

# **Google Calendar Project**

## **Dokumentation zu Teil 1 und Teil 2**

Name: Dominik Mitzel

Matrikelnummer: 6633800

Studiengang: Angewandte Informatik

Modul: Programmieren

Bei: Prof. Dr. Holger D. Hofmann

Datum: 28. August 2025

# Inhaltsverzeichnis

<b>Aufgabenstellungen und Installation</b>	<b>3</b>
0.1 Überblick über die Aufgabenstellungen . . . . .	3
0.2 Lokale Installation . . . . .	3
0.3 Hinweis (Disclaimer) . . . . .	4
<b>I Aufgabe 1 - iCal Export via CLI</b>	<b>5</b>
<b>1 Programmbeschreibung</b>	<b>5</b>
1.1 Kernfunktionalität . . . . .	5
1.2 Eingabemodi . . . . .	5
1.3 Eingabevalidierung . . . . .	5
1.4 Fehlerbehandlung . . . . .	6
<b>2 Dokumentation</b>	<b>6</b>
2.1 Projektverzeichnis . . . . .	6
2.2 Klassendiagramm . . . . .	8
2.3 CalenderApiConnector Klasse . . . . .	10
2.4 MyEvent-Struktur: Event-Modellierung und -Erstellung . . . . .	13
2.5 ICal - Zentrale Container-Klasse für Kalenderdaten . . . . .	22
2.6 InputValidator - Utility-Klasse für Eingabevalidierung . . . . .	26
2.7 G2iCal - Hauptanwendungsklasse . . . . .	27
2.8 Ablaufdarstellung als UML Sequenz-Diagramm . . . . .	30
<b>3 Fragen</b>	<b>32</b>
3.1 Welche Funktionen bietet das Tool Gradle? Wofür kann es eingesetzt werden? . . . . .	32
3.2 Was bedeuten die Schritte „API aktivieren“ und „OAuth-Zustimmungsbildschirm konfigurieren“? . . . . .	33
3.3 Wie kann man in einer IDE die Google Calendar API verfügbar machen? . . . . .	34
3.4 Erkläre die Code-Zeilen mit GoogleNetHttpTransport und Calendar.Builder . . . . .	35
<b>II Aufgabe 2 - GUI-Erweiterung mit Swing</b>	<b>36</b>
<b>4 Programmbeschreibung</b>	<b>36</b>
4.1 UI-Screenshots . . . . .	37
<b>5 Dokumentation</b>	<b>40</b>
5.1 UI/UX-Design . . . . .	40
5.2 Projektverzeichnis . . . . .	40
5.3 Umsetzung des Model-View-Controller-Patterns . . . . .	41
5.4 Klassendiagramm . . . . .	42
<b>6 Fragen</b>	<b>50</b>
6.1 Wie definiert man Zeilen/Spalten für ein JTable-Objekt . . . . .	50
6.2 Was muss beachtet werden, wenn bei einem JTable-Objekt die Spaltenüberschriften angezeigt werden sollen? . . . . .	51

6.3	Welche Layoutmanager haben Sie verwendet? Begründen Sie Ihre Wahl. . .	52
6.4	Wie definiert man, welche Aktion bei der Auswahl eines Menüs ausgeführt werden soll? . . . . .	53

## Abbildungsverzeichnis

1	Klassendiagramm CLI . . . . .	9
2	Sequenz Diagramm . . . . .	31
3	UI Screenshots – Screens . . . . .	38
4	UI Screenshots – Dialoge und Popups 2 . . . . .	39
5	Planung der UI . . . . .	40
6	MVC Pattern (Bildquelle: Wikipedia) . . . . .	41
7	Klassendiagramm Teil 2 . . . . .	43

# Aufgabenstellungen und Installation

## 0.1 Überblick über die Aufgabenstellungen

**Teil 1:** Entwicklung eines Java-Programms zur Abfrage eines Google-Kalenders über die Google Calendar API. Die Ergebnisse sollen im iCalendar-Format exportiert werden. Der Aufruf erfolgt über die Kommandozeile.

**Teil 2:** Erweiterung des Programms um eine grafische Benutzeroberfläche (z.B. Swing oder JavaFX), inklusive Menüs und Dialogfenstern zur Eingabe und Anzeige der Kalenderdaten.

