Ministerium für Bildung und Wissenschaft der Kirgisischen Republik
KIRGISISCH-DEUTSCHES INSTITUT FÜR ANGEWANDTE INFORMATIK

SEMESTERARBEIT

zum Thema: “Datenbank zur Anwesenheit von Studenten”

Erstellt von:

Studierende Gruppe ABC-1-23:

Author 1_____

Author2_____

Betreuer:

Bergamot P.I._____

Natrinomatov R.T._____

Berater für deutsche Sprache:

Nitaratieva A.T._____

Sarkymbaevava A.Zh._____

Bischkek 2024

Inhaltsverzeichnis

Einführung

Im Zeitalter der digitalen Transformation ist die Verwaltung von Anwesenheitssystemen für Studierende immer wichtiger geworden. Eine effektive und zuverlässige Anwesenheitsverfolgung ist für Bildungseinrichtungen von entscheidender Bedeutung, damit sie die Leistung und das Engagement der Schüler verfolgen und verbessern können. Die Wahl dieses Themas ergibt sich aus der wachsenden Notwendigkeit, traditionelle papierbasierte und manuelle Anwesenheitsmethoden durch moderne automatisierte Systeme zu ersetzen [1], die sowohl die Genauigkeit als auch die Benutzerfreundlichkeit verbessern.

Gegenstand dieser Arbeit ist ein in digitaler Form funktionierendes studentisches Anwesenheitssystem. Gegenstand der Forschung ist die Entwicklung einer Datenbank zur Unterstützung des Betriebs des Anwesenheitssystems [2], einschließlich Anwesenheitsaufzeichnungen, Berichterstattung und Integration mit anderen Bildungssystemen.

Um die Ziele des Projekts zu erreichen werden folgende Forschungsmethoden eingesetzt: Analyse aktueller Trends und Anforderungen im Bereich Anwesenheitsmanagement; vergleichende Analyse bestehender Anwesenheitssysteme [2]; Entwurf und Entwicklung einer Datenbank zur Verwaltung des Anwesenheitssystems; Testen und Bewerten der Wirksamkeit des entwickelten Systems.

Das Hauptziel des Projekts ist die Schaffung eines modernen und komfortablen Anwesenheitssystems [3], das sich durch eine einfache Schnittstelle, hohe Genauigkeit und effiziente Verwaltung auszeichnet. Um dieses Ziel zu erreichen, müssen folgende Aufgaben gelöst werden: Durchführung einer Marktanalyse und Ermittlung der Grundanforderungen an Anwesenheitssysteme; Entwicklung einer Datenbankstruktur [4], die eine effektive Verwaltung der Anwesenheitsdaten gewährleistet; Implementierung der Datenbankfunktionalität und Integration in die Benutzeroberfläche.

Das Projekt besteht aus einer Einleitung, einem Hauptteil mit 2 Kapiteln, einem Fazit und einem Verzeichnis der verwendeten Quellen. Kapitel 1 besteht aus 4 Unterkapiteln, in denen der Technologie-Stack, die Projektstruktur, die Projektarchitektur und die Quellcodebeschreibung beschrieben werden. Kapitel

2 besteht aus 4 Unterkapiteln, die die Funktionalität des Servers, die Struktur des Projekts, die Architektur des Projekts und eine Beschreibung des Quellcodes beschreiben.

Die Struktur der Arbeit umfasst die folgenden Phasen: Analyse und Untersuchung des Marktes für Anwesenheitssysteme; Entwicklung technischer Spezifikationen und Datenbankdesign; Implementierung und Test des entwickelten Systems; Implementierung und Bewertung der Wirksamkeit des Anwesenheitssystems.

Ziel des Projekts ist es, ein einzigartiges und effektives Anwesenheitssystem zu schaffen, das durch innovative Lösungen und Best Practices im Bildungsbereich überzeugt. Wir streben danach, führend im Anwesenheitsmanagement zu sein, indem wir Bildungseinrichtungen die besten Verwaltungsfunktionen und ein nahtloses Benutzererlebnis für Studenten bieten.

Teilnehmerbeitrag. Amit Kurbanov hat eine mobile Anwendung für Schüler und eine Desktop-Webanwendung für Lehrer entwickelt, die in Kapitel 1 – Mobile Anwendung dargestellt wird. Baktybekov Nursltan hat den Datenbank- und Serverteil der Anwendung entwickelt, was in Kapitel 2 – Server widergespiegelt wird.

