trigger it manually. I could have used a library for this but instead I opted to let the pertinent methods manually sync with the server and render views explicitly.

In addCard function -->
```
list.cards.add({title: title, id: this.collection.cardSerial++});
Backbone.sync('create', list.cards.at(-1));
this.appendCard(list);
```

Ultimately this might not be the best use of the features backbone has to offer, but I like that it is very clear to see what each method is doing.

API endpoints

While building the front end of the app, I just used just one API endpoint to get all of the data and to post it. To keep things simple at the outset, I posted the entire data object after every change to a list or card.

Once the frontend code was nearing completion, I added specific put and post routes to backup changes to properties and order of the cards and lists.

Because of the nested structure of the data, the API endpoints for editing and creating cards are organized like this: '/lists/:listID/cards/:cardID', lists are created and modified here: '/lists/:listID’, cards are reordered in their lists here '/lists' and lists reordered on the board here ‘/board’.

Performance (ajax)

With every operation that changes the data on the front end, I made sure to send an ajax request so the server would back up the data. I tried to keep the payload as small as possible. For example, when changing the order of cards or lists I tried to just send the bare minimum to the server to make the change to its data. In this example, we sent just an array of list IDs to the server to the “/board” endpoint so the server can handle the work of arranging the lists on the board in the right order. Here is what it looks like within the route.

```
var order = JSON.parse(req.body);
var result = order.map(el => _.where(lists, {id: el})[0]);
```

Libraries

As I said earlier, I opted not to use a Backbone library to handle nested collections. I did include the underscore library on the node server, but I only ended up using the “findWhere” method. Looking back, I could have gone without it.

For handling drags I found dragula to be the most promising, lightweight library focused on just dragging. I did have challenges at first, that I thought were caused by the “.gu-mirror” class that styles the appearance of the mirror element being dragged. Instead of getting to the bottom of it, I just commented the css class and moved on. Eventually, I found that it wasn’t just a css bug, it turned out that I was creating exponentially more drag event trackers with which each drag.

More on Dragula

I learned that it is only necessary to set both the card drag tracker and the list drag tracker just once, at the beginning of the initial render. From there, the “lists” container is never removed so it never needs to be updated. But, because new lists might get created, the new containers created for the cards need to be shared within the card drag tracker “container” property whenever a new list is added to the board.

```
this.cardDrags.containers = [$('.draggable')];
```

In conclusion

I learned a tremendous amount on this project, I could go on and on. I don’t think that I would do anything differently as far as my approach; I think I made good choices modeling the my data and in keeping new libraries to a minimum.

Here are some things I’d like to add and improve to the app in the future:

-a cardView
-ability to delete lists and cards (garbage can that you could drag cards/lists into)
-multiple boards
-multiple users
-ajax error handling