Grockart – Group 10

Group Report

Q. How you implemented the continuous integration?

A. Continuous integration was implemented with Jenkins. There was no hard-coded. configuration. All the database connections are accessed via Grockart. Credentials dll which contains the required configuration. That dll should be deployed on the server which will get back the connection string.

Q. List the design patterns used, and why were they used?

A. We implemented the following design patterns.

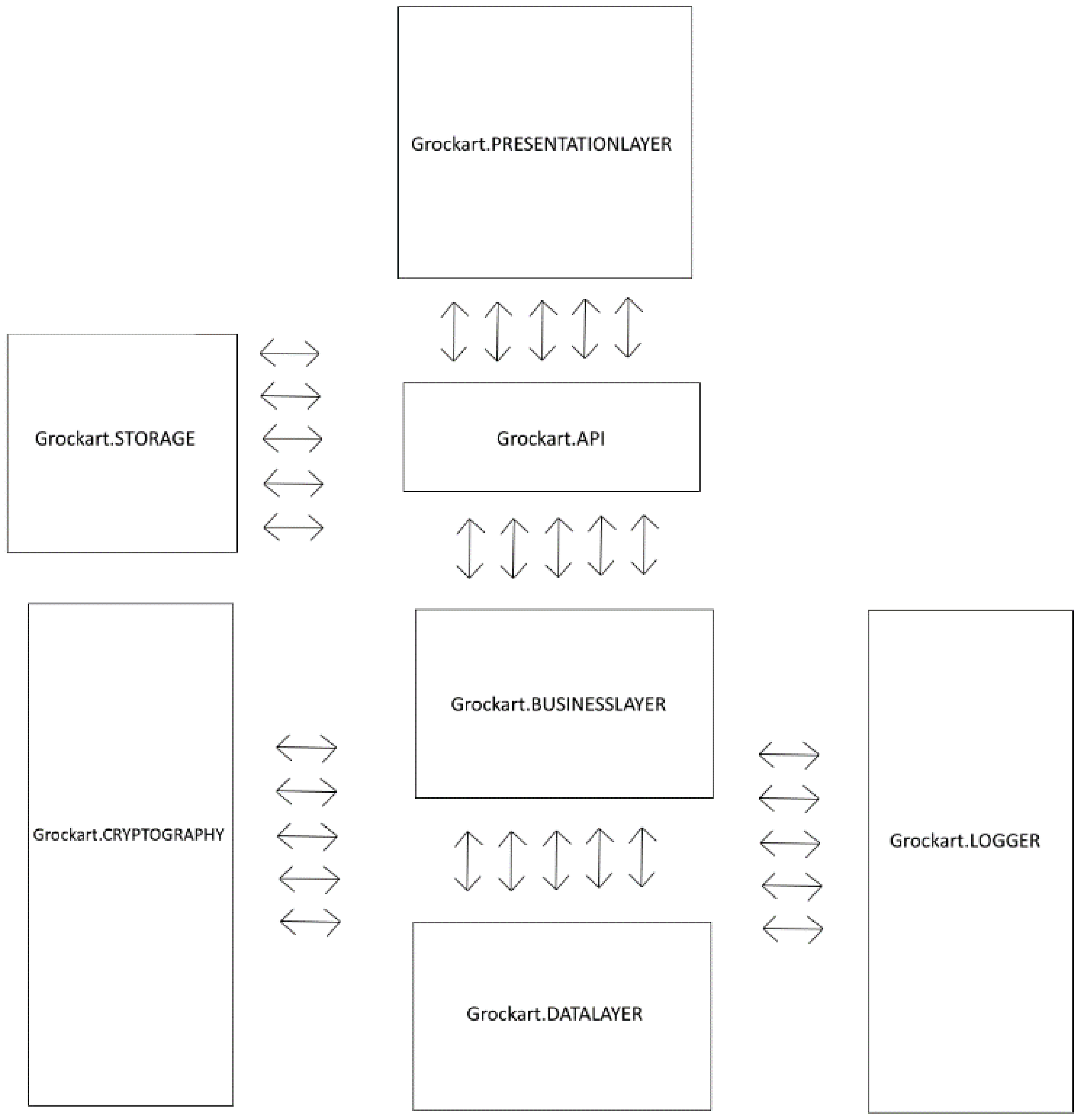
- Builder Pattern
  - Building different orders based on the order type (fetching Cancelled order, shipped order).
  - Building search result, when a user searches for a grocery, builder pattern is used which fetches all the product details.
- Singleton Pattern
  - Fetching Logs, adding log
  - Fetching database connection
  - Fetching Storage items like cookies, sessions, and cache.
- Proxy Pattern
  - Fetching the type of cookies, session and cache values. Singleton pattern will fetch the instance but actual insertions, deletion, and modification will happen through a proxy
- Template Pattern
  - All CRUD operations for Product, category, store, card details, address details, order status

Q. How did you achieve the separation of presentation, business and data layer?

A.

