Project Report: Building a Task Management Application with Go Backend and React Frontend

Table of Contents

---

1. Executive Summary

This report details the development of a task management application built with a Go backend and React frontend. The project leverages RESTful APIs, PostgreSQL database management, and React components to deliver a user-friendly interface. The main objective was to create a full-stack application that allows users to create, update, delete, and manage tasks seamlessly.

---

2. Introduction

Task management applications are essential for organizing tasks and increasing productivity. This project explores the process of building such an application using modern technologies—Go for backend development and React for frontend. Go was selected for its performance and simplicity in building RESTful services, while React enables the creation of responsive and modular user interfaces.

---

3. Project Objectives

- Develop a full-stack task management application with Go and React.
- Implement CRUD operations and more complex queries using GORM.
- Build RESTful API endpoints for task management.
- Ensure a responsive and user-friendly UI with React components.
- Use JSON for seamless data exchange between the backend and frontend.

---

4. Introduction to Go

Development Environment

- Installed Go from the official website.
- Configured the environment with `$GOPATH` and `$GOROOT`.

Syntax Basics

Example of variable declaration and a simple function in Go:

```
package main

import "fmt"

func main() {
    message := "Hello, Go!"
    fmt.Println(message)
}
```

This basic setup introduced control structures such as loops and conditional statements.

---

5. Object-Oriented Programming in Golang

Structs and Methods

The `Task` struct models the task entity:

```
type Task struct {
    gorm.Model
    ID          uint      `json:"ID" gorm:"primaryKey"`
    Title       string    `json:"title" gorm:"not null"`
    Description string    `json:"description"`
    DueDate     time.Time `json:"dueDate"`
    Priority    Priority  `json:"priority"`
    Status      string    `json:"status"`
    UserID      uint      `json:"userId"`
    User        User      `json:"user"`
}

type User struct {
    gorm.Model
    Email     string `json:"email" gorm:"uniqueIndex;not null"`
    Password  string `json:"-" gorm:"not null"` // "-" prevents password from being included in JSON
    FirstName string `json:"firstName"`
    LastName  string `json:"lastName"`
    Role      string `json:"role" gorm:"default:'user'"`
    Tasks     []Task `json:"tasks,omitempty" gorm:"foreignKey:UserID"`
}
```

Methods for CRUD operations were implemented in service files to encapsulate task behavior.

Encapsulation

Data and methods were encapsulated within services, ensuring that task manipulation logic resides in the backend.

---

6. Dependency Management

Go Modules

The project uses Go modules for dependency management: go.mod:

```
module backend

go 1.23.2

require (
        github.com/rs/cors v1.11.1
        gorm.io/driver/postgres v1.5.9
)

require (
        github.com/bytedance/sonic v1.11.6 // indirect
        github.com/bytedance/sonic/loader v0.1.1 // indirect
        github.com/cloudwego/base64x v0.1.4 // indirect
        github.com/cloudwego/iasm v0.2.0 // indirect
        github.com/gabriel-vasile/mimetype v1.4.3 // indirect
        github.com/gin-contrib/sse v0.1.0 // indirect
        github.com/go-playground/locales v0.14.1 // indirect
        github.com/go-playground/universal-translator v0.18.1 // indirect
        github.com/go-playground/validator/v10 v10.20.0 // indirect
        github.com/goccy/go-json v0.10.2 // indirect
        github.com/json-iterator/go v1.1.12 // indirect
        github.com/klauspost/cpuid/v2 v2.2.7 // indirect
        github.com/leodido/go-urn v1.4.0 // indirect
        github.com/mattn/go-isatty v0.0.20 // indirect
        github.com/modern-go/concurrent v0.0.0-20180306012644-bacd9c7ef1dd // indirect
        github.com/modern-go/reflect2 v1.0.2 // indirect
        github.com/pelletier/go-toml/v2 v2.2.2 // indirect
        github.com/twitchyliquid64/golang-asm v0.15.1 // indirect
        github.com/ugorji/go/codec v1.2.12 // indirect
        golang.org/x/arch v0.8.0 // indirect
        golang.org/x/net v0.25.0 // indirect
        golang.org/x/sys v0.20.0 // indirect
        google.golang.org/protobuf v1.34.1 // indirect
        gopkg.in/yaml.v3 v3.0.1 // indirect
        //.....
)
```

