Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт цифровых технологий, электроники и физики (ИЦТЭФ)

Кафедра вычислительной техники и электроники (ВТиЭ)

**Отчет по лабораторной работе**

по курсу “Технологии разработки программного обеспечения”

“**Проектный менеджер”**

Выполнил: студенты 506 группы:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.В. Осипенко

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.С. Маркевцев

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ф.С. Калугарев

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.А. Дмуха

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Проверил: доц. кафедры ВТиЭ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Грязнов

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Лабораторная работа защищена

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

[Цель работы 3](#_v1jr9h3jq19)

[Команда 3](#_9y1a1eb1904f)

[Проектирование 3](#_n89eq3fp1q32)

[1. Описание 3](#_56indidjo4k8)

[2. Системный дизайн 3](#_z5q4fjk6v7zd)

[2.1. Функциональные требования 3](#_w3ofmvdvy6zx)

[2.2. Навигация 4](#_3lbj82tg23vq)

[2.3. Страницы 4](#_4djghgmmagzi)

[2.3.1. Index page 4](#_lhkvknwry15f)

[2.3.2. Login/Register page 5](#_gccd9061iruz)

[2.3.3. Profile page 5](#_po31k1axv1re)

[2.3.3.1. Overview 5](#_p8lfs73o9f8g)

[2.3.3.2. Projects/Tasks 5](#_ueabwcc13z9t)

[2.3.3.3. Settings 5](#_rm7vk0ktwcf8)

[2.3.4. Project page 5](#_mo3oxds1fcg4)

[2.3.4.1. Overview 5](#_awbgfxetzgi1)

[2.3.4.2. Tasks 5](#_357in232f5j7)

[2.3.4.3. Settings 6](#_trqsrzhpkyln)

[2.4. Домены и их зависимости 6](#_1u0f0o5ypn7g)

[2.5. Модели 6](#_4q1mq9czhafw)

[Реализация 7](#_9132xrjpewmr)

[1. Дизайн 7](#_luipirxh3tcg)

[2. Frontend 9](#_291cu0kfi9ja)

[3. База данных 9](#_neujxdva4jkv)

[4. Backend 11](#_j54qrzh2clcf)

[Вывод 19](#_ox6v8vc1679a)

# 

# Цель работы

Разработать простое онлайн приложения для ведения проектного менеджмента. Получить навыки командной разработки.

# Команда

* Осипенко Д. – Тим. лид., проектировщик, backend разработчик.
* Маркевцев В. – Дизайнер.
* Калугарев Ф. – frontend разработчик.
* Дмуха П. – разработчик баз данных.

# Проектирование

## 1. Описание

Projager – это веб приложение с простым и интуитивным дизайном, созданной для осуществления эффективного и понятного проектного менеджмента, с участием одного или нескольких человек.

## 2. Системный дизайн

### 2.1. Функциональные требования

- Создать новый проект

- Создать новую задачу, относящуюся к проекту

- Изменить статус существующей задачи

- Изменить описание и название задачи

- Изменить название и описание проекта

- Удалить задачу, относящуюся к проекту

- Создать новый аккаунт

- Авторизоваться существующим аккаунтом

- Изменить имя пользователя

- Изменить пароль пользователя

- Пригласить пользователя к проекту

- Создатель проекта может назначить исполнителя задачи

- Получить список задач, относящихся к проекту

- Получить список проектов, относящихся к пользователю

- Получить список задач, относящихся к пользователю

- Получить список пользователей, относящихся к проекту

### 2.2. Навигация

Визуальное представление навигации по страницам приложения.

