

Stundenplaner-Projekt

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
1				
2				
3				
4				

D24a

**Kenan Ustalic
Sanjivan Selvarasa
Berat özgür
Halldor Andri Omarsson**

IPT 5.1

INHALTSVERZEICHNIS

- 01 Deckblatt
- 02 Projektbeschreibung
- 03 Anforderungen und Use-Cases
- 04 Benutzeranleitung
- 05 Qualitätskriterien und Bewertung
- 06 Reflexion Stundenplaner Projekt
- 07 Qualitätskriterien und Bewertung
- 08 Tests und beispielaufgabe
- 09 Reflexion and Teamarbeit
- 10 Arbeitsrapport/ Zeitaufwand

02 PROJEKTBESCHREIBUNG

- In diesem Projekt entwickeln wir ein Stundenplaner-Tool, mit dem Lehrpersonen Stundenpläne erstellen, bearbeiten und verwalten können. Das Programm soll automatisch einen Stundenplan für ein Semester generieren, indem es Schüler, Lehrpersonen, Räume und Fächer kombiniert. Dabei werden auch Verfügbarkeiten und verschiedene Kriterien wie Randstunden oder Zwischenstunden berücksichtigt. Ziel ist es, einen möglichst sinnvollen und ausgewogenen Stundenplan zu erstellen. Die Daten werden gespeichert, damit sie nach einem Neustart erhalten bleiben.

03 ANFORDERUNG

- Verwaltung Ressourcen

- Schüler:innen, Lehrpersonen, Räume, Fächer erstellen, bearbeiten und löschen.

- Verfügbarkeit bestimmen

- Angeben, von wann bis wann eine Lehrperson unterrichtet.

- Sinnvolle Stundenplan Struktur

- Stundenplan wird so gebaut, dass es klar und deutlich ist, Raum, Lehrer, Klasse.

- Automatische Stundenplanerstellung

- Aus den angegebenen Daten einen passenden Stundenplan erstellen.

- Persistente Datenspeicherung

- Der Stundenplan soll auch nach Neustart wiederherstellbar sein.

03 USE CASES

Akteure: Lehrperson, Schüler, System

USE CASE 1

Akteur: Lehrperson

Ziel: Räume verwalten

Ablauf:

1. Lehrperson öffnet Programm
2. Lehrperson geht zur Stunde, wo man den Raum ändern will.
3. System überprüft, ob Raum schon besetzt ist.
4. Wenn nein, wird Raum geändert.

Alternative:

- Falls Raum schon besetzt ist, muss man halt ein anderer Raum finden.

USE CASE 2

Akteur: Schüler:innen

Ziel: Lesen, welches Fach die Klasse am Montagnachmittag hat und wo.

Ablauf:

1. Schüler:innen öffnet Stundenplan.
2. Wenn Stundenplan klar und deutlich strukturiert ist, können die Schüler:innen sehr schnell lesen, was sie am Montagnachmittag haben.
3. Lesen wo sie dieses Fach haben.

USE CASE 3

Akteur: System

Ziel: Der Stundenplan wird persistent gespeichert, sodass die Lehrperson am nächsten Tag weiter dran arbeiten kann.

Ablauf:

1. Alle Daten vom Stundenplan werden gesammelt.
2. Wird in einer JSON-Datei gespeichert.
3. Wenn man das Programm wieder startet, lädt das System die JSON-Datei und ladet alle Daten wieder her.

Szenarien

- Lehrperson

- Fügt ein neues Fach hinzu z.B. Englisch und dies wird im Stundenplan gespeichert.
- Lehrperson kann nächste Woche am Montag nicht unterrichten, deswegen muss die Lehrperson die Verfügbarkeit ändern und im Stundenplan das so speichern.

- System

- Stundenplan wurde nach Änderung von Lehrperson geschlossen. Am nächsten Tag arbeitet die Lehrperson weiter an dem Stundenplan und die Arbeit von gestern wurde persistent gespeichert.

04 USE-CASE DIAGRAMM

Architekturentscheidungen:

Wir werden als Programmiersprache C# verwenden. Wir werden als Klassen definitiv Schüler, Lehrer, Schulklasse, Räume und Fächer verwenden, dazu brauchen wir für die Erfassung der Zeiten eine Klasse und für den Algorithmus den an sich und eine Hilfklasse Kombination. Das Vorgehen wird so aussehen, dass wir heuristisch, auf Zufallsbasierter Basis den besten Plan bestimmen werden. Alle Klassen die relevant für den Algorithmus sind sollen in den hineinlaufen. Der Algorithmus soll der Dreh- und Angelpunkt des Programms sein. Als Speicherung nehmen wir JSON.

Datenmodell:

1. Person (abstrakt)

Basisklasse für Personen im System.

Attribute:

- FirstName: string
- LastName: string
- Email: string

Diese Klasse wird von Teacher und Student geerbt.

2. Teacher (erbt von Person)

Repräsentiert eine Lehrperson.

Attribute:

- TeachingSubjects: List<Subject>
- AvailableBlocks: List<TimeBlock>

Beziehungen:

- 1 Teacher → n Subjects (kann mehrere Fächer unterrichten)
- 1 Teacher → n TimeBlocks (definierte Verfügbarkeit)

3. Student (erbt von Person)

Repräsentiert einen Lernenden.

Beziehungen:

- Ein Student gehört genau zu einer SchoolClass.

4. Subject

Ein Schulfach.

Attribute:

- Name: string

Beziehungen:

- 1 Subject → n Teachers (mehrere Lehrpersonen können es unterrichten)

5. Room

Ein Unterrichtsraum.

Attribute:

- RoomId: string
- MaxStudent: int

Beziehungen:

- 1 Room → n TimeBlocks (kann zu verschiedenen Zeiten genutzt werden)

6. SchoolClass

Repräsentiert eine Schulklasse mit ihren Lernenden und dem resultierenden Stundenplan.

Attribute:

- Name: string
- Students: List<Student>
- Timetable: Dictionary<TimeBlock, Combination>

Beziehungen:

- 1 SchoolClass → n Students
- 1 SchoolClass → n TimeBlocks (über Timetable)

7. TimeBlock

Eine Zeitspanne im Stundenplan (z. B. Montag Block 2).

Attribute:

- Day: Weekday (enum)
- BlockIndex: int
- Hours: string[] (legt Stunden fest, z. B. „08:20–09:05“)

Beziehungen:

- Wird in Teacher, SchoolClass und Combination verwendet.

8. Combination

Kombiniert die drei Ressourcen für eine Lektion:

- Subject
- Teacher
- Room

Beziehungen:

- 1 Combination gehört zu genau einem TimeBlock im Timetable einer Klasse.

9. CurriculumAlgo

Verwendet alle Klassen zur Berechnung des optimalen Stundenplans.

Benötigte Daten:

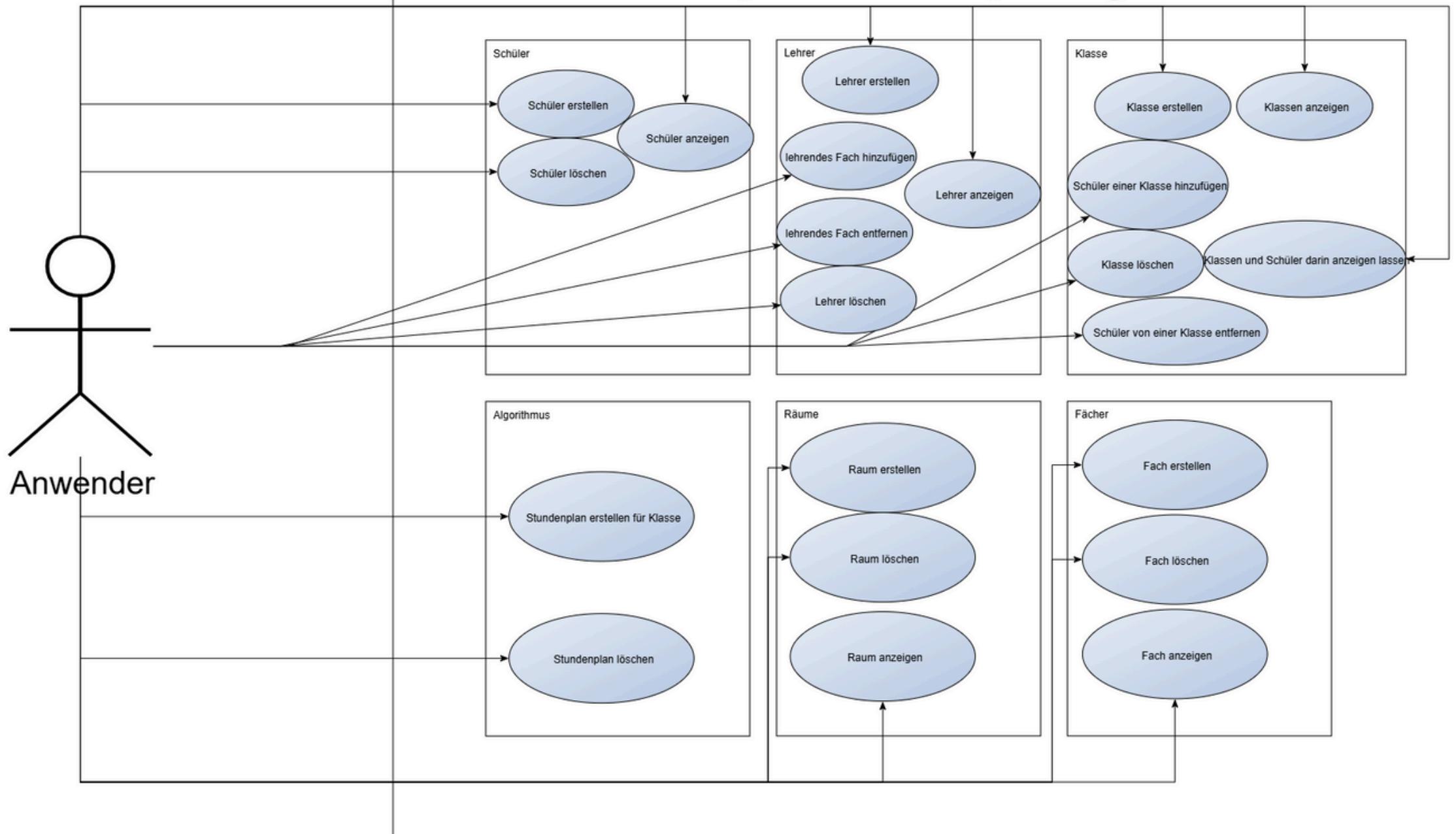
- List<SchoolClass>
- List<Subject>
- List<Teacher>
- List<Room>

Ergebnis:

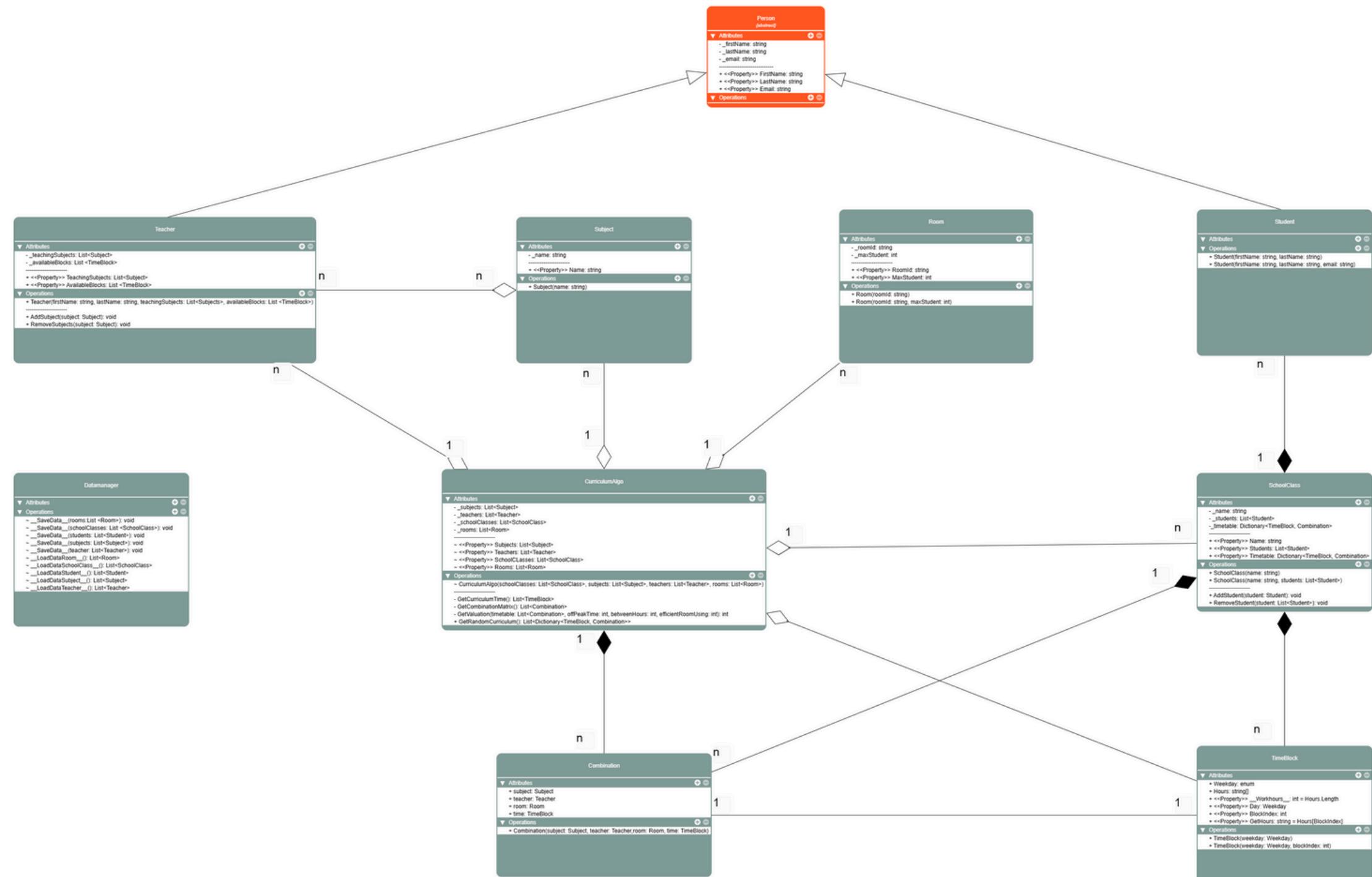
- Dictionary<TimeBlock, Combination>

04 USE CASE DIAGRAMM

Stundenplanerprojekt



04 KLASSENEDIAGRAMM



05 UMSETZUNG

Combination:

Generell:

Der Nutzen dieser Klasse besteht darin, die Beziehungen der anderen notwendigen Klassen zusammenfassen damit der Algorithmus einfacher wird.

Properties:

Wir haben vier öffentliche Properties erstellt, die dazu dienen, zusammen später in der Combinationsmatrix in der Algorithmuskasse benutzt zu werden. Deswegen jeweils:

« public Klassename propertynname » bsp. « public Subject subject »

```
4 Verweise
public Subject Subject { get; }
```

Konstruktor :

Hier fassen wir alle 4 Beziehungen auf und verwenden sie später im Algorithmus.

```
2 Verweise
public Combination(Subject subject, Teacher teacher, Room room, TimeBlock time)
{
    this.Subject = subject;
    this.Teacher = teacher;
    this.Room = room;
    this.Room = room;
    this.Time = time;
}
```

TimeBlock:

Mit dieser Klasse sieht man, erkennt man wann jeder Lehrer frei ist, und benutzt diese Daten dann im Algorithmus, damit man keine redundanten Lektionen erhältet. Die Klasse gibt also die genauen Daten der Lektionen im Stundenplan an.

Attribute:

Wie haben in dieser Klasse zwei Attribute. Als erstes haben wir ein «enum», welches uns den richtigen Schultag angibt.

```
public enum Weekday { Montag = 1, Dienstag = 2, Mittwoch = 3, Donnerstag = 4, Freitag = 5 }
```

Dann haben wir noch «Hours», welches ein «Array» ist mit dem man, die Verfügbaren Zeiten auswählen kann.

```
public readonly static string[] Hours =
{
    "8:00 - 9:00",
    "9:00 - 10:00",
    "10:00 - 11:00",
    "11:00 - 12:00",
    "12:00 - 13:00",
    "13:00 - 14:00",
    "14:00 - 15:00",
    "15:00 - 16:00",
    "16:00 - 17:00"
};
```

Properties:

Wir haben 4 Properties und 1 davon stammt aus den Attribut «Weekday», welches bereits erklärt wurde. Dieses gibt nur den genauen Tag aus.

```
public Weekday Day { get; }
```

Eine der anderen heisst «WorkHours» und mit ihr kann man die gesamten Arbeitsstunden wiedergeben. Dafür braucht man die Länge von «Hours».

```
public static int WorkHours { get; } = Hours.Length;
```

Die nächste heisst «BlockIndex» und mit dieser kann man die genaue Stunde aus «Hours» herausnehmen.

```
public int BlockIndex { get; }
```

Die nächste heisst «GetHours» und mit dieser kann man die genaue Stunde aus «Hours» herausnehmen.

```
public string GetHours => Hours[BlockIndex];
```

Konstruktoren:

In dieser Klasse haben wir zwei Konstruktoren.
Einmal mit nur dem Parameter «Weekday».

```
public TimeBlock(Weekday day)
{
    Day = day;
}
```

Und ein weiterer mit den zwei Konstruktoren «Weekday» und «BlockIndex»

```
public TimeBlock(Weekday weekday, int blockIndex) : this(weekday)
{
    BlockIndex = blockIndex;
}
```

Datenmanager:

Generell:

Diese Klasse kann man sich wie als den Datenspeicher des Projektes vorstellen. Wir verlagern unsere Daten hier, sodass sie selbst nach dem Schliessen des Projektes noch abgespeichert sind.