## 0.2 Lokale Installation

In diesem Abschnitt wird die Installation und Einrichtung des Projekts beschrieben. Es wird vorausgesetzt, dass **Java** (Version 17 oder neuer), **Git** sowie **Gradle** lokal installiert sind.

### 0.2.1 Google Cloud Projekt und API-Zugang

Um auf den Google Kalender zugreifen zu können, wird ein Google-Cloud-Projekt mit aktivierter Calendar API benötigt.

1. Erstellen Sie ein neues Projekt oder wählen Sie ein bestehendes Projekt aus. (siehe Google-Cloud-Erklärung)
2. Aktivieren Sie in der Google Cloud Console die Google Calendar API: klicken Sie hier
3. Gehen Sie zu **APIs & Dienste** → **OAuth** und erstellen Sie:
  - Einen **OAuth 2.0 Client** für eine Desktop-Anwendung.
  - Laden Sie die erzeugte Datei `credentials.json` herunter.
4. Fügen Sie unter **OAuth-Zustimmungsbildschirm** den verwendeten Google-Account als **Testnutzer** hinzu.

### 0.2.2 Projekt von GitHub klonen

Das Projekt ist auf GitHub verfügbar und kann wie folgt heruntergeladen werden:

```
git clone https://github.com/nikmtl/G2iCal
```

### 0.2.3 Hinzufügen der Credentials

Die heruntergeladene Datei `credentials.json` muss in den Ordner `src/main/resources/` des Projekts kopiert werden. Ohne diese Datei ist kein Zugriff auf die Google Calendar API möglich. Bitte stellen Sie sicher, dass die Datei exakt `credentials.json` heißt.

### 0.2.4 Starten der Anwendung

Das Projekt besteht aus zwei Komponenten: einer reinen CLI-Version (Teil 1) und einer erweiterten GUI-Version (Teil 2).

### **Start der CLI-Version (Teil 1):**

```
./gradlew runCli --args=""
```

Bitte beachten Sie: Im interaktiven Modus muss ein Terminal verwendet werden, das Benutzereingaben unterstützt. Andernfalls erscheint eine Fehlermeldung wie „Error: No input available. Please run the program in an interactive terminal.“

### **Start der GUI-Version (Teil 2):**

```
./gradlew runGui
```

Beim ersten Start öffnet sich ein Browserfenster für den OAuth-Anmeldeprozess. Nach erfolgreicher Anmeldung werden die Zugangsdaten lokal gespeichert, sodass der Login beim nächsten Start entfällt.

## **0.3 Hinweis (Disclaimer)**

Das Projekt wird in dieser Form für Test- und Studienzwecke bereitgestellt. Die Ausführung erfolgt aktuell als Quellcode-Projekt, das lokal mit eigenen `credentials.json`-Dateien arbeitet. In einer final bereitgestellten, offiziell freigegebenen Version würde der Code kompiliert ausgeliefert, und es käme ein von Google bestätigtes Projekt zum Einsatz. Dadurch müssten keine eigenen Credentials mehr eingefügt werden, was die Nutzung deutlich vereinfachen und nutzerfreundlicher gestalten würde.

### **Warum wird `./gradlew` verwendet?**

Der Start erfolgt über Gradle, da dies den Vorteil bietet, die unterschiedlichen Startpunkte (CLI oder GUI) direkt auszuführen, ohne dass der Nutzer die Main-Klasse kennen oder komplexe java-Kommandos eingeben muss. Darüber hinaus übernimmt Gradle technische Aufgaben wie das Setzen des Klassenpfads und stellt sicher, dass alle Abhängigkeiten korrekt eingebunden sind. Dies erleichtert die Nutzung erheblich und vermeidet Fehler beim manuellen Start.

# Teil I

# Aufgabe 1 - iCal Export via CLI

## 1 Programmbeschreibung

**G2iCal** ist eine Java-basierte Kommandozeilenanwendung, die Termine aus Google Calendar in das iCal-Format (.ics-Dateien) exportiert. Das Programm bietet sowohl einen interaktiven als auch einen Kommandozeilenmodus für flexible Nutzung.

