Tugas Study Case Chapter 1
Nama Anggota:

- Helsa Nesta Dhaifullah
- Naily Khairiya

1. **Find your website (system) for your group!**
   Our group number is 3, according to the distribution website, we get pijarsekolah website on https://pijarsekolah.id/

2. **Create 5 main functional requirements and 5 main non-functional requirements!**
   Functional requirements
   a. User registration.
   b. Customer service / chat helper.
   c. Payment subscription.
   d. Content management system.
   e. Task management system.

   Non-Functional requirements
   a. The website should be responsive and have a user-friendly interface.
   b. The website should load quickly and provide a seamless user experience, even under heavy traffic loads.
   c. The website should be able to handle increasing amounts of traffic without compromising performance.
   d. The website must have robust security measures in place to protect user data and transactions.
   e. The website should be accessible to users with disabilities, such as using screen readers.

3. **Create User Story based on those Functional Requirement and Non-Functional Requirement!**
   Functional requirements

User registration	***As a new user***, ***I want to*** create an account on https://pijarsekolah.id/ ***so that I can*** access the educational materials and features available on the website. Acceptance Criteria: 1. The registration process is simple and quick, with no unnecessary steps or confusing language. 2. The user is asked to provide their name, school name, email address, telephone number, and a password to create their account. 3. The password field is secure, with minimum password length requirements. 4. the user receives a confirmation email to verify their account. 5. Once the account is verified, the user can log in and access the website's features and educational materials.

Customer service / chat helper	***As a user*** of https://pijarsekolah.id/, ***I want to*** be able to get help and support quickly and easily when I need it, ***so that*** I can troubleshoot any issues I'm experiencing with the website or the educational materials. Acceptance criteria: 1. The customer service or chat helper feature is easily accessible from any page on the website, with a prominent button or icon. 2. The feature is available 24/7 and provides quick responses.
Payment subscription	***As a user*** of https://pijarsekolah.id/, ***I want to*** be able to subscribe to pijarsekolah content and features on the website, ***so that*** I can access more comprehensive features in pijarsekolah. Acceptance criteria: 1. The website provides clear and easy-to-understand information about the available subscription plans and pricing. 2. The user can easily select and sign up for the subscription plan that best fits their needs and budget. 3. The payment process is secure and reliable.
Content management system (cms)	***As an admin user (teachers)*** of https://pijarsekolah.id/, ***I want to*** be able to manage and publish educational content on the website, ***so that*** I can provide up-to-date and relevant materials to the users (students). Acceptance criteria: 1. The cms provides a clear and intuitive interface for creating, editing, and deleting different types of educational content. 2. The cms includes tools for formatting and styling content, adding multimedia, and organizing content into categories.
Task management system	***As a student or teacher*** using https://pijarsekolah.id/, ***I want to*** be able to manage and assign or organize tasks and assignments, ***so that students*** can keep track of their progress and submit their tasks and assignments under the deadline. ***Then the teacher*** can provide them with clear expectations and track their progress more effectively.

	Acceptance criteria: 1. The task management system provides a clear and user-friendly interface for creating, assigning, and tracking tasks and assignments. 2. The task management system includes options for setting deadlines, priorities, requirements, and feedback, and provides notifications or reminders to help users stay on top of their workload.

Non-Functional requirements

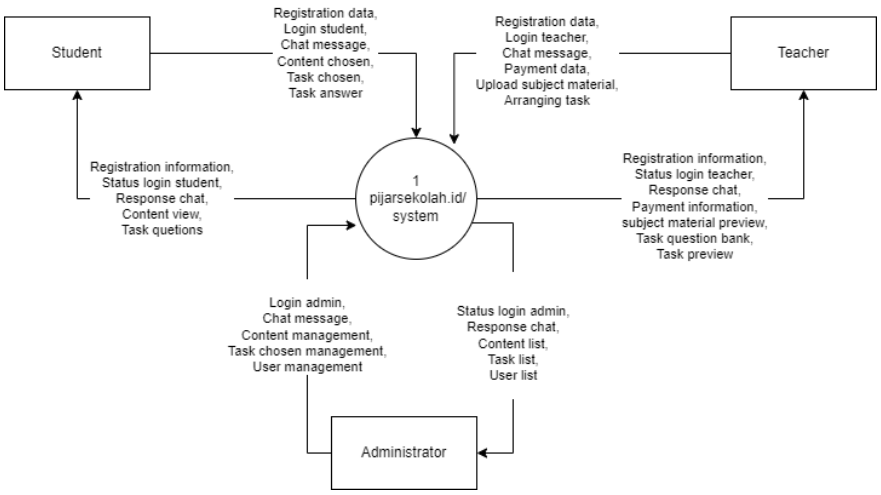
The website should be responsive and have a user-friendly interface	***As a user*** of https://pijarsekolah.id/, ***I want*** the website to be responsive and have a user-friendly interface, ***so that*** I can access and navigate the website easily from any device and location. Acceptance criteria: 1. The website should be tested on multiple devices and screen sizes to ensure that it is responsive and adjusts the layout and content accordingly.
The website should load quickly and provide a seamless user experience, even under heavy traffic loads	***As a user*** of https://pijarsekolah.id/, ***I want*** the website to load quickly and provide a seamless user experience, even under heavy traffic loads, ***so that*** I can access and use the website without delays or interruptions. Acceptance criteria: 1. The website should be tested for performance using load testing tools. 2. The website should be monitored for uptime and availability using automated tools or services.
The website should be able to handle increasing amounts of traffic without compromising performance	***As a user*** of https://pijarsekolah.id/, ***I want*** the website to be able to handle increasing amounts of traffic without compromising performance, ***so that*** I can access and use the website without delays or interruptions even as the user base grows. Acceptance criteria: 1. The website should be tested for scalability using load testing tools. 2. The website should be monitored for resource utilization and performance using automated tools or services.
The website must have robust security measures in place to protect user data and transactions.	***As a user*** of https://pijarsekolah.id/, ***I want*** the website to have robust security measures in place to protect my personal information and transactions, ***so that*** I can use the website with confidence and trust.

	Acceptance criteria: 1. The website should use industry-standard secure encryption protocols (SSL, HTTPS, or TLS) to prevent unauthorized access or interception of sensitive information. 2. The website should undergo regular security audits and vulnerability assessments by qualified third-party experts to ensure that all security measures are up-to-date and effective in preventing common threats and attacks.
The website should be accessible to users with disabilities, such as using screen readers	***As a user*** of https://pijarsekolah.id/ who may have a disability or use assistive technologies such as screen readers, ***I want*** the website to be fully accessible and compatible with these tools, ***so that*** I can access all of the website's content and functionality without any difficulties. Acceptance criteria: 1. All images, videos, and other non-text content should have alternative text descriptions that accurately convey their meaning and purpose. 2. The website should use clear and consistent labeling and formatting for all headings, links, buttons, and other interactive elements, and should avoid using color or visual cues as the only means of conveying important information or functionality.

4. Create DFD, Usecase, and Activity Diagram related to the Requirement!
Diagram : https://app.diagrams.net/#G1CIGuj3FVcNFzZEGD21DCV9Hl9FPjz5bz