go.sum:

```
github.com/bytedance/sonic v1.11.6 h1:oUp34TzMlL+OY1OUWxHqsdkgC/Zfc85zGqw9siXjrc0=
github.com/bytedance/sonic v1.11.6/go.mod h1:LysEHSvpvDySVdC2f87zGWf6CIKJcAvqab1ZaiQtds4=
github.com/bytedance/sonic/loader v0.1.1 h1:c+e5Pt1k/cy5wMveRDyk2X4B9hF4g7an8N3zCYjJFNM=
github.com/bytedance/sonic/loader v0.1.1/go.mod h1:ncP89zfokxS5LZrJxl5z0UJcsk4M4yY2JpfqGeCtNLU=
github.com/cloudwego/base64x v0.1.4 h1:jwCgWpFanWmN8xoIUHa2rtzmkd5J2plF/dnLS6Xd/0Y=
github.com/cloudwego/base64x v0.1.4/go.mod h1:0zlkT4Wn5C6NdauXdJRhSKRlJvmclQ1hhJgA0rcu/8w=
github.com/cloudwego/iasm v0.2.0 h1:1KNIy1I1H9hNNFEEH3DVnI4UujN+1zjpuk6gwHLTssg=
github.com/cloudwego/iasm v0.2.0/go.mod h1:8rXZaNYT2n95jn+zTI1sDr+IgcD2GVs0nlbbQPiEFhY=
github.com/davecgh/go-spew v1.1.0/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/davecgh/go-spew v1.1.1/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/gabriel-vasile/mimetype v1.4.3 h1:in2uUcidCuFcDKtdcBxlR0rJ1+fsokWf+uqxgUFjbI0=
github.com/gabriel-vasile/mimetype v1.4.3/go.mod h1:d8uq/6HKRL6CGdk+aubisF/M5GcPfT7nKyLpA0lbSSk=
github.com/gin-contrib/sse v0.1.0 h1:Y/yl/+YNO8GZSjAhjMsSuLt29uWRFHdHYUb5lYOV9qE=
github.com/gin-contrib/sse v0.1.0/go.mod h1:RHrZQHXnP2xjPF+u1gW/2HnVO7nvIa9PG3Gm+fLHvGI=
github.com/gin-gonic/gin v1.10.0 h1:nTuyha1TYqgedzytsKYqna+DfLos46nTv2ygFy86HFU=
github.com/gin-gonic/gin v1.10.0/go.mod h1:4PMNQiOhvDRa013RKVbsiNwoyezlm2rm0uX/T7kzp5Y=
//.....
```

Project Structure

The project is structured as follows:

```
task-management/
├── backend/
│   ├── cmd/
│   │   └── main.go
│   ├── internal/
│   │   ├── models/
│   │   │   ├── task.go
│   │   │   └── user.go
│   │   ├── handlers/
│   │   │   ├── task_handler.go
│   │   │   └── user_handler.go
│   │   ├── middleware/
│   │   │   ├── auth.go
│   │   │   └── logging.go
│   │   ├── repository/
│   │   │   └── database.go
│   │   └── service/
│   │       ├── task_service.go
│   │       └── user_service.go
│   ├── config/
│   │   └── config.go
│   ├── migrations/
│   │   └── init.sql
│   └── go.mod
├── frontend/
│   ├── public/
│   ├── src/
│   │   ├── components/
│   │   │   ├── Task/
│   │   │   ├── User/
│   │   │
│   │   ├── contexts/
│   │   ├── services/
│   │   ├── types/
│   │   └── App.tsx
│   ├── package.json
│   └── tsconfig.json
└── README.md
```

---

7. Working with Database

7.1 Part 1: Basic CRUD Operations

- **Database Setup:** PostgreSQL was configured to manage tasks.
- **CRUD Operations:** GORM was used to simplify CRUD operations:

```
import (
    "gorm.io/gorm"
    "gorm.io/driver/postgres"
    "backend/internal/models"
)

type Repository struct {
    db *gorm.DB
}

func NewUserRepository(db *gorm.DB) *UserRepository {
    return &UserRepository{db: db}
}

func (r *Repository) DB() *gorm.DB {
    return r.db
}

func (r *Repository) CreateTask(task *models.Task) error {
    return r.db.Create(task).Error
}

func (r *UserRepository) CreateUser(user *models.User) error {
    return r.db.Create(user).Error
}
```

