 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Розрахунково-графічна робота**

з дисципліни “[ВЕБ-технології та ВЕБ-дизайн-2. Веб-орієнтована розробка програмного забезпечення](http://wiki.kpi.ua/index.php/%D0%92%D0%95%D0%91-%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97%20%D1%82%D0%B0%20%D0%92%D0%95%D0%91-%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD-2.%20%D0%92%D0%B5%D0%B1-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0%20%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F_(35103510))”

**Виконав:**

студент гр. БС-81

Погребенко В.О.

**Перевірив:**

[ас. Давидько О. Б.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=b92c0571-df7d-410d-8013-82552f62f3cf),

[ас. Матвійчук О. В.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=dab62327-7bea-4b83-ba3f-a2aa2848e036)

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2020

# Розділи

1. Вибір та опис специфікації проекту.
2. Вибір та обґрунтування технологій для реалізації веб сервісу.
3. Опис архітектурних рішень що використовувались в ході розробки проекту.
4. Розгортання програмного продукту.
5. Висновки.
6. Додаток.

# **Вибір та опис специфікації проекту**

**Тематика веб-додатку**

Тематика – моніторинг активності працівників і відділів певного підприємства, за допомогою якого можна буде оцінити продуктивність та активність роботи.

**Базові вимоги до додатку:**

Функціональні вимоги:

* + 1. Можливість отримати список відділів
    2. Можливість отримати користувачів по відділам
    3. Можливість редактувати список відділв (додавати\видаляти\редактувати)
    4. Можливість редактувати список користувачів (додавати\видаляти\редактувати)
    5. Можливість отримати активність по користувачам за вказанний період
    6. Можливість отримати активність по відділам за вказанний період

Нефункціональні вимоги:

*Кількість користувачів*: сервіс можуть використовувати велика кількість користувачів одночасно. Максимальна кількість користувачів обмежена «залізом», може бути визначена більш точно після проведення лоад та стрес тестування.

*Downtime*: під час апдейтів.

*Інтеграції з іншими сервісами*: Не передбачено. При необхідності, взаємодія з іншими сервісами буде проводитися через API.

*Доступність для людей з обмеженими можливостями*: Не розроблюється. При необхідності може бути підтримана у фронтєнді.

*Безпека*: авторизація, зберігання паролів у вигляді хешу з використанням «солі» та використання JWT токенів.

*Вимоги до тестування*: Вимоги:

- перевірка доступу до хендлерів без авторизації

- базова перевірка функціоналу всіх хендлерів

# **Вибір та обґрунтування технологій для реалізації веб сервісу**

* **Вибір мови програмування для реалізації веб сервісу**

Було обрано мову програмування Golang(Go), по наступним причинам:

* Наявність певного досвіду роботи з golang у IT компанії.
* За допомогою golang можна просто організовувати паралелізм за допомогою goroutines та channels.
* За допомогою golang можна порівняно просто писати веб-сервіси.
* У golang просто проводити юніт-тестування за допомогою вбудованого фреймворку testing (import "testing").
* **Вибір фреймворка для обробки мережевих запитів**

Для роботи з мережею був обраний стандартний пакет net, та написаний на ньому пакет gorilla (<https://github.com/gorilla>):

Причини вибору:

* + Gorilla – один з найстаріших та найбільших toolkit’ів для golang, що має дуже високий рейтинг.
  + Gorilla має найбільше англійське community з усіх інших подібних пакетів.
  + Gorilla має багато пакетів, серед яких є пакети для роботи з маршрутизацією та websocket’ами, що будуть використовуватися у лабораторних.
  + Gorilla сумісна з базовим пакетом net та використовується разом з ним.
  + net - net/http – стандартний пакет, що має все необхідне для того, щоб виконувати http запити.
* **Опис механізмів виконання паралельних та асинхронних завдань в обраній мові**

Паралельні обчислення реалізуються за допомогою Goroutine:

func hello() {

fmt.Println("Hello world")

}

func main() {

go hello()

}

Синхронізація зазвичай відбувається за допомогою:

* channels – канали використовуються для обміну інформацією між горутинами та для іх синхронізації за допомогою операцій запису та читання з каналу (ch <- 1, someVar <-ch)
* sync – за допомогою пакету sync та його функціоналу. Наприклад: sync.Mutex (m.Lock(), m.Unlock()), sync.WaitGroup(wg.Add(), wg.Wait()), sync.atomic(AddUint…, LoadUint…), etc.

