Research Project- John Shop

Nicholas Jumpp- 1501298

Kimone V. Roper Ist -1403541

University of Technology Jamaica

**Abstract**

This advanced programming research paper explores common/popular/recommended design patterns and principles both in concept and through application. Select patterns are implemented throughout a JavaScript application to achieve relatively low coupling through abstraction. The web application is centered around a snack delivery system. It facilitates students ordering snacks as well as the business owner and delivery personnel viewing the orders made.

**Table Of Content**

[S.O.L.I.D. Principles 3](#_Toc15027)

[Design Patterns 6](#_Toc28829)

[MVC Pattern 6](#_Toc18385)

[Model 6](#_Toc20105)

[View 6](#_Toc28629)

[Controller 7](#_Toc7236)

[Repository Pattern 7](#_Toc5104)

[Singleton Pattern 8](#_Toc27699)

[Factory Pattern 9](#_Toc17153)

[References 10](#_Toc1925)

**S.O.L.I.D. Principles**

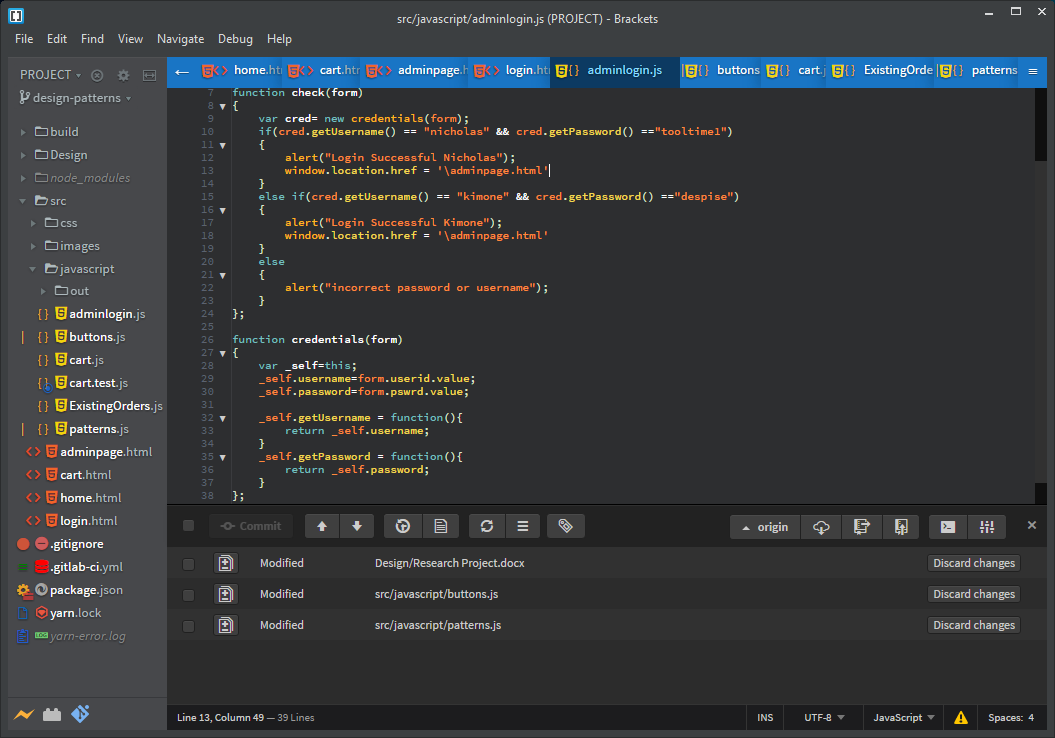
“Object-Oriented type of programming brought a new design to software development. This enables developers to combine data with the same purpose/functionality in one class to deal with the sole purpose there, regardless of the entire application.” (Nnamdi, 2018).