Pfade:

Wir haben 5 Pfade für unsere Daten erstellt, welche in ein «Json file» führen, damit diese später noch erhältlich sind.

```
private const string pathRoom = "room.json";  
  
private const string pathSchoolClass = "schoolClass.json";  
//  
  
private const string pathStudent = "student.json";  
//  
  
private const string pathSubject = "subject.json";  
//  
  
private const string pathTeacher = "teacher.json";
```

Methoden:

In dieser Klasse haben wir zwei Arten von Methoden. Als erstes die «SaveData» Methoden. Mit diesen kann man die Daten speichern. Das haben wir für alle 5 Pfade erstellt.

```
internal static void SaveData(List<Room> rooms)  
{  
    string json = JsonSerializer.Serialize(rooms);  
    File.WriteAllText(pathRoom, json);  
}  
  
internal static void SaveData(List<SchoolClass> schoolClasses)  
{  
    string json = JsonSerializer.Serialize(schoolClasses);  
    File.WriteAllText(pathSchoolClass, json);  
}  
  
internal static void SaveData(List<Student> students)  
{  
    string json = JsonSerializer.Serialize(students);  
    File.WriteAllText(pathStudent, json);  
}
```

```
internal static void SaveData(List<Subject> subjects)
{
    string json = JsonSerializer.Serialize(subjects);
    File.WriteAllText(pathSubject, json);
}
```

```
internal static void SaveData(List<Teacher> teacher)
{
    string json = JsonSerializer.Serialize(teacher);
    File.WriteAllText(pathTeacher, json);
}
```

Die Zweite Art von Methoden sind die «LoadData» Methoden. Mit diesen kann man die jeweiligen Daten aus den Pfaden laden.

```
internal static List<Room> LoadDataRoom()
{
    if (!File.Exists(pathRoom)) return new List<Room>();

    string json = File.ReadAllText(pathRoom);
    return JsonSerializer.Deserialize<List<Room>>(json);
}
```

```
internal static List<SchoolClass> LoadDataSchoolClass()
{
    if (!File.Exists(pathSchoolClass)) return new List<SchoolClass>();

    string json = File.ReadAllText(pathSchoolClass);
    return JsonSerializer.Deserialize<List<SchoolClass>>(json);
}
```

```
internal static List<Student> LoadDataStudent()
{
    if (!File.Exists(pathStudent)) return new List<Student>();

    string json = File.ReadAllText(pathStudent);
    return JsonSerializer.Deserialize<List<Student>>(json);
}
```

```
internal static List<Subject> LoadDataSubject()
{
    if (!File.Exists(pathSubject)) return new List<Subject>();

    string json = File.ReadAllText(pathSubject);
    return JsonSerializer.Deserialize<List<Subject>>(json);
}
```

```
internal static List<Teacher> LoadDataTeacher()
{
    if (!File.Exists(pathTeacher)) return new List<Teacher>();

    string json = File.ReadAllText(pathTeacher);
    return JsonSerializer.Deserialize<List<Teacher>>(json);
}
```

05 UMSETZUNG ERKLÄRUNG ALGORITHMUS

Nutzen:

Die CurriculumAlgo-Klasse erstellt automatisch einen Stundenplan für alle vorhandenen Schulklassen.

Es werden dabei die folgenden Aspekte berücksichtigt:

- Fachzuweisung: Welche Lehrer sind berechtigt, welche Fächer zu lehren.
- Lehrpersonenverfügbarkeit: Nur die Zeitabschnitte, in denen der Lehrer verfügbar ist, werden genutzt.
- Räume: Es kommen festgelegte, Räume zum Einsatz.
- Planqualität: Die Randstunden, Zwischenstunden und eine effiziente Raumnutzung werden durch eine Bewertungsfunktion gewichtet.

Das Resultat ist ein heuristisch ermittelter, „nahezu optimaler“ Stundenplan, der für jede Klasse in deren Timetable-Eigenschaft abgelegt und zugleich als Rückgabewert bereitgestellt wird.

Konstruktor:

Bevor der Algorithmus verwendet werden kann, muss er mit allen relevanten Daten instanziiert werden.

```
internal CurriculumAlgo(List<SchoolClass> schoolClasses, List<Subject> subjects, List<Teacher> teachers, List<Room> rooms)
{
    SchoolClasses = schoolClasses;
    Subjects = subjects;
    Teachers = teachers;
    Rooms = rooms;
}
```

Parameter:

- schoolClasses: alle Schulklassen, für die ein Stundenplan erstellt werden soll.
- subjects: alle Fächer, die im Stundenplan vorkommen dürfen.
- teachers: alle aktiven Lehrpersonen, inklusive ihrer unterrichtbaren Fächer und Verfügbarkeiten.
- rooms: alle Räume, die für den Unterricht zur Verfügung stehen.

GetCurriculumTime()

```
private List<TimeBlock> GetCurriculumTime()
{
    List<TimeBlock> timetable = new();
    for (int i = 1; i <= 5; i++)
        for (int j = 0; j < WorkHours; j++)
            timetable.Add(new TimeBlock((Weekday)i, j));
    return timetable;
}
```

Zweck:

Erstellt alle möglichen Zeitblöcke (Montag–Freitag, über alle Arbeitsstunden) für den Stundenplan.

Funktionsweise:

- Erzeugt eine leere Liste timetable.
- Schleife über die Wochentage 1 bis 5 (Montag bis Freitag).
- Schleife über alle WorkHours-Blöcke pro Tag.
- Für jede Kombination aus Tag und Block wird ein TimeBlock erzeugt und der Liste hinzugefügt.
- Rückgabe der vollständigen Liste aller TimeBlock-Objekte.

GetCombinationMatrix()

```
private List<Combination> GetCombinationMatrix()
{
    var allComb =
        from time in GetCurriculumTime()
        from subject in Subjects
        from teacher in Teachers
        where teacher.TeachingSubjects.Any(s => s.Name == subject.Name) &&
        from room in Rooms
        select new Combination(subject, teacher, room, time);
    return allComb.ToList();
}
```

Zweck:

Erstellt eine „Matrix“ aus allen logisch möglichen Kombinationen aus Fach, Lehrer, Raum und Zeitblock.

Funktionsweise (LINQ-Abfrage):

- Nimmt alle TimeBlock-Einträge aus GetCurriculumTime().
- Für jeden Zeitblock werden alle Fächer (Subjects) durchlaufen.
- Für jedes Fach werden alle Lehrer (Teachers) angeschaut, die:
 - dieses Fach unterrichten (teacher.TeachingSubjects.Any(s => s.Name == subject.Name)) und
 - zum entsprechenden Tag und Block verfügbar sind (teacher.AvailableBlocks mit passendem Day und BlockIndex).
- Für jede gültige Fach–Lehrer-Kombination werden alle Räume (Rooms) durchlaufen.
- Für jede gültige Kombination wird ein neues Combination-Objekt erstellt: new Combination(subject, teacher, room, time)
- Das Ergebnis wird als Liste zurückgegeben.

GetValuation()

```
private int GetValuation(List<Combination> timetable, int offPeakTime = 5, int betweenHours = 5, int efficientRoomUsing = 20)
{
    int value = 1000;
    HashSet<string> memorizeRooms = new();
    foreach (var t in timetable)
    {
        if ((t.Time.BlockIndex == 0) || (t.Time.BlockIndex == WorkHours - 1))
        {
            value -= offPeakTime;
        }
        if (!memorizeRooms.Contains(t.Room.RoomId))
            memorizeRooms.Add(t.Room.RoomId);
        else
            value -= betweenHours;
    }

    int firstHour = int.MaxValue;
    foreach (var item in timetable)
        if (item.Time.BlockIndex < firstHour)
            firstHour = item.Time.BlockIndex;

    for (int i = firstHour; i < WorkHours - 1; i++)
    {
        bool isValue = false;
        foreach (var item in timetable)
        {
            if (item.Time.BlockIndex == i)
            {
                isValue = true;
                break;
            }
        }
        if (!isValue) value -= efficientRoomUsing;
    }
    return value;
}
```

Zweck:

Bewertet einen Stundenplan (für eine einzelne Klasse) anhand verschiedener Kriterien.