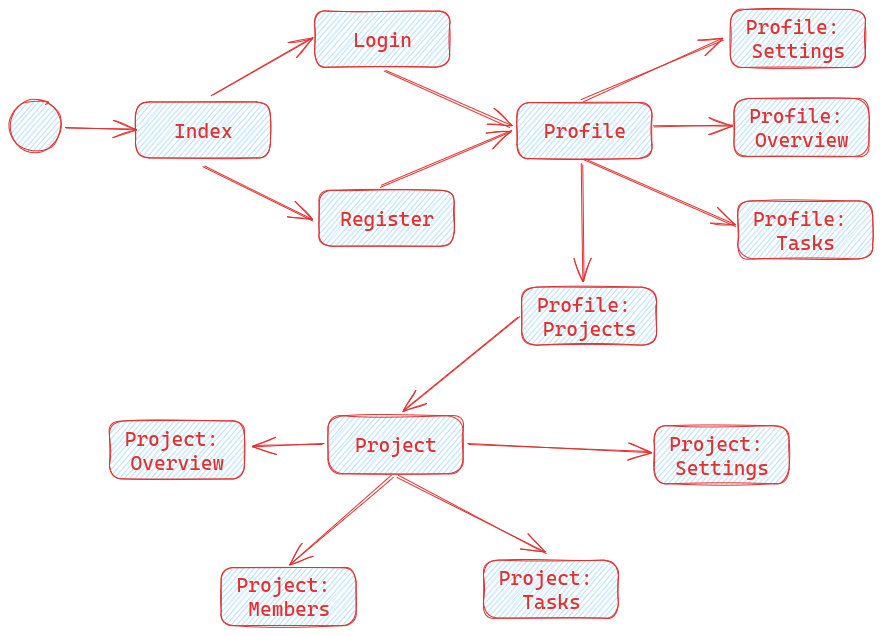


Рис 1. Карта навигации по сайту.

### 2.3. Страницы

#### 2.3.1. Index page

Входная точка в приложение. Содержит в себе логотип и название приложения, ссылочное меню (Вход/Регистрация или Профиль/Выход), различные слоганы и описание приложения, изображение человека/команды выполняющей какую либо проектную деятельность.

#### 2.3.2. Login/Register page

Служит для создания нового или авторизацией для существующего пользователя. Содержит в себе логотип и меню (Вход/Регистрация), форму входа/регистрации, абстрактное изображение.

#### 2.3.3. Profile page

Служит для управления пользовательским аккаунтом, просмотром его активности и навигации по относящимся к нему проектам и задачам.

##### 2.3.3.1. Overview

Используется для отображения активности и обновлений, связанных с данным пользователем. Содержит график активности, список приглашений, ленту обновлений с связанными проектами и задачами.

##### 2.3.3.2. Projects/Tasks

Используется для отображения проектов/задач, частью которых является пользователь. Содержит список проектов/задач.

##### 2.3.3.3. Settings

Используется для осуществления настроек пользовательского аккаунта. Содержит список настроек и действий с аккаунтом.

#### 2.3.4. Project page

Служит для отображения информации, связанной с конкретным проектом.

##### 2.3.4.1. Overview

Используется для отображения активности проекта. Содержит график активности, списоки обновленных задач и самых результативных пользователей.

##### 2.3.4.2. Tasks

Используется для отображения задач/пользователей и манипуляций с ними. Содержит список задач/пользователей.

##### 2.3.4.3. Settings

Используется для настройки проекта. Содержит настраиваемые поля и действия.

### 2.4. Домены и их зависимости

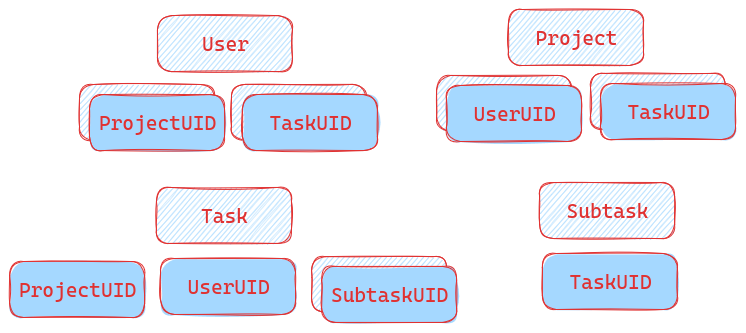


Рис. 2 Отображение основных доменов и их зависимостей.

