BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 319 - Object Oriented Software Engineering

S2T9-undefined

Class Diagram & Design Patterns -D5

10/12/2023

Group T9

Yassin Younis - 22101310

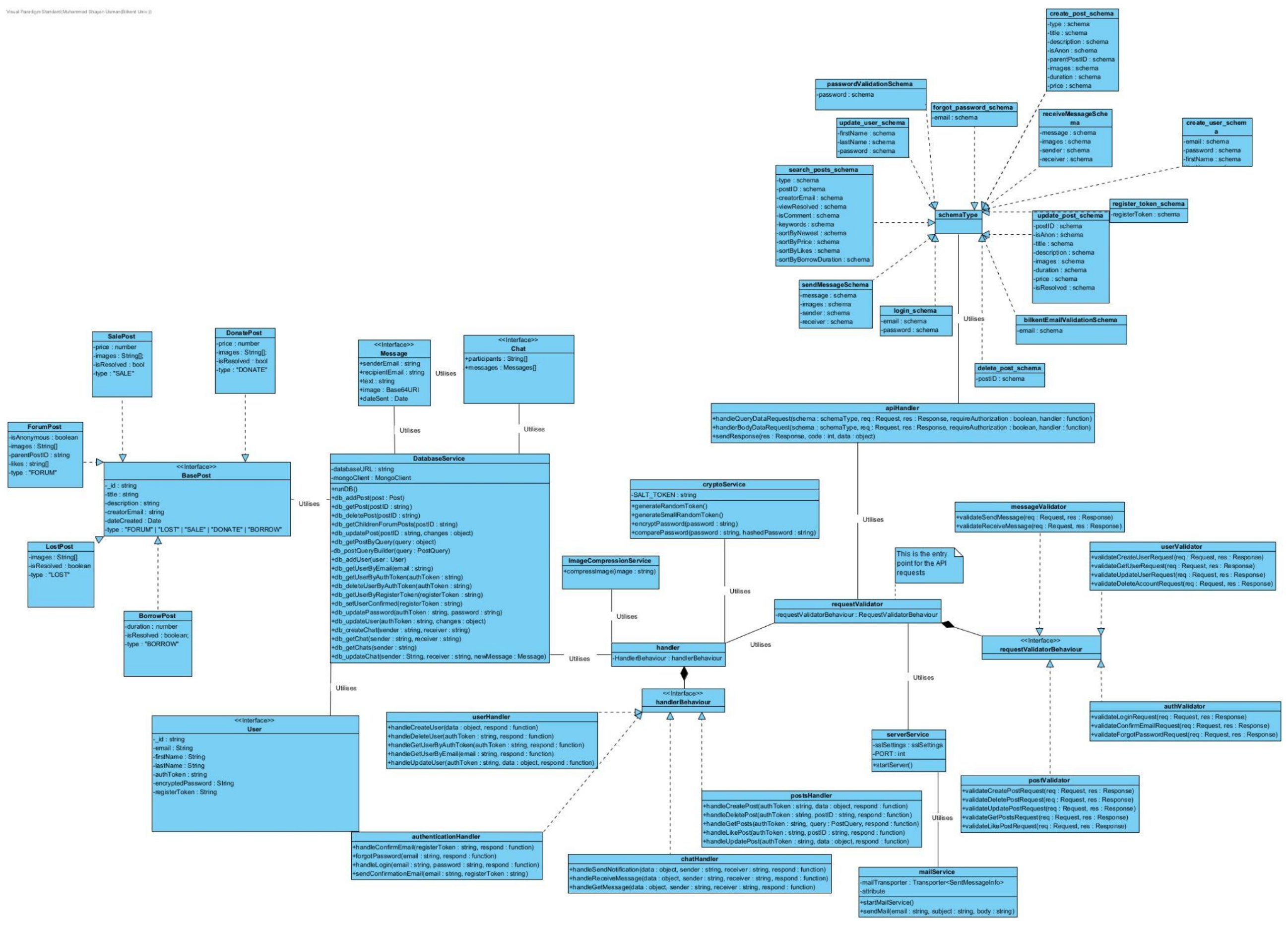
Muhammed Shayan Usman - 22101343

Zahaab Khawaja - 22101038

Table of Contents

1. Class Diagram(updated)

Full resolution image, along with .vpp file available in D5 folder in repo: link.