### 1.1 Kernfunktionalität

Das Programm folgt diesem Arbeitsablauf:

1. **Authentifizierung & API-Verbindung:** Stellt eine sichere Verbindung zur Google Calendar API über OAuth 2.0-Anmelddaten her
2. **Kalenderauswahl:** Ruft die verfügbaren Google-Kalender des Benutzers ab und ermöglicht die Auswahl über einen Index
3. **Datumsbereich festlegen:** Akzeptiert Start- und Enddatum zur Definition des Exportzeitraums
4. **Termindaten abrufen:** Holt alle Termine aus dem ausgewählten Kalender innerhalb des angegebenen Datumsbereichs
5. **iCal-Konvertierung:** Wandelt Google Calendar-Termine in das Standard-iCal-Format um
6. **Dateiexport:** Speichert die konvertierten Termine als .ics-Datei im Downloads-Ordner des Benutzers

### 1.2 Eingabemodi

**Kommandozeilenmodus:** Akzeptiert alle Parameter als Argumente im Format:

```
./gradlew runCli --args="<Start-Datum> <End-Datum> <Dateiname> <Kalender-Index>"
```

**Interaktiver Modus:** Falls Kommandozeilenargumente fehlen oder ungültig sind, fordert das Programm den Benutzer zur Eingabe auf mit Validierung und Wiederholungsmechanismen.

### 1.3 Eingabeverifikation

Das Programm validiert alle Benutzereingaben:

- **Daten:** Müssen im Format YYYY-MM-DD vorliegen, gültige Daten repräsentieren und in richtiger Reihenfolge sein. Damit auch nur ein Tag ausgewählt werden kann ist das Startdatum immer bei 00:00 Uhr und das Enddatum bei 12:59 Uhr.
- **Dateinamen:** Werden auf gültige Zeichen geprüft und erhalten automatisch die .ics-Erweiterung, falls sie fehlt

- **Kalender-Index:** Wird gegen die verfügbare Kalenderliste geprüft

## 1.4 Fehlerbehandlung

Die Anwendung umfasst umfassende Fehlerbehandlung für:

- API-Verbindungsfehler
- Ungültige Datumsformate oder -bereiche
- Dateisystemfehler beim Export
- Netzwerkverbindungsprobleme
- Authentifizierungsprobleme

Das Programm gewährleistet ein elegantes Verhalten bei Fehlern mit informativen Fehlermeldungen und angemessenen Exit-Codes.

## 2 Dokumentation

### 2.1 Projektverzeichnis

Das G2iCal-Projekt folgt der Standard-Gradle-Verzeichnisstruktur und ist in logische Packages unterteilt:

```
google_calendar_iCal/
|-- .gradle/
|-- build/                               # Gradle Build-Ausgabe
|-- docs/                                # Documentation
|-- gradle/                             # Gradle Wrapper-Dateien

|-- src/main/
|   |-- java/
|       |-- G2iCalGUI.java                # Main GUI Version
|       |-- G2iCal.java                  # Main CLI Version
|       |-- calendar/
|           |-- CalendarEvent.java      # Event Class Interface
|           |-- ICal.java              # ICal Class
|           |-- MyEvent.java          # MyEvent Class
|           +-+ MyEventBuilder.java    # MyEvent Builder Class
|       |-- exceptions/
|           +-+ ICalExportException.java # Exception-Handling for ICal export
|       +-+ utils/
|           |-- CalendarApiConnector.java # Connection to the Google API
|           |-- EventConverter.java     # Convert Google event into MyEvent
|           |-- EventFactory.java      # Eventfactory MyEvent
|           +-+ InputValidator.java    # Input Validator for Date, Path and Name
|   +-+ resources/
|       +-+ credentials.json          # Google Cloud Credentials
|-- tokens/                                # Google API Token-Speicher
```

```
|-- .gitignore
|-- build.gradle                               # Gradle Build-Konfiguration
|-- gradlew                                     # Gradle Wrapper (Unix)
|-- gradlew.bat                                 # Gradle Wrapper (Windows)
|-- settings.gradle.kts                         # Gradle Einstellungen
```

### 2.1.1 Package-Beschreibungen

#### calendar/ Package

Enthält die Kern-Domain-Objekte für Kalenderereignisse und iCal-Container.