### 2.5. Модели

1. User

- UID: string

- Email: string

- PasswordHash: string

- FirstName: string

- LastName: string

- Projects: []Project.UID

- Tasks: []Task.UID

2. Project

- UID: string

- Title: string

- Description: string

- Creator: User.UID

- Members: []User.UID

- Tasks: []Task.UID

3. Task

- UID: string

- Name: string

- Status: string

- ProjectUID: Project.UID

- Todo: []Subtask.UID

- User: User.UID

4. Subtask

- UID: string

- Todo: string

- Done: bool

- Task: Task.UID

# Реализация

## 1. Дизайн

Дизайн выполнялся с помощью бесплатного приложения Figma, позволяющего создавать дизайн любого уровня и сложности. Часть результатов работы приведены на рисунках 4 и 5. Цветовая схема была создана с помощью бесплатного приложения Coolors.

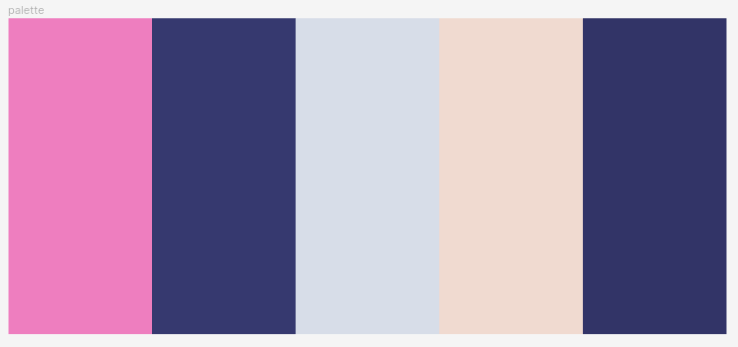


Рис. 3 Цветовая схема.



Рис. 4 Пример дизайна страницы Index.

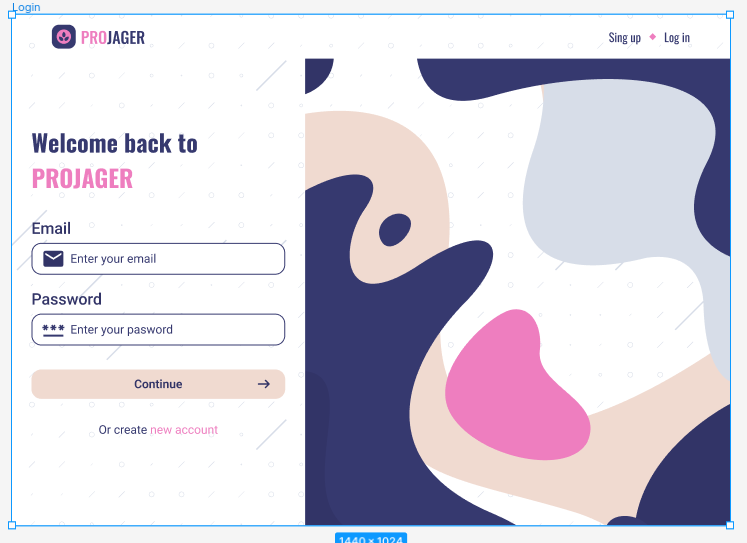


Рис. 5 Пример дизайна страницы Login.

## 2. Frontend

Страницы создавались на основе представленного ранее дизайна с использование HTML, и в качесте стилей использовались библиотеки TailwindCSS и DaisyUI.



Рис. 6 Пример созданной страницы Index.

## 3. База данных

В качестве базы данных была выбрана Sqlite3, за ее простоту и легковесность. Для нее была создана схема и пачка запросов. На основе этих схем и запросов, с помощью инструмента SQLC был сгенерирован рабочий Golang код, с помощью которого можно было обращаться к базе данных.