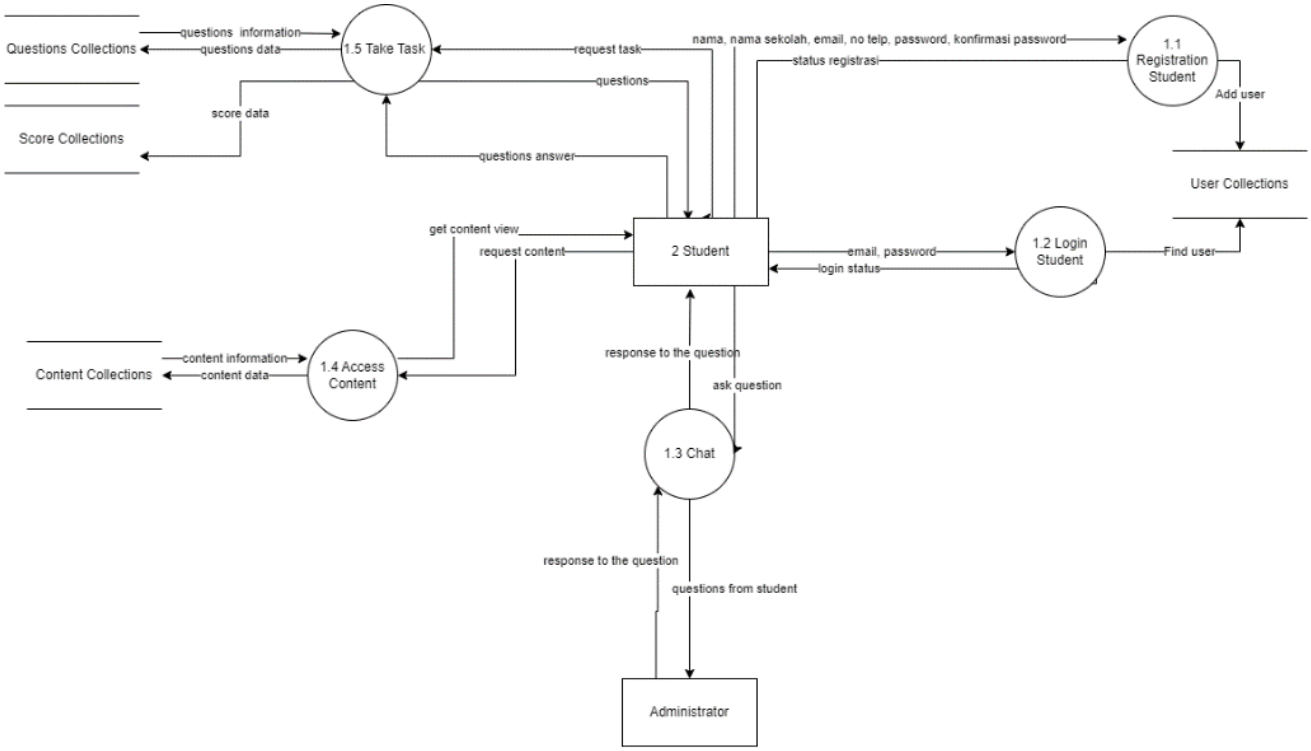
A. DFD

a. Context Diagram

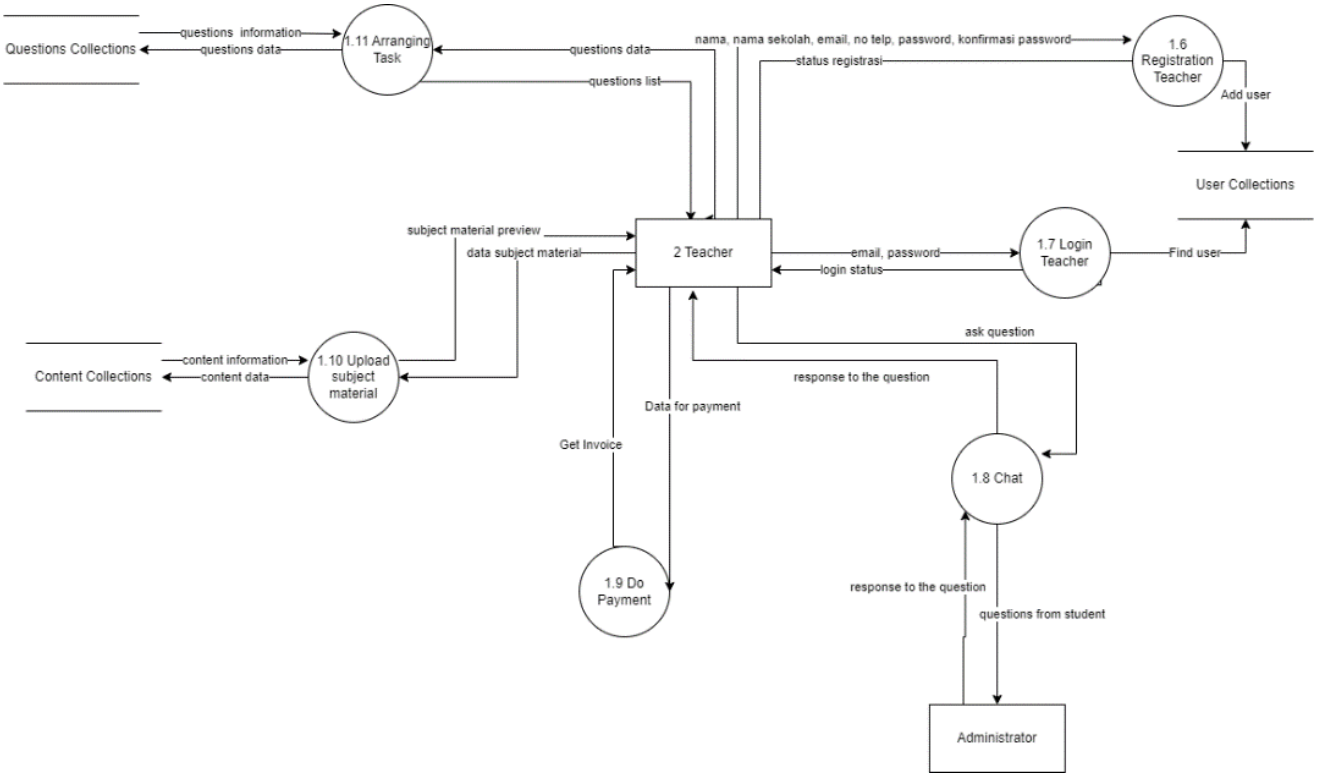


b. DFD Level 1

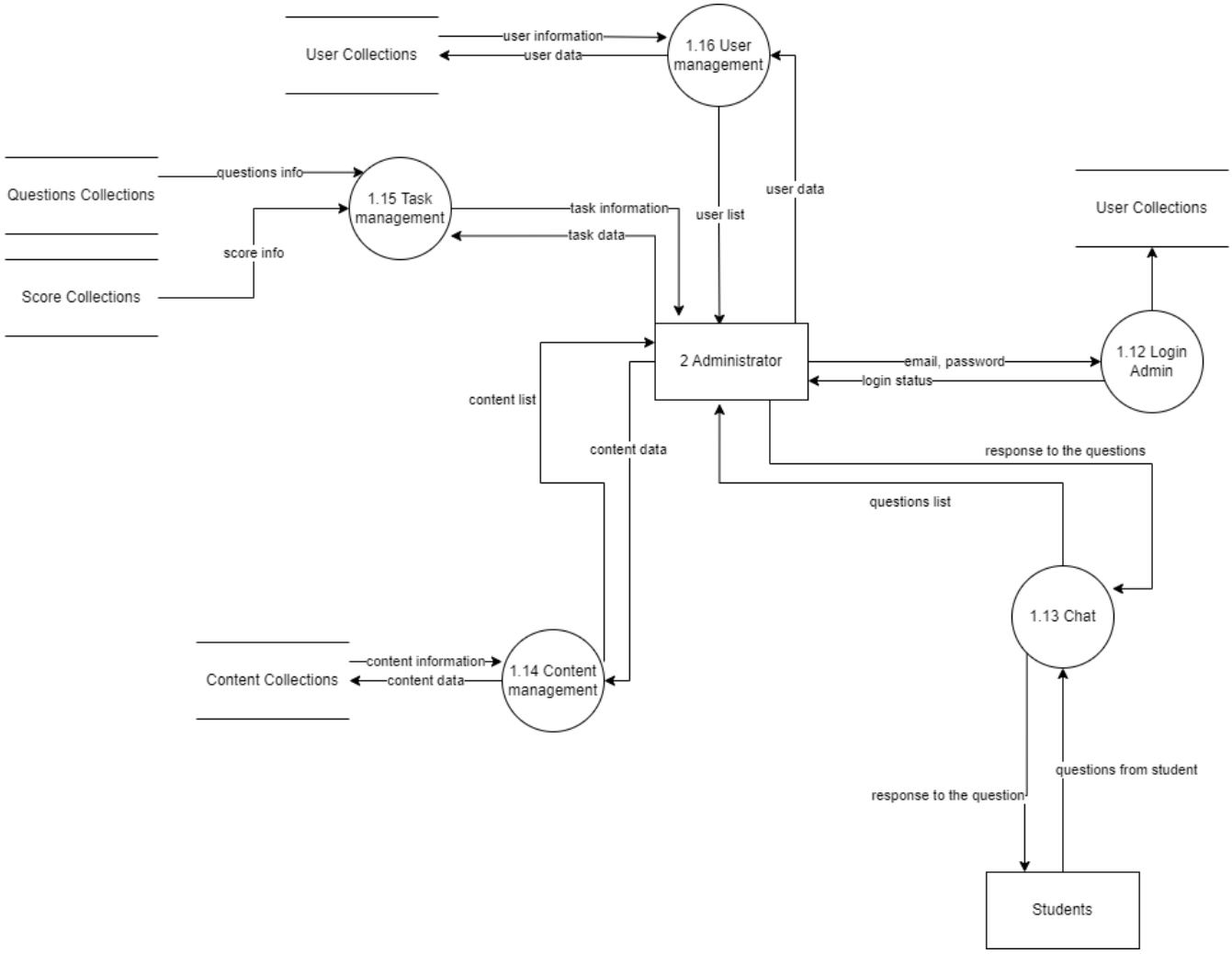
i. Student Process



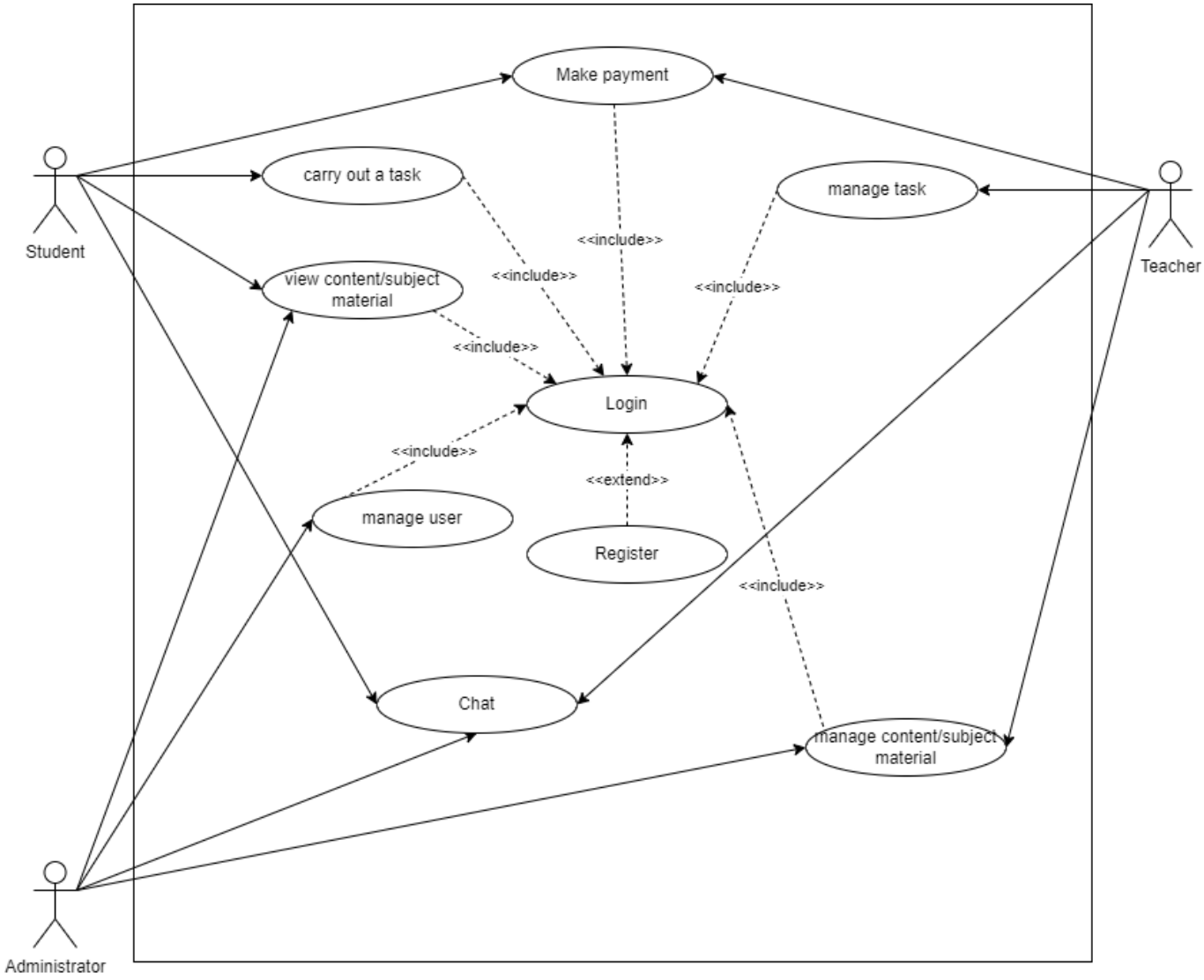
ii. Teacher Process



iii. Administrator Process

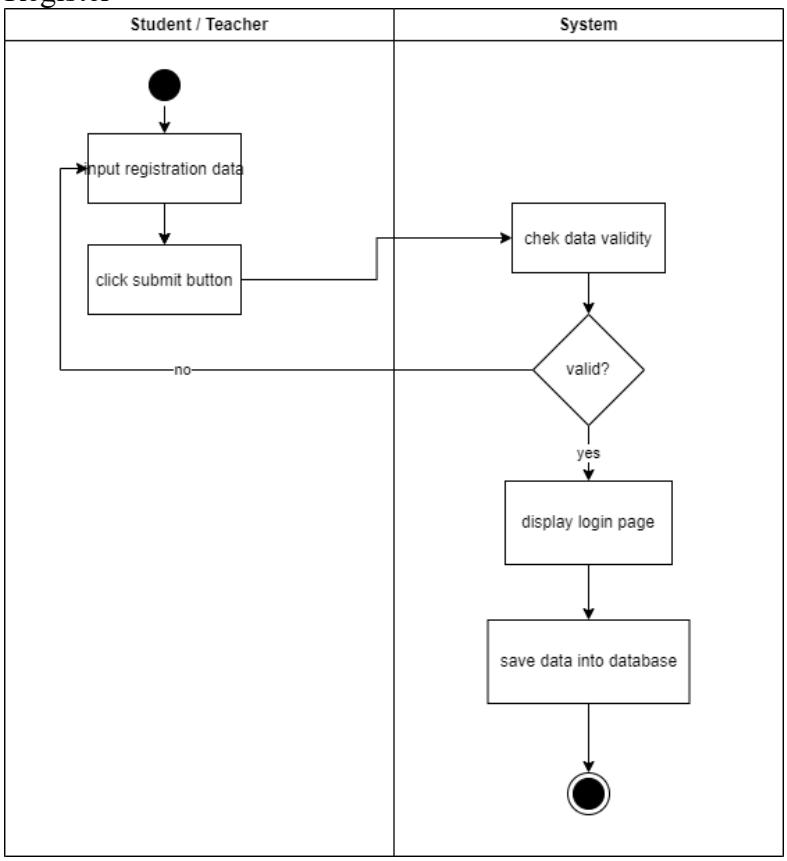


B. Usecase

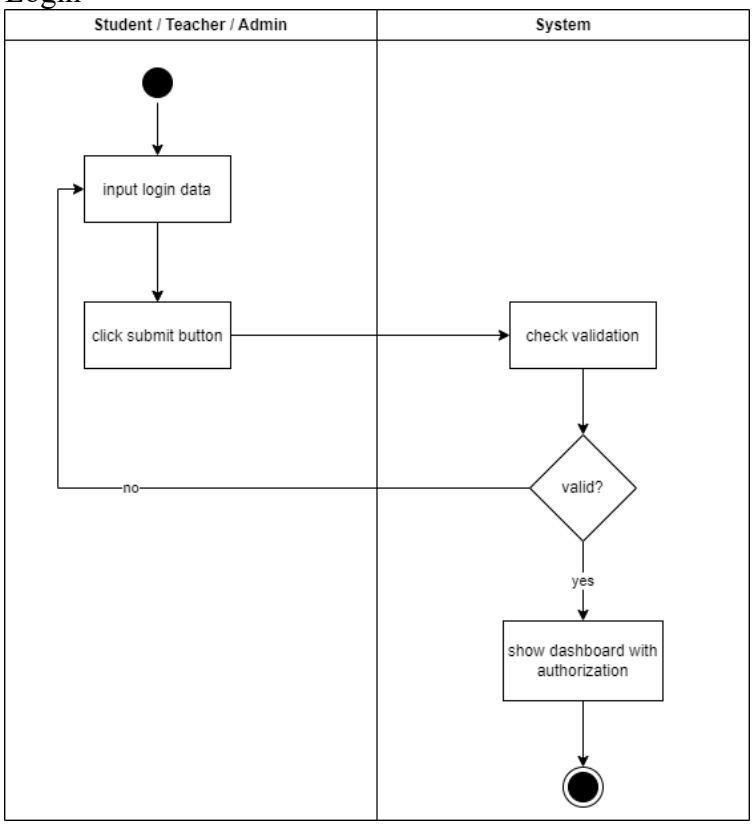

Make payment
carry out a task
manage task
view content/subject material
Login
manage user
Register
Chat
manage content/subject material
Student
Teacher
Administrator
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<extend>>
<<include>>