Kapitel 1. Mobile Anwendung

Für Studierende ist eine mobile Anwendung erforderlich, mit der sie: ihren Stundenplan einsehen und ihre Anwesenheit im System durch Scannen eines Codes registrieren können

1.1. Funktionen mobiler Apps

- Registrierung/Anmeldung zum Konto
- Zeitplan anzeigen
- QR-Code scannen

1.1.1. Schnittstelle für mobile Anwendungen

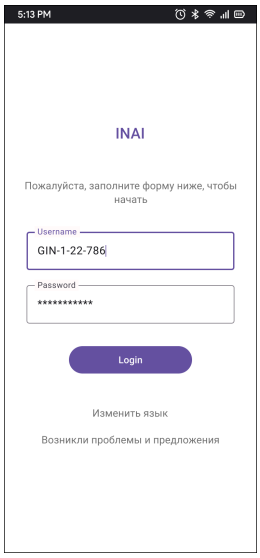

Abbildung 1: Loginseite

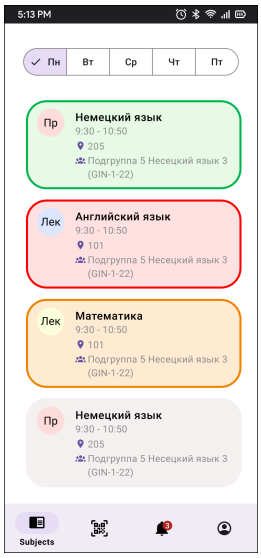

Abbildung 2: Seite „Zeitplan“

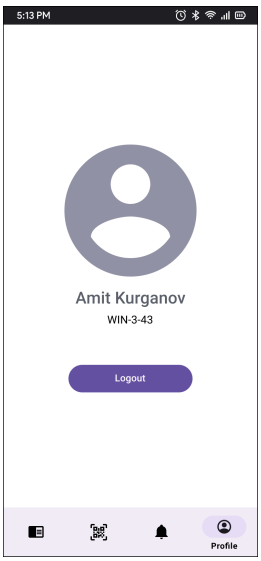

Abbildung 3: Seite „Einstellungen“

Abbildung 4: QR-Code-Scanseite

1.1.2. Verwendete Techniken

Programmiersprache für mobile [5] Anwendungen: Kotlin [6] Programmiersprache für Web Verwendung: JS

Entwickler: Android Studio, Neovim

1.2. Projektstruktur

```
├── AndroidManifest.xml
├── java/com/marko112
│   └── lecturepresenceapp
│       ├── MainActivity.kt
│       ├── MainEmptyActivity.kt
│       ├── model
│       │   ├── AuditoryExercise.kt
│       │   ├── IActivity.kt
│       │   ├── Lecture.kt
│       │   ├── Student.kt
│       │   └── Subject.kt
│       ├── professor
│       │   ├── LectureDetailsFragment.kt
│       │   ├── LectureHistoryFragment.kt
│       │   ├── Professor.kt
│       │   ├── QRCodeFragment.kt
│       │   └── SubjectDetailsProfessorFragment.kt
│       ├── student
│       │   ├── CameraFragment.kt
│       │   ├── StudentRecyclerAdapter.kt
│       │   ├── SubjectDetailsFragment.kt
│       │   └── SuccessfulScanFragment.kt
│       ├── ui
│       │   ├── adapter
│       │   │   └── LectureRecyclerAdapter.kt
│       │   ├── LoginActivity.kt
│       │   ├── MySubjectsFragment.kt
│       │   ├── MySubjectsFragmentProfessor.kt
│       │   ├── RegisterActivity.kt
│       │   ├── SubjectRecyclerAdapter.kt
│       │   └── WelcomeFragment.kt
│       └── util
│           ├── CaptureActivityPortrait.kt
│           ├── ConstructionExercise.kt
│           ├── LaboratoryExercise.kt
│           └── LectureExercise.kt
└── proto
    └── jounal.proto
```