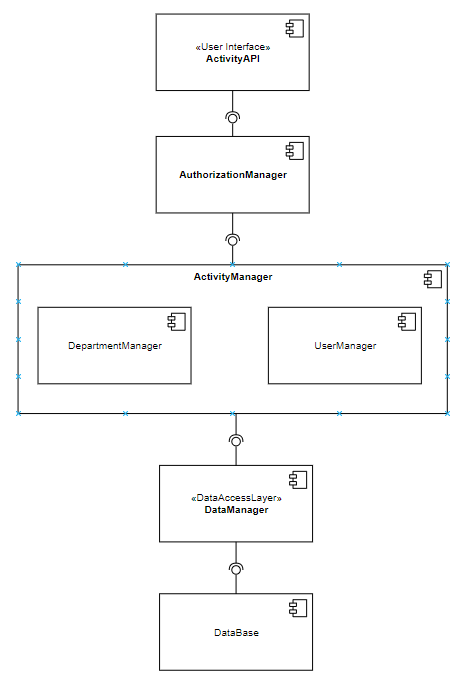
# **Опис архітектурних рішень що використовувались в ході розробки проекту.**

* **Опис високорівневої архітектури:**

Було обрано монолітну архітектуру, бо:

* розроблюваний додаток не є достатньо великим для використання мікросервісів або SOA. І використання мікросервісів тільки додасть зайві складнощі, та може не дати ніякого позитивного ефекту.
* додаток можна буде простіше та швидше розроблювати, тестувати да розгортати.

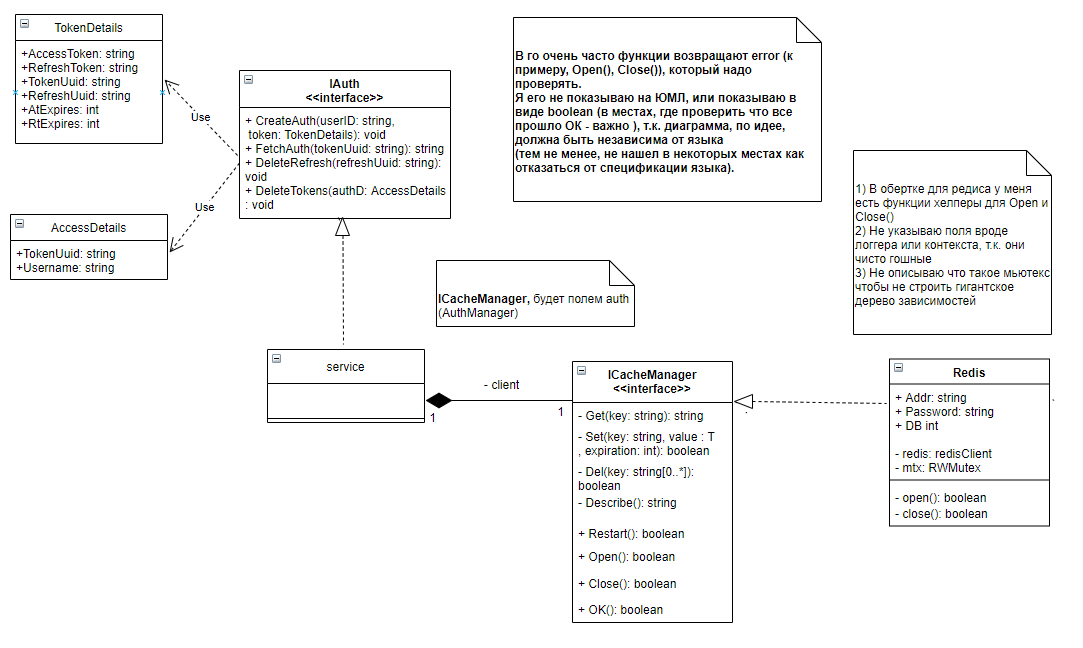
**Діаграма компонентів та їх взаємодій (HLD)**



* **Опис низькорівневої архітектури для одного із сервісів:**

Модуль авторизації («IAuth\serivce») відповідає за авторизацію користувача. Він зберігає данні про логін у редіс для подальшого швидкого доступу до них.

Модуль кешування (ICacheManager\Redis) відовідає за кешування данних додатку для можливості подальшого швидкого доступу до них.