C. Activity Diagram

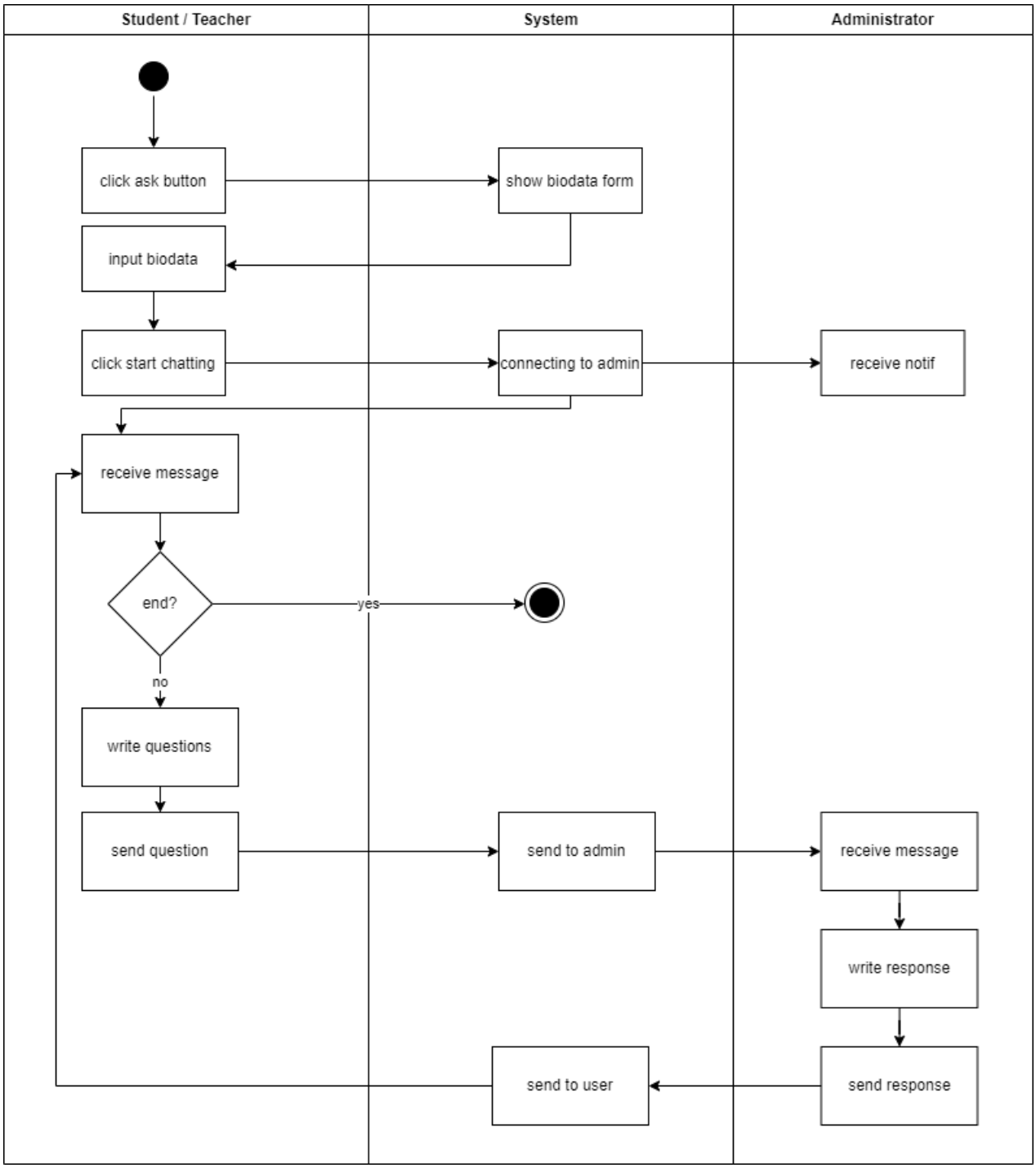
a. Register



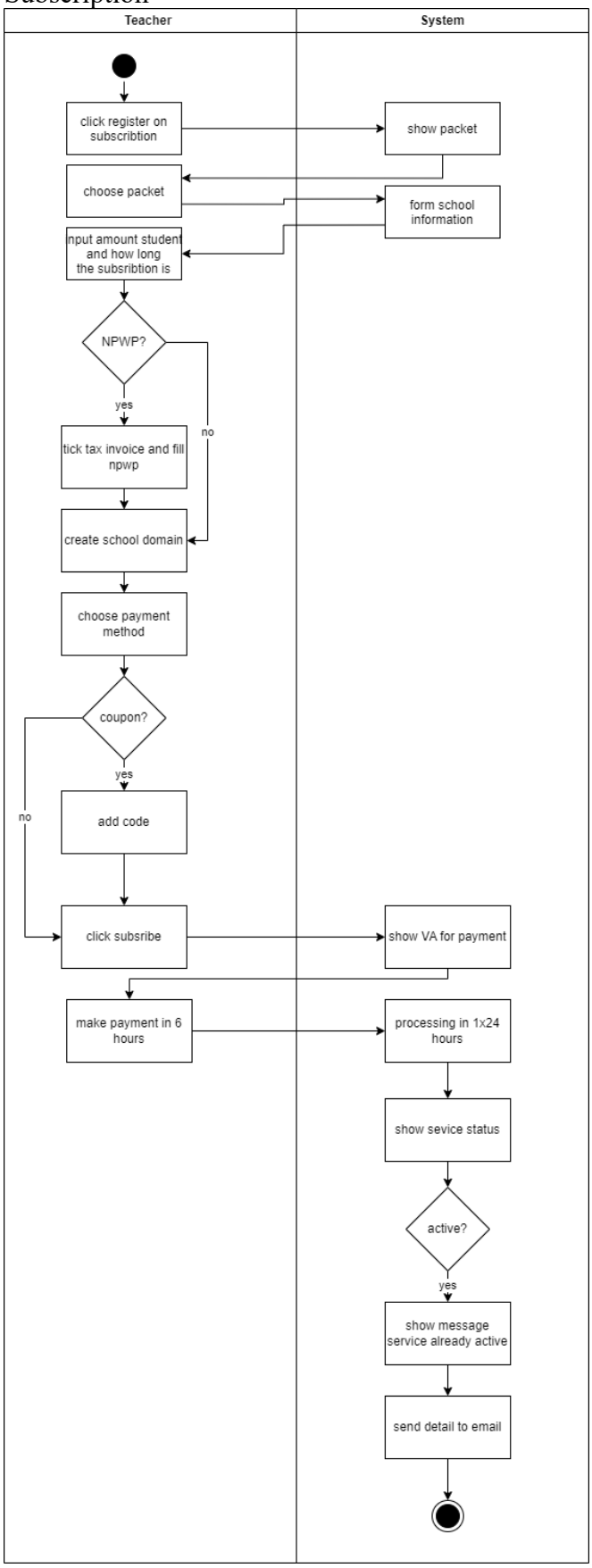
b. Login



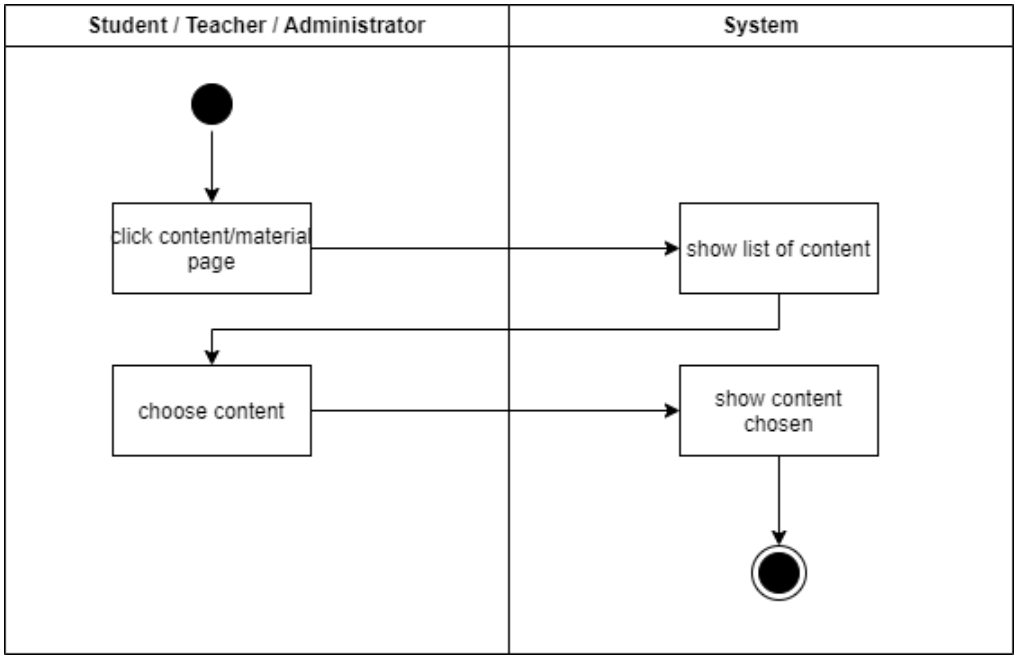
c. Chat


Student / Teacher
System
Administrator
click ask button
show biodata form
input biodata
click start chatting
connecting to admin
receive notif
receive message
end?
yes
no
write questions
send question
send to admin
receive message
write response
send response
send to user

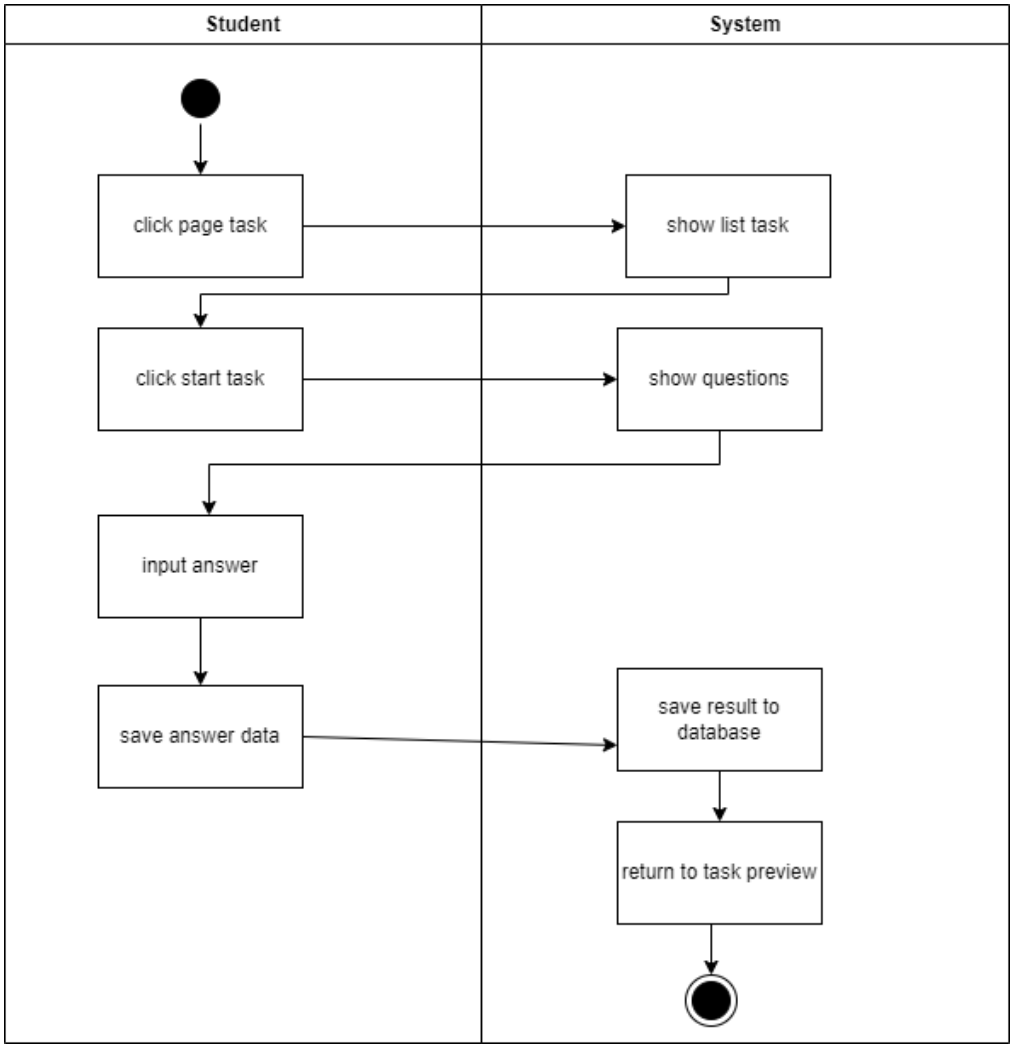
d. Subscription



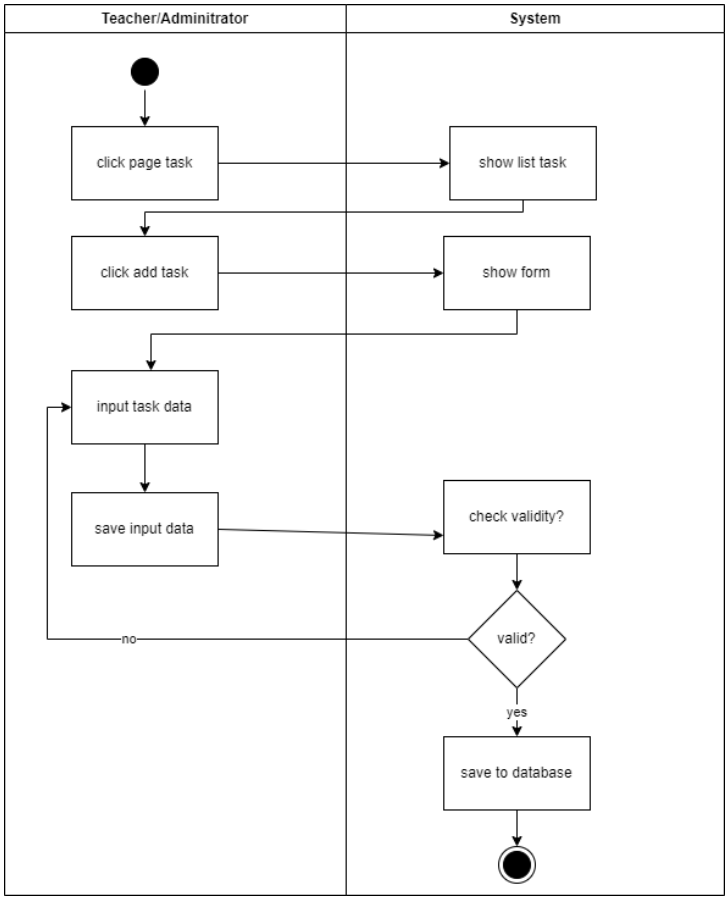
e. Access Content



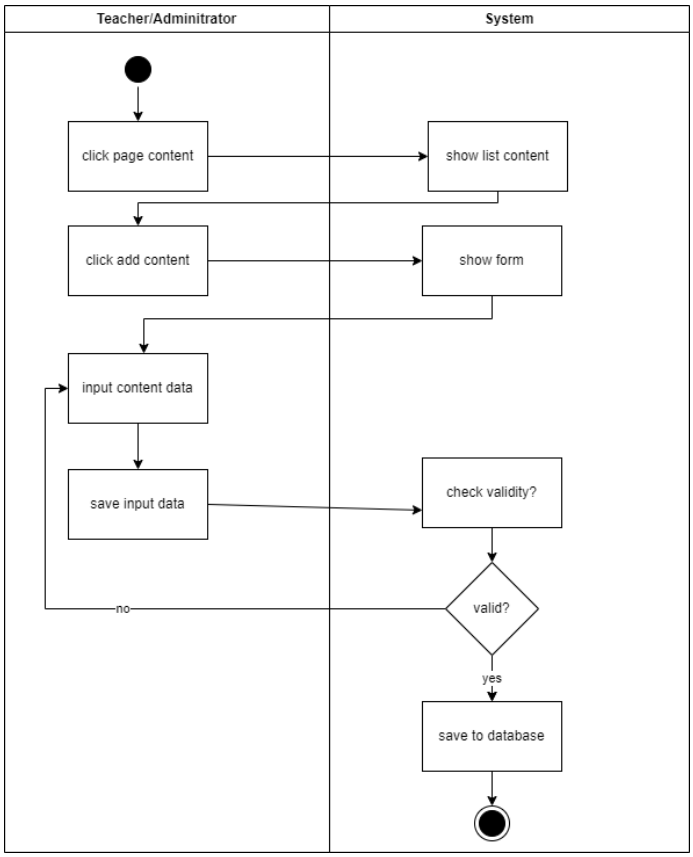
f. Take Task



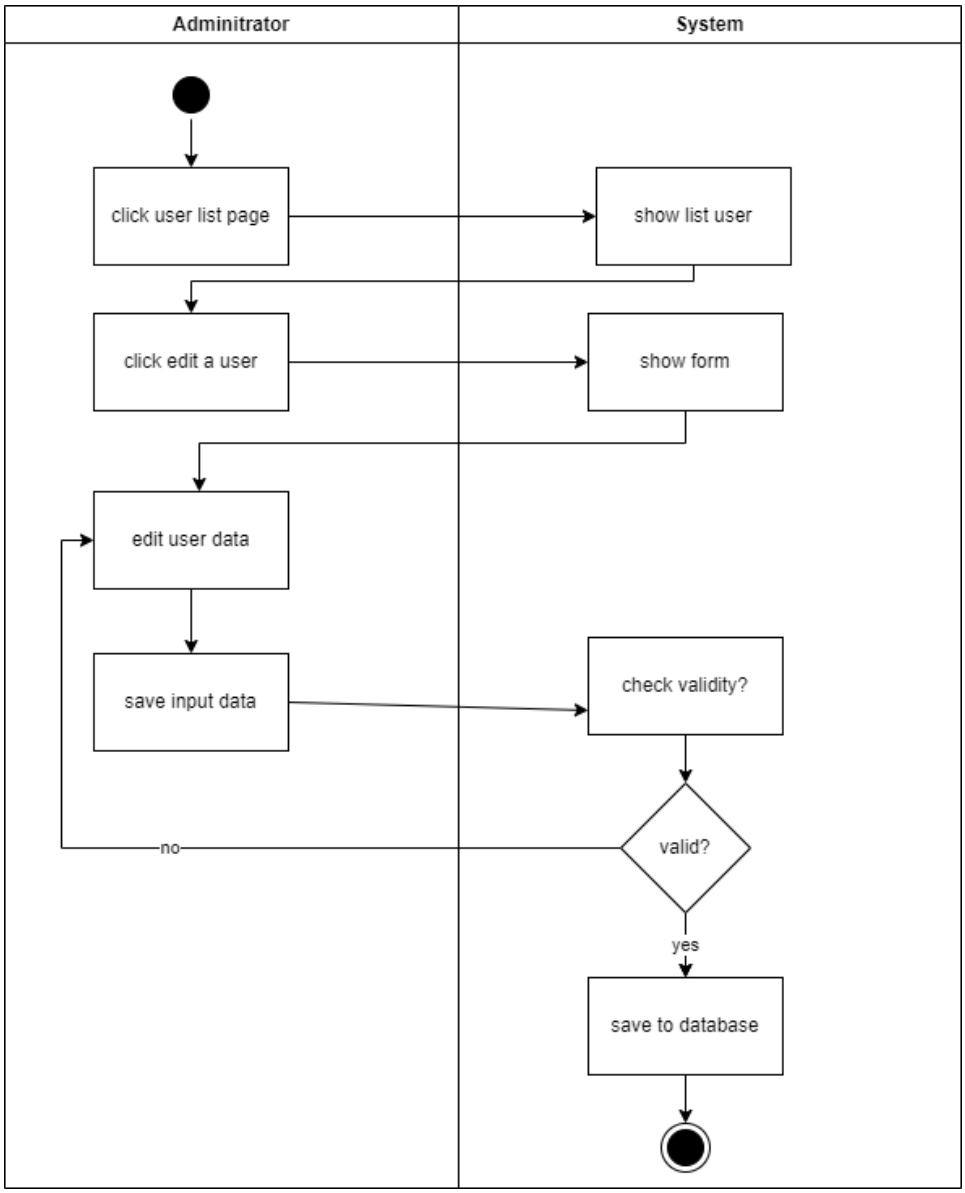
g. Add Task



h. Add Content



i. Edit User



5. **Create API to fulfill all of those User Stories!**
   This link to our **repository** : https://github.com/nailykhry/TEFA-STUDYCASE-1.git