##### Klassen:

- `CalendarEvent.java` – Event-Interface (Abstraktion)
- `ICal.java` – Haupt-iCal-Container-Klasse
- `MyEvent.java` – Konkretes Event-Objekt
- `MyEventBuilder.java` – Builder Pattern für Event-Erstellung

##### Verantwortlichkeiten:

- Definition der Event-Struktur
- iCal-Format-Repräsentation
- Event-Objekterstellung mit Builder Pattern

#### exceptions/ Package

Behandlung anwendungsspezifischer Ausnahmen und Fehlerbehandlung.

##### Klassen:

- `ICalExportException.java` – Export-spezifische Ausnahmen

##### Verantwortlichkeiten:

- Behandlung von Export-Fehlern
- Strukturierte Fehlerbehandlung
- Benutzerfreundliche Fehlermeldungen

#### utils/ Package

Utility-Klassen für unterstützende Funktionalitäten.

##### Klassen:

- `CalendarApiConnector.java` – Google Calendar API Integration
- `EventConverter.java` – Event-Konvertierungslogik
- `EventFactory.java` – Factory Pattern für Event-Erstellung
- `InputValidator.java` – Eingabeverifikation

## **Verantwortlichkeiten:**

- API-Konnektivität zu Google Calendar
- Datenkonvertierung zwischen Formaten
- Objekterstellung über Factory Pattern
- Validierung von Benutzereingaben

### **2.1.2 Hauptklasse**

**G2iCal.java:** Die Hauptklasse im Root-Package fungiert als Einstiegspunkt und Kommandozeilen-Interface der Anwendung.

### **2.1.3 Konfiguration und Ressourcen**

- `src/main/resources/credentials.json` – Google API Authentifizierungsdaten
- `tokens/StoredCredential` – OAuth-Token-Speicher
- `build.gradle` – Abhängigkeiten und Build-Konfiguration

## **2.2 Klassendiagramm**

Das Klassendiagramm in Abbildung 1 zeigt die Architektur der G2iCal-Anwendung mit ihren Hauptkomponenten und deren Beziehungen. Die Anwendung folgt objektorientierten Design-Prinzipien und verwendet verschiedene Design Patterns.