7.2 Part 2: Complex Queries and Migrations

- **Complex Queries:** Queries were implemented to filter tasks by user and status.
- **Migrations:** GORM auto-migration ensured smooth schema updates:

```
func NewRepository(dsn string) (*Repository, error) {
    db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
    if err != nil {
        return nil, err
    }

    // Auto migrate schemas
    err = db.AutoMigrate(&models.User{}, &models.Task{})
    if err != nil {
        return nil, err
    }

    return &Repository{db: db}, nil
}



func (r *Repository) GetTaskByID(id uint) (*models.Task, error) {
    var task models.Task
    err := r.db.Preload("User").First(&task, id).Error
    return &task, err
}

func (r *UserRepository) GetUserByEmail(email string) (*models.User, error) {
    var user models.User
    err := r.db.Where("email = ?", email).First(&user).Error
    if err != nil {
        return nil, err
    }
    return &user, nil
}
```

---

8. Connecting with Frontend

API Endpoints

The following endpoints were created:

- **POST** `/api/auth/register`: Register a new user.
- **GET** `/api/auth/login` : Login user with JWT
- **POST** `/api/tasks`: Create a new task.
- **GET** `/api/tasks`: Retrieve all tasks.
- **PUT** `/api/tasks/:id`: Update a task.
- **DELETE** `/api/tasks/:id`: Delete a task.

```
        api := router.Group("/api")
        {
                api.POST("/auth/register", userHandler.Register)
                api.POST("/auth/login", userHandler.Login)

                authGroup := api.Group("/tasks")
                authGroup.Use(middleware.AuthMiddleware(jwtSecret))
                {
                        authGroup.POST("", taskHandler.CreateTask)
                        authGroup.GET("", taskHandler.GetTasks)
                        authGroup.PUT("/:id", taskHandler.UpdateTask)
                        authGroup.DELETE("/:id", taskHandler.DeleteTask)
                }
        }
```

JSON Handling

The API exchanges data in JSON format:

```
{
  "title": "Finish report",
  "description": "Complete the project report by Monday",
  "dueDate": "2024-10-23T10:00:00Z"
  "priority": medium
}
```

---

9. Frontend Development with React

9.1 Components and Routing

React Setup

The React application was created using:

```
npx create-react-app task-manager-frontend --template typescript
```

Key Components

- **`TaskList` Component:** Displays a list of tasks.

```
    <div>
        <h1>Task List</h1>
        <TaskForm onSuccess={fetchTasks} />
        {tasks.map((task) => (
            <div key={task.ID}>
                <h3>{task.title}</h3>
                <p>Task ID: {task.ID}</p>
                <p>{task.description}</p>
                <button onClick={() => handleEdit(task)}>Edit</button>
                <button onClick={() => handleDelete(task.ID)}>Delete</button>
            </div>
        ))}

        {editingTask && (
            <form onSubmit={handleUpdate}>
                <input
                    type="text"
                    value={editingTask.title}
                    onChange={(e) =>
                        setEditingTask({ ...editingTask, title: e.target.value })
                    }
                    required
                />
                <textarea
                    value={editingTask.description}
                    onChange={(e) =>
                        setEditingTask({ ...editingTask, description: e.target.value })
                    }
                />
                <button type="submit">Save</button>
                <button onClick={() => setEditingTask(null)}>Cancel</button>
            </form>
        )}
    </div>
```

