

BARISTA VIEW DOCUMENTATION

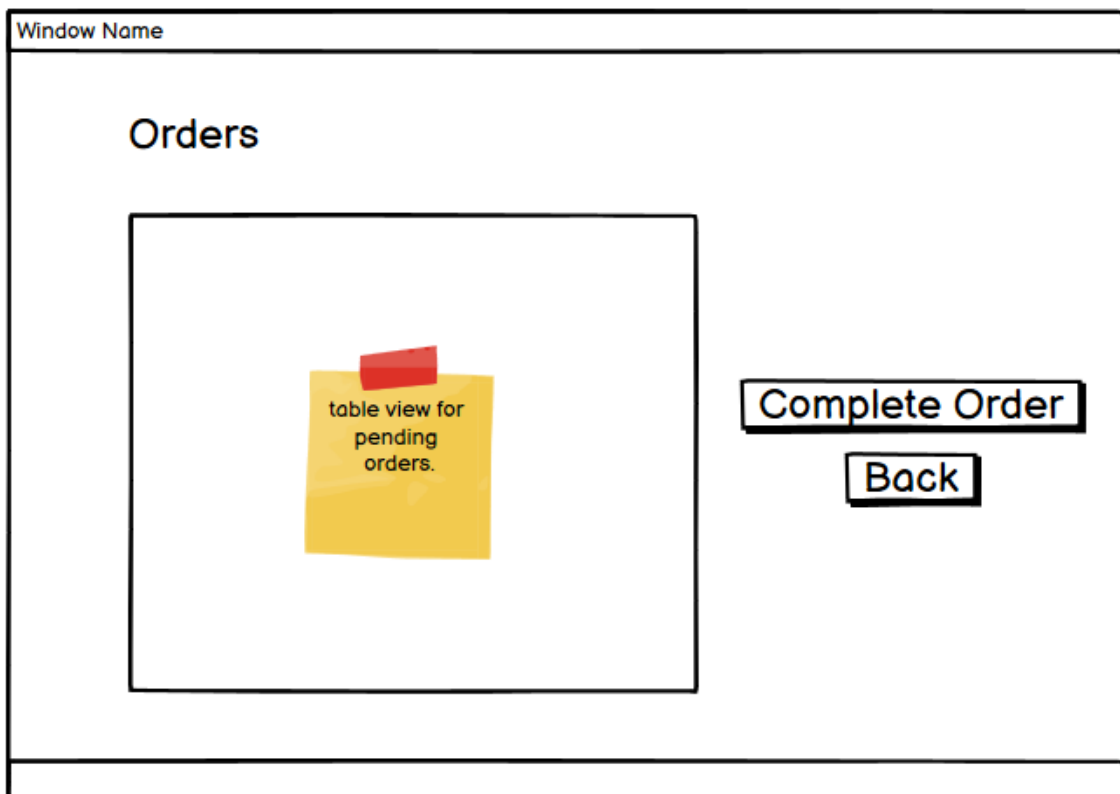
DESIGN CHOICES

The Product Backlog Items that this view fulfills are the following:

As a barista, I want to see all the pending orders so that I can start one at a time.

As a barista, I want to mark an order as complete so that a customer can collect it.

In order to achieve this, it was necessary to have a window with a Table View of the currently pending orders. It would also be necessary to have a button able to complete an order. With this in mind, the first design was the following:



However, it was later realized that the Table View did not display the details of the order – neither the items nor the possible comment to the order. So it would not make sense for a Barista to be able to mark an order as complete without access to its contents. **Therefore, the “Complete Order” button was changed to “See Details”,** which opens a new window. This new window would feature detailed information about the selected order. It is in the following page.

Window Name

Order Details

Items

table view for
items in order.

Comment

text area for
the comment

Back

Complete

With this design, both Product Backlog Items could be fulfilled.

IMPLEMENTATION

As for implementation details, there is a lot to delve into.

The ViewController is simple, but there is a crucial detail.

It has injectable FXML fields for the TableView and for all its Columns. It also has a selection model in order for it to be possible to know which order is pressed so that the new window can open the details of the right order.