The five principles are as follows:

* S – Single Responsibility Principle (SRP)
* O – Open Closed Principle (OCP)
* L – Liskov Substitution Principle (LSP)
* I – Interface Segregation Principle (ISP)
* D – Dependency Inversion Principle (DIP)

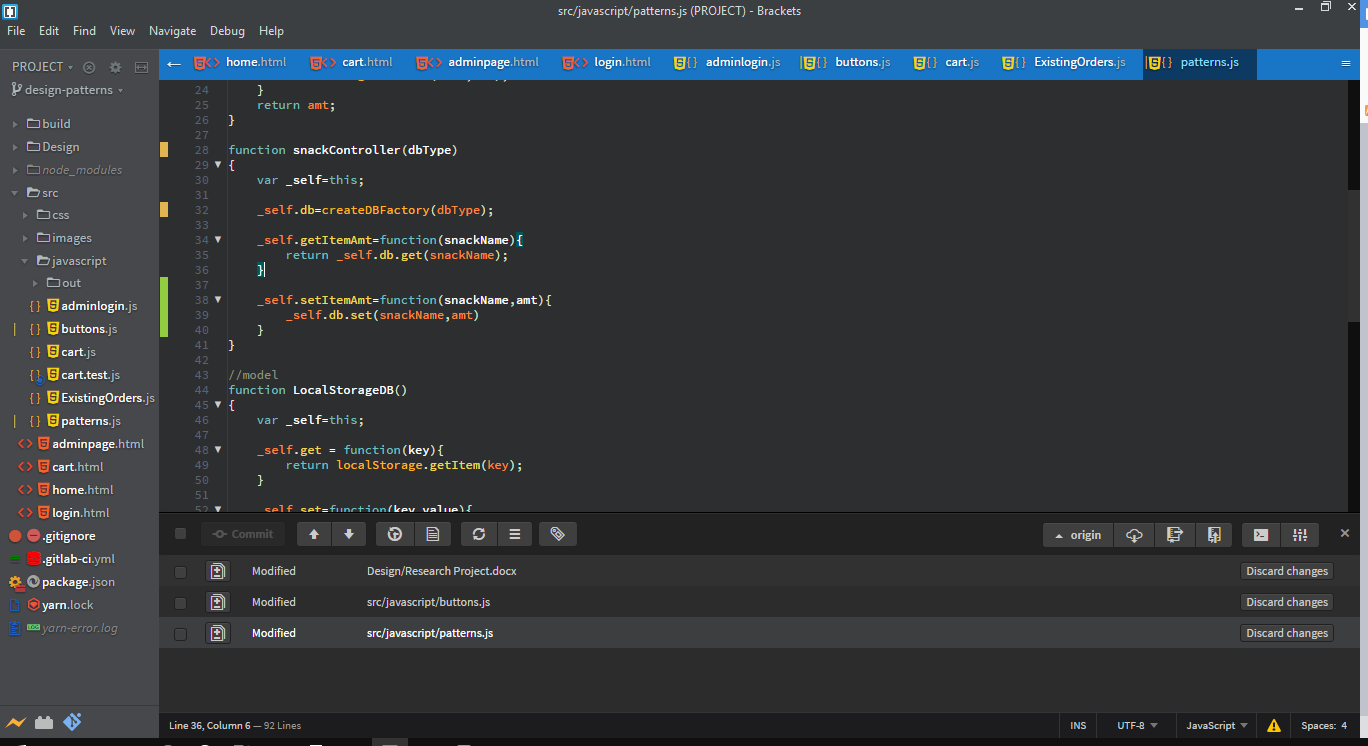
The Single Responsibility Principle (SRP) states that “A class should have one, and only one job.”(Barber, 2015)

* The SRP is seen in the adminlogin.js file as its only job is to verify a user for successful login. As seen in the screenshot below the check function simply verifies the login of a an admin that being the manager or the delivery person. The credentials function simply acquires the credentials from the textboxes on the login.hml page and returns the username and password using the \_self.getUsername function and\_self.getPassword respectively when invoked



The Open Closed Principle (OCP) states that “You should be able to extend a classes behavior, without modifying.”(Barber, 2015)

* OCP is seen in the pattern.js file with the snackController function open for extension in which the functionality can be extended by specifying the database type but closed for modification in which it can be modified by any subclass.