Die Bewertung startet bei 1000 Punkten und wird durch „schlechte“ Eigenschaften reduziert.

Parameter:

- timetable: Liste von Combination-Einträgen (Stundenplan einer Klasse).
- offPeakTime: Gewichtung für Randstunden (erste/letzte Stunde).
- betweenHours: Gewichtung für „Doppelbelegung“ eines Raums / unerwünschte Situation bei Raumnutzung.
- efficientRoomUsing: Gewichtung für Lücken (Zwischenstunden) innerhalb des Tages.

Logik im Detail:

- Grundwert:
 - value wird auf 1000 gesetzt.
- Randzeiten:
 - Schleife über jede Combination im timetable.
 - Wenn der Block der ersten oder letzten Stunde des Tages entspricht (BlockIndex == 0 oder BlockIndex == WorkHours - 1), wird value um offPeakTime reduziert.
- Raumnutzung:
 - Es wird ein HashSet<string> memorizeRooms gemacht:
 - Ist ein Raum zum ersten Mal in dieser Klasse enthalten, wird er hinzugefügt.
 - Ist der gleiche Raum wieder da, wird value um betweenHours reduziert.
- Zwischenstunden / Lücken im Tagesverlauf:
 - Es wird die früheste Stunde (firstHour) des Tages in diesem Stundenplan ermittelt.
 - Für jede Stunde vom firstHour bis WorkHours - 2 wird geprüft:
 - Gibt es in diesem Zeitblock irgendeine Combination im Stundenplan?
 - Wenn nein, wird value um efficientRoomUsing reduziert.
 - So werden Lücken zwischen erster und letzter belegter Stunde bestraft (Zwischenstunden).
- Rückgabe:
 - Gibt den berechneten Score als int zurück.

GetRandomCurriculum()

```
private List<Dictionary<TimeBlock, Combination>> GetRandomCurriculum()
{
    List<Dictionary<TimeBlock, Combination>> allCurr = new();
    Random rnd = new();
    List<Combination> allCombinations = GetCombinationMatrix();
    foreach (var schoolClass in SchoolClasses)
    {
        Dictionary<TimeBlock, Combination> tempComb = new();
        for (int i = 1; i <= 5; i++)
        {
            for (int j = 0; j <= 4; j++)
            {
                TimeBlock timeBlock;
                Combination combination;
                do
                {
                    combination = allCombinations[rnd.Next(0, allCombinations.Count - 1)];
                    timeBlock = new TimeBlock((Weekday)i, combination.Time.BlockIndex);
                } while (tempComb.Keys.Any(time => ((time.BlockIndex == timeBlock.BlockIndex) && (time.Day == timeBlock.Day))));
                tempComb.Add(timeBlock, combination);
                allCombinations.Remove(combination);
            }
        }
        allCurr.Add(tempComb);
    }
    return allCurr;
}
```

Zweck:

Erzeugt für alle Klassen einen zufälligen Stundenplan basierend auf der zuvor erstellten Kombinationsmatrix.

Funktionsweise:

1. allComb wird aus GetCombinationMatrix() geladen (alle möglichen Kombinationen).
2. Für jede SchoolClass:
 - Es wird ein temporäres Dictionary tempComb erzeugt, das TimeBlock und Combination enthält.
 - Schleifen über die Wochentage (1 bis 5) und über fünf Stunden pro Tag.
 - Zufallsauswahl:
 - Mit Random wird eine zufällige Combination aus allCombinations gewählt.
 - Der passende TimeBlock wird aus dem Tag (Weekday) und dem BlockIndex dieser Kombination gebildet.
 - Prüfung:
 - Es wird so lange neu gezogen, bis für den gewählten Tag+Block noch kein Eintrag im tempComb existiert (keine Doppelbelegung eines Zeitblocks in dieser Klasse).
 - Die gewählte Kombination wird dem Dictionary hinzugefügt und gleichzeitig aus allCombinations entfernt, das sie nicht mehrfach verwendet wird.
3. Der fertige Plan für diese Klasse (tempComb) wird der Liste allCurr hinzugefügt.
4. Rückgabe von allCurr (Liste von Stundenplänen, je Klasse ein Dictionary).

GetBestPlan()

```
public List<Dictionary<TimeBlock, Combination>> GetBestPlan(int offPeakTime = 5, int betweenHours = 5, int efficientRoomUsing = 20)
{
    List<Dictionary<TimeBlock, Combination>> curriculums = new();
    int bestScore = 0;
    for (int i = 0; i < 100; i++)
    {
        List<Dictionary<TimeBlock, Combination>> currList = GetRandomCurriculum();
        int avgVal = 0;
        foreach (var cur in currList)
            avgVal += GetValuation(cur.Values.ToList(), offPeakTime, betweenHours, efficientRoomUsing);
        avgVal /= currList.Count;

        if (avgVal > bestScore)
        {
            curriculums = currList;
            bestScore = avgVal;
        }
    }

    List<Dictionary<TimeBlock, Combination>> tempDic = new();
    for (int i = 0; i < curriculums.Count - 1; i++)
    {
        if (SchoolClasses[i].Timetable == null) SchoolClasses[i].Timetable = new Dictionary<TimeBlock, Combination>();
        Dictionary<TimeBlock, Combination> sortedDic = new();
        foreach (var t in curriculums[i].OrderBy(k => k.Key.Day).ThenBy(k => k.Key.BlockIndex))
            sortedDic.Add(t.Key, new Combination(t.Value.Subject, t.Value.Teacher, t.Value.Room, t.Key));
        SchoolClasses[i].Timetable.Clear();
        foreach (var s in sortedDic)
            SchoolClasses[i].Timetable.Add(s.Key, s.Value);
        tempDic.Add(sortedDic);
    }
    curriculums = tempDic;
    return curriculums;
}
```

Nutzen:

Die Methode findet einen nahezu optimalen Stundenplan für alle Schulklassen, indem viele zufällige Lösungen erzeugt, bewertet und die beste auswählt. Dadurch wird ein Stundenplan generiert, der möglichst wenige Randstunden, Zwischenstunden und ineffiziente Raumbelegungen enthält. Der beste gefundene Plan wird anschließend sortiert und direkt in den Timetable-Eigenschaften der Schulklassen gespeichert, damit andere Klassen oder die UI ihn sofort weiterverwenden können.

Ablauf:

1. Die Methode definiert Variablen für den besten bisher gefundenen Plan (curriculums) und den höchsten Score (bestScore).
2. Danach werden 100 zufällige Stundenpläne mit GetRandomCurriculum() erzeugt.
3. Jeder dieser Pläne enthält für jede Klasse einen kompletten Wochenstundenplan.
4. Für jeden dieser Zufallspläne wird die Qualität bestimmt:
5. Die Methode ruft für jede Klasse GetValuation(...) auf und berechnet daraus einen Durchschnittswert.
6. Wenn der gefundene Plan besser ist als zuvor gespeicherte, wird er als neuer bester Plan übernommen.
7. Nach Abschluss aller 100 Durchläufe wird der beste Plan sortiert:
 - Jede Klasse erhält ein nach Tag und Block indexiertes Dictionary.
 - Vorhandene Einträge in SchoolClasses[i].Timetable werden geleert und durch den sortierten Plan ersetzt.
8. Die Methode gibt danach die Liste der sortierten Stundenpläne zurück.