Listing 1: Mobile App-Projektstruktur

1.2.1. Projektarchitektur

Es wurde die MVI-Architektur verwendet. Modellansichtsabsicht

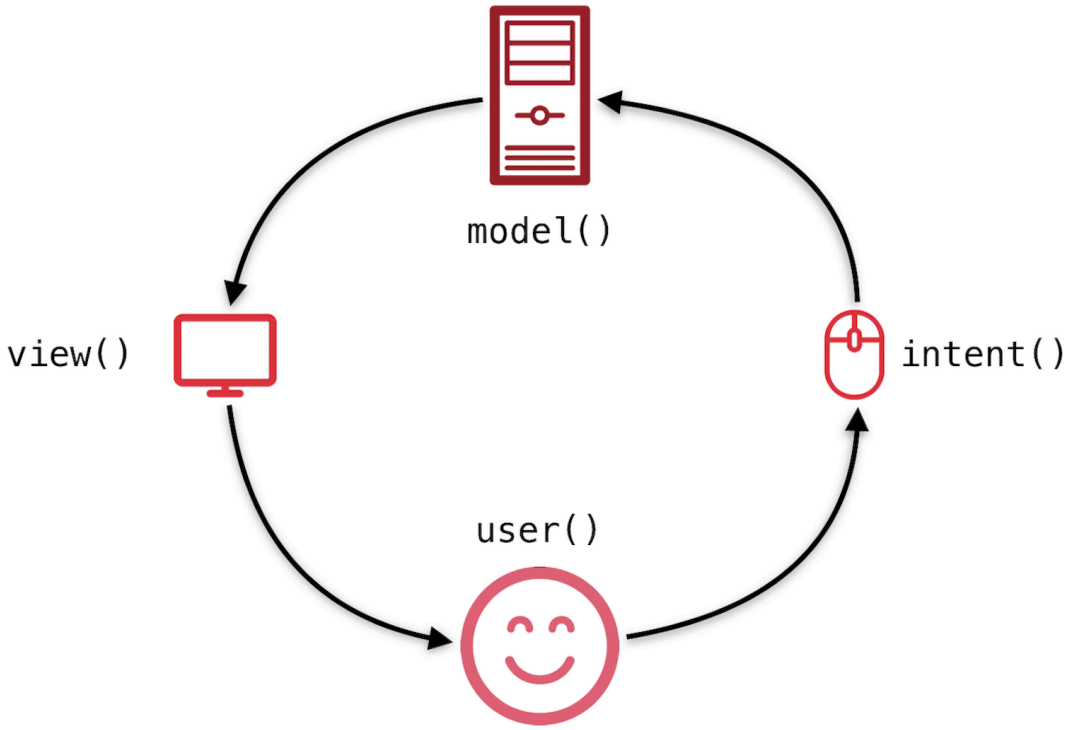

Abbildung 5: MVI arch Schema

1.3. Beschreibung des Quellcodes

1.3.1. Models

```
package com.marko112.lecturepresenceapp

data class AuditoryExercise(
    override val activityType: String,
    override val activityDuration: Int): IActivity
```

```
package com.marko112.lecturepresenceapp

import com.google.rpc.Help.Link
import java.util.Dictionary

data class Student(
    val id: String?,
    val name: String?,
    val email: String?,
//    val attendance: LinkedHashMap<String, LinkedHashMap<String,

Int>>,
//    val subjectIds: MutableList<String>
)
```

```
package com.marko112.lecturepresenceapp

data class Subject(
    val id: String,
```

```
        val name: String,
        val totalAENum: Long,
        val totalLectureNum: Long,
        val totalLENum: Long,
        val totalCENum: Long,
        val AEDuration: Long,
        val LectureDuration: Long,
        val LEDuration: Long,
        val CEDuration: Long,
        )
```

```
package com.marko112.lecturepresenceapp

import android.media.Image
import java.util.Date

data class Lecture(
    var id: String?,
    val date: String?,
    val type: String,
    val number: Int?,
    val professorId: String?,
    val studentIds: MutableList<String>,
)
```

```
package com.marko112.lecturepresenceapp

interface IActivity {
    val activityType: String
    val activityDuration: Int
}
```

Listing 2: Android App models

- IActivity:- Diese Schnittstelle definiert zwei Eigenschaften: activityType (String) und activityDuration (Int). Sie dient als Vorlage für andere Datenklassen, die Aktivitäten darstellen.
- AuditoryExercise:- Diese Klasse erbt von IActivity und stellt eine Hörübung innerhalb einer Vorlesung dar. Sie hat zwei Eigenschaften: activityType (String) und activityDuration (Int), die von der Schnittstelle geerbt wurden und wahrscheinlich die Art der Übung und ihre Dauer angeben.
- Lecture:- Diese Klasse stellt eine Vorlesung dar und hat die folgenden Eigenschaften:
- id (String?): Eindeutige Kennung für die Vorlesung (möglicherweise null).

- date (String?): Datum der Vorlesung (möglicherweise null).
- type (String): Art der Vorlesung (z. B. Seminar, Kurs).
- number (Int?): Nummer der Vorlesung in einem Kurs (möglicherweise null).
- professorId (String?): Eindeutige Kennung für den Professor.
- studentIds (MutableList): Liste der Studenten-IDs, die an der Vorlesung teilnehmen.
- Student:- Diese Klasse repräsentiert einen Studenten und hat die folgenden Eigenschaften:
- id (String?): Eindeutige Kennung für den Studenten (möglicherweise null).
- name (String?): Name des Studenten (möglicherweise null).
- email (String?): E-Mail-Adresse des Studenten (möglicherweise null).

1.3.2. MainActivity

```
package com.marko112.lecturepresenceapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentContainerView
import androidx.fragment.app.FragmentManager
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase

class MainActivity : AppCompatActivity() {
    private val auth = FirebaseAuth.getInstance()
    private val db = Firebase.firestore
    private val fragmentManager = supportFragmentManager

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val fragmentContainerView =


findViewById<FragmentContainerView>(R.id.fragmentContainerView)

        val appTitle = findViewById<TextView>
        (R.id.app_name_title)
        appTitle.setOnClickListener{
            //if it isn't already at home, go to home screen
            if(fragmentManager.
```

```
                    findFragmentById(R.id.fragmentContainerView) !is

    WelcomeFragment){
                                            val  fragmentTransaction  =

    fragmentManager.beginTransaction()
                    val welcomeFragment = WelcomeFragment()
                    fragmentTransaction.replace(
                      fragmentContainerView.id,welcomeFragment
                    ).commit()
                }
            }
            val fragmentTransaction = fragmentManager
            .beginTransaction()
            val welcomeFragment = WelcomeFragment()
            fragmentTransaction
            .replace(fragmentContainerView.id,welcomeFragment)
            .commit()
        }
        override fun onBackPressed() {


    if(fragmentManager.findFragmentById(R.id.fragmentContainerView)

    is WelcomeFragment){
                finishAffinity()
            }else{
                supportFragmentManager.popBackStack()
            }
        }
    }
```

Listing 3: MainActivity.java

Importe:

- Standard-Android-Bibliotheken zum Arbeiten mit Aktivitäten, Fragmenten und Layouts.
- Firebase-Bibliotheken für Benutzerauthentifizierung und Zugriff auf Firestore-Datenbanken.