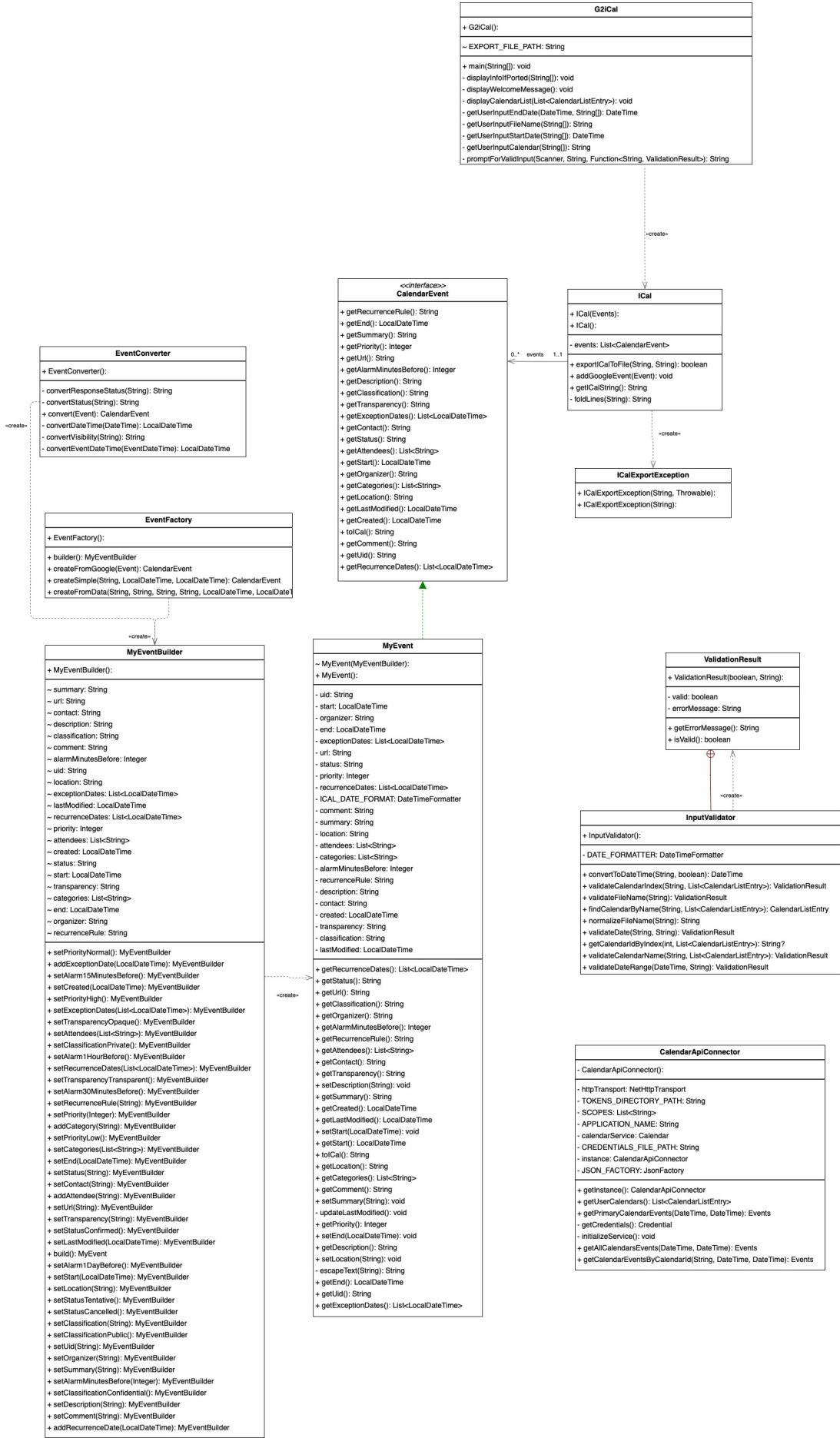


Abbildung 1: Klassendiagramm CLI  
9

## 2.3 CalenderApiConnector Klasse

Die `CalendarApiConnector`-Klasse ist das zentrale Element für die Kommunikation mit der Google Calendar API. Sie implementiert das Singleton Pattern und verwaltet die OAuth2-Authentifizierung sowie alle API-Interaktionen.

### 2.3.1 Singleton Pattern Implementation

Die Klasse verwendet das Singleton Pattern, um sicherzustellen, dass nur eine Instanz der API-Verbindung existiert:

```
1 public class CalendarApiConnector {  
2     // Singleton instance  
3     private static CalendarApiConnector instance;  
4  
5     // Private constructor to prevent direct instantiation  
6     private CalendarApiConnector() throws IOException,  
7         GeneralSecurityException {  
8         initializeService();  
9     }  
10  
11    // Thread-safe singleton access  
12    public static synchronized CalendarApiConnector getInstance()  
13        throws IOException, GeneralSecurityException {  
14        if (instance == null) instance = new CalendarApiConnector()  
15        ();  
16        return instance;  
17    }  
18}
```

#### Vorteile des Singleton Patterns:

- Verhindert mehrfache Authentifizierung
- Zentralisierte API-Verbindung
- Thread-sichere Implementierung durch `synchronized`
- Ressourcenschonung durch Wiederverwendung