* **Опис сторонніх компонентів та причини їх вибору:**

Сторонніми компонентами в даній роботі є:

1. БД – використовується SQLite для збереження даних системи.
2. Redis – сервіс кешування для полегшення та оптимізування системи авторизації.

# **4. Розгортання програмного продукту.**

1. Скрипт контейнеризації сервісу Dockerfile:

# Dockerfile References: https://docs.docker.com/engine/reference/builder/  
  
# Start from the latest golang base image  
FROM golang:latest as builder  
  
# Add Maintainer Info  
LABEL maintainer="Vasily\_BS81"  
  
# Set the Current Working Directory inside the container  
WORKDIR /app  
  
# Copy go mod and sum files  
COPY go.mod go.sum ./  
  
# Download all dependancies. Dependencies will be cached if the go.mod and go.sum files are not changed  
RUN go mod download  
  
# Copy the source from the current directory to the Working Directory inside the container  
COPY . .  
  
# Build the Go app  
# "-linkmode external -extldflags -static" used for solving problem with sql mattn  
# Due to that there is warning during building docker-compose : https://github.com/mattn/go-sqlite3/issues/457  
# And as I understand, there is no simple way to handle it.  
RUN go build -ldflags "-linkmode external -extldflags -static" -o main .  
  
  
######## Start a new stage from scratch #######  
FROM golang:1.15-alpine   
  
RUN apk --no-cache add ca-certificates  
  
WORKDIR /root/  
  
# Copy the Pre-built binary file from the previous stage  
COPY --from=builder /app/main .  
# See messsage at the top of docker-compose.yml  
COPY config.json /root/config.json  
  
# Expose port 9332 to the outside world  
EXPOSE 9332  
  
# Command to run the executable  
CMD ["./main"]

1. Скрипт оркестрації сервісів docker-compose.yml:

# https://dantehranian.wordpress.com/2015/03/25/how-should-i-get-application-configuration-into-my-docker-containers/  
# Used 1 way due to API was already using config.json,  
# and I didn't have enough time for googling and implementing the best practices  
  
version: '3.9'  
services:  
 web:  
 container\_name: activity\_api  
 # Path to dockerfile.  
 # '.' represents the current directory in which  
 # docker-compose.yml is present.  
 build: .  
 # tag an image to avoid unnecessary image rebuilding   
 image: activity\_api:latest  
 # map port  
 ports:  
 - "9332:9332"  
 # Launch Redis first  
 depends\_on:  
 - redis  
 # I don't see any need in this lab to map volumes.  
# # map root dir to /activity\_api, this allows to make changes to the project during the local development process (for possible future unit testing?)  
# volumes:  
# - .:/activity\_api  
 redis:  
 image: "redis:alpine"  
 # Set password (api gets pass from config json, see comment at the top of the file)  
 command: redis-server --requirepass pass\_for\_development\_purposes\_only  
 ports:  
 - "6379:6379"  
# I didn't specify connections/links because docker should take care of it by itself

# **5. Висновки**

**Аналіз архітектури та обраних інструментів:**

Вибір моноліту для архітектури додатку було правильним рішенням, бо у конкретному випадку невеликого проекту вона дозволила швидко та просто приєднати БД та Redis до додатку, та їх задеплоіти.

SQLite дозволив швидко розробити додаток та забезпечив необхідний для додатку функціонал, однак у майбутньому, у випадку розширення додатку, він може бути змінений на іншу ДБ

Redis дозволив пришвидшити та спростити реалізацію автентифікації та авторизації користувачів.

За допомогою можливостей golang був реалізований смоук тест всього додатку, та кілька простих юніт тестів. Крім того, завдяки можливостям testing можна так само просто реалізувати, наприклад, бенчмарки.

**Дослідження правил розгортання та маштабованості розробленого сервісу:**

На практиці було розглянуто ключові моменти розгортання та масштабованості розробленого сервісу:

Розгортання:

1. Краще передавати у додаток його конфігурацію як environment variables, та отримувати її всередені додатку через os.Getnev
2. Краще використовувати багатошарову структуру докерфалу, та організовувати його таким чином, щоб найрідше змінювані шари (як, наприклад, образ голанг) приходилось перебудовувати як можна рідше (у випадку змін). Це дозволить як зменшити розмір імеджу, так і пришвидшити його білд у більшості випадків.
3. Хост компонентів таких як редіс потрібно надавати в залежності від відповідних імен сервісів в docker-compose.yml. Наприклад, якщо:

version: '3.9'  
 services:

redis:

…………..