   Folder Structure :

```
📦 TEFA-STUDYCASE-1
├ 📂 controllers
│  ├ 📜 auth.go
│  ├ 📜 content.go
│  ├ 📜 task.go
│  ├ 📜 userChat.go
│  └ 📜 userSub.go
├ 📂 database
│  └ 📜 db.go
```

```
├ 📂 middleware
│ └ 📄 Authentication.go
├ 📂 models
│ ├ 📄 content.go
│ ├ 📄 task.go
│ ├ 📄 user.go
│ ├ 📄 userChat.go
│ └ 📄 userSub.go
├ 📂 repository
│ ├ 📄 contens.go
│ ├ 📄 task.go
│ ├ 📄 userChat.go
│ ├ 📄 users.go
│ └ 📄 userSub.go
├ 📂 routes
│ ├ 📄 auth.go
│ ├ 📄 content.go
│ ├ 📄 task.go
│ ├ 📄 userChat.go
│ └ 📄 userSub.go
├ 📂 security
│ ├ 📄 password.go
│ ├ 📄 token.go
├ 📂 util
│ ├ 📄 errors.go
│ └ 📄 util.go
├ 📄 .env
├ 📄 .gitignore
├ 📄 go.mod
├ 📄 go.sum
└ 📄 main.go
```

Directory and File Explanation :

📦 TEFA-STUDYCASE-1	Root
├ 📂 controllers	This directory contains the implementation of the application's controllers, which handle requests from the client and return appropriate responses.
│ ├ 📄 auth.go	It provides various endpoints for sign-up, sign-in, getting, updating and deleting users. The code follows SOLID principles, especially the Single Responsibility Principle, as each module/class has only one responsibility, and the Open-Closed Principle, as it's open for extension but closed for

	modification. The implementation of these principles can be seen in the structure of the code and the way different responsibilities are separated into modules.
┃ ┣ content.go	This code defines a ContentsController interface and a contentsController struct that implements it. The contentsController struct has methods to handle HTTP requests related to content management, such as uploading, retrieving, updating, and deleting content. The code demonstrates the Single Responsibility Principle (SRP) and the Interface Segregation Principle (ISP) of the SOLID principles. The ContentsController interface defines the contract for content management, and the contentsController struct implements the interface. By doing this, the code separates concerns and keeps each function focused on one task. Additionally, the interface defines only the necessary methods, allowing the code to follow the ISP and avoid unnecessary coupling between different parts of the system.
┃ ┣ task.go	Controller for task. Nearly same as the contentController.
┃ ┣ userChat.go	Controller for chat. Nearly same as the contentController.
┃ ┗ userSub.go	Controller for subscription. Nearly same as the contentController.
┣ database	This directory contains the implementation of the database for the application.
┃ ┗ db.go	This code is for connecting to a MongoDB database. It defines an interface to create new connection and close DB and implement it.
┣ middleware	This directory for middleware that is a http.Handler that wraps another http.Handler to do some pre- and/or post-processing of the request.
┃ ┗ Authentication.go	This code defines a middleware package that includes an AuthRequired function to handle JWT authentication. The AuthRequired function uses Fiber's JWT middleware to verify the user's authorization token and returns a JSON error response if the token is invalid.
┣ models	This directory contains the implementation of the application's data models.
┃ ┣ content.go ┃ ┣ task.go ┃ ┣ user.go ┃ ┣ userChat.go ┃ ┗ userSub.go	Each file have struct for data model.
┣ repository	This directory contains the implementation of the application's repository layer, which handles data access.

┃ ┣ contens.go ┃ ┣ task.go ┃ ┣ userChat.go ┃ ┣ users.go ┃ ┗ userSub.go	File in repository defines a contents repository interface and implementation, which allows for CRUD operations on contents. It utilizes the Dependency Inversion Principle (DIP) by depending on an abstraction, the database.Connection interface, rather than a concrete implementation, allowing for more flexibility in the future. It also follows the Single Responsibility Principle (SRP) by separating concerns between the repository and the database connection, and the Interface Segregation Principle (ISP) by providing only the necessary methods in the ContentsRepository interface.
┣ routes	This directory contains the implementation of the application's routes.
┃ ┣ auth.go	The code in routes defines a set of routes for each model data. The interface defines a method to install routes in a Fiber app. The struct implements the routes interface and defines specific routes and controller. The NewModelRoutes function creates an instance of Routes with the given Controller.
┃ ┣ content.go	
┃ ┣ task.go	
┃ ┣ userChat.go	
┃ ┗ userSub.go	
┣ security	This directory contains the implementation of security-related functionality for the application. Here, this folder handle password and token from jwt.
┃ ┣ password.go	Use to encrypt password.
┃ ┣ token.go	Handle all related token like generate new token, validate signed methos, and parse token.
┣ util	This directory contains utility functions for the application.
┃ ┣ errors.go	This code defines several error variables with corresponding error messages. These errors are used in different parts of the application to handle specific error scenarios.
┃ ┗ util.go	Contains utility functions for the application.
┣ .env	Configurations file
┣ .gitignore	File that must be ignore by VCS.
┣ go.mod	Golang Modules
┣ go.sum	Keeps track of the checksums for all the dependencies used in a Go project.
┗ main.go	It imports necessary packages, sets up a connection to a database, creates a new instance of the Fiber web framework, adds middleware, defines routes for different endpoints, creates instances of controllers and repositories for different models, and starts the application to listen on port 8080.

6. Implement SOLID for each API and explain the Big O calculation!
   - Database/db

``go
import (
"fmt" //O(1)
"log" //O(1)
"os" //O(1)
"strconv" //O(1)
 	
"gopkg.in/mgo.v2" //O(1)
)
 	
type Connection interface {
Close() //O(1)
DB() *mgo.Database //O(1)
}
 	
type conn struct {
session *mgo.Session //O(1)
}
``
``go
func NewConnection() Connection {
var c conn //O(1)
var err error //O(1)
url := getURL() //O(1)
c.session, err = mgo.Dial(url) //O(1)
if err != nil { //O(1)
log.Panicln(err.Error()) //O(1)
}
return &c //O(1)
}
``
``go
func (c *conn) Close() {
c.session.Close() //O(1)
}
``
``go
func (c *conn) DB() *mgo.Database {
return c.session.DB(os.Getenv("DATABASE_NAME")) //O(1)
}
``
``go
func getURL() string {
port, err := strconv.Atoi(os.Getenv("DATABASE_PORT")) //O(1)
if err != nil { //O(1)
log.Println("error on load db port from env:", err.Error()) //O(1)
port = 27017 //O(1)
}
return fmt.Sprintf("mongodb://%s:%d/%s", //O(1)
os.Getenv("DATABASE_HOST"),
port,
os.Getenv("DATABASE_NAME"))
}
``
The complexity of the functions in this file is very minimal, as they mostly consist of basic operations to connect to the MongoDB database. There are no complex operations or algorithms in this code, so the complexity of this file can be considered as O(1) or constant.
		Import, create interface, and create struct only O(1) so the complexity is contants
		
		Function NewConnection has contants complexity because its only call database. And other simple operation that only need O(1).
		
		Function Close is for close connection to database and it only need O(1)
		
		DB for return connected databased need O(1)
		
		GetURL function read environment to get url for connecting to database. This simple operation need O(1)

   - Models

``go
import (
"time" //O(1)
 	
"gopkg.in/mgo.v2/bson" //O(1)
)
 	
type User struct {
Id bson.ObjectId `json:"id" bson:"_id"` //O(1)
Email string `json:"email" bson:"email"` //O(1)
Nama string `json:"nama" bson:"nama"` //O(1)
Nama_Sekolah string `json:"nama_sekolah" bson:"nama_sekolah"` //O(1)
Telp string `json:"telp" bson:"telp"` //O(1)
Role string `json:"role" bson:"role"` //O(1)
Password string `json:"password" bson:"password"` //O(1)
CreatedAt time.Time `json:"created_at" bson:"created_at"` //O(1)
UpdatedAt time.Time `json:"updated_at" bson:"updated_at"` //O(1)
}
``
All models only need O(1). Because its file only contains import package and creating model struct. This is example for user models. It shows all only need O(1).		

- Security

```go
package security

import "golang.org/x/crypto/bcrypt" //O(1)

func EncryptPassword(password string) (string, error) {
    hashed, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.DefaultCost) //O(n)
    if err != nil {                                                                   //O(1)
        return "", err //O(1)
    }
    return string(hashed), nil //O(1)
}

func VerifyPassword(hashed, password string) error {
    return bcrypt.CompareHashAndPassword([]byte(hashed), []byte(password)) //O(n)
}
```

The complexity of the EncryptPassword and VerifyPassword functions depends on the complexity of the encryption and decryption algorithm used by the golang.org/x/crypto/bcrypt package. The bcrypt algorithm itself uses random iterations and loops in the password hashing process.
In this case, the complexity of the EncryptPassword and VerifyPassword functions is O(n), where n is the length of the password provided. This is because the bcrypt algorithm will encrypt the password in a certain number of iterations.

```go
func NewToken(userId string, userRole string) (string, error) {
    claims := jwt.StandardClaims{ //O(1)
        Id:        userId,
        Subject:   userRole,
        Issuer:    userId,
        IssuedAt:  time.Now().Unix(),
        ExpiresAt: time.Now().Add(time.Minute * 30).Unix(),
    }
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims) //O(n)
    return token.SignedString(JwtSecretKey)                    //O(1)
}

func validateSignedMethod(token *jwt.Token) (interface{}, error) {
    if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok { //O(1)
        return nil, fmt.Errorf("unexpected signing method: %v", token.Header["alg"]) //O(1)
    }
    return JwtSecretKey, nil //O(1)
}

func ParseToken(tokenString string) (*jwt.StandardClaims, error) {
    claims := new(jwt.StandardClaims)                                          //O(1)
    token, err := jwt.ParseWithClaims(tokenString, claims, validateSignedMethod) //O(n)
    if err != nil {                                                            //O(1)
        return nil, err //O(1)
    }
    var ok bool                                  //O(1)
    claims, ok = token.Claims.(*jwt.StandardClaims) //O(1)
    if !ok || !token.Valid {                     //O(1)
        return nil, util.ErrInvalidAuthToken //O(1)
    }
    return claims, nil //O(1)
}
```

The **NewToken** function has a complexity of O(n) because the jwt.NewWithClaims function has a time complexity of O(n), where n is the length of the claims data.

The **validateSignedMethod** function has a complexity of O(1) because it performs only basic operations to validate the signing method used in the token.

The **ParseToken** function has a complexity of O(n) because the jwt.ParseWithClaims function has a time complexity of O(n), where n is the length of the claims data in the token. The rest of the function performs basic operations and has a complexity of O(1).

- Utils

```go
type JError struct {
    Error string `json:"error"` //O(1)
}

func NewJError(err error) JError {
    jerr := JError{"generic error"} //O(1)
    if err != nil {                 //O(1)
        jerr.Error = err.Error() //O(1)
    }
    return jerr //O(1)
}

func NormalizeEmail(email string) string {
    return strings.TrimSpace(strings.ToLower(email)) //O(1)
}
```

In file util its only contains simple operation so the complexity is O(1) or constant

```go
var (
    ErrInvalidEmail       = errors.New("invalid email")           //O(1)
    ErrEmailAlreadyExists = errors.New("email already exists")    //O(1)
    ErrEmptyPassword      = errors.New("password can't be empty") //O(1)
    ErrInvalidAuthToken   = errors.New("invalid auth-token")      //O(1)
    ErrInvalidCredentials = errors.New("invalid credentials")     //O(1)
    ErrUnauthorized       = errors.New("Unauthorized")            //O(1)
)
```

In file error it declare custom error and it also has constant complexity or O(1)

- Repository

```go
type ContentsRepository interface {
    UploadContent(content *models.Content) error            //O(1)
    Update(content *models.Content) error                   //O(1)
    GetById(id string) (content *models.Content, err error) //O(1)
    GetAll() (contents []*models.Content, err error)        //O(1)
    Delete(id string) error                                 //O(1)
}

type contentsRepository struct {
    c *mgo.Collection //O(1)
}

func NewContentRepository(conn database.Connection) ContentsRepository {
    return &contentsRepository{conn.DB().C(ContentsCollection)} //O(1)
}

func (r *contentsRepository) UploadContent(content *models.Content) error {
    return r.c.Insert(content) //O(1)
}

func (r *contentsRepository) Update(content *models.Content) error {
    return r.c.UpdateId(content.Id, content) //O(1)
}

func (r *contentsRepository) GetById(id string) (content *models.Content, err error) {
    err = r.c.FindId(bson.ObjectIdHex(id)).One(&content) //O(1)
    return content, err                                  //O(1)
}

func (r *contentsRepository) GetAll() (contents []*models.Content, err error) {
    err = r.c.Find(bson.M{}).All(&contents) //O(1)
    return contents, err                    //O(1)
}

func (r *contentsRepository) Delete(id string) error {
    return r.c.RemoveId(bson.ObjectIdHex(id)) //O(1)
```

Here example for content repository. We choose one because for other repository nearly same as this one. So, this file contains basic operation for data manipulation, get data from database, and create data in database. Mostly, it take O(1). Buat for GetAll its depend on the number of data in collections. So, for big data we can say the complexity is O(n)

- Middleware

```go
import (
    "TEFA-STUDYCASE-1/security" //O(1)
    "TEFA-STUDYCASE-1/util"     //O(1)
    "net/http"                  //O(1)

    "github.com/gofiber/fiber/v2"       //O(1)
    jwtware "github.com/gofiber/jwt/v2" //O(1)
)

func AuthRequired(ctx *fiber.Ctx) error {
    return jwtware.New(jwtware.Config{ //O(1)
        SigningKey:    security.JwtSecretKey,
        SigningMethod: security.JwtSigningMethod,
        TokenLookup:   "header:Authorization",
        ErrorHandler: func(c *fiber.Ctx, err error) error {
            return c.
                Status(http.StatusUnauthorized).
                JSON(util.NewJError(err))
        },
    })(ctx)
}
```

The complexity of this program is O(1). The AuthRequired function only creates and returns a jwtware.New function, which has a constant time complexity. It does not perform any operations that depend on the size of the input.

```go
import (
    "TEFA-STUDYCASE-1/security" //O(1)
    "TEFA-STUDYCASE-1/util"     //O(1)

    "github.com/form3tech-oss/jwt-go" //O(1)
    "github.com/gofiber/fiber/v2"     //O(1)
)

func AdminMiddleware(ctx *fiber.Ctx) error {
    token := ctx.Locals("user").(*jwt.Token)       //O(1)
    payload, err := security.ParseToken(token.Raw) //O(n)
    if err != nil {                                //O(1)
        return err //O(1)
    }
    if payload.Subject != "admin" { //O(1)
        return util.ErrUnauthorized //O(1)
    }
    return ctx.Next() //O(1)
}
```

Complexity for AdminMiddleware is O(n). Because its require ParseToken that has complexity O(n). And other basic operation O(1).

```go
import (
    "TEFA-STUDYCASE-1/security" //O(1)
    "TEFA-STUDYCASE-1/util"     //O(1)

    "github.com/form3tech-oss/jwt-go" //O(1)
    "github.com/gofiber/fiber/v2"     //O(1)
)

func TeacherMiddleware(ctx *fiber.Ctx) error {
    token := ctx.Locals("user").(*jwt.Token)       //O(1)
    payload, err := security.ParseToken(token.Raw) //O(n)
    if err != nil {                                //O(1)
        return err //O(1)
    }
    if payload.Subject != "teacher" { //O(1)
        return util.ErrUnauthorized //O(1)
    }
    return ctx.Next() //O(1)
}
```

Complexity for TeacherMiddleware is O(n). Because its require ParseToken that has complexity O(n). And other basic operation O(1).