### 2.3.2 OAuth2-Authentifizierung

Die Authentifizierung erfolgt über OAuth2 mit persistenter Token-Speicherung:

```
1 private static final List<String> SCOPES =  
2     Collections.singletonList(CalendarScopes.CALENDAR_READONLY);  
3 private static final String CREDENTIALS_FILE_PATH = "/credentials  
4     .json";  
5 private static final String TOKENS_DIRECTORY_PATH = "tokens";  
6  
6 private Credential getCredentials() throws IOException {  
7     // Load client secrets from credentials.json  
8     InputStream in = CalendarApiConnector.class  
9         .getResourceAsStream(CREDENTIALS_FILE_PATH);
```

```

10    if (in == null) {
11        throw new FileNotFoundException(
12            "Credential object resource not found: " +
13            CREDENTIALS_FILE_PATH);
14    }
15
16    GoogleClientSecrets clientSecrets = GoogleClientSecrets
17        .load(JSON_FACTORY, new InputStreamReader(in));
18
19    // Build authorization flow
20    GoogleAuthorizationCodeFlow flow = new
21        GoogleAuthorizationCodeFlow.Builder(
22            httpTransport, JSON_FACTORY, clientSecrets, SCOPES)
23            .setDataStoreFactory(new FileDataStoreFactory(
24                new java.io.File(TOKENS_DIRECTORY_PATH)))
25            .setAccessType("offline")
26            .build();
27
28    LocalServerReceiver receiver = new LocalServerReceiver.
29        Builder()
30        .setPort(8888).build();

31    return new AuthorizationCodeInstalledApp(flow, receiver).
32        authorize("user");
33}

```

### Authentifizierungs-Workflow:

1. Laden der Client-Secrets aus `credentials.json`
2. Aufbau des OAuth2-Flows mit Read-Only-Berechtigung
3. Persistente Token-Speicherung im `tokens/-Verzeichnis`
4. Lokaler Server auf Port 8888 für Callback-Handling
5. Offline-Zugriff für langfristige Token-Nutzung

### 2.3.3 Service-Initialisierung

Die Google Calendar API wird beim ersten Zugriff initialisiert:

```

1 private Calendar calendarService;
2 private NetHttpTransport httpTransport;
3
4 private void initializeService() throws IOException,
5     GeneralSecurityException {
6     // Build trusted HTTP transport
7     httpTransport = GoogleNetHttpTransport.newTrustedTransport();
8
9     // Create Calendar service with authenticated credentials
10    calendarService = new Calendar.Builder(httpTransport,
11        JSON_FACTORY, getCredentials())
12        .setApplicationName(APPLICATION_NAME)
13
14}

```

```

11     .build();
12 }
```

### Service-Komponenten:

- `NetHttpTransport`: Sichere HTTPS-Verbindung
- `GsonFactory`: JSON-Parsing für API-Responses
- `Credential`: OAuth2-Authentifizierung
- `APPLICATION_NAME`: "G2iCal" für API-Identifikation

#### 2.3.4 API-Interaktionsmethoden

##### Kalender-Listen-Abruf

```

1 public List<CalendarListEntry> getUserCalendars() throws
2   IOException {
3   CalendarList calendarList = calendarService.calendarList().
4     list().execute();
5   return calendarList.getItems();
6 }
```

Diese Methode ruft alle dem Benutzer zugänglichen Kalender ab und gibt sie als Liste zurück.

**Event-Abruf nach Kalender-ID und Datum** Diese Methode bildet den hauptsächlichen API-Aufruf, bei dem Kalenderevents in einem bestimmtem Zeitraum geladen werden.

```

1 public Events getCalendarEventsByCalendarId(String calendarId,
2   DateTime startTime, DateTime endTime) throws IOException
3   {
4   return calendarService.events().list(calendarId)
5     .setTimeMin(startTime) // Events must start after
6       this time
7     .setTimeMax(endTime) // Events must start
8       before this time
9     .setOrderBy("startTime") // Order events by start
10      time
11     .setSingleEvents(true) // Expand recurring events
12     .execute();
13 }
```

##### Query-Parameter:

- `setTimeMin/setTimeMax`: Zeitbereich-Filterung
- `setOrderBy("startTime")`: Chronologische Sortierung
- `setSingleEvents(true)`: Auflösung wiederkehrender Termine

### 2.3.5 Sicherheits- und Fehlerbehandlung

#### Exception-Handling:

- IOException: Netzwerk- und API-Kommunikationsfehler
- GeneralSecurityException: Sicherheits- und Zertifikatsfehler
- FileNotFoundException: Fehlende Credentials-Datei

#### Sicherheitsmerkmale:

- Read-Only-Berechtigung (CALENDAR\_READONLY)
- Sichere HTTPS-Verbindung über GoogleNetHttpTransport
- Lokale Token-Speicherung für Offline-Zugriff
- Thread-sichere Singleton-Implementierung

### 2.3.6 Konfigurationskonstanten

```
1 private static final String APPLICATION_NAME = "G2iCal";
2 private static final JsonFactory JSON_FACTORY = GsonFactory.
    getDefaultInstance();
3 private static final String TOKENS_DIRECTORY_PATH = "tokens";
4 private static final List<String> SCOPES =
    Collections.singletonList(CalendarScopes.CALENDAR_READONLY);
5 private static final String CREDENTIALS_FILE_PATH = "/credentials
    .json";
```

Diese Konstanten definieren die Anwendungsidentität, JSON-Verarbeitung, Token-Speicherort, API-Berechtigungen und Credentials-Pfad.