То адрес редісу у додатку також потрібно надати як: redis:<port>

Масштабовність:

1. Так як моноліти складно масштабувати (і масштабуються вони, в загалому, тільки вертикально), може виникнути потреба розбити моноліт на мікросервіси. У такому випадку, обгортання модулів у інтерфейси, документування коду, етс, може допомогти у цій задачі у майбутьному.
2. Для зручності масштабування, балансування навантаження, етс систем зазвичай використовують ***orchestration engine,*** такий як, наприклад, Kubernetes

# 6. Додаток.

Повний код можна знайти за посиланням: https://github.com/PogrebenkoBS81/Web\_course3/tree/master/lab4-5/activity\_api

**Моделі:**

package models  
  
// Admin - admin user of AAService.  
type Admin struct {  
 Username string `db:"admin\_name"`  
 // Hash - password hash (written to DB with salt).  
 // Password should be hashed on the client side, and then it would be hashed again on the server side.  
 Hash string `db:"password\_hash"`  
}  
  
// Department - AAService Department.  
type Department struct {  
 DepartmentID int64 `db:"department\_id"`  
 DepartmentName string `db:"department\_name"`  
}  
  
// User - AAService User.  
type User struct {  
 UserID int64 `db:"user\_id"`  
 UserName string `db:"user\_name"`  
 // DepartmentID - department in which the user participates.  
 DepartmentID int64 `db:"department\_id"`  
}  
  
// Activity - AAService activity.  
type Activity struct {  
 // UserID - id of user which activity were recorded.  
 UserID int64 `db:"user\_id"`  
 RecordID int64 `db:"record\_id"`  
 // TotalTime - total user work time.  
 TotalTime int64 `db:"total\_time"`  
 // ActiveTime - active user time from all total time.  
 ActiveTime int64 `db:"active\_time"`  
 // Date - time when activity record were taken.  
 Date int64 `db:"activity\_date"`  
}  
  
// UserActivity - data about user activity  
// Due to rows could be null (which would lead to an err during unmarshalling), pointer are used  
type UserActivity struct {  
 UserID \*int64 `db:"user\_id"`  
 ActiveTime \*int64 `db:"active\_time"`  
 TotalTime \*int64 `db:"total\_time"`  
}  
  
// DepartmentActivity - data about department activity  
// Due to rows could be null (which would lead to an err during unmarshalling), pointer are used  
type DepartmentActivity struct {  
 DepartmentID \*int64 `db:"department\_id"`  
 ActiveTime \*int64 `db:"active\_time"`  
 TotalTime \*int64 `db:"total\_time"`  
}  
  
// ObjectID - used when returning last inserted id or rows affected.  
type ObjectID struct {  
 ID int64  
}

**Робота з кешем:**

**Інтерфейс кеш-менеджера (Редіс обгорнутий у структуру що доповнює його можливості та реалізовує цей інтерфейс)**

// Can't create common "core" here like in SQL,  
// due to cache client doesn't have common connection interface  
// (and i (cant/don't want to thinks how to) create it)  
package cache  
  
import (  
 "activity\_api/data\_manager/cache/cache\_mock"  
 "activity\_api/data\_manager/cache/redis"  
 "context"  
 "github.com/sirupsen/logrus"  
 "time"  
)  
  
const (  
 ICacheMock = iota  
 Redis = iota  
 // Memcache  
 // ...  
)  
  
// ICacheManager - cache manager interface fr AAService.  
type ICacheManager interface {  
 Get(key string) (string, error)  
 Set(key string, value interface{}, expiration time.Duration) error  
 Del(keys ...string) error  
 Describe() string  
  
 Restart() error  
 Open() error  
 Close() error  
 OK() error  
}  
  
// ICacheConfig - common config for cache services.  
type ICacheConfig struct {  
 Address string  
 Password string  
 DB int  
}  
  
// NewCacheManager - returns new cache manager.  
func NewCacheManager(  
 cacheType int,  
 cacheConfig \*ICacheConfig,  
 ctx context.Context,  
 logger logrus.FieldLogger,  
) ICacheManager {  
 switch cacheType {  
 case ICacheMock:  
 return cache\_mock.NewCacheMock(logger)  
 case Redis:  
 return redis.NewRedisManager(cacheConfig.Address, cacheConfig.Password, cacheConfig.DB, ctx, logger)  
 default:  
 logger.WithField("func", "NewCacheManager").  
 Warnf("Unsupported cacheType: %d, using default: Redis", Redis)  
  
 return redis.NewRedisManager(cacheConfig.Address, cacheConfig.Password, cacheConfig.DB, ctx, logger)  
 }  
}