- Controllers/auth

```go
import (
    "TEFA-STUDYCASE-1/models"     //O(1)
    "TEFA-STUDYCASE-1/repository" //O(1)
    "TEFA-STUDYCASE-1/security"   //O(1)
    "TEFA-STUDYCASE-1/util"       //O(1)
    "fmt"                         //O(1)
    "log"                         //O(1)
    "net/http"                    //O(1)
    "strings"                     //O(1)
    "time"                        //O(1)

    "github.com/form3tech-oss/jwt-go"    //O(1)
    "github.com/gofiber/fiber/v2"        //O(1)
    "gopkg.in/asaskevich/govalidator.v9" //O(1)
    "gopkg.in/mgo.v2"                    //O(1)
    "gopkg.in/mgo.v2/bson"               //O(1)
)

type AuthController interface {
    SignUp(ctx *fiber.Ctx) error     //O(1)
    SignIn(ctx *fiber.Ctx) error     //O(1)
    GetUser(ctx *fiber.Ctx) error    //O(1)
    GetUsers(ctx *fiber.Ctx) error   //O(1)
    PutUser(ctx *fiber.Ctx) error    //O(1)
    DeleteUser(ctx *fiber.Ctx) error //O(1)
}

type authController struct {
    usersRepo repository.UsersRepository //O(1)
}

func NewAuthController(usersRepo repository.UsersRepository) AuthController {
    return &authController{usersRepo} //O(1)
}
```

Importing packages and other dependencies is O(1), which means it takes constant time to execute them.
Declaring an interface AuthController is O(1), since it simply declares an interface without doing any complex operations.
Creating a function NewAuthController that returns an instance of tasksController has O(1), because it just creates an instance of the authController.
Creating a struct authController that has a field authRepo that implements the authRepository interface is O(1) because it simply declares a struct and a field.

```go
func (c *authController) SignUp(ctx *fiber.Ctx) error {
    var newUser models.User        //O(1)
    err := ctx.BodyParser(&newUser) //O(1)
    if err != nil {
        return ctx. //O(1)
                Status(http.StatusUnprocessableEntity).
                JSON(util.NewJError(err))
    }
    newUser.Email = util.NormalizeEmail(newUser.Email) //O(1)
    if !govalidator.IsEmail(newUser.Email) {           //O(1)
        return ctx. //O(1)
                Status(http.StatusBadRequest).
                JSON(util.NewJError(util.ErrInvalidEmail))
    }
    exists, err := c.usersRepo.GetByEmail(newUser.Email) //O(n)
    if err == mgo.ErrNotFound {                          //O(1)
        if strings.TrimSpace(newUser.Password) == "" { //O(1)
            return ctx. //O(1)
                    Status(http.StatusBadRequest).
                    JSON(util.NewJError(util.ErrEmptyPassword))
        }
        newUser.Password, err = security.EncryptPassword(newUser.Password) //O(n)
        if err != nil {                                                    //O(1)
            return ctx. //O(1)
                    Status(http.StatusBadRequest).
                    JSON(util.NewJError(err))
        }
        newUser.CreatedAt = time.Now()        //O(1)
        newUser.UpdatedAt = newUser.CreatedAt //O(1)
        newUser.Role = "student"              //O(1)
        newUser.Id = bson.NewObjectId()       //O(1)
        err = c.usersRepo.Save(&newUser)      //O(1)
        if err != nil {                       //O(1)
            return ctx. //O(1)
                    Status(http.StatusBadRequest).
```

Function SignUp has complexity O(n). For the basic operations need O(1). But we have to call function GetByEmail that has complexity O(n) and EncryptPassword O(n).

``go
JSON(util.NewJError(err))
}
return ctx. //O(1)
Status(http.StatusCreated).
JSON(newUser)
}
 	
if exists != nil { //O(1)
err = util.ErrEmailAlreadyExists //O(1)
}
 	
return ctx. //O(1)
Status(http.StatusBadRequest).
JSON(util.NewJError(err))
}
``
``go
func (c *authController) SignIn(ctx *fiber.Ctx) error {
var input models.User //O(1)
err := ctx.BodyParser(&input) //O(1)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnprocessableEntity).
JSON(util.NewJError(err))
}
input.Email = util.NormalizeEmail(input.Email) //O(1)
user, err := c.usersRepo.GetByEmail(input.Email) //O(1)
if err != nil { //O(1)
log.Printf("%s signin failed: %v\n", input.Email, err.Error()) //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(util.ErrInvalidCredentials))
}
err = security.VerifyPassword(user.Password, input.Password) //O(n)
if err != nil { //O(1)
log.Printf("%s signin failed: %v\n", input.Email, err.Error()) //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(util.ErrInvalidCredentials))
}
token, err := security.NewToken(user.Id.Hex(), user.Role) //O(n)
if err != nil { //O(1)
log.Printf("%s signin failed: %v\n", input.Email, err.Error()) //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(err))
}
return ctx. //O(1)
Status(http.StatusOK). //O(1)
JSON(fiber.Map{
"user": user,
"token": fmt.Sprintf("Bearer %s", token),
})
}
``
``go
func (c *authController) GetUser(ctx *fiber.Ctx) error {
payload, err := AuthRequestWithId(ctx) //O(n)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(err))
}
user, err := c.usersRepo.GetById(payload.Id) //O(1)
if err != nil { //O(1)
return ctx.
Status(http.StatusInternalServerError).
JSON(util.NewJError(err))
}
return ctx. //O(1)
Status(http.StatusOK).
JSON(user)
}
``
``go
func (c *authController) GetUsers(ctx *fiber.Ctx) error {
users, err := c.usersRepo.GetAll() //O(n)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusInternalServerError).
JSON(util.NewJError(err))
}
return ctx. //O(1)
Status(http.StatusOK).
JSON(users)
}
``
Function SignIn has complexity O(n). For the basic operations need O(1). But we have to call function **NewToken** that has complexity O(n)
Function Get User has complexity O(n). This is because for checking authority need O(n) complexity.
Creating a function GetUsers that receives a fiber context and returns all users. The complexity of retrieving all contents using the contentsRepo is O(n), where n is the number of contents in the database because we need to retrieve all contents from the database.

``go
func (c *authController) PutUser(ctx *fiber.Ctx) error {
payload, err := AuthRequestWithId(ctx) //O(n)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(err))
}
var update models.User //O(1)
err = ctx.BodyParser(&update) //O(1)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnprocessableEntity).
JSON(util.NewJError(err))
}
 	
update.UpdatedAt = time.Now() //O(1)
update.Id = bson.ObjectIdHex(payload.Id) //O(1)
err = c.usersRepo.Update(&update) //O(1)
 	
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusBadRequest).
JSON(util.NewJError(err))
}
 	
return ctx. //O(1)
Status(http.StatusCreated).
JSON(update)
}
``
	Creating a function PutUser that receives a fiber context and updates a content by ID. The complexity of parsing the task ID and request payload is O(1). The complexity of updating the content in the database using the contentRepo is O(1) as well, since it takes constant time. But again we must check the authorization and its needs O(n).

``go
func (c *authController) DeleteUser(ctx *fiber.Ctx) error {
payload, err := AuthRequestWithId(ctx) //O(n)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(err))
}
err = c.usersRepo.Delete(payload.Id) //O(1)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusInternalServerError).
JSON(util.NewJError(err))
}
ctx.Set("Entity", payload.Id) //O(1)
return ctx.SendStatus(http.StatusNoContent) //O(1)
}
``
	Creating a function DeleteUser that receives a fiber context and deletes a user by ID. The complexity of parsing the content ID is O(1). The complexity of deleting the user from the database using the userRepo is O(1) as well, since it takes constant time.

``go
func AuthRequestWithId(ctx *fiber.Ctx) (*jwt.StandardClaims, error) {
id := ctx.Params("id") //O(1)
if !bson.IsObjectIdHex(id) { //O(1)
return nil, util.ErrUnauthorized //O(1)
}
token := ctx.Locals("user").(*jwt.Token) //O(1)
payload, err := security.ParseToken(token.Raw) //O(n)
if err != nil { //O(1)
return nil, err //O(1)
}
if payload.Id != id || payload.Issuer != id { //O(1)
return nil, util.ErrUnauthorized //O(1)
}
return payload, nil //O(1)
}
``
	AuthRequestWithId for checking authorization has complexity O(n) for Parsing token. Other basic operation have O(1) for the complexity.

``go
func PayloadID(ctx *fiber.Ctx) (*jwt.StandardClaims, error) {
token := ctx.Locals("user").(*jwt.Token) //O(1)
payload, err := security.ParseToken(token.Raw) //O(n)
if err != nil { //O(1)
return nil, err //O(1)
}
return payload, nil //O(1)
}
``
	PayloadID for get ID from authorized user has complexity O(n) for Parsing token. Other basic operation have O(1) for the complexity.

Overall function in this file have complexity O(n).

- controllers/content

``go
import (
"TEFA-STUDYCASE-1/models" //O(1)
"TEFA-STUDYCASE-1/repository" //O(1)
"TEFA-STUDYCASE-1/util" //O(1)
"errors" //O(1)
"net/http" //O(1)
"time" //O(1)
 	
"github.com/gofiber/fiber/v2" //O(1)
"gopkg.in/mgo.v2/bson" //O(1)
)
 	
type ContentsController interface {
UploadContent(ctx *fiber.Ctx) error //O(1)
GetContent(ctx *fiber.Ctx) error //O(1)
GetContents(ctx *fiber.Ctx) error //O(1)
PutContent(ctx *fiber.Ctx) error //O(1)
DeleteContent(ctx *fiber.Ctx) error //O(1)
}
 	
func NewContentController(contentsRepo repository.ContentsRepository) ContentsController {
return &contentsController{contentsRepo} //O(1)
}
 	
type contentsController struct {
contentRepo repository.ContentsRepository //O(1)
}
``
Importing packages and other dependencies is O(1), which means it takes constant time to execute them.
Declaring an interface ContentsController is O(1), since it simply declares an interface without doing any complex operations.
Creating a function NewContentController that returns an instance of tasksController has O(1), because it just creates an instance of the contentsController.
Creating a struct contentsController that has a field contentRepo that implements the ContentsRepository interface is O(1) because it simply declares a struct and a field.

``go
func (c *contentsController) UploadContent(ctx *fiber.Ctx) error {
payload, err := PayloadID(ctx) //O(n)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(err))
}
var newContent models.Content //O(1)
err = ctx.BodyParser(&newContent) //O(1)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnprocessableEntity).
JSON(util.NewJError(err))
}
 	
if newContent.Title == "" || newContent.Content == "" { //O(1)
return ctx. //O(1)
Status(http.StatusBadRequest).
JSON(util.NewJError(errors.New("bad request: invalid content")))
}
 	
newContent.CreatedAt = time.Now() //O(1)
newContent.UpdatedAt = newContent.CreatedAt //O(1)
newContent.User_ID = payload.Id //O(1)
newContent.Id = bson.NewObjectId() //O(1)
err = c.contentRepo.UploadContent(&newContent) //O(1)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusBadRequest).
JSON(util.NewJError(err))
}
return ctx. //O(1)
Status(http.StatusCreated).
JSON(newContent)
}
``
Creating a function UploadContent that receives a fiber context and creates a new content has a complexity of O(1) for parsing the request and validating the input. The complexity of creating a new content, generating an object ID, and inserting the task into the database using the contentRepo is O(1) as well. But at first we need to get payload ID. And this complexity is O(n). So, overall the complexity is O(n)

``go
func (c *contentsController) GetContent(ctx *fiber.Ctx) error {
contentID := ctx.Params("id") //O(1)
if !bson.IsObjectIdHex(contentID) { //O(1)
return ctx.Status(http.StatusBadRequest). //O(1)
JSON(util.NewJError(errors.New("invalid content id")))
}
 	
content, err := c.contentRepo.GetById(contentID) //O(1)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusInternalServerError).
JSON(util.NewJError(err))
}
//cek authorize
err = ContentRequestWithId(ctx, content.User_ID) //O(n)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusUnauthorized).
JSON(util.NewJError(err))
}
return ctx. //O(1)
Status(http.StatusOK). //O(1)
JSON(content)
}
``
Creating a function GetContent that receives a fiber context and returns a content by ID is O(1), because GetConetntById, it directly accesses the database record using the unique identifier and returns a single task. But we must check is the user has authorize for this content. And it is needs O(n)