**Листинг 1**: Схема базы данных.

create table if not exists users (

uid text primary key,

email text not null unique,

password text not null,

first\_name text,

last\_name text

);

create table if not exists projects (

uid text primary key,

title text not null,

description text not null,

creator text not null,

owner text not null

);

create table if not exists tasks (

uid text primary key,

name text not null,

project\_uid text not null,

user\_uid text not null,

status text not null

);

**Листинг 2**: Запросы к базе данных.

-- name: CreateUser :exec

insert into users (uid, email, password, first\_name, last\_name)

values (?, ?, ?, ?, ?);

-- name: GetUserUID :one

select uid from users

where email = ? and password = ?;

-- name: GetProjects :many

select \* from projects;

-- name: CreateProject :exec

insert into projects (uid, title, description, creator, owner)

values (?, ?, ?, ?, ?);

-- name: UpdateProject :exec

update projects set

title = ?,

description = ?

where uid = ?;

-- name: GetTasks :many

select \* from tasks;

-- name: CreateTask :exec

insert into tasks (uid, name, project\_uid, user\_uid, status)

values (?, ?, ?, ?, ?);

-- name: UpdateTask :exec

update tasks set

name = ?,

status = ?

where uid = ?;

-- name: DeleteTask :exec

DELETE FROM tasks

WHERE uid = ?;

## 4. Backend

Сервер разработан на языке Golang, с использованием библиотеки Echo. Шаблоны реализованы с помощью Templ и HTMX. Архитектура сервера представляет из себя слоистую архитектуру, где четко выделены:

- cmd – главный модуль, выполняющий запуск сервера.

- handlers – модуль, осуществляющий обработку HTTP запросов.

- auth – модуль, осуществляющий авторизацию и аутентификацию.

- services – бизнес логика приложения.

- models – основные модели приложения.

- db – сгенерированный модуль, осуществляющий взаимодействие с базой данных.

**Листинг 3**: main.go

package main

import (

"log"

"github.com/labstack/echo/v4"

"projager.io/handlers"

)

func main() {

app := echo.New()

app.Static("/static", "public")

if err := handlers.InitRoutes(app); err != nil {

log.Fatalf("can't init routes: %s", err)

}

if err := app.Start(":8080"); err != nil {

log.Fatal(err)

}

}

**Листинг 4**: handlers.go

package handlers

import (

"context"

"database/sql"

"io"

"github.com/a-h/templ"

"github.com/labstack/echo/v4"

\_ "github.com/mattn/go-sqlite3"

"projager.io/auth"

"projager.io/db"

"projager.io/services"

)

func InitRoutes(app \*echo.Echo) error {

queries, err := openDB()

if err != nil {

return err

}

index := Index{}

index.InitRoutes(app)

login := Login{

UserService: &services.UserService{

Queries: queries,

},

}

login.InitRoutes(app)

signup := Signup{

UserService: &services.UserService{

Queries: queries,

},

}

signup.InitRoutes(app)

logout := Logout{}

logout.InitRoutes(app)

profile := Profile{

Content: context.Background(),

ProjectService: &services.ProjectService{

Queries: queries,

},

}

profile.InitRoutes(app)

project := Project{

Content: context.Background(),

TaskService: &services.TaskService{

Queries: queries,

},

ProjectService: &services.ProjectService{

Queries: queries,

},

}

project.InitRoutes(app)

return nil

}

func render(component templ.Component, ctx context.Context, resp io.Writer) error {

return component.Render(ctx, resp)

}

func getUserUID(ctx echo.Context) string {

authCookie, err := ctx.Cookie("auth")

if err != nil {

return ""

}

return auth.GetUserUID(authCookie.Value)

}

func openDB() (\*db.Queries, error) {

d, err := sql.Open("sqlite3", "./db/sqlite.db")

if err != nil {

return nil, err

}

return db.New(d), nil

}

**Листинг 5**: models.go

package models

import (

"database/sql"

"projager.io/db"