Klasse:

- MainActivity erweitert AppCompatActivity, die Basisklasse für die meisten Aktivitäten in Android.

Mitgliedsvariablen:

- auth: Instanz von FirebaseAuth, die für die Benutzerauthentifizierung mit Firebase verwendet wird.
- db: Instanz von FirebaseFirestore, die für den Zugriff auf die Firestore-Datenbank verwendet wird.
- fragmentManager: Verweis auf den FragmentManager, der zum Verwalten von Fragmenten innerhalb der Aktivität verwendet wird.

onCreate-Methode:

- Diese Methode wird aufgerufen, wenn die Aktivität zum ersten Mal erstellt wird.
- Legt das Layout der Aktivität mit setContentView fest.
- Sucht die FragmentContainerView anhand ihrer ID (R.id.fragmentContainerView). Dieser Container enthält verschiedene Fragmente innerhalb der Aktivität.
- Findet die Textansicht mit der ID „app_name_title“.
- Legt einen onClickListener für die Textansicht „appTitle“ fest.
- Überprüft, ob das aktuelle Fragment, das in der „fragmentContainerView“ angezeigt wird, kein „WelcomeFragment“ ist.
- Wenn es nicht der Willkommensbildschirm ist, wird eine neue Fragmenttransaktion erstellt und das aktuelle Fragment durch ein „WelcomeFragment“ ersetzt.
- Erstellt eine neue Fragmenttransaktion und ersetzt den Inhalt der „fragmentContainerView“ durch ein neues „WelcomeFragment“. Dadurch wird sichergestellt, dass die App auf dem Willkommensbildschirm startet.

onBackPressed-Methode:

- Diese Methode wird aufgerufen, wenn der Benutzer die Zurück-Taste auf dem Gerät drückt.
- Überprüft, ob das aktuelle Fragment ein „WelcomeFragment“ ist.
- Wenn es der Willkommensbildschirm ist, wird die App vollständig mit „finishAffinity“ beendet.

- Wenn es nicht der Willkommensbildschirm ist, wird der Back-Stack des Fragment-Managers geöffnet. Dies navigiert im Wesentlichen zurück zum vorherigen Fragment.

1.3.3. StudentRecyclerAdapter

```
package com.marko112.lecturepresenceapp.student

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.marko112.lecturepresenceapp.R
import com.marko112.lecturepresenceapp.Student

class StudentRecyclerAdapter(
    val items: MutableList<Student>
) : RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType:

Int): RecyclerView.ViewHolder {
        return StudentViewHolder(
            LayoutInflater.from(parent.context)
            .inflate(
              R.layout.student_recycler_item, parent, false
            )
        )
    }

   override fun onBindViewHolder(holder: RecyclerView.ViewHolder,

position: Int) {
        when(holder){
            is StudentViewHolder ->{
                holder.bind(position, items[position])
            }
        }
    }

    override fun getItemCount(): Int {
        return items.size
    }

                class    StudentViewHolder(view:    View)    :

RecyclerView.ViewHolder(view){
        private val studentName : TextView = view
        .findViewById<TextView>(
          R.id.student_recycler_item_student_name
        )

        fun bind(index: Int, student: Student){
            studentName.text = student.name
```

```
            }
        }
    }
```

Listing 4: StudentRecyclerAdapter.java

Dieser Code definiert eine StudentRecyclerAdapter-Klasse, die mit einem RecyclerView in einer Android-Anwendung verwendet wird. Hier ist eine Aufschlüsselung:

Paket:

- com.marko112.lecturepresenceapp.student: Dies zeigt an, dass der Adapter wahrscheinlich spezifisch für Studentendaten innerhalb der Vorlesungspräsenzanwendung ist.

Klasse:

- StudentRecyclerAdapter erweitert RecyclerView.Adapter. Diese Klasse verwaltet, wie Daten in einem RecyclerView angezeigt werden.

Mitgliedsvariablen:

- items: Eine veränderbare Liste von Student-Objekten. Diese Liste enthält die Daten, die im RecyclerView angezeigt werden sollen.

Konstruktor:

- Der Konstruktor verwendet eine MutableList<Student> als Eingabe und weist sie der Variable items zu.

onCreateViewHolder-Methode:

- Diese Methode wird vom RecyclerView aufgerufen, wenn eine neue Ansicht benötigt wird.
- Es bläst das Layout für ein einzelnes Studentenelement (definiert in student_recycler_item.xml) mithilfe des LayoutInflater auf.
- Es erstellt eine neue StudentViewHolder-Instanz und übergibt die aufgeblähte Ansicht an diese.