``go
func (c *contentsController) GetContents(ctx *fiber.Ctx) error {
contents, err := c.contentRepo.GetAll() //O(n)
if err != nil { //O(1)
return ctx. //O(1)
Status(http.StatusInternalServerError).
JSON(util.NewJError(err))
}
return ctx. //O(1)
Status(http.StatusOK).
JSON(contents)
}
``
Creating a function GetContents that receives a fiber context and returns all contents. The complexity of retrieving all contents using the contentsRepo is O(n), where n is the number of contents in the database because we need to retrieve all contents from the database.

```go
func (c *contentsController) PutContent(ctx *fiber.Ctx) error {
    contentID := ctx.Params("id")        //O(1)
    if !bson.IsObjectIdHex(contentID) { //O(1)
        return ctx.Status(http.StatusBadRequest). //O(1)
                                JSON(util.NewJError(errors.New("invalid content id")))
    }
    var update models.Content       //O(1)
    err := ctx.BodyParser(&update) //O(1)
    if err != nil {                //O(1)
        return ctx. //O(1)
                Status(http.StatusUnprocessableEntity).
                JSON(util.NewJError(err))
    }

    //cek authorize
    err = ContentRequestWithId(ctx, update.User_ID) //O(n)
    if err != nil {                                 //O(1)
        return ctx. //O(1)
                Status(http.StatusUnauthorized).
                JSON(util.NewJError(err))
    }

    update.UpdatedAt = time.Now()           //O(1)
    update.Id = bson.ObjectIdHex(contentID) //O(1)
    err = c.contentRepo.Update(&update)     //O(1)

    if err != nil { //O(1)
        return ctx. //O(1)
                Status(http.StatusBadRequest).
                JSON(util.NewJError(err))
    }

    return ctx. //O(1)
            Status(http.StatusCreated).
            JSON(update)
}
```

Creating a function PutContent that receives a fiber context and updates a content by ID. The complexity of parsing the task ID and request payload is O(1). The complexity of updating the content in the database using the contentRepo is O(1) as well, since it takes constant time. But again we must check the authorization and its needs O(n).

```go
func (c *contentsController) DeleteContent(ctx *fiber.Ctx) error {
    contentID := ctx.Params("id")       //O(1)
    if !bson.IsObjectIdHex(contentID) { //O(1)
        return ctx.Status(http.StatusBadRequest). //O(1)
                                JSON(util.NewJError(errors.New("invalid content id")))
    }

    err := c.contentRepo.Delete(contentID) //O(1)
    if err != nil {                        //O(1)
        return ctx. //O(1)
                Status(http.StatusInternalServerError).
                JSON(util.NewJError(err))
    }
    ctx.Set("Entity", contentID)         //O(1)
    return ctx.SendStatus(http.StatusOK) //O(1)
}
```

Creating a function DeleteContent that receives a fiber context and deletes a content by ID. The complexity of parsing the content ID is O(1). The complexity of deleting the content from the database using the contentRepo is O(1) as well, since it takes constant time.

```go
// AUTHORIZATION
func ContentRequestWithId(ctx *fiber.Ctx, id string) error {
    token := ctx.Locals("user").(*jwt.Token)        //O(1)
    payload, err := security.ParseToken(token.Raw) //O(n)
    if err != nil {                                //O(1)
        return err //O(1)
    }

    if payload.Id != id || payload.Issuer != id { //O(1)
        return util.ErrUnauthorized //O(1)
    }
    return nil //O(1)
}
```

Function ContentRequestWithId being create to check user authorization. To call locals it require O(1). ParseToken is O(n) because inside function ParseToken need to hash token that dep

Overall, the complexity of the code is mainly O(n).

- controllers/task.go

```go
import (
    "TEFA-STUDYCASE-1/models"     // O(1)
    "TEFA-STUDYCASE-1/repository" // O(1)
    "TEFA-STUDYCASE-1/util"       // O(1)
    "errors"                      // O(1)
    "net/http"                    // O(1)
    "time"                        // O(1)

    "github.com/gofiber/fiber/v2" // O(1)
    "gopkg.in/mgo.v2/bson"        // O(1)
)

type TasksController interface {
    CreateTask(ctx *fiber.Ctx) error // O(1)
    GetTask(ctx *fiber.Ctx) error    // O(1)
    GetTasks(ctx *fiber.Ctx) error   // O(1)
    UpdateTask(ctx *fiber.Ctx) error // O(1)
    DeleteTask(ctx *fiber.Ctx) error // O(1)
}

func NewTaskController(tasksRepo repository.TasksRepository) TasksController {
    return &tasksController{tasksRepo} // O(1)
}

type tasksController struct {
    taskRepo repository.TasksRepository // O(1)
}
```

Importing packages and other dependencies is O(1), which means it takes constant time to execute them.
Declaring an interface TasksController is O(1), since it simply declares an interface without doing any complex operations.
Creating a function NewTaskController that returns an instance of tasksController has O(1), because it just creates an instance of the tasksController.
Creating a struct tasksController that has a field taskRepo that implements the TasksRepository interface is O(1) because it simply declares a struct and a field.

```go
func (c *tasksController) CreateTask(ctx *fiber.Ctx) error {
    var newTask models.Task          // O(1)
    err := ctx.BodyParser(&newTask) // O(1)
    if err != nil {                  // O(1)
        return ctx. // O(1)
                Status(http.StatusUnprocessableEntity).
                JSON(util.NewJError(err))
    }

    if newTask.Title == "" || newTask.Description == "" { // O(1)
        return ctx. // O(1)
                Status(http.StatusBadRequest).
                JSON(util.NewJError(errors.New("bad request: invalid task")))
    }

    newTask.CreatedAt = time.Now()        // O(1)
    newTask.UpdatedAt = newTask.CreatedAt // O(1)
    newTask.Id = bson.NewObjectId()       // O(1)

    err = c.taskRepo.CreateTask(&newTask) // O(1)
    if err != nil {                       // O(1)
        return ctx. // O(1)
                Status(http.StatusBadRequest).
                JSON(util.NewJError(err))
    }
    return ctx. // O(1)
            Status(http.StatusCreated).
            JSON(newTask)
}
```

Creating a function CreateTask that receives a fiber context and creates a new task has a complexity of O(1) for parsing the request and validating the input. The complexity of creating a new task, generating an object ID, and inserting the task into the database using the taskRepo is O(1) as well, since they all take constant time.

```go
func (c *tasksController) GetTask(ctx *fiber.Ctx) error {
    taskID := ctx.Params("id")        // O(1)
    if !bson.IsObjectIdHex(taskID) { // O(1)
        return ctx.Status(http.StatusBadRequest).
            JSON(util.NewJError(errors.New("invalid task id")))
    }
    user, err := c.taskRepo.GetTaskById(taskID) // O(1)
    if err != nil {                             // O(1)
        return ctx. // O(1)
                Status(http.StatusInternalServerError).
                JSON(util.NewJError(err))
    }
    return ctx. // O(1)
            Status(http.StatusOK).
            JSON(user)
}
```

Creating a function GetTask that receives a fiber context and returns a task by ID is O(1), because GetTaskById, it directly accesses the database record using the unique identifier and returns a single task.

```go
func (c *tasksController) GetTasks(ctx *fiber.Ctx) error {
    tasks, err := c.taskRepo.GetAllTasks() // O(n)
    if err != nil {                        // O(1)
        return ctx. // O(1)
                Status(http.StatusInternalServerError).
                JSON(util.NewJError(err))
    }
    return ctx. // O(1)
            Status(http.StatusOK).
            JSON(tasks)
}
```

Creating a function GetTasks that receives a fiber context and returns all tasks. The complexity of retrieving all tasks using the taskRepo is O(n), where n is the number of tasks in the database because we need to retrieve all tasks from the database.

```go
func (c *tasksController) UpdateTask(ctx *fiber.Ctx) error {
    taskID := ctx.Params("id")      // O(1)
    var update models.Task          // O(1)
    err := ctx.BodyParser(&update) // O(1)
    if err != nil {                 // O(1)
        return ctx.
            Status(http.StatusUnprocessableEntity).
            JSON(util.NewJError(err))
    }

    update.UpdatedAt = time.Now()        // O(1)
    update.Id = bson.ObjectIdHex(taskID) // O(1)

    err = c.taskRepo.UpdateTask(&update) // O(1)
    if err != nil {                      // O(1)
        return ctx. // O(1)
                Status(http.StatusBadRequest).
                JSON(util.NewJError(err))
    }

    return ctx. // O(1)
            Status(http.StatusOK).
            JSON(update)
}
```

Creating a function UpdateTask that receives a fiber context and updates a task by ID. The complexity of parsing the task ID and request payload is O(1). The complexity of updating the task in the database using the taskRepo is O(1) as well, since it takes constant time.

```go
func (c *tasksController) DeleteTask(ctx *fiber.Ctx) error {
    taskID := ctx.Params("id")        // O(1)
    if !bson.IsObjectIdHex(taskID) { // O(1)
        return ctx.Status(http.StatusBadRequest). // O(1)
                JSON(util.NewJError(errors.New("invalid task id")))
    }

    err := c.taskRepo.DeleteTask(taskID) // O(1)
    if err != nil {                      // O(1)
        return ctx. // O(1)
                Status(http.StatusInternalServerError).
                JSON(util.NewJError(err))
    }
    ctx.Set("Entity", taskID)            // O(1)
    return ctx.SendStatus(http.StatusOK) // O(1)
}
```

Creating a function DeleteTask that receives a fiber context and deletes a task by ID. The complexity of parsing the task ID is O(1). The complexity of deleting the task from the database using the taskRepo is O(1) as well, since it takes constant time.

Overall, the complexity of the code is mainly O(n) when retrieving or returning all tasks, and O(1) for the rest of the operations. This is because we have not connect this function with token in jwt or something relate to password. Therefore, according to Big O rules, we know that we have to drop non-dominant terms, so the complexity of controllers/task.go is O(n).

- controllers/userchat.go

```go
import (
    "TEFA-STUDYCASE-1/models"     // O(1)
    "TEFA-STUDYCASE-1/repository" // O(1)
    "TEFA-STUDYCASE-1/util"       // O(1)
    "errors"                      // O(1)
    "net/http"                    // O(1)
    "time"                        // O(1)

    "github.com/gofiber/fiber/v2" // O(1)
    "gopkg.in/mgo.v2/bson"        // O(1)
)

type UserchatsController interface {
    CreateUserchat(ctx *fiber.Ctx) error // O(1)
    GetUserchat(ctx *fiber.Ctx) error    // O(1)
    GetUserchats(ctx *fiber.Ctx) error   // O(1)
    UpdateUserchat(ctx *fiber.Ctx) error // O(1)
}

func NewUserchatController(userchatsRepo repository.UserchatsRepository) UserchatsController {
    return &userchatsController{userchatsRepo} // O(1)
}

type userchatsController struct {
    userchatRepo repository.UserchatsRepository // O(1)
}
```

Importing packages and other dependencies is O(1), which means it takes constant time to execute them.
Declaring an interface UserchatsController is O(1), since it simply declares an interface without doing any complex operations.
Creating a function NewUserchatController that returns an instance of userchatsController has O(1), because it just creates an instance of the userchatsController.
Creating a struct userchatsController that has a field userchatRepo that implements the UserchatsRepository interface is O(1) because it simply declares a struct and a field.

```go
func (c *userchatsController) CreateUserchat(ctx *fiber.Ctx) error {
    var newUserchat models.Userchat     // O(1)
    err := ctx.BodyParser(&newUserchat) // O(1)
    if err != nil {                     // O(1)
        return ctx. // O(1)
                Status(http.StatusUnprocessableEntity).
                JSON(util.NewJError(err))
    }

    if newUserchat.Title == "" || newUserchat.Question == "" { // O(1)
        return ctx. // O(1)
                Status(http.StatusBadRequest).
                JSON(util.NewJError(errors.New("bad request: invalid userchat")))
    }

    newUserchat.Status = false                  // O(1)
    newUserchat.CreatedAt = time.Now()          // O(1)
    newUserchat.UpdatedAt = newUserchat.CreatedAt // O(1)
    newUserchat.Id = bson.NewObjectId()         // O(1)

    err = c.userchatRepo.CreateUserchat(&newUserchat) // O(1)
    if err != nil {                                   // O(1)
        return ctx. // O(1)
                Status(http.StatusBadRequest).
                JSON(util.NewJError(err))
    }
    return ctx. // O(1)
            Status(http.StatusCreated).
            JSON(newUserchat)
}
```

Creating a function CreateUserchat that receives a fiber context and creates a new userchat has a complexity of O(1) for parsing the request and validating the input. The complexity of creating a new userchat, generating an object ID, and inserting the userchat into the database using the userchatRepo is O(1) as well, since they all take constant time.

``go
func (c *userchatsController) GetUserchat(ctx *fiber.Ctx) error {
userchatID := ctx.Params("id") // O(1)
if !bson.IsObjectIdHex(userchatID) { // O(1)
return ctx.Status(http.StatusBadRequest). // O(1)
JSON(util.NewJError(errors.New("invalid userchat id")))
}
user, err := c.userchatRepo.GetUserchatById(userchatID) // O(1)
if err != nil { // O(1)
return ctx. // O(1)
Status(http.StatusInternalServerError).
JSON(util.NewJError(err))
}
return ctx. // O(1)
Status(http.StatusOK).
JSON(user)
}
``
``go
func (c *userchatsController) GetUserchats(ctx *fiber.Ctx) error {
userchats, err := c.userchatRepo.GetAllUserchats() // O(n)
if err != nil { // O(1)
return ctx. // O(1)
Status(http.StatusInternalServerError).
JSON(util.NewJError(err))
}
return ctx. // O(1)
Status(http.StatusOK).
JSON(userchats)
}
``
``go
func (c *userchatsController) UpdateUserchat(ctx *fiber.Ctx) error {
userchatID := ctx.Params("id") // O(1)
var update models.Userchat // O(1)
err := ctx.BodyParser(&update) // O(1)
if err != nil { // O(1)
return ctx.
Status(http.StatusUnprocessableEntity).
JSON(util.NewJError(err))
}
update.Status = true // O(1)
update.UpdatedAt = time.Now() // O(1)
update.Id = bson.ObjectIdHex(userchatID) // O(1)
 	
err = c.userchatRepo.UpdateUserchat(&update) // O(1)
if err != nil { // O(1)
return ctx. // O(1)
Status(http.StatusBadRequest).
JSON(util.NewJError(err))
}
 	
return ctx. // O(1)
Status(http.StatusOK).
JSON(update)
}
``
Overall, the complexity of the code is mainly O(n) when retrieving or returning all userchats, and O(1) for the rest of the operations. This is because we have not connect this function with token in jwt or something relate to password or something with complex operations. Therefore, according to Big O rules, we know that we have to drop non-dominant terms, so the complexity of controllers/userchat.go is O(n).
	Creating a function GetUserchat that receives a fiber context and returns a userchat by ID is O(1), because GetUserchatById, it directly accesses the database record using the unique identifier and returns a single userchat.
	Creating a function GetUserchats that receives a fiber context and returns all userchats. The complexity of retrieving all userchats using the userchatRepo is O(n), where n is the number of userchats in the database because we need to retrieve all userchats from the database.
	Creating a function UpdateUserchat that receives a fiber context and updates a userchat by ID. The complexity of parsing the userchat ID and request payload is O(1). The complexity of updating the userchat in the database using the userchatRepo is O(1) as well, since it takes constant time.