)

type Status struct {

status string

}

func (s \*Status) Status() string {

return s.status

}

var (

StatusInProgress = Status{"In progress"}

StatusIdle = Status{"Idle"}

StatusComplite = Status{"Complite"}

)

type User struct {

UID string

Email string

PasswordHash string

FirstName string

LastName string

}

type Project struct {

UID string

Title string

Description string

Creator string

}

type Task struct {

UID string

Name string

Status Status

Project string

User string

}

type Subtask struct {

UID string

Todo string

IsDone bool

Task string

}

func UserFromDTO(u db.User) User {

return User{

UID: u.Uid,

Email: u.Email,

PasswordHash: u.Password,

FirstName: u.FirstName.String,

LastName: u.LastName.String,

}

}

func UserToDTO(u User) db.User {

return db.User{

Uid: u.UID,

Email: u.Email,

Password: u.PasswordHash,

FirstName: sql.NullString{

String: u.FirstName,

},

LastName: sql.NullString{

String: u.LastName,

},

}

}

func ProjectFromDTO(p db.Project) Project {

return Project{

UID: p.Uid,

Title: p.Title,

Description: p.Description,

Creator: p.Creator,

}

}

func ProjectToDTO(p Project) db.Project {

return db.Project{

Uid: p.UID,

Title: p.Title,

Description: p.Description,

Creator: p.Creator,

Owner: p.Creator,

}

}

func TaskFromDTO(t db.Task) Task {

status := Status{}

switch t.Status {

case "Complite":

status = StatusComplite

case "In progress":

status = StatusInProgress

case "Idle":

fallthrough

default:

status = StatusIdle

}

return Task{

UID: t.Uid,

Name: t.Name,

Project: t.ProjectUid,

Status: status,

}

}

func TaskToDTO(t Task) db.Task {

return db.Task{

Uid: t.UID,

Name: t.Name,

ProjectUid: t.Project,

UserUid: t.User,

Status: t.Status.Status(),

}

}

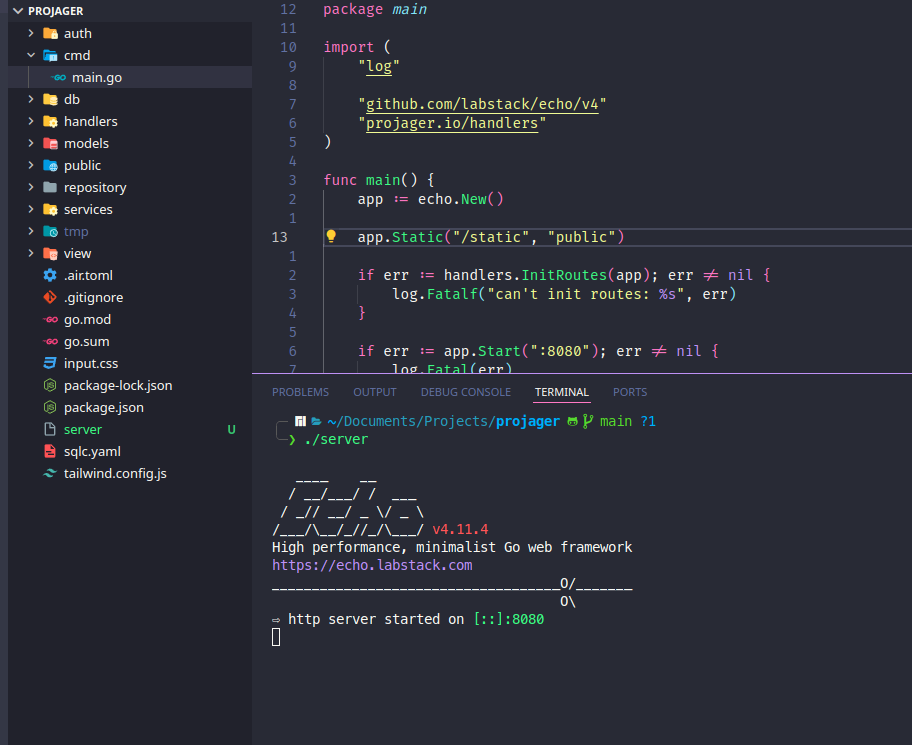


Рис. 7 Запуск сервера

# Вывод

В Ходе данной работы была осуществлена командная деятельность по созданию веб приложения. Получены навыки командной работы, разработки веб приложений.