Visual Paradigm Standard(Muhammad Shayan Usman(Bilkent Univ.))
SalePost
-price : number
-images : String[];
-isResolved : bool
-type : "SALE"
DonatePost
-price : number
-images : String[];
-isResolved : bool
-type : "DONATE"
ForumPost
-isAnonymous : boolean
-images : String[]
-parentPostID : string
-likes : string[]
-type : "FORUM"
<<Interface>>
BasePost
-_id : string
-title : string
-description : string
-creatorEmail : string
-dateCreated : Date
-type : "FORUM" | "LOST" | "SALE" | "DONATE" | "BORROW"
LostPost
-images : String[]
-isResolved : boolean
-type : "LOST"
BorrowPost
-duration : number
-isResolved : boolean;
-type : "BORROW"
<<Interface>>
Message
+senderEmail : string
+recipientEmail : string
+text : string
+image : Base64URI
+dateSent : Date
<<Interface>>
Chat
+participants : String[]
+messages : Messages[]
DatabaseService
-databaseURL : string
-mongoClient : MongoClient
+runDB()
+db_addPost(post : Post)
+db_getPost(postID : string)
+db_deletePost(postID : string)
+db_getChildrenForumPosts(postID : string)
+db_updatePost(postID : string, changes : object)
+db_getPostByQuery(query : object)
-db_postQueryBuilder(query : PostQuery)
+db_addUser(user : User)
+db_getUserByEmail(email : string)
+db_getUserByAuthToken(authToken : string)
+db_deleteUserByAuthToken(authToken : string)
+db_getUserByRegisterToken(registerToken : string)
+db_setUserConfirmed(registerToken : string)
+db_updatePassword(authToken : string, password : string)
+db_updateUser(authToken : string, changes : object)
+db_createChat(sender : string, receiver : string)
+db_getChat(sender : string, receiver : string)
+db_getChats(sender : string)
+db_updateChat(sender : String, receiver : string, newMessage : Message)
<<Interface>>
User
-_id : string
-email : String
-firstName : String
-lastName : String
-authToken : string
-encryptedPassword : String
-registerToken : String
userHandler
+handleCreateUser(data : object, respond : function)
+handleDeleteUser(authToken : string, respond : function)
+handleGetUserByAuthToken(authToken : string, respond : function)
+handleGetUserByEmail(email : string, respond : function)
+handleUpdateUser(authToken : string, data : object, respond : function)
authenticationHandler
+handleConfirmEmail(registerToken : string, respond : function)
+forgotPassword(email : string, respond : function)
+handleLogin(email : string, password : string, respond : function)
+sendConfirmationEmail(email : string, registerToken : string)
passwordValidationSchema
-password : schema
forgot_password_schema
-email : schema
update_user_schema
-firstName : schema
-lastName : schema
-password : schema
search_posts_schema
-type : schema
-postID : schema
-creatorEmail : schema
-viewResolved : schema
-isComment : schema
-keywords : schema
-sortByNewest : schema
-sortByPrice : schema
-sortByLikes : schema
-sortByBorrowDuration : schema
sendMessageSchema
-message : schema
-images : schema
-sender : schema
-receiver : schema
login_schema
-email : schema
-password : schema
schemaType
create_post_schema
-type : schema
-title : schema
-description : schema
-isAnon : schema
-parentPostID : schema
-images : schema
-duration : schema
-price : schema
receiveMessageSchema
-message : schema
-images : schema
-sender : schema
-receiver : schema
create_user_schema
-email : schema
-password : schema
-firstName : schema
register_token_schema
-registerToken : schema
update_post_schema
-postID : schema
-isAnon : schema
-title : schema
-description : schema
-images : schema
-duration : schema
-price : schema
-isResolved : schema
bilkentEmailValidationSchema
-email : schema
delete_post_schema
-postID : schema
apiHandler
+handleQueryDataRequest(schema : schemaType, req : Request, res : Response, requireAuthorization : boolean, handler : function)
+handlerBodyDataRequest(schema : schemaType, req : Request, res : Response, requireAuthorization : boolean, handler : function)
+sendResponse(res : Response, code : int, data : object)
cryptoService
-SALT_TOKEN : string
+generateRandomToken()
+generateSmallRandomToken()
+encryptPassword(password : string)
+comparePassword(password : string, hashedPassword : string)
ImageCompressionService
+compressImage(image : string)
handler
-HandlerBehaviour : handlerBehaviour
<<Interface>>
handlerBehaviour
requestValidator
-requestValidatorBehaviour : RequestValidatorBehaviour
This is the entry point for the API requests
messageValidator
+validateSendMessage(req : Request, res : Response)
+validateReceiveMessage(req : Request, res : Response)
userValidator
+validateCreateUserRequest(req : Request, res : Response)
+validateGetUserRequest(req : Request, res : Response)
+validateUpdateUserRequest(req : Request, res : Response)
+validateDeleteAccountRequest(req : Request, res : Response)
<<Interface>>
requestValidatorBehaviour
authValidator
+validateLoginRequest(req : Request, res : Response)
+validateConfirmEmailRequest(req : Request, res : Response)
+validateForgotPasswordRequest(req : Request, res : Response)
serverService
-sslSettings : sslSettings
-PORT : int
+startServer()
postValidator
+validateCreatePostRequest(req : Request, res : Response)
+validateDeletePostRequest(req : Request, res : Response)
+validateUpdatePostRequest(req : Request, res : Response)
+validateGetPostsRequest(req : Request, res : Response)
+validateLikePostRequest(req : Request, res : Response)
postsHandler
+handleCreatePost(authToken : string, data : object, respond : function)
+handleDeletePost(authToken : string, postID : string, respond : function)
+handleGetPosts(authToken : string, query : PostQuery, respond : function)
+handleLikePost(authToken : string, postID : string, respond : function)
+handleUpdatePost(authToken : string, data : object, respond : function)
chatHandler
+handleSendNotification(data : object, sender : string, receiver : string, respond : function)
+handleReceiveMessage(data : object, sender : string, receiver : string, respond : function)
+handleGetMessage(data : object, sender : string, receiver : string, respond : function)
mailService
-mailTransporter : Transporter<SentMessageInfo>
-attribute
+startMailService()
+sendMail(email : string, subject : string, body : string)
Utilises