**Робота з БД:**

**Інтерфейс БД**

package core  
  
import "activity\_api/common/models"  
  
// TODO: Add "update" queries  
// TODO: Segregate interface into something like: UserManager, DepartmentManager, etc  
// ISQLDatabase - database interface for AAService  
type ISQLDatabase interface {  
 ISQLCore  
  
 CreateDB() error  
 Describe() string  
  
 CreateAdmin(admin \*models.Admin) (int64, error)  
 GetAdmin(name string) (\*models.Admin, error)  
 DeleteAdmin(name string) (int64, error)  
  
 CreateDepartment(depart \*models.Department) (int64, error)  
 GetDepartments() ([]\*models.Department, error)  
 GetDepartment(departID string) (\*models.Department, error)  
 DeleteDepartment(departID string) (int64, error)  
  
 CreateUser(user \*models.User) (int64, error)  
 GetUsers(depID string) ([]\*models.User, error)  
 GetUser(userID string) (\*models.User, error)  
 DeleteUser(userID string) (int64, error)  
  
 CreateActivity(activity \*models.Activity) (int64, error)  
 GetActivities() ([]\*models.Activity, error)  
 GetActivity(activityID string) (\*models.Activity, error)  
 DeleteActivity(activityID string) (int64, error)  
  
 GetUserActivity(userID, timeBefore, timeAfter string) (\*models.UserActivity, error)  
 GetDepartmentActivity(departID, timeBefore, timeAfter string) (\*models.DepartmentActivity, error)  
}

**Ядро для роботи з SQL БД:**

package core  
  
import (  
 "database/sql"  
 "errors"  
 "fmt"  
 "github.com/jmoiron/sqlx"  
 "github.com/sirupsen/logrus"  
 "sync"  
)  
  
// ISQLCore - common interface for all SQL databases.  
// With it, a lot of copy-paste code would be avoided during integration of new SQL DB,  
// like MySQL, MSSQL, etc  
type ISQLCore interface {  
 Exec(query string, args ...interface{}) (sql.Result, error)  
 Get(dest interface{}, query string, args ...interface{}) error  
 Pick(dest interface{}, query string, args ...interface{}) error  
  
 Open() error  
 Close() error  
 Restart() error  
 OK() error  
}  
  
// SQL core struct  
type SQL struct {  
 driver string // Driver of given SQL DB  
 connString string // Conn string of given SQL DB  
 mtx sync.RWMutex // RWMutex to improve performance  
 db \*sqlx.DB // Connection to DB  
  
 logger logrus.FieldLogger  
}  
  
// NewSQL - returns new SQL core struct  
func NewSQL(driver, connString string, logger logrus.FieldLogger) \*SQL {  
 return &SQL{  
 driver: driver,  
 connString: connString,  
 logger: logger.WithField("module", "SQLCore"),  
 }  
}  
  
// Restart - restarts the SQL db, can be user when service is down.  
func (s \*SQL) Restart() (err error) {  
 s.mtx.Lock()  
 defer s.mtx.Unlock()  
  
 entry := s.logger.WithField("func", "Restart")  
 entry.Info("Restarting SQL database...")  
  
 if err = s.close(); err != nil {  
 entry.Errorf("close redis connection error")  
 }  
  
 if err = s.open(); err != nil {  
 entry.Errorf("close redis connection error")  
 }  
  
 return  
}  
  
// Open - opens connection with given SQL db  
func (s \*SQL) Open() error {  
 s.mtx.Lock()  
 defer s.mtx.Unlock()  
  
 return s.open()  
}  
  
// open - open helper. Will be user in Restart and in Open.  
func (s \*SQL) open() (err error) {  
 s.logger.WithField("func", "open").Info("Opening DB connection...")  
  
 if s.db != nil {  
 return errors.New("sql connection already exist")  
 }  
  
 s.db, err = sqlx.Open(s.driver, s.connString)  
 if err != nil {  
 return fmt.Errorf("SQL Open(): %w", err)  
 }  
  
 return  
}  
  
// Close - closes connection with given DB  
func (s \*SQL) Close() (err error) {  
 s.mtx.Lock()  
 defer s.mtx.Unlock()  
  
 return s.close()  
}  
  
// close - close helper. Will be user in Restart and in Close.  
func (s \*SQL) close() (err error) {  
 s.logger.WithField("func", "close").Info("Closing DB connection...")  
  