The Presentation layer interacts with the API layer which interacts with the Business layer and then the business layer interacts with the data layer. Cryptography layers handle the encryption and decryption for each object while Grockart.Logger will log each event from the business layer and the data layer.

Please check the image below for the architecture



Q. Explain the naming convention and the spacing convention you agreed upon and why?

A. Naming Convention: Pascal Naming convention for being consistent across the complete codebase.

Spacing Convention: Tabs

Q. Examples of any refactoring you performed?

A. The following refactoring was performed

- **Decompose conditional:** Removed complicated ifelse statement in User Login operation and replaced it with class objects.

- **Duplicate observed data:** There was business logic in presentation layer which was later replaced to api folder.
- **Form template method:** Using the template design pattern for getting order details and performing CRUD operations.
- **Hide method:** Removed un-necessary public method declarations and converted it to private.
- **Introduce parameter object:** Removed unnecessary parameter declaration in constructor objects and replaced them with class objects.
- **Preserve the whole object:** We are returning and setting the calls as an object.
- **Replace conditional with polymorphism:** Removed complicated if-else statements and replaced them with conditional polymorphism.
- **Self-encapsulating field:** We have created the getters and setters method.

Q. What is considered a "technical debt" and how can it be resolved?

A. Technical Debt

- **Encapsulate field:** To print the data members in JSON format, we have converted the private member to public members.
- No test cases for abstract class
- **Remove setting method:** There are still some classes having a setting method, we must remove that method.
- **Parameterized method:** There are still some of the classes which use parameters in the method, we must remove that and convert it into class objects.
- **Incomplete functionalities:** Following functionalities couldn't be completed because of time constraints.
    - Group split
    - Forgot password action
    - Frontend code for admin panel

  Although backend functionalities are complete including the test cases.

Grockart

Individual Contributions

Team Member: Deep Prakash Singh

Contributions

Presentation Layer

- Admin.aspx/cs
- Admin_master.aspx/cs
- Admin_Category.aspx/cs
- Admin_Cart.aspx/cs
- Checkout.aspx/cs
- Default_home.aspx/cs
- Global.asax
- Login.aspx/cs
- Main.master/cs
- MyOrders.aspx/cs
- Products.aspx/cs
- Signout.aspx/cs

Presentation Layer API (aspx + cs files)

- Api/AddAdmin.aspx
- Api/AddCardDetails.apsx
- Api/AddToCart.aspx
- Api/AddUserAddress.aspx
- Api/AuthenticateUser.aspx
- Api/CancelOrder.aspx
- Api/ClearCart.aspx
- Api/DeleteCart.aspx
- Api/DeleteCard.aspx
- Api/DeleteCategory.aspx
- Api/DeleteStore.aspx
- Api/DeleteUserAddress.aspx
- Api/FetchAdminList.aspx
- Api/FetchCategoryList.aspx
- Api/FetchProducts.aspx
- Api/FetchUserList.aspx
- Api/GetCart.aspx
- Api/GetCardList.aspx

- Api/GetCityList.aspx
- Api/GetTaxValue.aspx
- Api/GetUserAddress.aspx
- Api/ModifyCart.aspx
- Api/ModifyCategory.aspx
- Api/PlaceOrder.aspx
- Api/RemoveAdmin.aspx
- Api/UserProfile.aspx
- Api/AllGroupOrderPlaced.aspx
- Api/orders/AllGroupOrderPlaced.aspx
- Api/orders/AllGroupOrders.aspx
- Api/orders/AllIndividualOrdersPlaced.aspx
- Api/orders/AllIndividualOrders.aspx
- Api/orders/CancelledGroupOrders.aspx
- Api/orders/CancelledIndividualOrders.aspx
- Api/orders/OrderPlacedGroupOrders.aspx
- Api/orders/OrderPlacedIndividualOrders.aspx
- Api/orders/ShippedGroupOrders.aspx
- Api/orders/ShippedIndividualOrders.aspx
- Api/orders/UnpaidGroupOrders.aspx
- Api/orders/UnpaidIndividualOrders.aspx