06 BEDIENUNGSANLEITUNG

Man startet, sobald man das Projekt beginnt. Danach sieht man dieses Fenster:



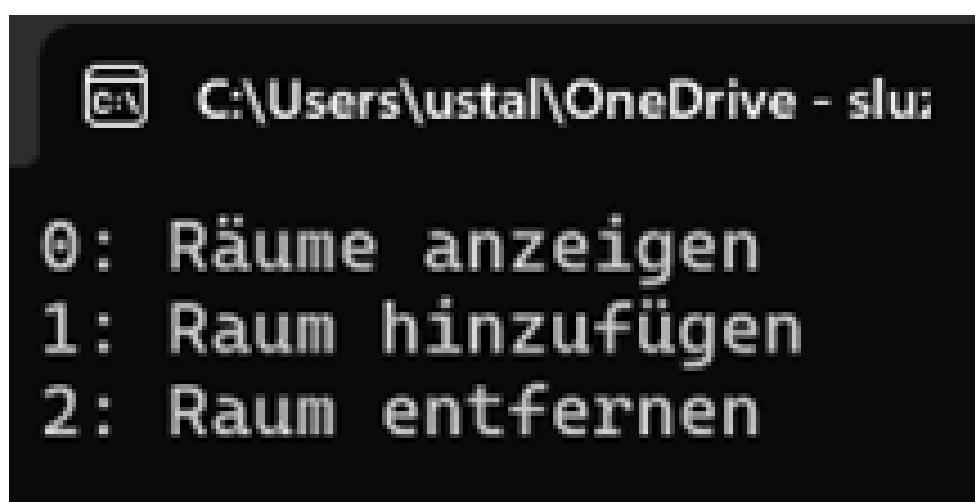
Danach kann man einer der Nummern eingeben, welche einen weiterführen.
In diesem Dokument gehe ich jeden Pfad schnell durch.

Generelle Infos:

- Wenn man die Software neu erhalten hat, sollte man davon ausgehen, dass es keine vorhandenen Daten im System hat. Deshalb sollte man genügend Lehrer, Klassen, Räume und Schüler anlegen, damit man den Stundenplan erstellen kann.

Raum infos/ bearbeiten:

Wenn man 0 klickt, kommt eine Seite, mit der man die Infos über die Räume sieht und diese bearbeiten kann:



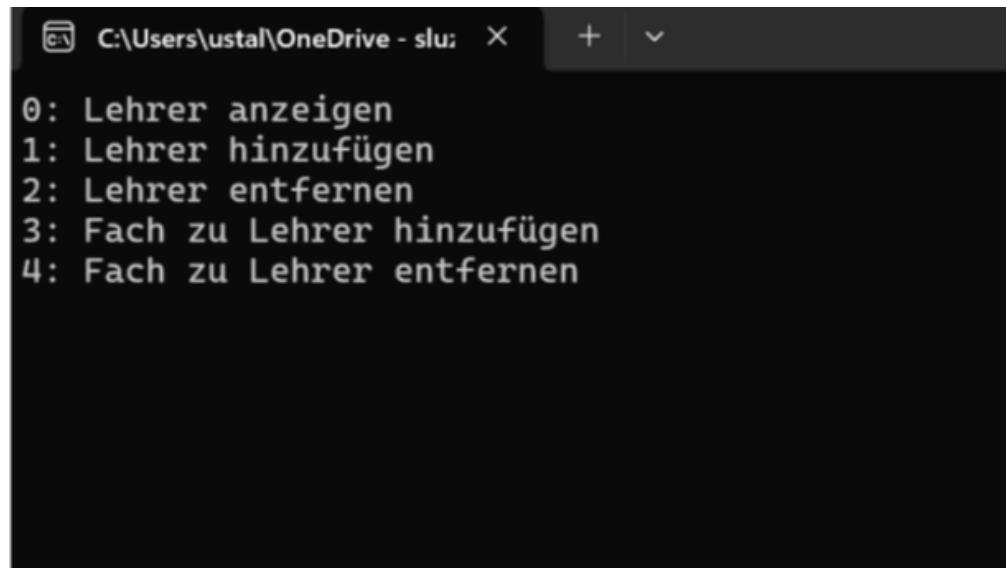
0: Wenn man 0 klickt, dann wird eine Liste mit allen erstellten Räumen angezeigt.

1: Wenn man 1 klickt, kann man einen Raum hinzufügen, den man dann in der Liste sieht

2: Wenn man 2 klickt, kann man einen Raum aus der Liste entfernen (Da gibt man den Index vom Raum an, welcher angezeigt wird, ähnlich wie bei «Räume anzeigen»)

1: Lehrer Infos/bearbeiten

Wenn man 1 klickt, kommt eine Seite, mit der man die Infos über die Lehrer sieht und diese bearbeiten kann:



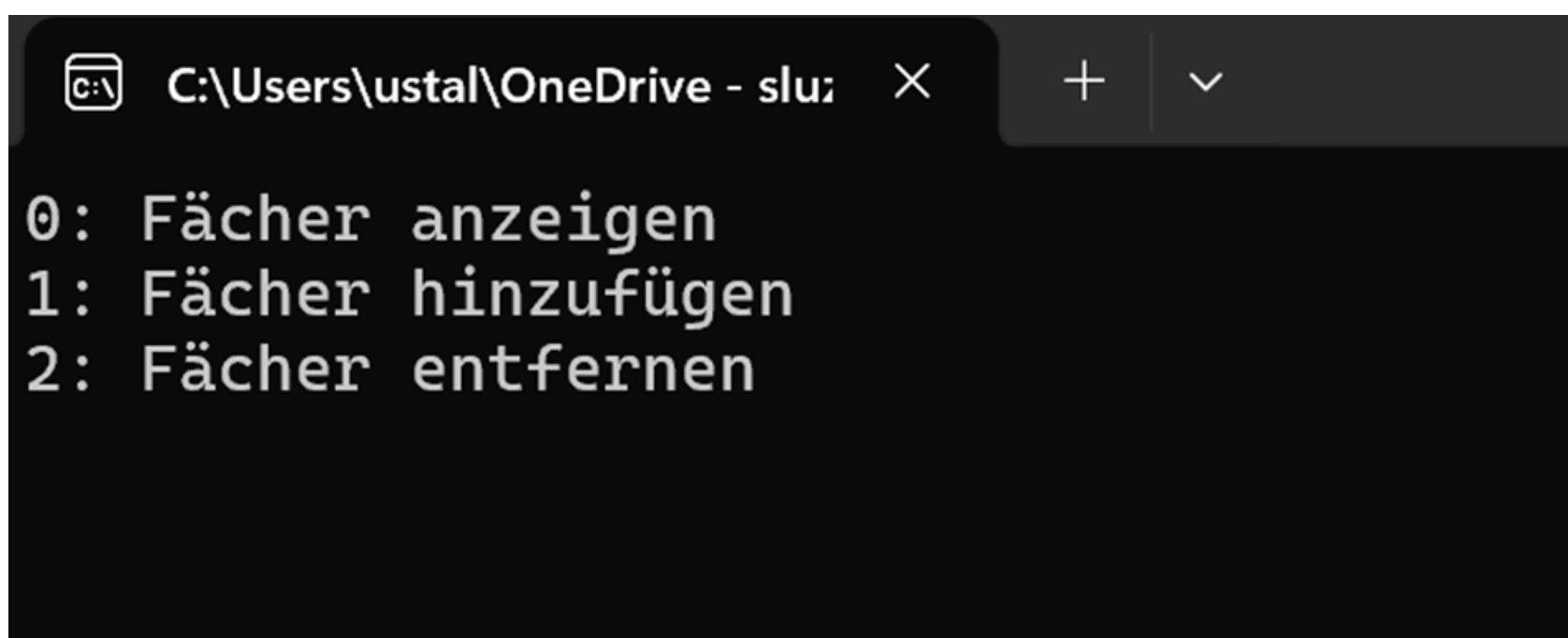
```
C:\Users\ustal\OneDrive - slu: + ▾
```

0: Lehrer anzeigen
1: Lehrer hinzufügen
2: Lehrer entfernen
3: Fach zu Lehrer hinzufügen
4: Fach zu Lehrer entfernen

- 0: Wenn man 0 klickt, wird eine Liste mit allen Lehrern angezeigt.
- 1: Wenn man 1 klickt, kann man einen Lehrer erstellen, welcher dann in der Liste angezeigt wird (Da muss man die Anweisungen befolgen).
- 2: Wenn man 2 klickt, kann man einen Lehrer aus der Liste entfernen.
- 3: Wenn man 3 klickt, kann man ein Fach zu einem Lehrer hinzufügen. Dann würde der Lehrer dies nun ebenfalls unterrichten
- 4: Wenn man 4 klickt, dann kann man ein Fach von einem Lehrer entfernen. Dann würde der Lehrer das nicht mehr unterrichten.

2: Fach Infos/bearbeiten

Wenn man 2 klickt, kommt eine Seite, mit der man die Infos über die Lehrer sieht und diese bearbeiten kann:



```
C:\Users\ustal\OneDrive - slu: + ▾
```

0: Fächer anzeigen
1: Fächer hinzufügen
2: Fächer entfernen

- 0: Wenn man 0 klickt, sieht man eine Liste mit allen Fächern, die es gibt.
- 1: Wenn man 1 klickt, kann man ein Fach zur Liste hinzufügen (Anweisungen befolgen).
- 2: Wenn man 2 klickt, kann man ein Fach aus der Liste entfernen (Angabe von angezeigtem Index).