 if s.db == nil {  
 return errors.New("sql connection doesn't exist")  
 }  
  
 if err := s.db.Close(); err != nil {  
 return fmt.Errorf("SQL Close(): %w", err)  
 }  
  
 s.db = nil // Some DB implementations doesn't require it, but I'b better be sure.  
  
 return  
}  
  
// OK - pings DB to check it status  
func (s \*SQL) OK() (err error) {  
 s.mtx.Lock()  
 defer s.mtx.Unlock()  
  
 if s.db == nil {  
 return errors.New("sql connection doesn't exist")  
 }  
  
 s.logger.WithField("func", "OK").Debug("Doing ping...")  
  
 // TODO: if ping will be used not only in pinger - create flag to reduce overhead.  
 return s.db.Ping()  
}  
  
// Exec - runs given query on db  
func (s \*SQL) Exec(query string, args ...interface{}) (sql.Result, error) {  
 s.mtx.Lock()  
 defer s.mtx.Unlock()  
  
 if s.db == nil {  
 return nil, errors.New("sql connection doesn't exist")  
 }  
  
 s.logger.WithField("func", "Exec").Debugf("Executing query: %s", query)  
  
 result, err := s.db.Exec(query, args...)  
  
 if err != nil {  
 return nil, fmt.Errorf("SQL Exec(): %w", err)  
 }  
  
 return result, nil  
}  
  
// Get - writes result of query to given interface.  
func (s \*SQL) Get(dest interface{}, query string, args ...interface{}) error {  
 s.mtx.RLock()  
 defer s.mtx.RUnlock()  
  
 if s.db == nil {  
 return errors.New("sql connection doesn't exist")  
 }  
  
 s.logger.WithField("func", "Get").Debugf("Get query: %s", query)  
  
 if err := s.db.Select(dest, query, args...); err != nil {  
 return fmt.Errorf("SQL conn.Select(): %w", err)  
 }  
  
 return nil  
}  
  
// Get - writes single object from sql query to given interface.  
func (s \*SQL) Pick(dest interface{}, query string, args ...interface{}) error {  
 s.mtx.RLock()  
 defer s.mtx.RUnlock()  
  
 if s.db == nil {  
 return errors.New("sql connection doesn't exist")  
 }  
  
 s.logger.WithField("func", "Pick").Debugf("Pick query: %s", query)  
  
 if err := s.db.Get(dest, query, args...); err != nil {  
 return fmt.Errorf("SQL conn.Get(): %w", err)  
 }  
  
 return nil  
}

**Обробники веб запитів:**

**Ядро апі**

package api  
  
import (  
 "activity\_api/api/auth"  
 "activity\_api/api/middleware"  
 "activity\_api/common/cancellation"  
 "activity\_api/data\_manager/cache"  
 "activity\_api/data\_manager/db/core"  
 "context"  
 "github.com/sirupsen/logrus"  
 "net/http"  
 "time"  
  
 "github.com/gorilla/mux"  
)  
  
// AApi - activity api for AAService  
type AApi struct {  
 router \*mux.Router  
 server \*http.Server  
 cancel \*cancellation.Token  
  
 auth auth.IAuth  
 token auth.IToken  
 password auth.IPassword  
  
 cacheManager cache.ICacheManager  
 sqlManager core.ISQLDatabase  
 logger logrus.FieldLogger  
}  
  
// NewAApi - returns new AApi  
func NewAApi(  
 addr string,  
 sqlManager core.ISQLDatabase,  
 cacheManager cache.ICacheManager,  
 ctx context.Context,  
 logger logrus.FieldLogger,  
) \*AApi {  
 api := &AApi{  
 router: mux.NewRouter(),  
 cancel: cancellation.NewCustomToken(ctx, 1).Close(),  
 auth: auth.NewAuth(cacheManager, logger),  
 password: new(auth.PasswordManager),  
 token: auth.NewToken(logger),  
 cacheManager: cacheManager,  
 sqlManager: sqlManager,  
 logger: logger.WithField("module", "AApi"),  
 }  
 // New http server for api  
 api.server = &http.Server{  
 Addr: addr,  
 Handler: api.router,  
 WriteTimeout: 15 \* time.Second,  
 ReadTimeout: 15 \* time.Second,  
 }  
 // init api routs  
 return api.initRoutes()  
}  
  