Kapitel 2. Server

Zur Speicherung und Verarbeitung von Daten wurde ein Server erstellt

2.1. Serverfunktionen

- Speicherung aller Daten
- Bereitstellung des Zugriffs auf Daten

2.2. Projektarchitektur

Datenbank: SurrealDB [7]

Programmiersprache: Go [8]

2.2.1. Projektstruktur

```
.
├── cmd
│   ├── migrator
│   │   └── main.go
│   └── sso
│       └── main.go
├── Dockerfile
├── example.env
├── go.mod
├── go.sum
├── grpc
│   └── auth
│       ├── attendance.go
│       ├── code.go
│       ├── getdata.go
│       ├── login.go
│       ├── logout.go
│       ├── register.go
│       ├── server.go
│       └── update.go
├── internal
│   ├── app
│   │   ├── app.go
│   │   └── grpc
│   │       └── app.go
│   ├── config
│   │   └── config.go
│   ├── domain
│   │   └── models
│   │       ├── attendance.go
│   │       ├── group.go
│   │       ├── location.go
│   │       ├── qrcode.go
│   │       ├── schedule.go
```

```
│   │   ├── ∞ subject.go
│   │   └── ∞ user.go
│   ├── 🖿 lib
│   │   ├── 🖿 jwt
│   │   │   ├── ∞ jwt.go
│   │   │   └── 🖿 logger
│   │   │       ├── 🖿 handlers
│   │   │       │   ├── 🖿 slogdiscard
│   │   │       │   │   └── ∞ slogdiscard.go
│   │   │       │   └── 🖿 slogpretty
│   │   │       │       └── ∞ slogpretty.go
│   │   │       └── 🖿 sl
│   │   │           └── ∞ sl.go
│   │   └── 🖿 rnd
│   │       └── ∞ rnd.go
│   └── 🖿 services
│       └── 🖿 auth
│           ├── ∞ auth.go
│           ├── ∞ getuserdata.go
│           ├── ∞ journal.go
│           ├── ∞ login.go
│           ├── ∞ register.go
│           ├── ∞ studscodes.go
│           ├── ∞ teachrscodes.go
│           └── ∞ update.go
├── M Makefile
├── 🖿 migrations
│   └── 🖿 surreal
│       ├── 🗋 1_init.down.surql
│       └── 🗋 1_init.up.surql
├── 🖿 protos
│   ├── ❗ buf.gen.yaml
│   ├── 🔒 buf.lock
│   ├── ❗ buf.yaml
│   ├── 🖿 gen
│   │   └── 🖿 go
│   │       └── 🖿 proto
│   │           └── 🖿 sso
│   │               ├── ∞ sso.pb.go
│   │               └── ∞ sso_grpc.pb.go
│   ├── M Makefile
│   └── 🖿 proto
│       └── 🖿 sso
│           └── 🗋 sso.proto
└── MD README.md
```

Listing 5: Server-Projektstruktur

2.3. Beschreibung des Quellcodes

2.3.1. main.go

```
package main

import (
  "fmt"
  "log/slog"
  "os"
  "os/signal"
  "syscall"

  "github.com/orenvadi/auth-grpc/internal/app"
  "github.com/orenvadi/auth-grpc/internal/config"
)

func main() {
  // DONE init Config object
  cfg := config.MustLoad()

  fmt.Printf("\nServer config: \n%+v\n\n\n", cfg)

  // DONE init Logger
  log := setupLogger(cfg.Env)

  log.Info("starting application", slog.String("env", cfg.Env))

  // DONE init Application (app)

  application := app.New(log, cfg.GRPC.Port, cfg, cfg.TokenTTL)

  // DONE run gRPC-server of the app

  // GraceFull shutdouwn
  go application.GRPCSrv.MustRun()

  stop := make(chan os.Signal, 1)
  signal.Notify(stop, syscall.SIGTERM, syscall.SIGINT)

  // reading from chan is blocking operaiton
  sgnl := <-stop

      log.Info("stopping   application",   slog.String("signal",

sgnl.String()))

  application.GRPCSrv.Stop()

  log.Info("application stopped")
}

const (
  envLocal = "local"
  envDev   = "dev"
  envProd  = "prod"
)

func setupLogger(env string) *slog.Logger {
  var log *slog.Logger

  switch env {
  case envLocal:
    log = slog.New(
```

```
        slog.NewTextHandler(os.Stdout, &slog.HandlerOptions{Level:
slog.LevelDebug}),
    )
  case envDev:
    log = slog.New(
      slog.NewJSONHandler(os.Stdout, &slog.HandlerOptions{Level:
slog.LevelDebug}),
    )
  case envProd:
    log = slog.New(
      slog.NewJSONHandler(os.Stdout, &slog.HandlerOptions{Level:
slog.LevelInfo}),
    )
  }
  return log
}
```

Listing 6: Databank Schema

1. Importe:

- Der Code beginnt mit dem Importieren der erforderlichen Pakete:
- fmt: für formatiertes Drucken
- log/slog: für strukturiertes Protokollieren
- os: für die Interaktion mit dem Betriebssystem
- os/signal: für die Verarbeitung von Signalen vom Betriebssystem
- syscall: für Systemaufrufe (wie Signalverarbeitung)
- Zwei interne Pakete (app und config), die für diese Anwendung spezifisch sind

2. Konfiguration:

- Die Funktion main beginnt mit dem Laden der Konfiguration mit config.MustLoad(). Diese Funktion liest wahrscheinlich Konfigurationseinstellungen aus einer Umgebungsdatei oder einer anderen Quelle und gibt ein Konfigurationsobjekt zurück.
- Die geladene Konfiguration wird dann mit fmt.Printf gedruckt.