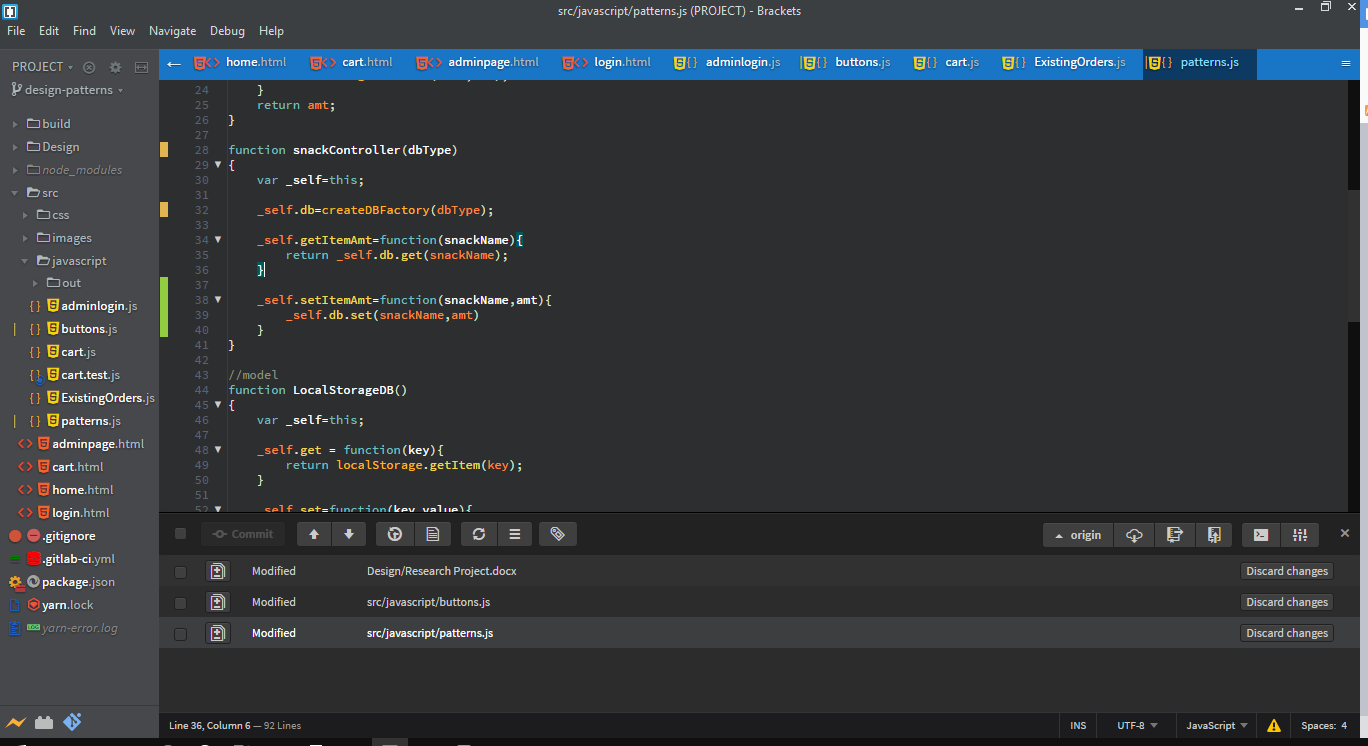
The Liskov Substitution Principle (LSP) states that “Derived classes must be substitutable for their base classes.”( Barber, 2015)

The Interface Segregation Principle (ISP) states “Make fine grained interfaces that are client specific.”( Barber, 2015)

* ISP was not implemented due to the fact interfaces are not able to be created because no built in support for traditional abstraction is in JavaScript.

The Dependency Inversion Principle (DIP) states “Depend on abstraction, not on concretion.”( Barber, 2015)

* The screenshot below demonstrates dependency inversion in which it is not being dependent on a specific database such as localStorage or MySQL. Simply place the database to be used in the parameter which will use the means of storage specified