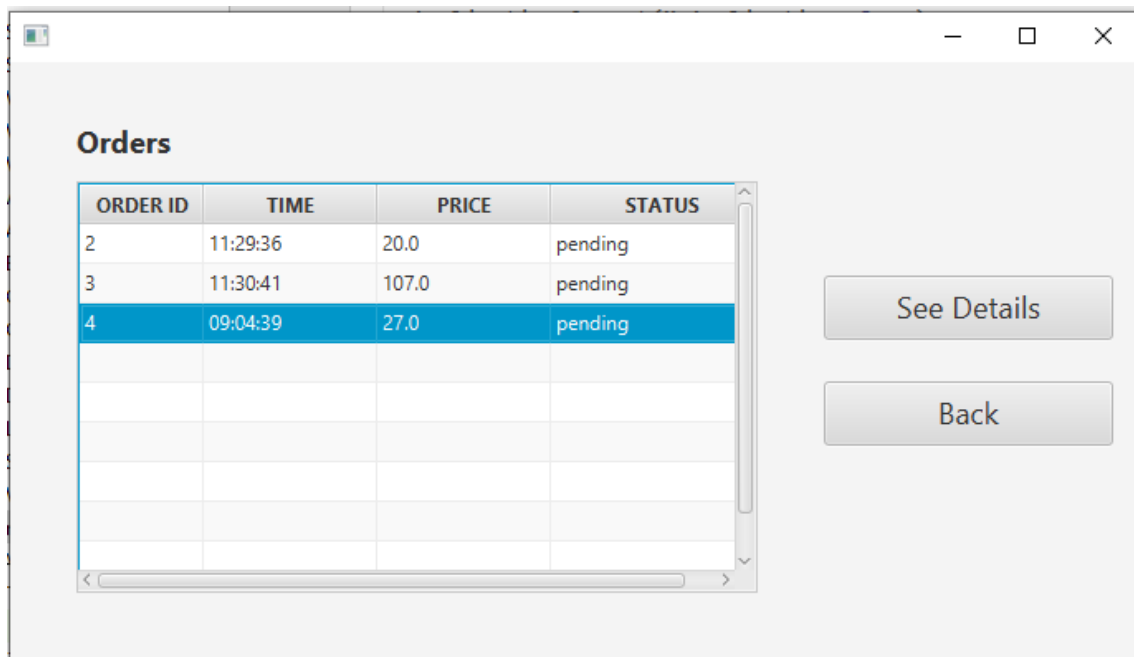
```
13 public class BaristaViewController extends ViewController
14 {
15     private @FXML TableView orderTableView;
16     private @FXML TableColumn idCol;
17     private @FXML TableColumn timeCol;
18     private @FXML TableColumn priceCol;
19     private @FXML TableColumn statusCol;
20     private BaristaViewModel viewModel;
21     private TableView.TableViewSelectionModel selectionModel;
```

The init() method sets the Cell Factories for the TableView, calls the reset() method (screenshot below) and initializes the selectionModel.

The reset() method calls the reset() method in the viewModel, making it so that the View package is not concerned with implementation in any way.

```
37 public void reset() {
38     viewModel.reset();
39     orderTableView.setItems(viewModel.getOrders());
40 }
```

The View looks like this:



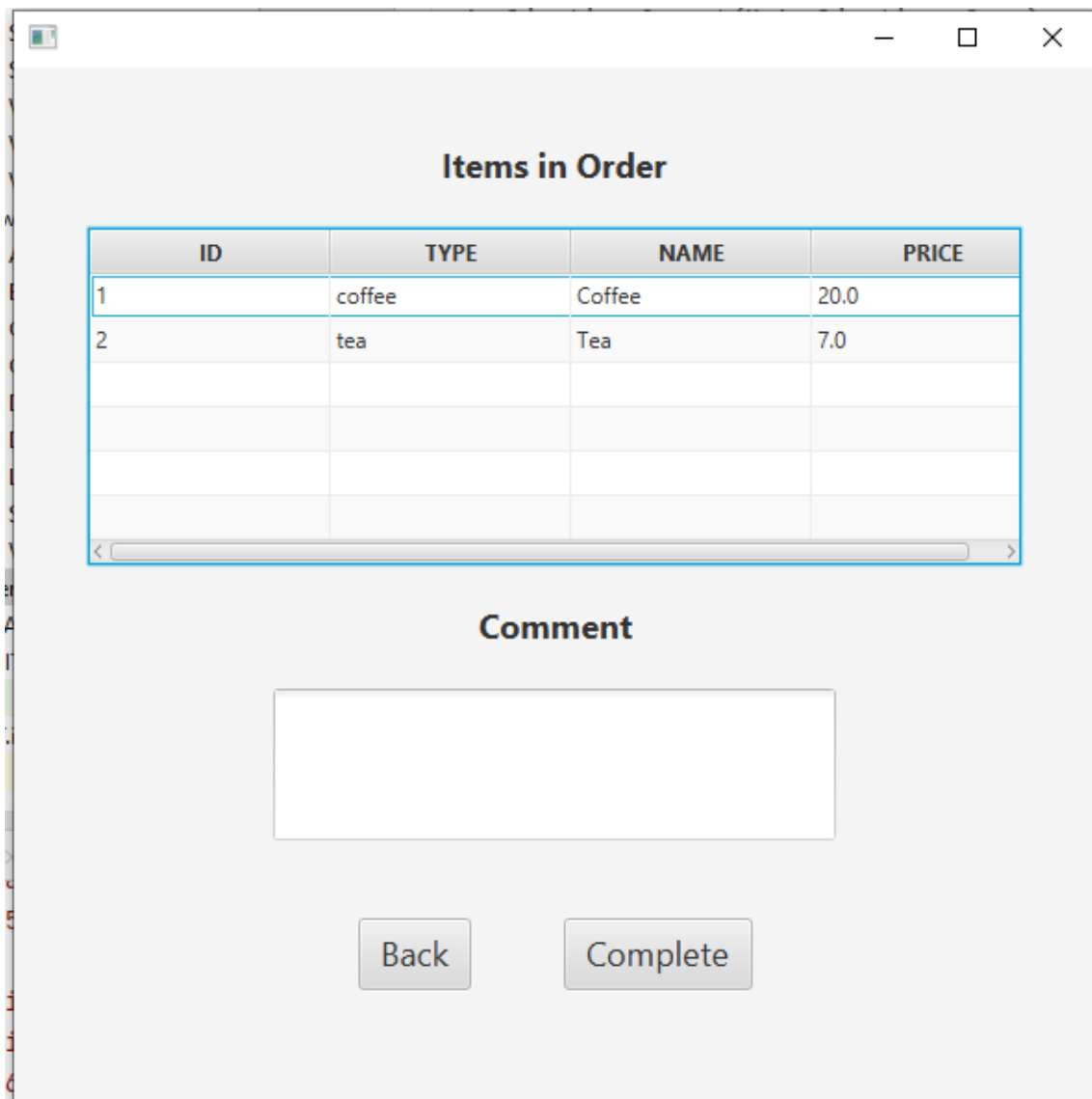
A method that should be mentioned is `seeDetailsPressed()`, which is ran when an order is selected and the button “See Details” is pressed. In this method, the selected order is fetched from the selection model, sent to the viewModel, and a new window is opened.

```
42 @FXML private void seeDetailsPressed() {  
43     OrderProperty order = (OrderProperty) selectionModel.getSelectedItem();  
44     viewModel.setSelectedOrder(order);  
45     getViewHandler().openView(id: "OrderDetailView.fxml");  
46 }
```

The functionality for this new window is the crucial detail mentioned before.

The new view, `OrderDetailView`, has the same viewModel as the `Barista View`. The reason why is because they must share the `Order` object that represents the selected order. This way, when in the “`seeDetailsPressed`” the “selected order” is set in the View Model, it will be accessible to the new window, because it has the same View Model.

The appearance of the new view is as follows:



The screenshot shows a window titled "Items in Order". Inside, there is a table with 4 columns: ID, TYPE, NAME, and PRICE. The table contains 2 rows of data. Below the table is a section titled "Comment" with a large text input area. At the bottom, there are two buttons: "Back" and "Complete".

ID	TYPE	NAME	PRICE
1	coffee	Coffee	20.0
2	tea	Tea	7.0

Comment

The barista can see the items inside the order and the comment. Right now, no order can have a comment because the Product Backlog item ***“As a customer, I want to be able to leave a comment in my order so that I can make a special request”*** was not completed. However, the functionality for fetching the comment is there.

The View Controller for Order Detail View is very similar to previously documented views: it has the FXML injectable fields for the TableView and a Text Area for the comment – nothing new. The init() method prepares the table and the reset() method populates it. The button “Back” takes the user back to the Barista View and the button “Complete” calls the following method:

```

55  @FXML void completePressed()
56  {
57      viewModel.completeOrder(viewModel.getSelectedOrder().getOrder());
58      backPressed();
59  }

```

It delegates the functionality to the View Model – the order is completed there, and then the user is taken back to the Barista View, because this order is done and forgotten.

So far, the View Model has been mentioned many times but never shown. It is time to look into it.

```

13  public class BaristaViewModel
14  {
15      private Model model;
16      private ObservableList<OrderProperty> orders;
17      private OrderProperty selectedOrder;

```

The View Model has three instance variables: a Model, of course; an observable list of Orders that represents the pending orders and an OrderProperty for the selected order. The OrderProperty class was created to facilitate the “observability” of the items inside the tables.

The main thing about the ViewModel is the completeOrder method.

```

40  public void completeOrder(Order order)
41  {
42
43      try
44      {
45          model.completeOrder(order);
46      }
47      catch (RemoteException e)
48      {
49          e.printStackTrace();
50      }
51
52      reset();
53  }

```

The View Model delegates this task to the Model and calls reset() to fetch the pending orders again. This way, the order just now completed will no longer be in the list and possible new pending orders will be fetched.

With this functionality, the Barista actor can do everything they were set out to do.