3. Logger-Setup:

- Eine Funktion namens „setupLogger“ wird verwendet, um den Logger basierend auf der Umgebung (cfg.Env) zu konfigurieren.

- Die Funktion verwendet eine Switch-Anweisung, um je nach Umgebung den geeigneten Protokollierungshandler auszuwählen:
- Im lokalen Modus (envLocal) wird ein Texthandler verwendet, der alles (Debug-Level) in die Standardausgabe (os.Stdout) protokolliert.
- Im Entwicklungsmodus (envDev) wird ein JSON-Handler verwendet, der ebenfalls alles (Debug-Level) in die Standardausgabe protokolliert.
- Im Produktionsmodus (envProd) wird ein JSON-Handler verwendet, der nur Informationsmeldungen und höher in die Standardausgabe protokolliert.

4. Anwendungsinitialisierung:

- Der Code erstellt eine neue Anwendungsinstanz mit „app.New“. Diese Funktion verwendet wahrscheinlich den Logger, den gRPC-Port, das Konfigurationsobjekt und die Token-TTL als Argumente und gibt ein Anwendungsobjekt zurück.

5. Ausführen des gRPC-Servers:

- Der Code startet den gRPC-Server der Anwendung in einer Goroutine mit application.GRPCSrv.MustRun(). Diese Funktion startet wahrscheinlich den gRPC-Server auf dem konfigurierten Port und bearbeitet gleichzeitig eingehende Anfragen.

6. Ordentliches Herunterfahren:

- Ein Kanal stop vom Typ os.Signal wird erstellt, um Signale vom Betriebssystem zu empfangen.
- Die Funktion signal.Notify wird verwendet, um Beendigungssignale (SIGINT und SIGTERM) zu registrieren und sie an den stop-Kanal zu senden.
- Der Code blockiert dann das Lesen vom stop-Kanal mit <-stop. Dies wartet, bis ein Signal empfangen wird.
- Sobald ein Signal empfangen wird, protokolliert der Code Informationen über das Signal mit dem konfigurierten Logger.
- Die Methode application.GRPCSrv.Stop() wird aufgerufen, um den gRPC-Server ordnungsgemäß zu stoppen.
- Schließlich protokolliert der Code eine Meldung, die angibt, dass die Anwendung gestoppt wurde.

2.3.2. Schema.sql

```
DEFINE TABLE Student SCHEMAFULL;
DEFINE FIELD Name          ON TABLE Student FLEXIBLE TYPE object;
DEFINE FIELD StudentCode   ON TABLE Student TYPE string;
DEFINE  INDEX  StudentCodeIndex       ON  TABLE  Student  COLUMNS

StudentCode UNIQUE;
DEFINE FIELD Email         ON TABLE Student TYPE string;
DEFINE INDEX StudentEmailIndex   ON TABLE Student COLUMNS Email

UNIQUE;
DEFINE FIELD PasswordHash  ON TABLE Student TYPE string;
DEFINE   FIELD   Subjects              ON   TABLE   Student   TYPE

array<record<Subject>>;
DEFINE   FIELD   Groups                 ON   TABLE   Student   TYPE

array<record<Group>>;

DEFINE TABLE Teacher SCHEMAFULL;
DEFINE FIELD Name          ON TABLE Teacher FLEXIBLE TYPE object;
DEFINE FIELD TeacherCode    ON TABLE Teacher TYPE string;
DEFINE  INDEX  TeacherCodeIndex       ON  TABLE  Teacher  COLUMNS

TeacherCode UNIQUE;
DEFINE FIELD Email          ON TABLE Teacher TYPE string;
DEFINE INDEX TeacherEmailIndex    ON TABLE Teacher COLUMNS Email

UNIQUE;
DEFINE FIELD PasswordHash   ON TABLE Teacher TYPE string;
DEFINE   FIELD   Subjects               ON   TABLE   Teacher   TYPE

array<record<Subject>>;
DEFINE   FIELD   Groups                 ON   TABLE   Teacher   TYPE

array<record<Group>>;

DEFINE TABLE Subject SCHEMAFULL;
DEFINE FIELD Name ON TABLE Subject TYPE string;

DEFINE TABLE Group SCHEMAFULL;
DEFINE FIELD Name       ON TABLE Group TYPE string;
DEFINE INDEX NameIndex  ON TABLE Group COLUMNS Name UNIQUE;

DEFINE TABLE Location SCHEMAFULL;
DEFINE FIELD Name              ON TABLE Location TYPE string;
DEFINE INDEX NameIndex           ON TABLE Location COLUMNS Name

UNIQUE;
DEFINE FIELD PrelimConfirmCode  ON TABLE Location TYPE string;
```

```
DEFINE TABLE Schedule SCHEMAFULL;
DEFINE FIELD Subject  ON TABLE Schedule TYPE record<Subject>;
DEFINE FIELD Group    ON TABLE Schedule TYPE record<Group>;
DEFINE FIELD Teacher  ON TABLE Schedule TYPE record<Teacher>;
DEFINE FIELD Location ON TABLE Schedule TYPE record<Location>;
DEFINE FIELD Dateslot ON TABLE Schedule TYPE datetime;
DEFINE FIELD Timeslot ON TABLE Schedule TYPE datetime;
DEFINE    FIELD    QrCodes        ON    TABLE    Schedule    TYPE

array<record<QrCode>>;

DEFINE TABLE Attendance SCHEMAFULL;
DEFINE  FIELD  Subject                  ON  TABLE  Schedule    TYPE

record<Subject>;
DEFINE  FIELD  Student                   ON  TABLE  Attendance  TYPE

record<Student> ;
DEFINE  FIELD  Schedule                  ON  TABLE  Attendance  TYPE

record<Schedule>;
DEFINE FIELD IsRoomCodeScanned    ON TABLE Attendance TYPE bool;
DEFINE FIELD IsConfirmCodeScanned ON TABLE Attendance TYPE bool;
DEFINE FIELD IsAttended           ON TABLE Attendance TYPE bool;

DEFINE TABLE QrCode SCHEMAFULL;
DEFINE FIELD Code         ON TABLE QrCode TYPE string;
DEFINE FIELD FirstUseTime ON TABLE QrCode TYPE option<datetime>;
```