**Design Patterns**

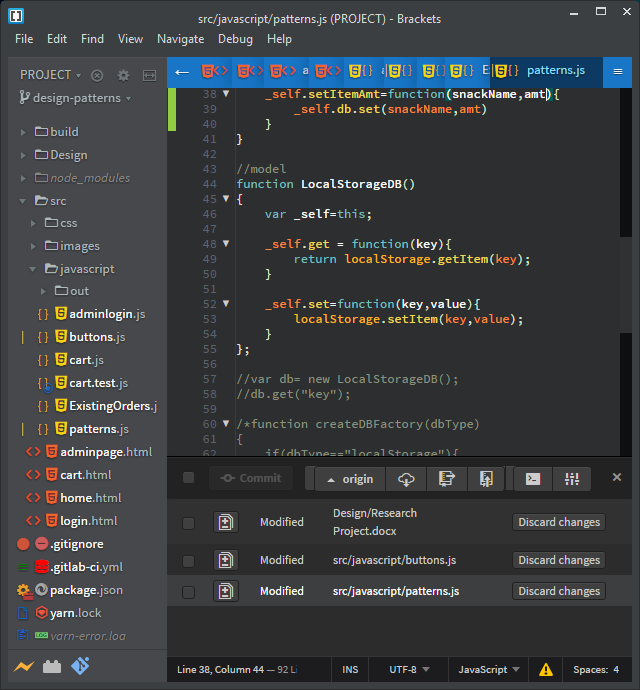
Model View Controller (MVC) Pattern

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns. (Tutorialspoint.com)

* Model - Model represents an object carrying data. It can also have logic to update controller if its data changes.
* View - View represents the visualization of the data that model contains.
* Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Model

The storage medium used in this research project is localStorage that is within modern browsers which stores necessary information from the user to be either presented back to them or to the manager or delivery man.



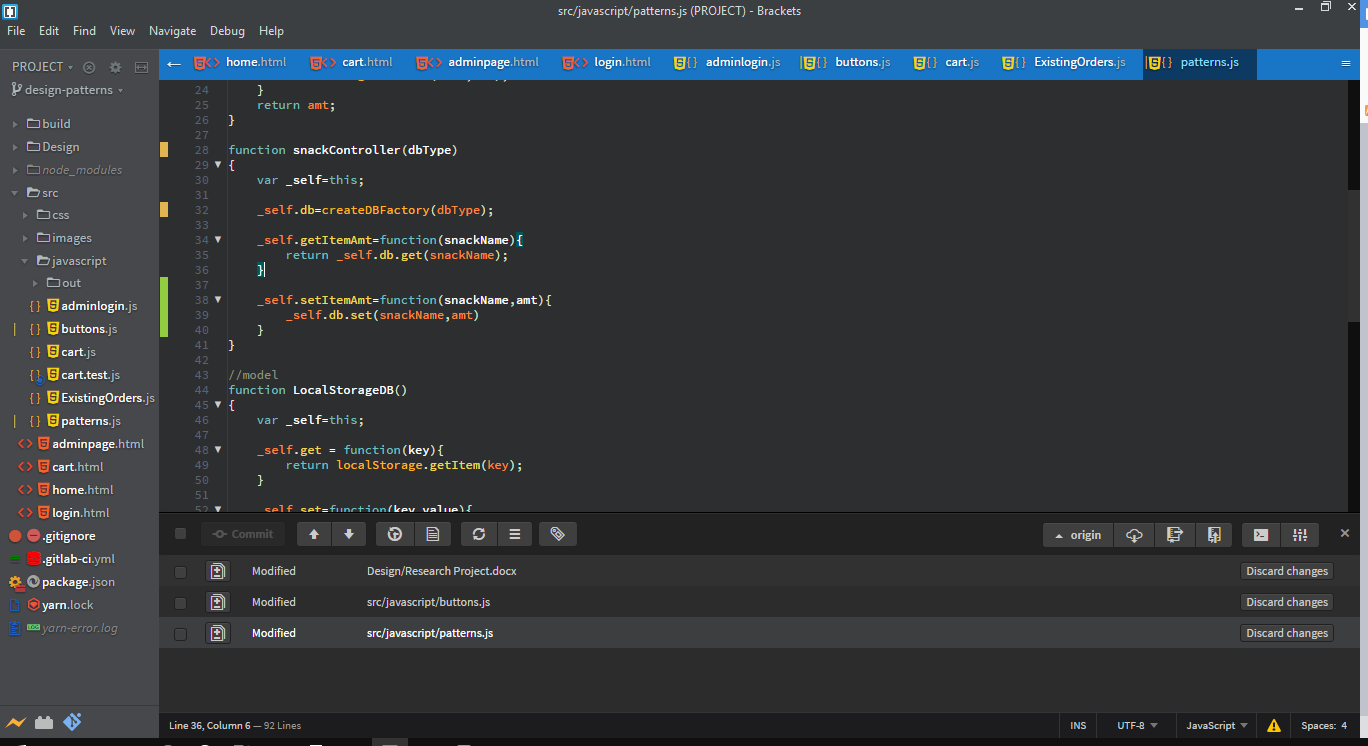
The screenshot above shows the model used which is localStorage to act as our database which contains the two functions to update(\_self.set) and retrieve(\_self.get).