3: Schulklassen Infos/bearbeiten

Wenn man 3 klickt, kommt eine Seite, mit der man die Infos über die Schulklassen sieht und diese bearbeiten kann:



0: Wenn man 0 klickt, sieht man eine Liste mit allen Schulklassen, die es gibt.

1: Wenn man 1 klickt, kann man eine Schulkasse zur Liste hinzufügen

2: Wenn man 2 klickt, kann man eine Schulkasse aus der Liste entfernen

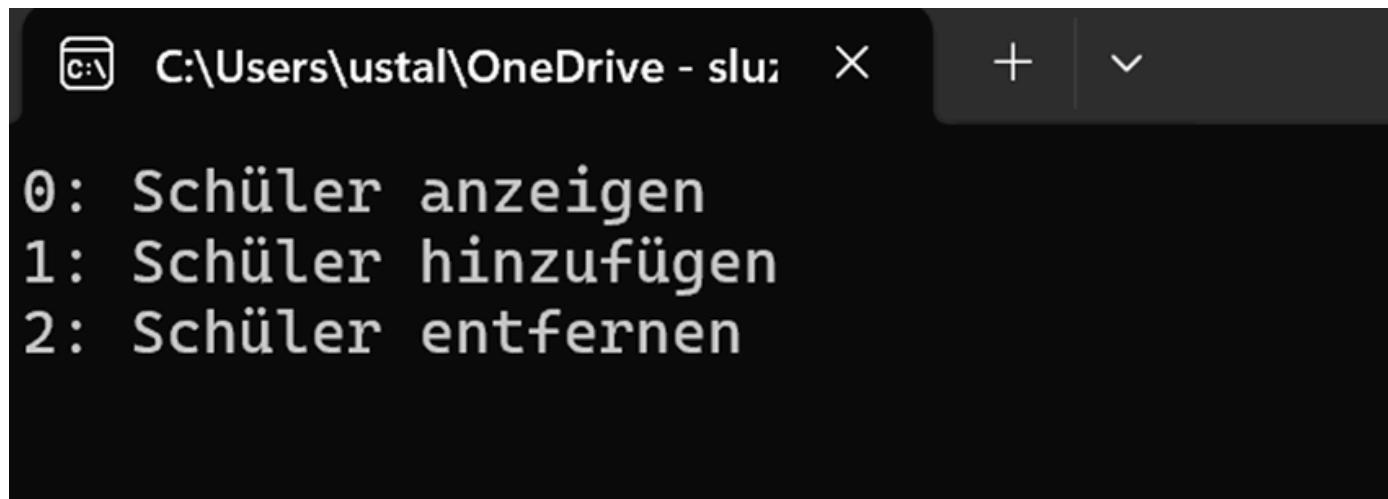
3: Wenn man 3 klickt, kann man Schüler einer Klasse hinzufügen

4: Wenn man 4 klickt, kann man Schüler aus einer Klasse entfernen

5: Wenn man 5 klickt kann man nachschauen welche Schüler in welchen Klassen sind

4: Schüler Infos/bearbeiten

Wenn man 4 klickt, kommt eine Seite, mit der man die Infos über die Lehrer sieht und diese bearbeiten kann:



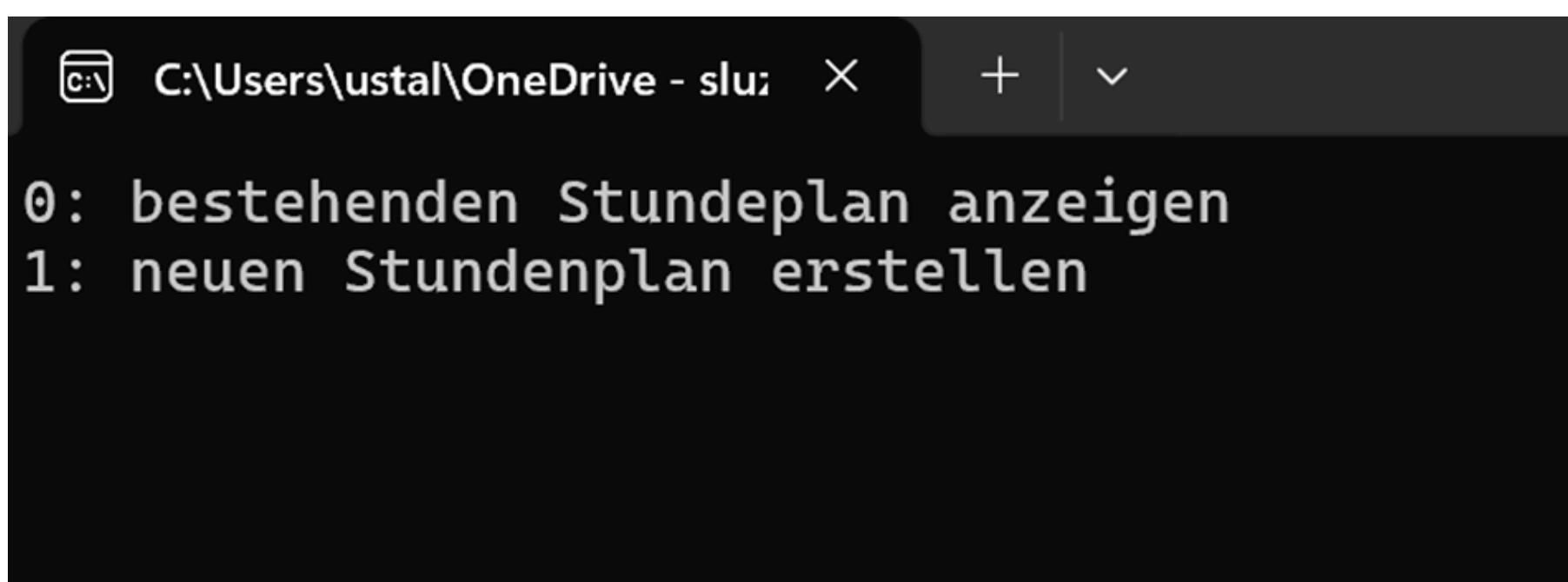
0: Wenn man 0 klickt, sieht man eine Liste mit allen Schülern, die es gibt.

1: Wenn man 1 klickt, kann man einen Schüler zur Liste hinzufügen
(Anweisungen vom Programm befolgen).

2: Wenn man 2 klickt, kann man einen Schüler aus der Liste entfernen.

5: Stundenplan Infos/bearbeiten

Wenn man 5 klickt, kommt eine Seite, mit der man die Infos über die Stundenpläne sieht und diese bearbeiten kann:



0: Wenn man 0 klickt, sieht man die Stundenpläne für alle Klassen, insofern man die erstellt hat.

1: Wenn man 1 klickt, erstellt man die Stundenpläne für alle Klassen, welche dann bestimmte Bedingungen erfüllt haben können, die man davor einstellen kann. Wie stark zum Beispiel Randzeiten, Zwischenstunden und Effizienz bewertet werden.

6: Beenden

Wenn man 6 klickt, beendet sich das Projekt mit einem weiteren Nutzen der Enter Taste.

07 QUALITÄTSKRITERIEN UND BEWERTUNG

Es ist wichtig, dass wir den besten Stundenplan haben, deswegen werden mehrere einzigartige Stundenpläne generiert. Jeder Stundenplan bekommt 1000 Punkte und wird runter gesetzt, wenn es eine Bedingung nicht erfüllt. Bei der Generierung werden auf drei Bedingungen geprüft, Randzeiten, Räume verwendet und Zwischenstunden. Es werden zuerst mal die Randzeiten bewertet. Wenn die Stunde an dem Tag um 8:00 Uhr beginnt oder die letzte stunde um 17 Uhr ist wird der Value runtersetzt. Die nächste Bewertung, die durchgeführt wird, ist die Bewertung, wie viele Räume pro Stundenplan benutzt werden. Dafür wird geprüft, ob der Raum schon benutzt wurde, falls nicht, wird ein Raum hinzugefügt. Wenn es schon benutzt wurde, wird Value runtersetzt. Man will so wenig wie möglich eine Raumänderung, sodass man auch verschiedene Fächer und Lektionen in dem gleichen Raum hat. Die letzte Validierung ist die Zwischenstunden, dafür wird die erste Stunde genommen um zum Beispiel 9:00. Dann wird geprüft ob in der nächsten Stunde eine Lektion stattfindet, falls nicht, werden Punkte abgezogen, weil man will, so wenige Zwischenstunden wie möglich. Nach alldem wird der Stundenplan mit den höchsten Value ausgegeben, gespeichert und genutzt.

```
1 Verweis
private int GetValuation(List<Combination> timetable)
{
    int value = 1000;
    HashSet<string> memorizeRooms = new();
    foreach (var t in timetable)
    {
        if ((t.Time.BlockIndex == 0) || (t.Time.BlockIndex == WorkHours - 1))
        {
            value -= 5;
        }
        if (!memorizeRooms.Contains(t.Room.RoomId))
            memorizeRooms.Add(t.Room.RoomId);
        else
            value -= 5;
    }

    int firstHour = int.MaxValue;
    foreach (var item in timetable)
        if (item.Time.BlockIndex < firstHour)
            firstHour = item.Time.BlockIndex;

    for (int i = firstHour; i < WorkHours - 1; i++)
    {
        bool isValue = false;
        foreach (var item in timetable)
        {
            if(item.Time.BlockIndex == i) {
                isValue = true;
                break;
            }
        }
        if (!isValue) value -= 20;
    }
    return value;
}
```