2. Design Patterns

Design patterns are often used to solve common organisational problems in code during development. They represent best practices, developed and refined over time by experienced developers. There are 3 main categories:

1. Creational
2. Structural
3. Behavioural

While developing our app, we realised the importance of design patterns and decided to apply them while writing the code. They were particularly useful in constructing the backend to ensure minimal redundancies and maximum flexibility for operations. Following are two of the design patterns we used in our app, and how we implemented them.

2.1. Strategy Pattern (Behavioural)

The Strategy Design pattern defines a group of algorithms, encapsulating each one with its own functionality, and makes them interchangeable, thus making them suited for different types of operations. It lets the algorithm vary independently from clients that use it.

We used this design pattern to implement the Handler Behaviour interface. When coming up with the design of the app, we quickly realised that the handler for each operation i.e. posts, chats, users and authentication, will all have a similar design but different implementation and will only vary at certain points of execution. Hence, we decided to group them together in a Strategy design pattern. This is done by encapsulating each of the post types, namely postsHandler, chatHandler, userHandler, authenticationHandler.

After choosing the HandlerBehaviour, we proceed to implement the selected handler, by instantiating an interface for the specific behaviours of that handler. Below is the structural diagram for the Design pattern for post type.

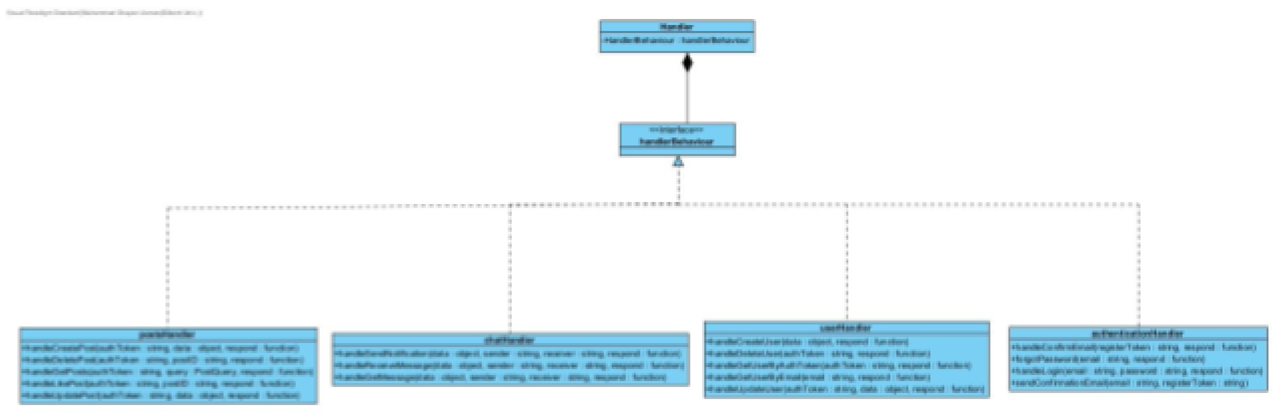


Fig 1: Strategy Design structure diagram for posts

2.2. Facade Pattern (Structural)

The Facade Design Pattern provides a simplified interface to a complex subsystem. This allows the program to abstract away the unnecessarily complicated parts which makes it easier and safer to use. In our app, we realised the need to abstract away complexities into further subclasses. In React, it is good practice to make use of predefined components that can serve as facades by encapsulating complex logic of choosing and instantiating sub-components based on the parameters received by the parent component.

For example, one of the facades used is the CreateForumPost which uses multiple sub-components, but only

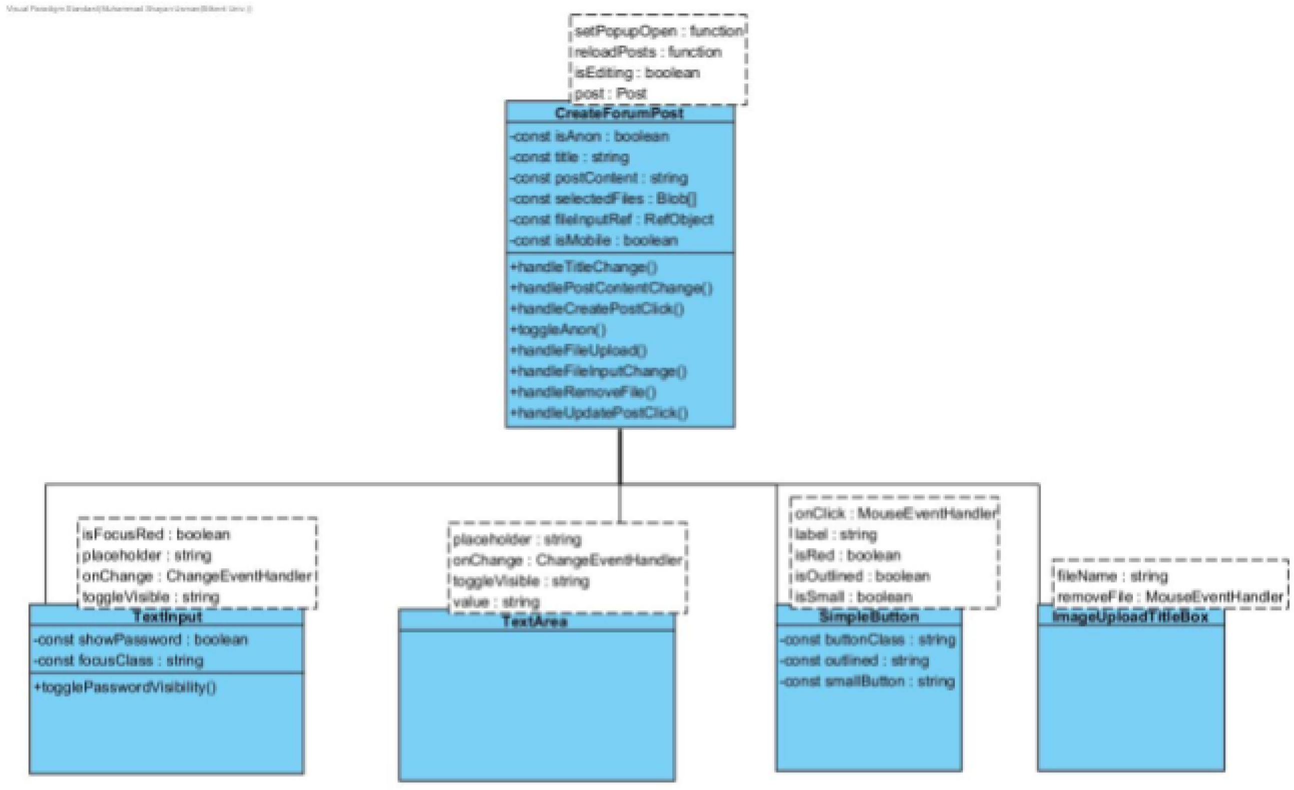


Fig 2: CreateForumPost acts as the facade that abstracts away the complexities of the sub components