View

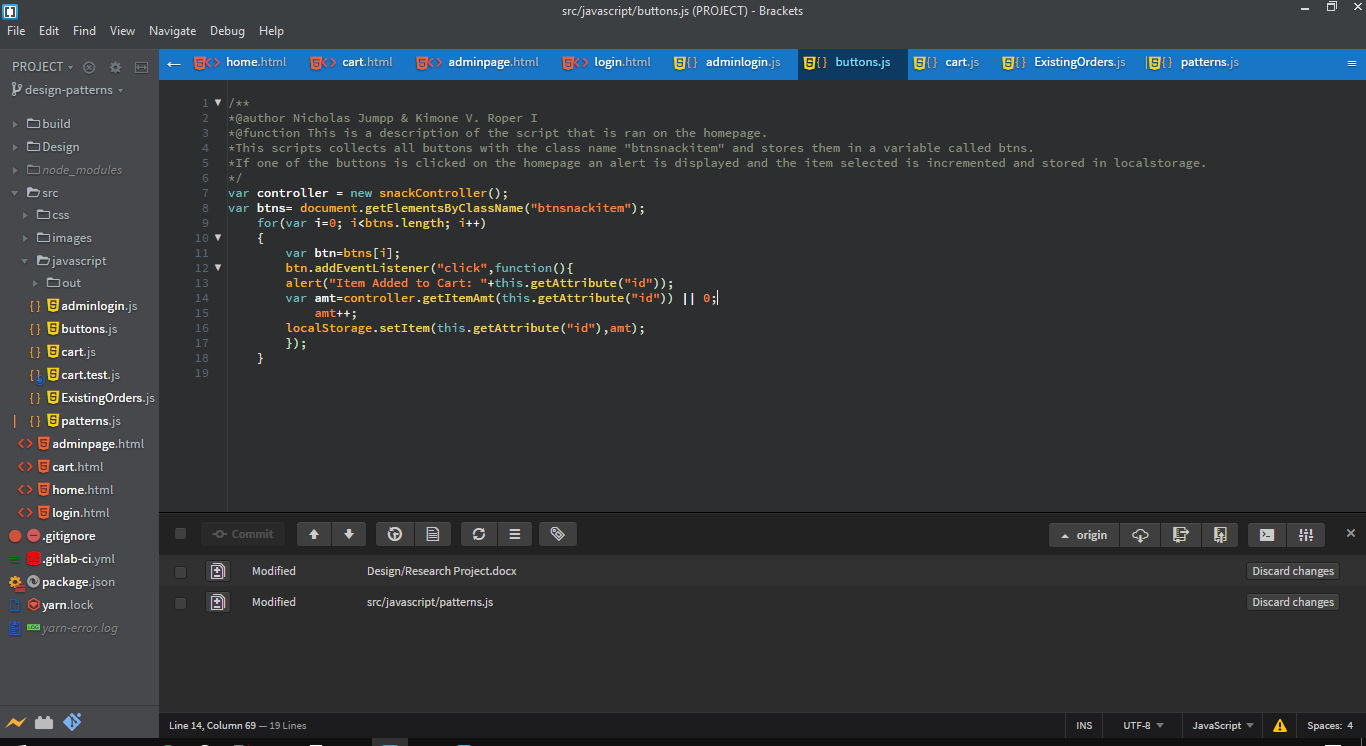
In our research project the view is our UI presented to the customer and manager/ delivery man of John Shop which was coded using HTML & CSS for styling.

Controller

The controller works in conjunction with the model and the view as to update the page as necessary if there has been any changes made from the view portion or from the model portion to the view.