func (a \*AApi) initRoutes() \*AApi {  
 a.logger.WithField("func", "initRoutes").Info("Initializing routes for api...")  
 // Init auth middleware  
 authMiddleware := middleware.NewAuthMiddleware(  
 a.logger,  
 routeLogin, // Exclude some routes from authz check  
 routeRegister,  
 routeRefresh,  
 )  
 //Init logging middleware  
 loggingMiddleware := middleware.NewLoggingMiddleware(a.logger)  
 // Add logging middleware and auth middleware to router  
 a.router.Use(loggingMiddleware.LogAuthMiddleware, authMiddleware.TokenAuthMiddleware)  
 // Default route  
 a.router.NotFoundHandler = http.HandlerFunc(a.defHandler)  
 // Init authz\auth routes  
 a.registerRoute(a.Login, routeLogin, http.MethodPost)  
 a.registerRoute(a.Logout, routeLogout, http.MethodPost)  
 a.registerRoute(a.Refresh, routeRefresh, http.MethodPost)  
  
 a.registerRoute(a.Register, routeRegister, http.MethodPost)  
 a.registerRoute(a.Unregister, routeUnregister, http.MethodDelete)  
 // Init department routes  
 a.registerRoute(a.CreateDepartment, routeDepartments, http.MethodPost)  
 a.registerRoute(a.GetDepartments, routeDepartments, http.MethodGet)  
 a.registerRoute(a.GetDepartment, routeDepartment, http.MethodGet)  
 a.registerRoute(a.DeleteDepartment, routeDepartment, http.MethodDelete)  
 // Init users routes  
 a.registerRoute(a.CreateUser, routeUsers, http.MethodPost)  
 a.registerRoute(a.GetUsers, routeUsers, http.MethodGet)  
 a.registerRoute(a.GetUser, routeUser, http.MethodGet)  
 a.registerRoute(a.DeleteUser, routeUser, http.MethodDelete)  
 // Init activity routes  
 a.registerRoute(a.CreateActivity, routeActivities, http.MethodPost)  
 a.registerRoute(a.GetActivities, routeActivities, http.MethodGet)  
 a.registerRoute(a.GetActivity, routeActivity, http.MethodGet)  
 a.registerRoute(a.DeleteActivity, routeActivity, http.MethodDelete)  
 // Init activity check routes  
 a.registerRoute(a.GetDepartmentsActivity, routeDepartmentsActivity, http.MethodGet)  
 a.registerRoute(a.GetUsersActivity, routeUsersActivity, http.MethodGet)  
  
 return a  
}  
  
// registerRoute - route init helper  
func (a \*AApi) registerRoute(f func(http.ResponseWriter, \*http.Request), path string, methods ...string) {  
 a.logger.WithField("func", "registerRoute").  
 Debugf("Initializing route %s with methods: %v", path, methods)  
 a.router.HandleFunc(path, f).Name(path).Methods(methods...) // Name if set for ability to exclude route from authz  
}  
  
// Start - starts api server  
func (a \*AApi) Start() {  
 a.logger.WithField("func", "Start").Info("Staring AApi on:", a.server.Addr)  
 a.logger.Info("Listen and serve:", a.server.ListenAndServe())  
}  
  
// Close - closes api server  
func (a \*AApi) Close() error {  
 a.logger.WithField("func", "Close").Info("Closing AApi....")  
 return a.server.Close()  
}

**Роути АПІ**

package api  
  
const (  
 routeLogin = "/login"  
 routeLogout = "/logout"  
 routeRefresh = "/refresh"  
 routeRegister = "/register"  
 routeUnregister = "/unregister"  
  
 routeDepartments = "/departments"  
 routeDepartment = routeDepartments + "/{id:[0-9]+}"  
  
 routeUsers = "/users"  
 routeUser = routeUsers + "/{id:[0-9]+}"  
  
 routeActivities = "/activities"  
 routeActivity = routeActivities + "/{id:[0-9]+}"  
  
 routeControl = "/control"  
 routeUsersActivity = routeControl + "/user/{id:[0-9]+}"  
 routeDepartmentsActivity = routeControl + "/department/{id:[0-9]+}"  
)