Grockart.BUSINESSLAYER

- AbstractProductBuilder.cs
- AdminUserTemplate.cs
- ConcreteProductBuilder.cs
- DBVersion.cs
- ISecurity.cs
- MaintenanceMode.cs
- Menu.cs
- OrderCreator.cs
- ProductBuilderResponse.cs
- ProductList.cs
- Province
- Security
- SettingsFromDB (BusinessLogic)
- TaxManagement
- UserActions

- UserTemplate

Grockart.CRYPTOGRAPHY

- AES256.cs
- IEncrypt.cs
- IHash.cs
- SHA256.cs

Grockart.DATALAYER (Including the SP (stored procedure) related to this layer)

- AdminUserTemplate.cs
- ICommands.cs
- Images.cs
- ISecurityDataLayer.cs
- MySQLCommands.cs
- NormalUserTemplate.cs
- OrderCreatedDataLayer.cs
- ProductMenu.cs
- ProductDataLayer.cs
- ProvinceDataLayer
- SecurityDataLayer
- ConfigurableBusinessLogic.cs (Fetching keys and values from DB)
- StoresList.cs
- TaxManagementDataLayer.cs
- UserActionsDataLayer.cs
- UserTemplate.cs

Grockart.LOGGER

- Debug.cs
- ENumLogType.cs
- Fatal.cs
- ILogType.cs
- Info.cs
- LogDebug.cs
- Logger.cs
- LogInfo.cs
- Warn.cs
- WarnDebug.cs

Grockart.STORAGE

- CacheProxy.cs
- CookieProxy.cs
- IStorage.cs
- SessionProxy.cs

Javascript files

- Js/Admin.js
- Js/cart.js
- Js/checkout.js
- js/grockart.js
- js/Login.js
- js/Orders.js
- js/Products.js
- js/register.js

Mohit Malik's Contribution

Before this Course:

- I was jack of many languages such as C, C++, C#, Java.
- I never did Test Driven Development
- I never wrote a single test case for any program/assignment/project.
- I only heard about SOLID principles but never used.
- I did not know anything about Design Patterns.
- I only heard of refactoring but never tried to apply it in my code.
- I never used interface or abstract classes while coding.
- I always preferred messy if-else and switch-case statements.
- I never cared about indents, spaces and tabs.

After this Course:

- Now I have become more proficient in C# and Java
- After using Test Driven Development in Assignment and in this project, I can easily perform TDD and will recommend others to perform TDD because of its long-term benefits.
- I got to know the value of unit test cases. From now onwards, I am always going write unit cases for my functions.
- Now I am comfortable in writing Code with keeping SOLID principle in my mind. Dependency Injection Principle has become my favorite.
- I cannot say I have become expertise in Design Patterns but now I am so much comfortable in using Template Pattern, Builder Pattern, Singleton, Abstract Factory.
- I always used to wonder how I can convert my messy code into a flexible one. And after taking this course I found my answer and that's "Refactoring".
- Now I hardly write any class without any interface or abstract class. Abstraction has completely changed my way of coding.
- I love TABS now.

Challenges:

- In starting of the project, I did not write any test cases. So whenever i used to write a new code or class I had to go back and check all the previous classes are working properly or not. And it was a cumbersome task. But when I started writing test cases it made my life lot easier.
- In the early phase of the project, due to lack of experience in Design Patterns me and my group member applied Factory Method design pattern for CRUD operations. But later on when we studied Design Pattern more carefully we found out Template Design Pattern. And then we had to go back and change all the classes which implemented Factory Method. It was good lesson but learnt the hard way.
- 2 or 3 weeks back we had some confusion in syncing our git branches and we did overwrite each other's classes and then we had to change it all back.

PRESENTATION LAYER:

FetchCategoryList.aspx.cs

Purpose: Fetch category list from database (via business layer and datalayer) and populate the list on web page.

Author: Me

Contributor: Deep (minor changes)

FetchProductAdminPanel.aspx.cs

Purpose: Fetch product list and populate on Admin panel

Author: Me

FetchStoreList.aspx.cs

Purpose: Fetch store list on populate on Admin panel

Author: Me

Contributor: Deep (minor changes)

ModifyCategory.aspx.cs

Purpose: Modify category in admin panel

Author: Me

Contributor: Deep (helped in JavaScript part)

Register.aspx.cs

Purpose: User Registration

Author: Me

Contributor: Deep (Helped in "On_LoadComplete" method with session issues)

BUSINESS LAYER:

AddressBusinessLayerTemplate.cs

Purpose: Performs all CRUD operation related to addresses.