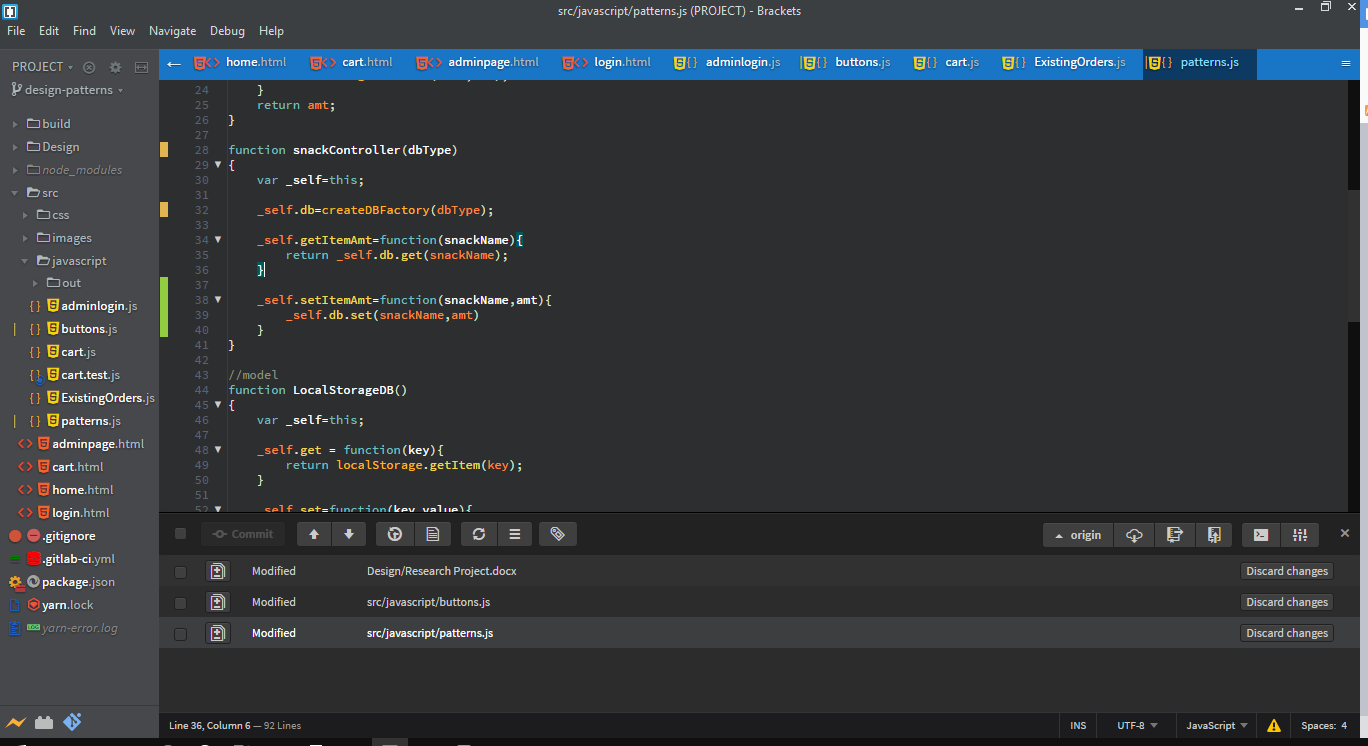
The screenshot above shows a function that is implementing the controller in the MVC pattern for snacks in the factory.js file. What this does is to call the factory method to create a localStorage object for us to access the model to retrieve information from it.



As seen in the screenshot above in the buttons.js file an instance of the snackController is created which is used to communicate between the model(localStorage) and the view which is the even listener that is is seen here.

Repository Pattern

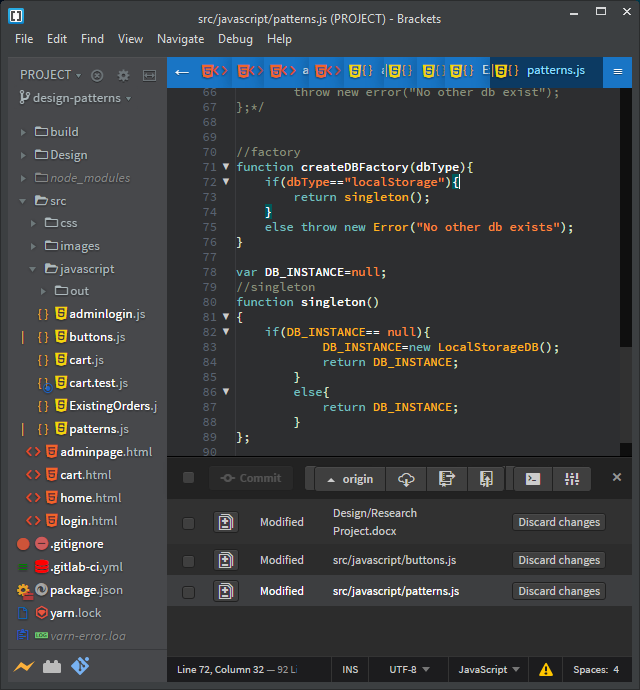
The Repository Pattern restricts us to work directly with the data in the application and creates new layers for database operations, business logic, and the application’s UI.