The code is annotated with three colored boxes highlighting specific logic:

- A yellow box highlights the logic for handling boundary times (Randzeiten):

```
if ((t.Time.BlockIndex == 0) || (t.Time.BlockIndex == WorkHours - 1))  
{  
    value -= 5;  
}
```
- A red box highlights the logic for tracking and penalizing the use of rooms (Verwendete Räume):

```
if (!memorizeRooms.Contains(t.Room.RoomId))  
    memorizeRooms.Add(t.Room.RoomId);  
else  
    value -= 5;
```
- A green box highlights the logic for calculating breaks between lessons (Zwischenstunden):

```
int firstHour = int.MaxValue;  
foreach (var item in timetable)  
    if (item.Time.BlockIndex < firstHour)  
        firstHour = item.Time.BlockIndex;  
  
for (int i = firstHour; i < WorkHours - 1; i++)  
{  
    bool isValue = false;  
    foreach (var item in timetable)  
    {  
        if(item.Time.BlockIndex == i) {  
            isValue = true;  
            break;  
        }  
    }  
    if (!isValue) value -= 20;  
}
```

Annotations with arrows point from the labels to the corresponding code blocks:

- A yellow arrow points to the yellow-highlighted boundary times logic.
- An orange arrow points to the red-highlighted room usage logic.
- A green arrow points to the green-highlighted break/lesson logic.

08 TESTS UND BEISPIELAUFGABE

1. Menüauswahl (0 / 1)

- 0: Zeigt einen bereits gespeicherten Stundenplan an.
1: Startet die Erstellung eines neuen Stundenplans.

0: bestehenden Stundeplan anzeigen
1: neuen Stundenplan erstellen

2. Gewichtung der Randzeiten

Du gibst an, wie stark der Plan vermeiden soll, dass Lektionen ganz am frühen Morgen oder späten Nachmittag liegen.

Wie hoch sollen Randzeiten gewichtet werden? (0 – 20):

3. Gewichtung der Zwischenstunden

Du bestimmst, wie wichtig es ist, dass möglichst keine langen Pausen zwischen Lektionen entstehen.

Wie hoch sollen Zwischenstunden gewichtet werden? (0 – 20):

4. Gewichtung der Raumnutzung

Du legst fest, wie sehr der Plan darauf achten soll, dass Räume effizient genutzt werden (z. B. keine unnötig leeren Zimmer).

Wie hoch soll die effiziente Nutzung der Räume gewichtet werden? (0 – 40):

08 BEISPIELAUFGABE LEHRPLAN

5. Erstellter Stundenplan

Der Stundenplan wird pro Klasse erstellt, und wird nach Tag und Zeit sortiert. Der Stundenplan zeigt das Fach pro Stunde an, folgend vom Lehrer mit Vor- und Nachnamen, und anschliessend in welchem Raum der Unterricht stattfindet.

----- 2A					
Montag	11:00 - 12:00	Informatik	Timo	Bauer	R107
Montag	12:00 - 13:00	Physik	Simon	Fischer	WIR1
Montag	13:00 - 14:00	Programmierung	Mila	Aydin	R110
Montag	15:00 - 16:00	Biologie	Eva	Arnold	ART1
Montag	16:00 - 17:00	Biologie	Eva	Arnold	LAB2
Dienstag	10:00 - 11:00	Wirtschaft	Daniel	Ziegler	LAB1
Dienstag	12:00 - 13:00	Physik	Simon	Fischer	R107
Dienstag	13:00 - 14:00	Wirtschaft	Daniel	Ziegler	PHIL1
Dienstag	14:00 - 15:00	Technik	Timo	Bauer	TEC1
Dienstag	15:00 - 16:00	Informatik	Mila	Aydin	R101
Mittwoch	9:00 - 10:00	Recht	Sophie	Lehmann	R109
Mittwoch	10:00 - 11:00	Französisch	Laura	Brunner	LAN1
Mittwoch	13:00 - 14:00	Biologie	Eva	Arnold	R108
Mittwoch	14:00 - 15:00	Physik	Anna	Meier	PHIL1
Mittwoch	16:00 - 17:00	Chemie	Simon	Fischer	LAB2
Donnerstag	8:00 - 9:00	Mathematik	Anna	Meier	R104
Donnerstag	9:00 - 10:00	Informatik	Timo	Bauer	R105
Donnerstag	12:00 - 13:00	Physik	Simon	Fischer	LAW1
Donnerstag	13:00 - 14:00	Kunst	Nora	Graf	TEC1
Donnerstag	16:00 - 17:00	Chemie	Simon	Fischer	R104
Freitag	8:00 - 9:00	Spanisch	Clara	Huber	R101
Freitag	9:00 - 10:00	Recht	Sophie	Lehmann	R102
Freitag	10:00 - 11:00	Geschichte	Laura	Brunner	R106
Freitag	12:00 - 13:00	Mathematik	Anna	Meier	GYM
Freitag	13:00 - 14:00	Deutsch	Lukas	Keller	LAW1

09 REFLEXION STUNDENPLANER PROJEKT

Halldór:

In diesem Projekt habe ich vor allem an der Dokumentation gearbeitet und in der Eintragung, von wer was gemacht hat, dazu wurde mir der Code vollständig vom Sanji erklärt und ich habe auch einen Teil vom code selbst geschrieben.

Kenan:

Ich habe in diesem Projekt hauptsächlich dokumentarische Teile bearbeitet, wie das UML-Diagramm, das Use-Case Diagramm oder die Benutzeranleitung. Ich musste, denn Code dennoch verstehen, damit ich diese Aufgaben erledigen konnte. Bei dem gab es bei mir grössere Schwierigkeiten, da ich den Code als sehr kompliziert empfunden habe. Nachdem ich Sanjivan mehrmals gefragt habe, lief es mir einfacher. Dank des Projektes hat sich mein Wissen in den Diagrammen vor allem deutlich erweitert. Code zu lesen, ist nun auch keine grosse Hürde mehr für mich. Generell würde ich sagen, dass dieses Projekt mein Wissen positiv erweitert hat.

Bei meinem nächstem Projekt werde ich persönlich mehr GitHub benutzen, da ich in diesem sehr viel Hilfe dabei erhalten habe. Um das zu erreichen werde ich mich in GitHub einlesen.

Sanjivan:

In dem Projekt habe ich die Rolle als Programmierer eingenommen, welcher auch die Konzeptionierung vorgenommen hat. Dazu hinaus habe ich mich auch an dem Management von Abläufen und To-do's konzentriert und beteiligt. Dieses Schulprojekt war eines der schwersten bis jetzt, wenn man die Planung und Einteilung berücksichtigt, welche wir machen, mussten während der Projektzeit. Ich habe da gelernt wie ich die Arbeit unter uns so aufteilt das jeder parallel an Aufgaben arbeiten, dass wir nicht ins Hintertreffen gelangen. Darüber hinaus war die Programmierung des Algorithmus, durchaus herausfordernd. Ich musste dafür vor der Programmierung ein Konzept entwerfen, wie das Vorgehen da das Beste wäre. Mein Unity Projekt, wo ich einen Roguelike Dungeon Explorer Spiel entwickelt habe, hat mir da sehr geholfen, evolutionsbasiert bzw. auf Zufallsgenerierter Basis Probleme zu lösen. Das Projekt war noch eine gute Möglichkeit meine C#-Skills weiter zu verbessern, ich habe beispielsweise gelernt, besser und häufiger Linq und Lambda zu benutzen, darüber hinaus weiss ich jetzt, wie ich Daten in JSON-Format speichern kann, was mir später durchaus nützlich werden könnte.