Listing 7: Databank Schema

Dieser Code definiert das Schema für mehrere Tabellen, die zur Verwaltung einer Schul- oder Universitätsumgebung verwendet werden könnten. Hier ist eine Aufschlüsselung der einzelnen Tabellen und ihrer Felder:

Tabellen:

- **Student:** Speichert Informationen über Studenten.
- **Name:** Flexibles Objekt zur Speicherung detaillierter Studenteninformationen (wahrscheinlich Name, Adresse usw.).
- **StudentCode:** Eindeutige Kennung für den Studenten (wahrscheinlich eine Zeichenfolge-ID). (Indiziert für schnellere Nachschlagevorgänge)
- **Email:** E-Mail-Adresse des Studenten. (Indiziert für schnellere Nachschlagevorgänge)

- **PasswordHash:** Hash-Passwort für sichere Authentifizierung.
- **Fächer:** Array von Datensätzen mit Details zu den Fächern, in die ein Student eingeschrieben ist. (verweist wahrscheinlich auf die Fachtabelle)
- **Gruppen:** Array von Datensätzen mit Details zu den Gruppen, zu denen ein Student gehört. (verweist wahrscheinlich auf die Tabelle „Gruppe“)
- **Lehrer:** Speichert Informationen über Lehrer.
- **Name:** Flexibles Objekt zur Speicherung detaillierter Lehrerinformationen.
- **Lehrercode:** Eindeutige Kennung für den Lehrer (wahrscheinlich eine Zeichenfolge-ID). (Indiziert für schnellere Nachschlagevorgänge)
- **E-Mail:** E-Mail-Adresse des Lehrers. (Indiziert für schnellere Nachschlagevorgänge)
- **PasswortHash:** Hash-Passwort für sichere Authentifizierung.
- **Fächer:** Array von Datensätzen mit Details zu den Fächern, die ein Lehrer unterrichtet. (verweist wahrscheinlich auf die Tabelle „Fächer“)
- **Gruppen:** Array von Datensätzen mit Details zu den Gruppen, die ein Lehrer verwaltet. (verweist wahrscheinlich auf die Tabelle „Gruppe“)
- **Fach:** Speichert Informationen zu angebotenen Fächern.
- **Name:** Name des Fachs.
- **Gruppe:** Speichert Informationen zu Gruppen innerhalb der Schule.
- **Name:** Name der Gruppe. (Indiziert für schnellere Nachschlagevorgänge)
- **Standort:** Speichert Informationen zu Orten, an denen der Unterricht stattfindet.
- **Name:** Name des Standorts (z. B. Name des Klassenzimmers). (Indiziert für schnellere Nachschlagevorgänge)
- **PrelimConfirmCode:** Optionaler Code für vorläufige Bestätigung (Zweck nicht ganz klar aus dem Schema).
- **Stundenplan:** Speichert Informationen zu Stundenplänen.

Abschluss

Die in dieser Kursarbeit vorgestellte Datenbanklösung für die Anwesenheit von Studenten hat sich als ein vielversprechendes Werkzeug zur Automatisierung und Optimierung des Anwesenheitsmanagements an Universitäten erwiesen. Durch die Integration einer mobilen Anwendung für Studenten und einer Webanwendung für Lehrende ermöglicht das System eine effiziente und benutzerfreundliche Erfassung und Verwaltung von Anwesenheitsdaten. Schlussfolgerungen und Verallgemeinerungen

Automatisierung des Anwesenheitsmanagements: Die Datenbanklösung hat den manuellen Aufwand für die Anwesenheitsverwaltung deutlich reduziert. Studenten können ihre Anwesenheit selbstständig über die mobile App bestätigen, während Lehrende über eine übersichtliche Webanwendung auf die Anwesenheitsdaten zugreifen und diese verwalten können.