Sometimes our controllers are too tightly coupled to a particular implementation. Abstraction can be used to reduce this coupling. In the screenshot above, it shows an example of the repository pattern. Instead of strictly declaring localStorage it takes a parameter dbType that specifies what means of storage is preferred to be used during run-time.

Singleton Pattern

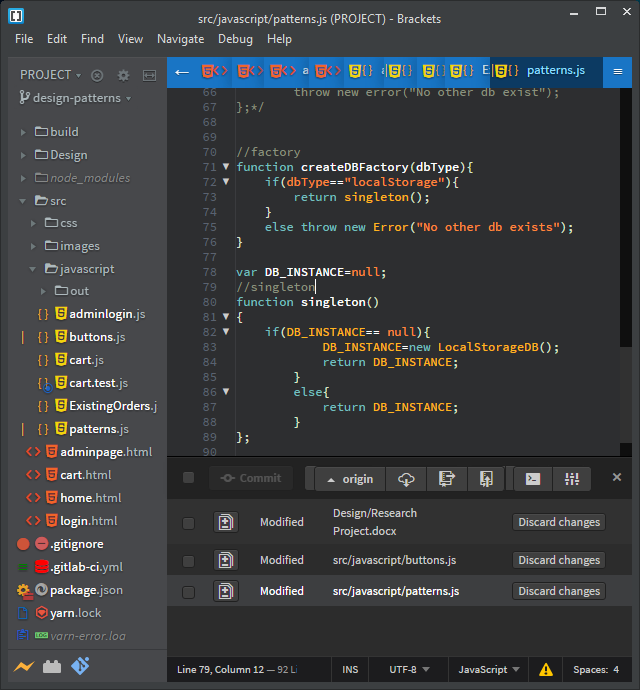
“The singleton pattern is a design pattern that restricts the instantiation of a class to one object” (Singleton, 2018).



The screenshot above demonstrates the singleton pattern in which only instance of the object/class should be present. How this is accomplished is that a variable called DB\_ISNTANCE is created and set to null from that the singleton() function in pattern.js checks if an instance has already been created. If an instance has not been created an instance is created of the LocalStorageDB and returned to be used by the function that requests access to the model. If an instance has already been created then it simply returns the instance that was made previously.

Factory Pattern

“A Factory Pattern says that just define an interface or abstract class for creating an object but let the sub-classes decide which class to instantiate” (JavaTPoint, n.d.).



The screenshot above demonstrates the factory pattern which checks the storage type being requested as seen with the parameter dbType. The if statement then checks as to which method of storage is being requested for example, checking to see if the localStorage Database is being requested and if true, then it calls the singleton to get an instance of the localStorage Database. It then returns the instance to the snackController() function.

**References**

Barber, A. P. (2015, August 06). SOLID Principles - Five Principles of Object-Oriented Programming and Design. Retrieved from <https://alanbarber.com/post/solid-principles-five-principles-of-objectoriented-pro> gramming-and-design/

JavaTPoint. (n.d.). Factory Method Design Pattern - Javatpoint. Retrieved from https://www.javatpoint.com/factory-method-design-pattern

Nnamdi, C. (2018, October 09). SOLID Principles every Developer Should Know. Retrieved from <https://blog.bitsrc.io/solid-principles-every-developer-should-know-b3bfa96bb68> 8

Singleton Design Pattern | Implementation. (2018, September 10). Retrieved from https://www.geeksforgeeks.org/singleton-design-pattern/

Tutorialspoint.com. (n.d.). Design Patterns MVC Pattern. Retrieved from https://www.tutorialspoint.com/design\_pattern/mvc\_pattern.htm