Berat:

In unserem Projekt war ich vor allem an den dokumentarischen Teil und an GitHub dran. Wie z.B. Arbeitsrapport, Anforderungen, Kriterien und halt das GitHub-Repository. Dennoch habe ich ein bisschen am Code gearbeitet.

Bei mir gab es vor allem Probleme bei GitHub. Ich konnte Daten von meinem lokalen PC nicht auf GitHub commiten oder pushen. Dazu konnten wir erst später eine Lösung finden, bis dann hat Sanjivan für mich meine Codezeilen commitet und gepusht. Am Anfang hatte ich noch sehr wenig Ahnung, was im Code passierte, deshalb erklärte Sanjivan mir den Code. Nach seiner Erklärung habe ich das meiste verstanden und auch viel neues gelernt.

Das Projekt hat mir sehr gefallen und ich konnte auch sehr viel neues lernen. Wenn wir wieder ein Projekt machen würden, müsste ich mich mehr bei GitHub einlesen, da ich noch sehr unerfahren bin und kaum GitHub benutze. Ich werde in meinem Alltag mehr GitHub benutzen, um mein Verständnis zu verbessern.

Gruppenfazit:

Als Gruppe konnten wir in diesem Projekt viel lernen sowohl fachlich als auch organisatorisch. Wir haben gemerkt, dass eine gute Zusammenarbeit vor allem dann funktioniert, wenn alle Beteiligten verstehen, wie der Code aufgebaut ist und wie die einzelnen Teile miteinander zusammenhängen. Besonders hilfreich war dabei, dass wir uns gegenseitig unterstützt haben: Wenn jemand Schwierigkeiten hatte, sei es beim Code oder bei GitHub, konnte er sich auf die anderen verlassen.

Ein wichtiger gemeinsamer Lernpunkt war der Umgang mit GitHub. Mehrere Gruppenmitglieder hatten am Anfang Mühe damit, wodurch wir gemerkt haben, wie zentral Versionsverwaltung für die Zusammenarbeit ist. Für zukünftige Projekte nehmen wir mit, dass sich alle früher und intensiver in GitHub einarbeiten sollten, um effizienter parallel arbeiten zu können.

Auch die Aufteilung der Aufgaben hat uns gezeigt, wie wichtig klare Rollen und gute Planung sind. Durch die Mischung aus Programmierung, Dokumentation, Diagrammen und Management konnten alle etwas beitragen, und jeder hat sein Wissen in einem bestimmten Bereich erweitert. Gleichzeitig wurde uns bewusst, dass gegenseitige Erklärungen und ein gemeinsames Verständnis der Code-Logik entscheidend sind, damit das Projekt als Ganzes funktioniert.

Zusammengefasst hat uns das Projekt gezeigt, wie wichtig Kommunikation, gegenseitige Unterstützung, strukturiertes Arbeiten und der richtige Umgang mit Tools wie GitHub sind. Als Gruppe konnten wir nicht nur unsere technischen Fähigkeiten, sondern auch unsere Teamkompetenzen deutlich stärken.

Arbeitsrapport

Datum	Person	Aufgabe	Zeit (min)	Status	Bemerkung
23.10.2025	Halldor	Arbeitsrapport Tag 1	30	Fertig	Wurde auf Canvas gemacht
23.10.2025	Berat	Git repository	30	Fertig	
23.10.2025	Sanjivan	Erstes UML Diagramm	150	Fertig	
23.10.2025	Halldor	Deckblatt	15	Fertig	Wurde auf Canvas gemacht
23.10.2025	Berat	Use-Case Beschreibung / Projektbeschreibung / Anforderungen	120	Fertig	
26.10.2025	Sanjivan	Code Erstellung Prototyp (Hauptklassen, Subject, Teacher, Student, Schoolclass, Room)	120	Fertig	Zu Hause
29.10.2025	Sanjivan	Code Implementierung Speicherung in JSON, Konzept Algorithmus Entwicklung	90	Fertig	Zu Hause
30.10.2025	Berat	Arbeitsrapport Tag 2	30	Fertig	Auf Excel gewechselt
30.10.2025	Kenan	UML Diagramm Erweiterung	180	Fertig	
30.10.2025	Berat	Sequenzdiagramm	30	Abgebrochen	Abgebrochen, weil wir einen Sequenzdiagramm nach dem Coden machen wollten, welches viel zu lange gebraucht hätte.
30.10.2025	Sanjivan	Code Ersteilung Matrix und Selektierung Stundenplan	120	Fertig	
30.10.2025	Berat, Kenan, Sanjivan	Konzept Algorithmus und persistente Speicherung vom Code erklärt	30	Fertig	Sanjivan hat Berat und Kenan Konzept Algorithmus und persistente Speicherung im Code erklärt
30.10.2025	Berat	Use-Cases Szenarien	10	Fertig	
31.10.2025	Kenan	UML Diagramm Erweiterung	30	Fertig	
31.10.2025	Berat	Github Repository docx	10	Fertig	
04.11.2025	Sanjivan	Code Validation Stundenpläne und Selektierung nach höchste Value	150	Fertig	Zu Hause
05.11.2025	Sanjivan	Code Bugfixes Algorithmus, Validationskriterien eingeführt, Zwischenstunden	120	Fertig	Zu Hause
06.11.2025	Berat	Use-Cases Beschreibungen Verbesserung	20	Fertig	Die verschiedenen Use-Cases verbessert
06.11.2025	Kenan	UML Diagramm Erweiterung	30	Fertig	
06.11.2025	Sanjivan	Code Erweiterung	60	Fertig	
06.11.2025	Halldor	Einführung Ablauf Algorithmus	30	Fertig	
06.11.2025	Berat	Arbeitsrapport Tag 3	30	Fertig	
06.11.2025	Berat	Github Repository docx	30	Fehler	Problem beim Pushen
12.11.2025	Sanjivan	Benutzerinteraktionen codiert (Program), unnötigen Relationen gelöscht	120	Teilweise	Zu Hause
		Finales Projekt			
13.11.2025	Berat	Arbeitsrapport Tag 4	30	Fertig	Wollte Ampel-System für den Status einfügen, doch hatte Probleme mit den Rules und funktionierte nicht mehr.
13.11.2025	Kenan	Use-Case Diagram	60	Fertig	
13.11.2025	Halldor	Code Klasse: Teacher, Subject	30	Fertig	Gepusht
13.11.2025	Berat	Code Klasse: Room, SchoolClass	30	Fertig	Fehler beim pushen. Sanjivan hat für Berat gepusht und committed
13.11.2025	Sanjivan	Code Klasse: Datamanager	20	Fertig	Gepusht
19.11.2025	Sanjivan	Code: Zusammengesetzt, Schlüsselklassen vom Prototypen importiert	60	Fertig	Zu Hause; Klassen: Algo, Combination, Timeblock
19.11.2025	Sanjivan	Bugfixes	120	Fertig	Zu Hause
20.11.2025	Kenan	Benutzeranleitung	100	Fertig	
20.11.2025	Berat	Arbeitsrapport Tag 5	30	Fertig	
20.11.2025	Berat	Qualitätskriterien	80	Fertig	Sanjivan musste noch ganz kurz erklären wie der Algorithmus funktioniert. Gepusht
20.11.2025	Kenan	Kenans Reflexion	15	Fertig	
20.11.2025	Sanjivan	Halldor Algorithmus, Details vom Code erklärt	40	Fertig	

25.11.2025	Sanjivan	Bugfixes, Benutzeroberfläche erweitert	120	Fertig	Zu Hause
26.11.2025	Sanjivan	Bugfixes, Benutzeroberfläche erweitert, Kommentare hinzugefügt, JSON Speicherung eingesetzt	240	Fertig	Zu Hause
27.11.2025	Berat	Berats Reflexion	15	Fertig	
27.11.2025	Halldor	Halldors Reflexion	15	Fertig	
27.11.2025	Kenan	Use-Case Diagram Erweiterung	40	Fertig	
27.11.2025	Halldor	Code-Testing	30	Fertig	
27.11.2025	Sanjivan	Bugfixes	120	Fertig	Algo hat wegen persistente Speicherung nicht funktioniert
27.11.2025	Kenan	Umsetzung	120	Fertig	
27.11.2025	Berat	GitHub Repository finishes	10	Fertig	
27.11.2025	Berat	Demo	20	Fertig	
27.11.2025	Berat	Arbeitsrapport Tag 6	30	Fertig	
27.11.2025	Halldor	Alles auf Canvas übertragen	180	Fertig	
27.11.2025	Sanjivan	ProjektAbgabe	1 Sekunde	Fertig	