- controllers/usersub.go

``go
import (
"TEFA-STUDYCASE-1/models" // O(1)
"TEFA-STUDYCASE-1/repository" // O(1)
"TEFA-STUDYCASE-1/util" // O(1)
"errors" // O(1)
"net/http" // O(1)
"time" // O(1)
 	
"github.com/gofiber/fiber/v2" // O(1)
"gopkg.in/mgo.v2/bson" // O(1)
)
 	
type UsersubsController interface {
CreateUsersub(ctx *fiber.Ctx) error // O(1)
}
 	
func NewUsersubController(usersubsRepo repository.UsersubsRepository) UsersubsController {
return &usersubsController{usersubsRepo} // O(1)
}
 	
type usersubsController struct {
usersubRepo repository.UsersubsRepository // O(1)
}
``
Importing packages and other dependencies is O(1), which means it takes constant time to execute them.
Declaring an interface UsersubsController is O(1), since it simply declares an interface without doing any complex operations.
Creating a function NewUsersubController that returns an instance of usersubsController has O(1), because it just creates an instance of the usersubsController.
Creating a struct usersubsController that has a field usersubRepo that implements the UsersubsRepository	

	interface is O(1) because it simply declares a struct and a field.


func (c *usersubsController) CreateUsersub(ctx *fiber.Ctx) error {
var newUsersub models.Usersub // O(1)
err := ctx.BodyParser(&newUsersub) // O(1)
if err != nil { // O(1)
return ctx. // O(1)
Status(http.StatusUnprocessableEntity).
JSON(util.NewJError(err))
}
 	
if newUsersub.Plan == "" || newUsersub.Price == 0 { // O(1)
return ctx. // O(1)
Status(http.StatusBadRequest).
JSON(util.NewJError(errors.New("bad request: invalid usersub")))
}
 	
newUsersub.CreatedAt = time.Now() // O(1)
newUsersub.Id = bson.NewObjectId() // O(1)
 	
err = c.usersubRepo.CreateUsersub(&newUsersub) // O(1)
if err != nil { // O(1)
return ctx. // O(1)
Status(http.StatusBadRequest).
JSON(util.NewJError(err))
}
return ctx. // O(1)
Status(http.StatusCreated).
JSON(newUsersub)
}
	Creating a function CreateUsersub that receives a fiber context and creates a new usersub has a complexity of O(1) for parsing the request and validating the input. The complexity of creating a new usersub, generating an object ID, and inserting the usersub into the database using the usersubRepo is O(1) as well, since they all take constant time.

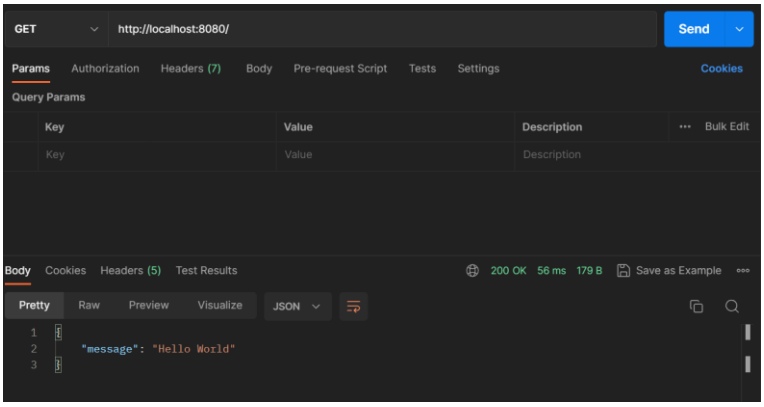
Overall, the complexity of controllers/usersub.go is O(1) since it just have a function CreateUserSub and others dependencies. This is because we have not connect this function with token in jwt or something relate to password or other complex operations.

7. **Create document also upload every doc and code to your git repository!**
   Already in github repo

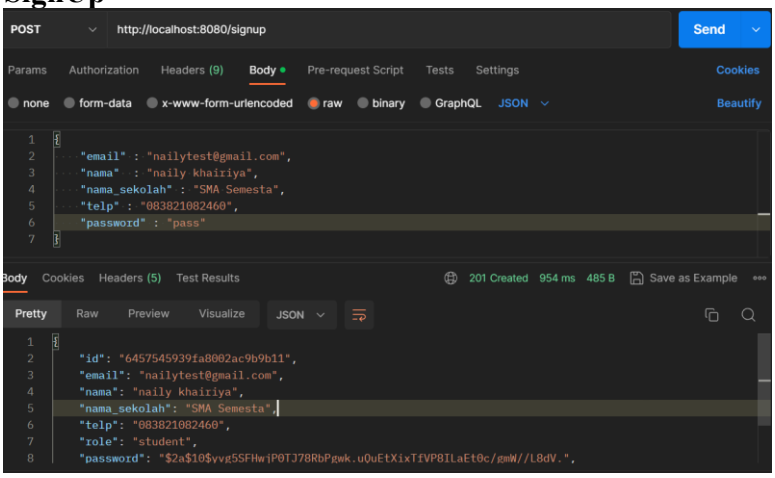
8. **Record your presentation(video) with duration between 15-20 minutes**
   Presentation Link : https://youtu.be/C0zcHwSBTPs

EXAMPLE TEST RESULT IN POSTMAN :