- **`TaskForm` Component:** Allows users to add or edit tasks.

```
    return (
        <form onSubmit={handleSubmit}>
            <input
                type="text"
                placeholder="Title"
                value={formData.title}
                onChange={(e) => setFormData({ ...formData, title: e.target.value })}
                required
            />
            <textarea
                placeholder="Description"
                value={formData.description}
                onChange={(e) => setFormData({ ...formData, description: e.target.value })}
            />
            <input
                type="datetime-local"
                value={formData.dueDate}
                onChange={(e) => setFormData({ ...formData, dueDate: e.target.value })}
                required
            />
            <select
                value={formData.priority}
                onChange={(e) => setFormData({ ...formData, priority: e.target.value as 'low' | 'medium' | 'high' })}
            >
                <option value="low">Low</option>
                <option value="medium">Medium</option>
                <option value="high">High</option>
            </select>
            <button type="submit">Create Task</button>
        </form>
    );
```

9.2 Pages Overview

Home Page:

Displays the greeting and provides a link to tasks.

```
<div>
    <h1>Welcome to Task Manager</h1>
    <p>Manage your tasks efficiently!</p>
    <Link to="/tasks">View Tasks</Link>
</div>
```

TaskList Page:

Contains a form for RUD new tasks and link to create task.

```
<div>
    <Link to="/tasks/create">Create Tasks</Link>
    <h1>Tasks</h1>
    {tasks.map((task) => (
        <div key={task.ID}>
            <h3>{task.title}</h3>
            <p>{task.description}</p>
            <Link to={`/tasks/${task.ID}`}>View Details</Link>
            <button onClick={() => handleEdit(task)}>Edit</button>
            <button onClick={() => handleDelete(task.ID)}>Delete</button>
        </div>
    ))}

    {editingTask && (
        <form onSubmit={handleUpdate}>
            <input
                type="text"
                value={editingTask.title}
                onChange={(e) => setEditingTask({ ...editingTask, title: e.target.value })}
            />
            <textarea
                value={editingTask.description}
                onChange={(e) => setEditingTask({ ...editingTask, description: e.target.value })}
            />
            <button type="submit">Update Task</button>
            <button onClick={() => setEditingTask(null)}>Cancel</button>
        </form>
    )}
</div>
```

Create Task Page:

Allows users to modify existing tasks.

```
        <form onSubmit={handleSubmit}>
            <input
                type="text"
                placeholder="Title"
                value={formData.title}
                onChange={(e) => setFormData({ ...formData, title: e.target.value })}
                required
            />
            <textarea
                placeholder="Description"
                value={formData.description}
                onChange={(e) => setFormData({ ...formData, description: e.target.value })}
            />
            <input
                type="datetime-local"
                value={formData.dueDate}
                onChange={(e) => setFormData({ ...formData, dueDate: e.target.value })}
                required
            />
            <select
                value={formData.priority}
                onChange={(e) => setFormData({ ...formData, priority: e.target.value as 'low' | 'medium' | 'high' })}
            >
                <option value="low">Low</option>
                <option value="medium">Medium</option>
                <option value="high">High</option>
            </select>
            <button type="submit">Add Task</button>
        </form>
    );
```

---

10. Conclusion

This project successfully demonstrates the development of a task management application using Go for the backend and React for the frontend. The combination of these technologies provided a robust and scalable solution for managing tasks. Future improvements may include adding authentication using JWT, enhancing the UI/UX, and deploying the application to the cloud for production use.

---

11. References

- Official Go Documentation: https://golang.org/doc (https://golang.org/doc)
- React Documentation: https://reactjs.org/ (https://reactjs.org/)
- GORM Documentation: https://gorm.io/docs (https://gorm.io/docs)
- PostgreSQL Documentation: https://www.postgresql.org/docs/ (https://www.postgresql.org/docs/)

---

12. Appendices

Appendix A: Full API Route List

Method	Endpoint	Description
POST	`/api/auth/register`	Register new user
GET	`/api/auth/login`	Login user with JWt
GET	`/api/tasks`	Retrieve all tasks
POST	`/api/tasks`	Create a new task

Method	Endpoint	Description
PUT	`/api/tasks/:id`	Update a specific task
DELETE	`/api/tasks/:id`	Delete a specific task

Appendix B: Task Data Model

```
{
  "ID": 1,
  "title": "Complete project report",
  "description": "Finish writing the final project report.",
  "dueDate": "2024-10-23T10:00:00Z",
  "priority": "high",
  "status": "pending",
  "userId": 1
}
```

This report template meets the requirements with a detailed breakdown of all project components. You can now modify it with project-specific insights or data before submission.