Author: Me

Contributor: Deep (Initialized AddressDataLayer object in constructor of above class)

CardDetailsBusinessLayerTemplate.cs

Purpose: Performs all CRUD operations related to addresses.

Author: Me (Original code committed by me. Commit ID: 83fe0387)

Contributor: Deep (Due to confusion in syncing of branches Deep is showing as contributor).

CategoryBusinessLayerTemplate.cs

Purpose: Performs all CRUD operations related to categories.

Author: Me

CRUDBusinessLayerTemplate.cs

Purpose: Abstract class for applying CRUD operations using Template Design Pattern in Business layer

Author: Me

OrderBusinessLayerTemplate.cs

Purpose: Performs all CRUD operations related to orders.

Author: Me

Contributor: Deep (added one extra method Cancel())

ProductBusinessLayerTemplate.cs

Purpose: Performs all CRUD operations related to products ONLY

Author: Me

Contributor: Deep (changed the namespace)

ProductByStoreBusinessLayerTemplate.cs

Purpose: Performs all CRUD operations on products based on the store

Author: Me

Contributor: Deep (changed the name space)

StoreBusinessLayerTemplate.cs

Purpose: Performs all CRUD operations related to Stores

Author: Me

RESPONSE_CLASSES:

(These classes are either setter & getter or Interface/ Abstract classes)

Address.cs

CardDetails.cs

CategoryListResponse.cs

Category.cs

IAddress.cs

ICardDetails.cs

ICategory.cs

IOrder.cs

IProduct.cs

Order.cs

ProductOnly.cs

DATA LAYER:

AddressDataLayer.cs

Puprose: Performs all CRUD operations related to addresses in Data Layer

Author: Me (Original code committed by me. Commit ID: 83fe0387)

Contributor: Deep (Due to confusion in syncing of branches Deep is showing as contributor)

CardDetailsDataLayer.cs

Purpose: Performs all CRUD operations related to card details in Data Layer

Author: Me (Original code committed by me. Commit ID: 83fe0387)

Contributor: Deep (Due to confusion in syncing of branches Deep is showing as contributor)

OrderDataLayer.cs

Purpose: Performs all CRUD operations related to orders in Data Layer

Author: Me

Contributor: Deep (added one extra method Cancel())

ProductTemplate.cs

Purpose: Performs all CRUD operations related to Products in Data layer

Author: Me

Contributor: Deep (Added 3 new methods)

TEST CASES:

Total test cases written by me: **140**

Note: Test cases starting with "MySQLProductDataLayer" written by me but due to changing the namespace, these are shown under Deep's name. (Commit ID: 33709728)

STORED_PROCEDURES:

Total stored procedures written by me: **21**

Individual Contributions

Team Member: Tanishka Chaturvedi

Take Away From this Course

I learnt a lot of new things (though hard to express everything here) within this course like –

- Earlier I just used to code now I can do good and a standard good.
- Familiar with industrial requirements.
- I got to know the real meaning of SOLID principle and Design patterns.
- Now I can say I can write industrial Standard of code.
- Got more familiar with Agile Methodology and its industrial standards.
- Time management and work pressure management within Sprints releases.
- Learn how to Work more efficiently in a Test-driven environment.
- Not all but got to know most of the advance software development challenges and flow.
- Why logging and error handling is important from industrial prospect.
- What real Refractoring looks like and how to apply those in my code.
- While completing assignment most of the main component got covered itself like Unit Testing, SOLID Principle, Design Pattern and Refractoring.
- Lastly, how to work good under pressure.

Real Challenges -

- Though I were having the enough experience to work on cloud by deployment of Jenkins over Azure was a task for me.
- Using Git as a mediator for Continuous integration was very new approach for me.
- Working on different database then moving code from Dev-Test-Prod was the real time feel and was a little challenge for me.
- Showing something new code to TA every Thursday was a big task for us because for that we had to have some new work to show, so meeting that expectation was also a task for us.
- Using try catch and handling exception at every layer was also a challenge for me.
- Completing all assigned Trollo task was initial a challenge later on got used of it.
- The most take-away that I have mentioned above are only the results of these challenges, but now I can say that these challenges are not the challenges for me anymore.

Note -

Though I haven't written as much classes as compare to the other members of my group, but I can claim that my classes are the backbone of this project and whatever work I have done is a quality work. As I have applied mostly two design patterns – **Template** and **Builder**, I refactored code two to three time as Changing a long parameter into objects, **Duplicate Observed Data**, **change value to reference** were main of Refractoring applied.

