

**Design Schema**

The 4Kotas system was designed and built using a layered architecture where each of the components on layer n are at least partly composed of the components on layer n-1. For example, the Starter Pack menu item is made up of the ingredients bread, chips, cheese and polony. A level up from there, the Starter Pack Line Item is made up of the Starter Pack menu item and other attributes including its name, quantity, unit price and cost properties. Since the system is written in JavaScript, which is object oriented, the interfaces to items at each level of the design are not found between said level and the one above it, but are rather encapsulated within each object.

The coding methodology combines the constructor pattern with prototypal inheritance to create the objects in the system and object prototypes are defined using object literal notation.

**Architectural Issues**

There is a major flaw in the design of the system which did not become problematic until I started considering how I would add more functionality to the application. For example, I would like the store owner to have an analysis page where he/she can view sales and stock figures over a given period to help them make strategic decisions for their business. The issue is that navigating away from the Till Slip page to this new Analysis Room wipes out all the data living in the application’s objects, the very same data required for analysis. This is not exactly an issue of bad design on my part but rather a consequence of how scripts are handled in the browser. Either way, I have to find a solution to this problem.

While I have never used local storage before, I figure there must be a way for me to keep my objects alive by using it. The other option is to use a database to store the data. The system would then be distributed in a classic client-server-database scheme. In this case, my initial idea was to put the object properties in the database while keeping the methods (i.e. the prototypes) on the page. I am sceptical about this decision because while doing calculations on the client side might relieve some pressure off the server, the repeated to and fro of request and responses across the network for the values required for those calculations may be just as costly, if not riskier as well. So maybe it is best to have the objects, in their entirety, live in the server and then complement them with proxy’s that will act as interfaces on the client side.

The client-server-database idea won over local storage because, as a learner, I figure I will be much more empowered once I have learned to implement a system of that type of complexity. That is why I am currently teaching myself Node.JS and MongoDB. The next changes you will see to the system will be done once I have learned to use these tools. I look forward to comments and critiques in the interim.