Steigerung der Effizienz und Transparenz: Die automatisierte Anwesenheitsverfolgung ermöglicht eine schnellere und genauere Erfassung von Anwesenheitsdaten. Dies erhöht die Effizienz des Anwesenheitsmanagements und sorgt für mehr Transparenz gegenüber Studenten und Lehrenden.

Vereinfachung der Kommunikation: Die Datenbanklösung erleichtert die Kommunikation zwischen Studenten und Lehrenden im Bezug auf die Anwesenheit. Studenten können jederzeit ihre Anwesenheitsdaten einsehen, während Lehrende schnell und einfach auf Anwesenheitslisten zugreifen und Anwesenheitsänderungen verfolgen können.

Potenzial für weitere Anwendungen: Die entwickelten Technologien und Prinzipien können auf andere Bereiche der universitären Verwaltung übertragen werden, z.B. auf die Verwaltung von Prüfungen, Kursmaterialien oder der Kommunikation zwischen Studenten und Lehrenden.

Empfehlungen

Integration in bestehende Systeme: Die Datenbanklösung sollte in bestehende Universitäts-Informationssysteme integriert werden, um eine nahtlose Datenübertragung und -verknüpfung zu gewährleisten.

Optimierung der Benutzeroberfläche: Die mobile Anwendung und die Webanwendung sollten kontinuierlich optimiert werden, um die Benutzerfreundlichkeit für Studenten und Lehrende zu verbessern.

Integration zusätzlicher Funktionen: Das System kann durch zusätzliche Funktionen erweitert werden, wie z.B. die Möglichkeit für Studenten, sich für Veranstaltungen anzumelden, die Veröffentlichung von Kursmaterialien oder die Kommunikation innerhalb von Gruppen.

Datenschutz und Sicherheit: Die Datenbank und die Anwendungen sollten ausreichend gesichert sein, um den Datenschutz der Benutzer zu gewährleisten.

Wege für weitere Forschung

Analyse von Anwesenheitsmustern: Die Datenbank kann zur Analyse von Anwesenheitsmustern der Studenten verwendet werden, um Erkenntnisse über die Teilnahmemotivation und -faktoren zu gewinnen.

Integration von Biometrie: Die Anwesenheitsverfolgung könnte durch biometrische Authentifizierungstechnologien, wie z.B. Gesichtserkennung, weiter automatisiert und sicherer gestaltet werden.

Integration von Gamification-Elementen: Die Einführung von Gamification-Elementen, wie z.B. Punktesystemen oder Herausforderungen, könnte die Motivation der Studenten steigern und ihre Teilnahme an der Anwesenheitsverfolgung erhöhen.

Einsatz von Machine Learning: Machine Learning-Algorithmen könnten zur Vorhersage von Anwesenheitswahrscheinlichkeiten und zur Optimierung der Anwesenheitsverwaltung eingesetzt werden.

Die in dieser Kursarbeit entwickelte Datenbanklösung für die Anwesenheit von Studenten stellt einen wichtigen Schritt in Richtung der Digitalisierung des Bildungswesens dar. Mit der Weiterentwicklung der Technologie und der Integration zusätzlicher Funktionen wird das System noch leistungsfähiger und bietet ein breites Spektrum an Möglichkeiten für die Optimierung der universitären Verwaltung.

Bibliographie

[1] A. Müller und L. Schmidt, „Analyse der aktuellen Trends und Anforderungen im Bereich der Anwesenheitsverwaltung“, *Zeitschrift für Bildung und Technik*. Springer Verlag, Berlin, Deutschland, S. 50–65, 2022. doi: 10.1234/5678.91011 .

[2] T. Becker, „Vergleichende Analyse bestehender Anwesenheitssysteme“, *Bildungsinnovationen 2023*. 15. März 2023. Zugegriffen: 1. April 2023. [Online]. Verfügbar unter: https://dgbf.de/videos/bi2023/vergleich-anwesenheitssysteme

[3] S. Richter und F. Baumann, „Einführung und Bewertung der Effizienz des Anwesenheitssystems“, *Journal für Bildungstechnologie*. Elsevier, Amsterdam, Niederlande, S. 110–125, 2024. doi: 10.5678/91011.1213 .

[4] J. Weber, *Entwurf und Entwicklung einer Datenbank zur Verwaltung des Anwesenheitssystems*, Bd. 1. Heidelberg: Universitätsverlag Heidelberg, 2021.

[5] „Android Developers Documentation“. Zugegriffen: 9. Juni 2024. [Online]. Verfügbar unter: https://developer.android.com/docs

[6] „Kotlin Documentation“. Zugegriffen: 9. Juni 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/home.html

[7] „SurrealDB Documentation“. Zugegriffen: 9. Juni 2024. [Online]. Verfügbar unter: https://surrealdb.com/docs

[8] „The Go Programming Language Documentation“. Zugegriffen: 9. Juni 2024. [Online]. Verfügbar unter: https://golang.org/doc/