Fully functional classes – Business Layer

AllOrdersTemplate.cs **Commit Id** 8e53fd4d
This class will have the template for calling All types of order 'Shipped','Unpaid','Cancelled' types of orders.
Design Pattern Used – Template
Learning Outcome- Template Pattern, Refractoring (Changed long parameter list into objects), Dependency Injection (DIP).
Other Contributors – (Deep)has moved one line from one method to another.

CancelledOrdersTemplate
Design Pattern Used **–** Template and Builder Pattern
Learning Outcome -Builder Pattern and Solid pattern**,**
Other Contributors – (Deep) Removed Builder response object.

GroupOrderTemplate commit Id 4fec35bc
Design Pattern Used **–** Template and Builder.
Learning Outcome- Template Pattern, Refractoring(Changed long parameter list into objects) , Dependency Injection. All the details of the group will come together.
Other Contributors – Deep(remove the singleton pattern used)

IndividualOrderTemplate Commit Id e6326536
Design Pattern Used **–** Template
Learning Outcome- Template Pattern, Refractoring(Changed long parameter list into objects) , Dependency Injection.
Other Contributors – (Deep)has removed OrderDetailsTemplate object.

OrderBuilder
Here we have used the builder pattern as we wanted to have the different values from different classes then we build all values together to build the order of user.
Design Pattern Used **– Builder Pattern**
Learning Outcome- Builder Pattern.
Other Contributors – none

OrderBuilderAbstract
Design Pattern Used **– Template**
Other Contributors – none.

OrderCreatedTemplate
Design Pattern Used **–**
Learning Outcome- Template Pattern, Refractoring(Changed long parameter list into objects) , Dependency Injection.
Other Contributors – none.

OrderDetailsTemplate
Design Pattern Used **–** Template
Learning Outcome- Template Pattern

Other Contributors –none.

OrderTypeTemplate
Other Contributors – none.

ShippedOrdersTemplate **Commit** Id 458813ee
Design Pattern Used **– Template.**
Learning Outcome- Template Pattern.
Other Contributors – (Deep)deleted the OrderId readonly int while refractoring.

UnpaidOrdersTemplate **Commit** Id 458813ee
Design Pattern Used **–** Template
Learning Outcome- Template Pattern.
Other Contributors – (Deep)Removed the IOrderBuilderResponse object.

IOrderBuilderResponse
Other Contributors – none.

IProductByStore
Other Contributors – none.

IStores
Other Contributors – none.

OrderBuilderResponse
Design Pattern Used **–**
Learning Outcome- Template Pattern, Refractoring(Changed long parameter list into objects) , Dependency Injection.
Other Contributors – (Deep)has moved one line from one method to another.

StoreListResponse **Commit Id** 8a330835
Design Pattern Used **– Template**
Learning Outcome- Template Pattern

Other Contributors – none.

Stores Commit Id 0c09ba9b
Other Contributors – Deep and Mohit added there one functionality.

IOrderDetailsDataLayer
Other Contributors none.

Fully Functional classes – Data layer

OrderDetailsDataLayer
Design Pattern Used
Other Contributors – (Deep)has removed unused spaces.
Store Procedure Used -
sp_FetchOrderDetailsByTypeAndStatus,sp_FetchOrderDetailsByType,sp_GetOrderDetails

ProductByStoreTemplate
Design Pattern Used **–**
Other Contributors – none.
Store Procedure Used -
sp_FetchProductByStore,sp_RemoveProduct,sp_AddProduct,sp_ModifyProduct

StoresTemplate
Design Pattern Used **–**
Other Contributors – none.
Store Procedure Used -
sp_FetchProductByStore,sp_RemoveProduct,sp_AddProduct,sp_ModifyProduct

CRUDTemplate Commit Id 7d846239
Design Pattern Used **–**
Other Contributors – none.

ProductByCategoryTemplate - Renamed now CategoryTemplate **Commit Id** 36fca25e

Design Pattern Used **–**
Other Contributors – Due to deletion of namespace and some sync issue file got renamed and showing deep as the contributor.
Store Procedure Used -
sp_FetchCategory, sp_RemoveCategory, sp_AddCategory, sp_ModifyCategory

Other classes – Presentation Layer

forgotPassword.aspx
api_FetchProductByStore.cs
Storebymanagement.cs
ViewOrderDetails.aspx

Note:- There are some minor functionalities class as well which are owned by other group members.

Total Test Cases:-

-111 (Grokart Data layer)

STORED_PROCEDURES:

Total stored procedures written by me: **16**