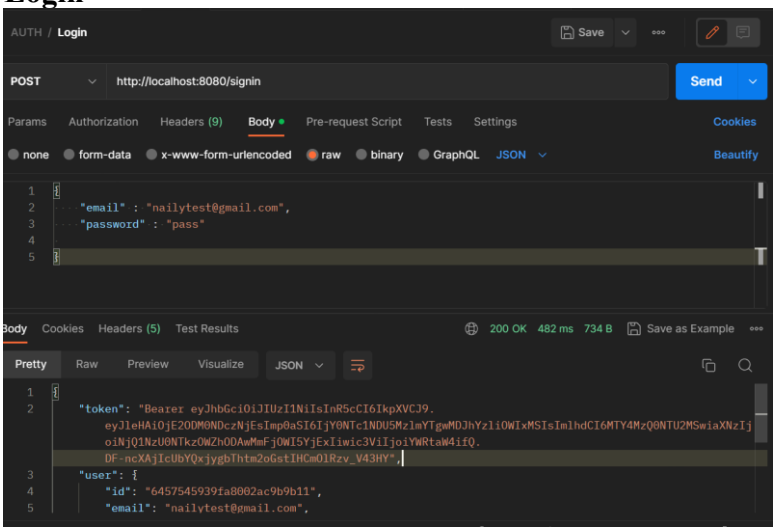
- /



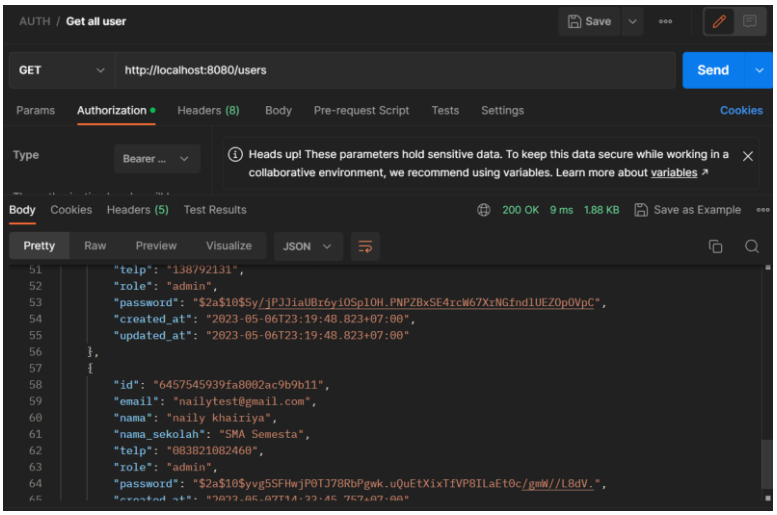
- **SignUp**



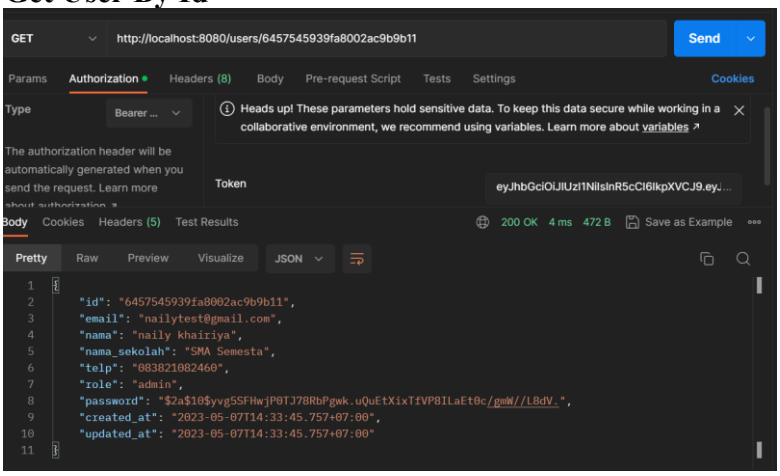
- **Login**



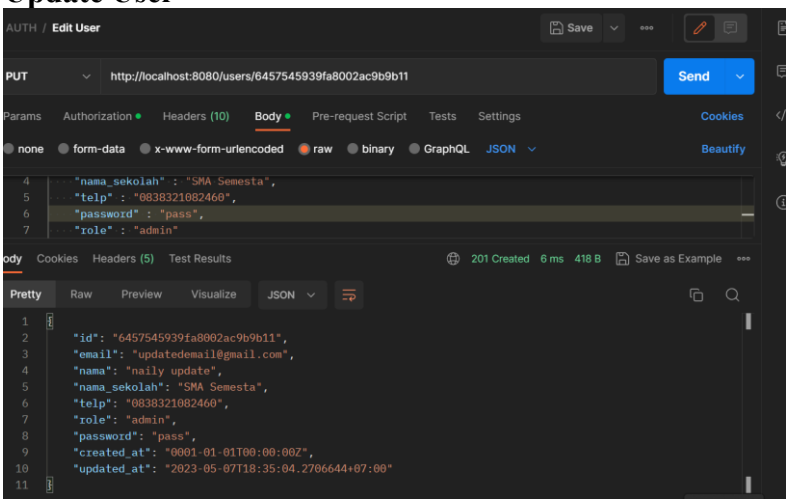
- **Get All User**



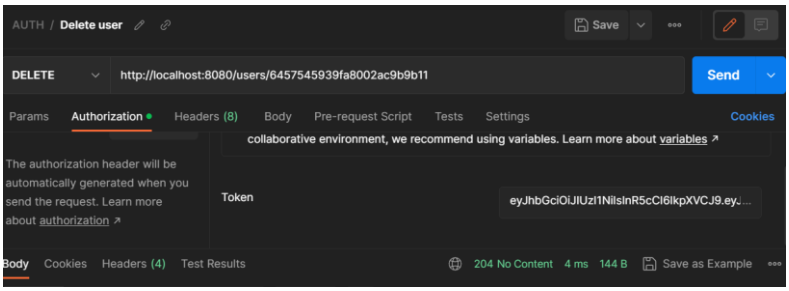
- **Get User By Id**



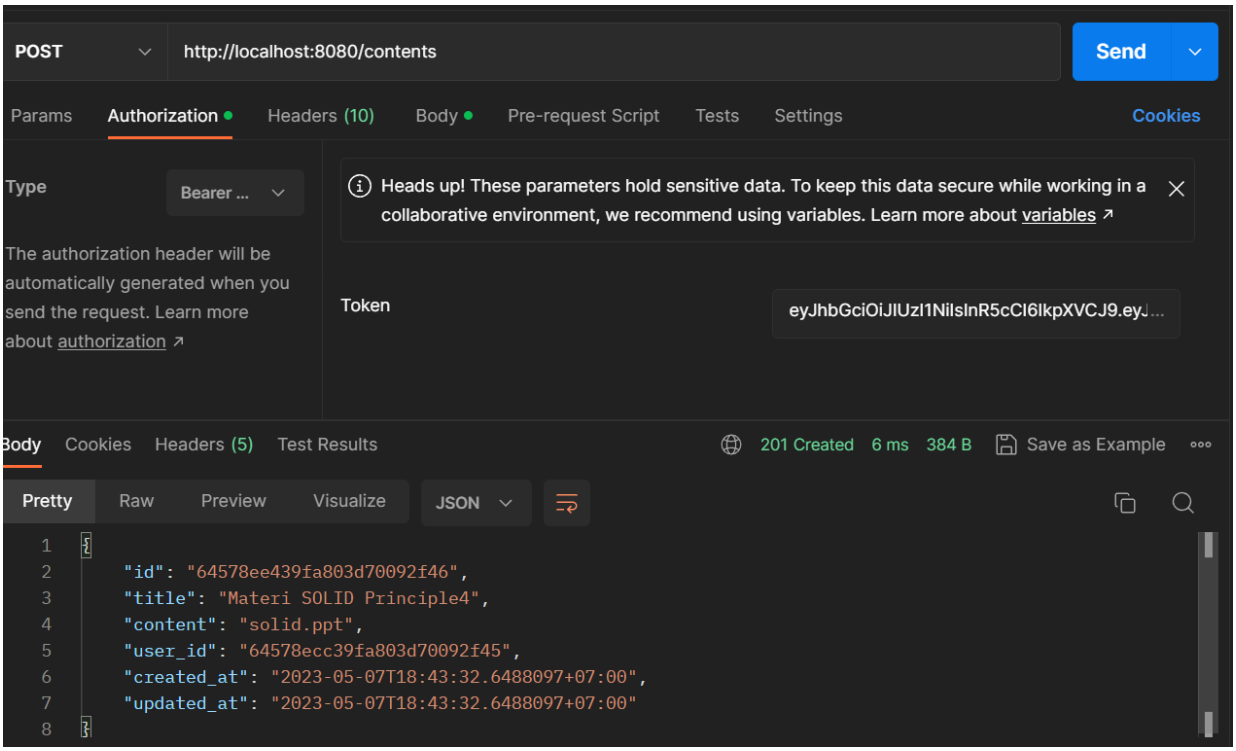
- **Update User**



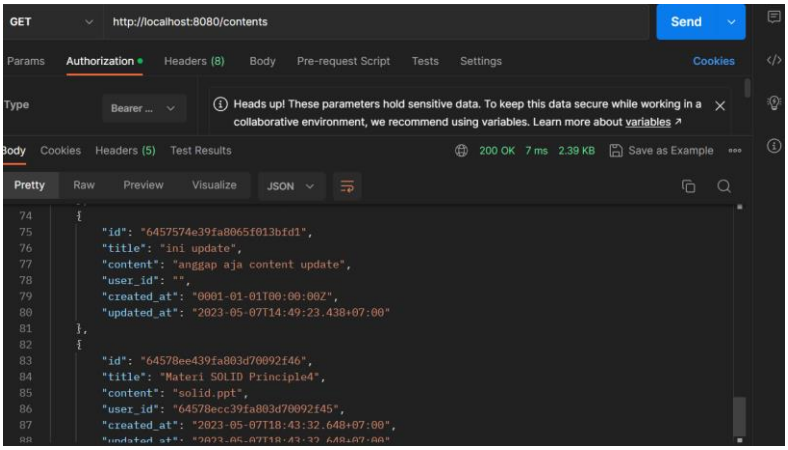
- **Delete User**



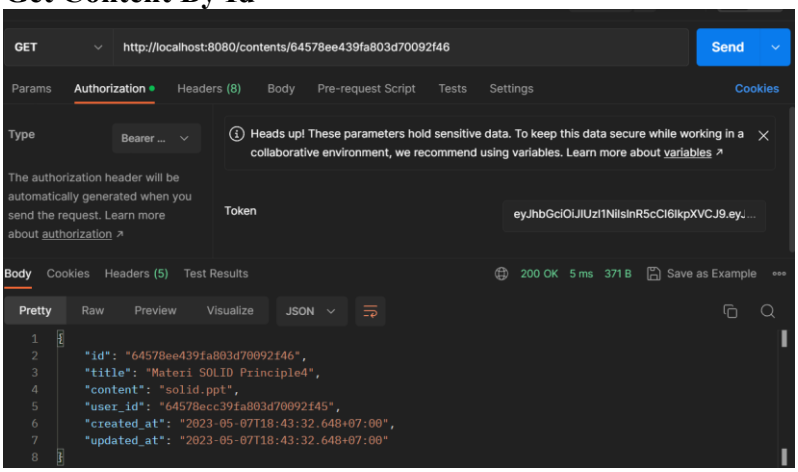
- **Create Content**



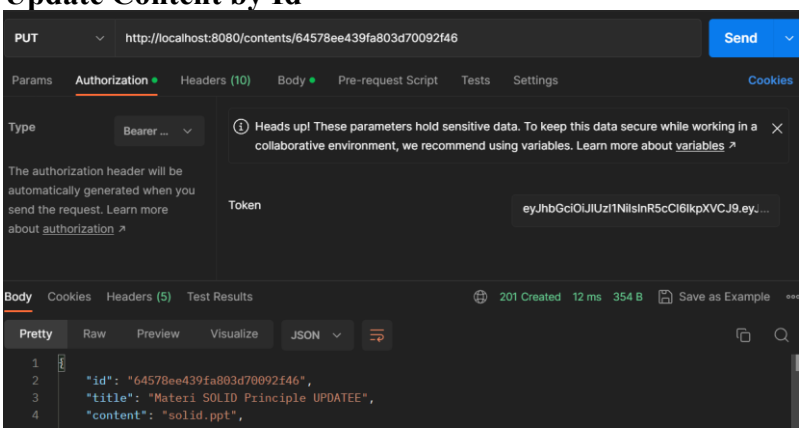
- **Get All Content**



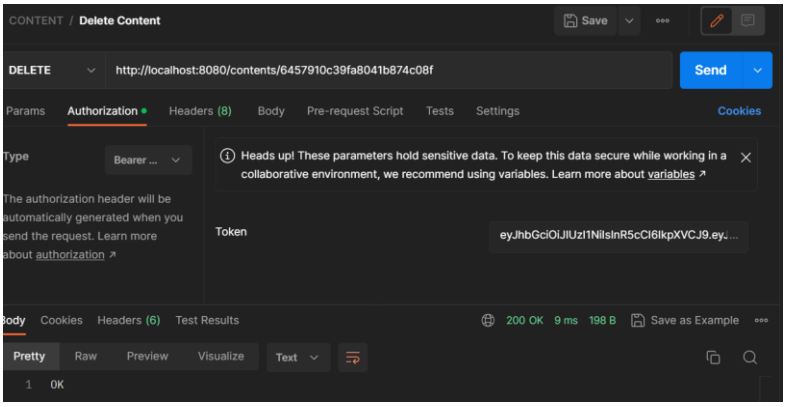
- **Get Content By Id**



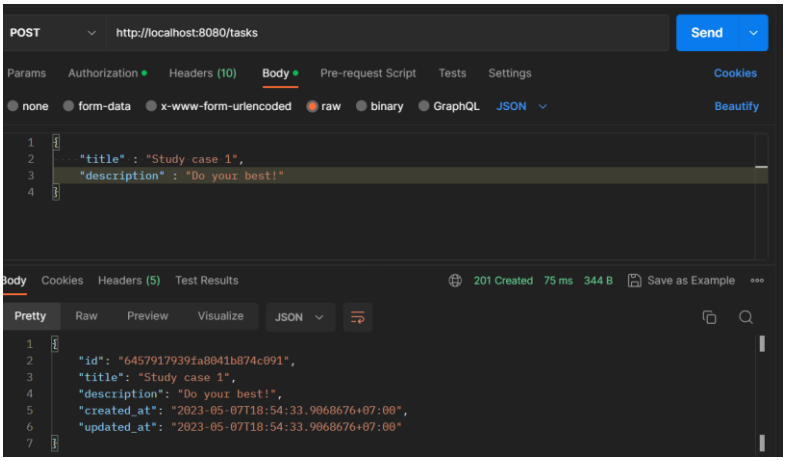
- **Update Content by Id**



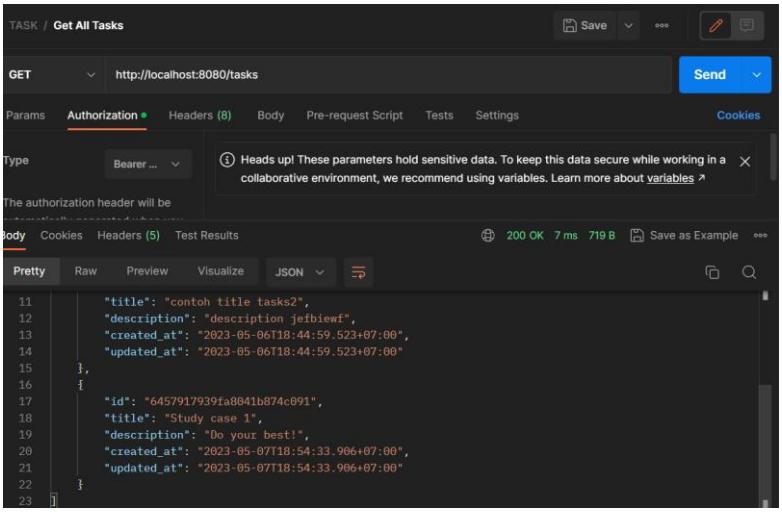
- **Delete Content**



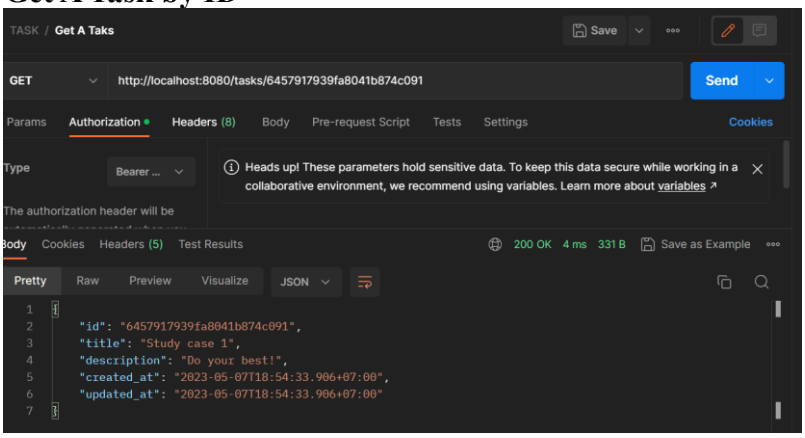
- **Create Task**



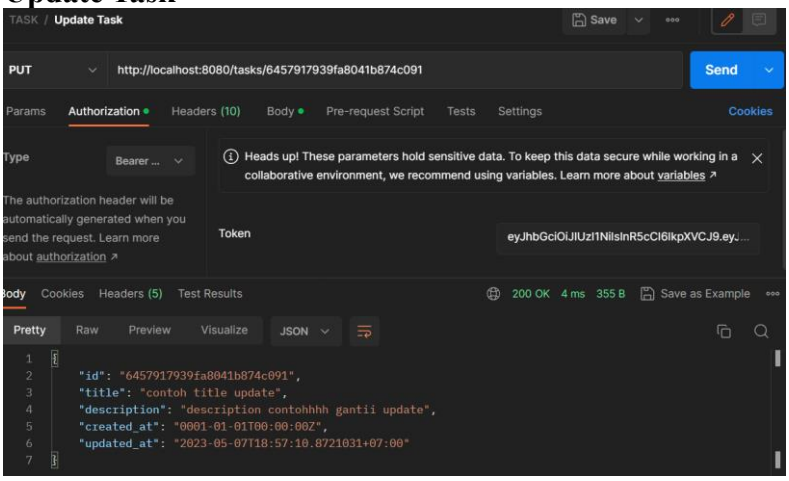
- **Get All Task**



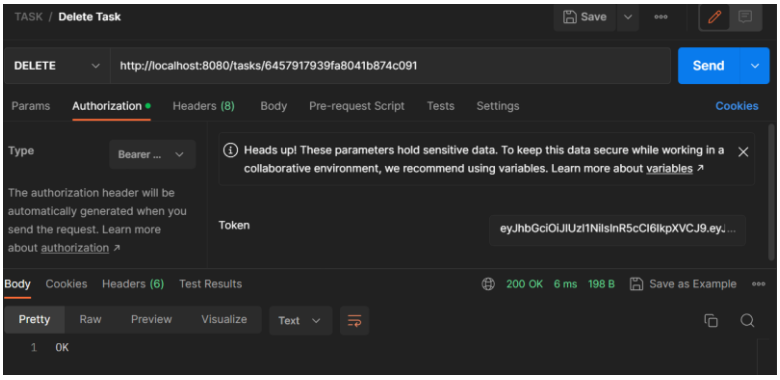
- **Get A Task by ID**



- **Update Task**



- **Delete Task**



- **Upload Chat**

USERCHAT / Upload Chats

POST http://localhost:8080/chats

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
  "title" : "general things",
  "question" : "How to subscribe?",
  "keyword" : "subsription",
  "status" : true
}
```

Body Cookies Headers (5) Test Results 201 Created 5 ms 371 B Save as Example

Pretty Raw Preview Visualize JSON

```
{
  "id": "6457929639fa8041b874c092",
  "title": "general things",
  "question": "How to subscribe?",
  "keyword": "subsription",
  "created_at": "2023-05-07T18:59:18.3666899+07:00",
  "updated_at": "2023-05-07T18:59:18.3666899+07:00"
}
```

- **Get A Chat**

USERCHAT / Get A Chat

GET http://localhost:8080/chats/6457929639fa8041b874c092

Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results 200 OK 4 ms 358 B Save as Example

Pretty Raw Preview Visualize JSON

```
{
  "id": "6457929639fa8041b874c092",
  "title": "general things",
  "question": "How to subscribe?",
  "keyword": "subsription",
  "created_at": "2023-05-07T18:59:18.366+07:00",
  "updated_at": "2023-05-07T18:59:18.366+07:00"
}
```

- **Update Chat**

USERCHAT / Update Chat

PUT http://localhost:8080/chats/64563b1e39fa802b442c97c2

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

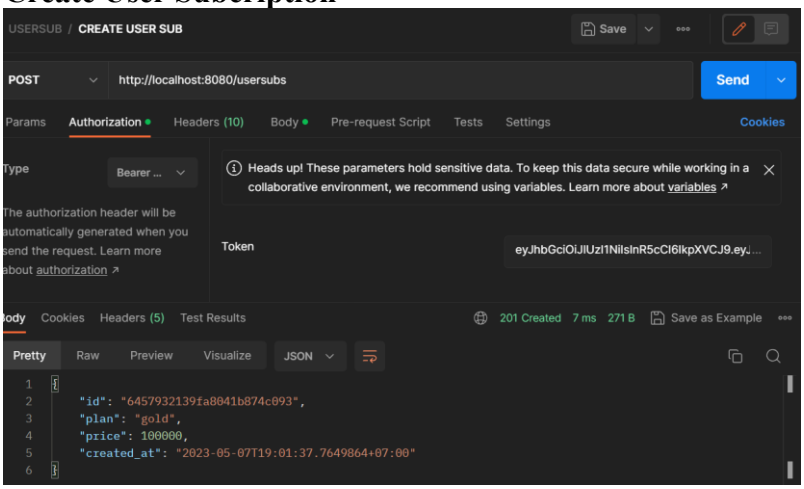
```
{
  "title" : "general things",
  "question" : "How to subscribe but for prepaid payment?",
  "keyword" : "subsription",
  "status" : true
}
```

Body Cookies Headers (5) Test Results 200 OK 4 ms

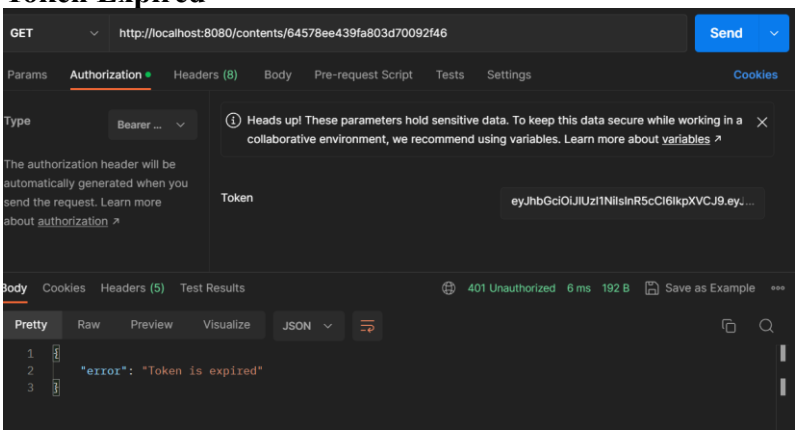
Pretty Raw Preview Visualize JSON

```
{
  "id": "64563b1e39fa802b442c97c2",
  "title": "general things",
  "question": "How to subscribe but for prepaid payment?",
  "keyword": "subsription",
  "status": true,
  "created_at": "0001-01-01T00:00:00Z",
  "updated_at": "2023-05-07T19:00:47.3171562+07:00"
}
```

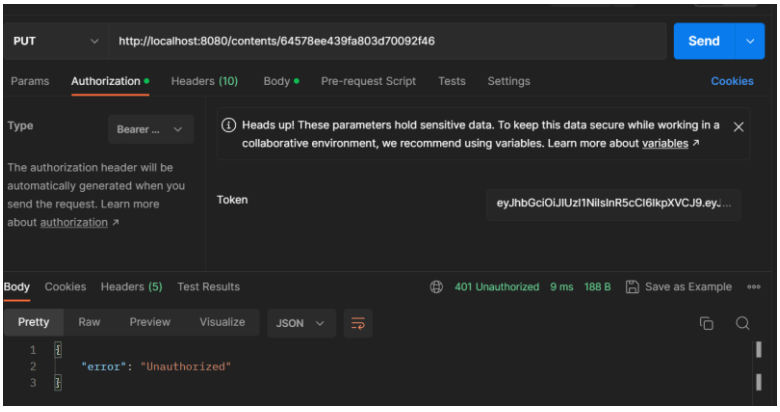
- **Create User Subcription**



- **Token Expired**



- **Unauthorized**



- **Non Authenticate**