## 2.4 MyEvent-Struktur: Event-Modellierung und -Erstellung

Die MyEvent-Struktur ist die zentrale Kalenderereignis-Modellierung und implementiert ein vollständiges Design Pattern-System für Event-Erstellung und -Verwaltung.

### 2.4.1 CalendarEvent Interface - Abstraktion

Das CalendarEvent Interface definiert eine Vorlage für Eventklassen:

```
1 public interface CalendarEvent {
2     // Basic event identification
3     String getUid();
4     String getSummary();
5     String getDescription();
6     String getLocation();
7
8     // Date and time fields
9     LocalDateTime getStart();
10    LocalDateTime getEnd();
11    LocalDateTime getCreated();
12    LocalDateTime getLastModified();
```

```

13
14 // People and organization
15 String getOrganizer();
16 List<String> getAttendees();
17
18 // Event properties
19 String getStatus(); // CONFIRMED, TENTATIVE, CANCELLED
20 String getTransparency(); // OPAQUE, TRANSPARENT
21 String getClassification(); // PUBLIC, PRIVATE, CONFIDENTIAL
22 Integer getPriority(); // 0-9
23
24 // Recurrence and timing
25 String getRecurrenceRule(); // RRULE
26 List<LocalDateTime> getRecurrenceDates(); // RDATE
27 List<LocalDateTime> getExceptionDates(); // EXDATE
28
29 // Additional fields
30 String getUrl();
31 List<String> getCategories();
32 String getComment();
33 String getContact();
34 Integer getAlarmMinutesBefore();
35
36 String toICal(); // iCal format conversion
37 }

```

### Interface-Vorteile:

- Abstraktion von konkreter Implementierung
- 23 Getter-Methoden für alle iCal-Eigenschaften
- Polymorphismus für verschiedene Event-Typen
- Einheitliche `toICal()`-Methode

### 2.4.2 MyEvent Implementation - Konkrete Klasse

Die MyEvent-Klasse implementiert das Interface mit iCal-Unterstützung:

```

1 public class MyEvent implements CalendarEvent {
2     // private fields for iCal properties
3     private String uid;
4     private String summary;
5     private String description;
6     private LocalDateTime start;
7     private LocalDateTime end;
8     // ... weitere Felder
9
10    // Default constructor with initialization
11    public MyEvent() {
12        this.uid = UUID.randomUUID().toString();
13        this.attendees = new ArrayList<>();

```

```

14     this.created = LocalDateTime.now();
15     this.lastModified = LocalDateTime.now();
16     this.status = "CONFIRMED";
17     this.transparency = "OPAQUE";
18     this.classification = "PUBLIC";
19 }
20
21 // Package-private constructor for Builder Pattern
22 MyEvent(MyEventBuilder builder) {
23     this.uid = builder.uid != null ? builder.uid : UUID.
24         randomUUID().toString();
25     this.summary = builder.summary;
26     this.start = builder.start;
27     this.end = builder.end;
28     // ... Mapping aller Builder-Felder
29 }

```

### iCal-Export mit RFC-konformer Formatierung:

```

1 @Override
2 public String toICal() {
3     StringBuilder ical = new StringBuilder();
4
5     ical.append("BEGIN:VEVENT\r\n");
6     ical.append("UID:").append(uid).append("\r\n");
7
8     if (start != null) {
9         ical.append("DTSTART:").append(start.format(
10             ICAL_DATE_FORMAT)).append("\r\n");
11    }
12    if (summary != null && !summary.isEmpty()) {
13        ical.append("SUMMARY:").append(escapeText(summary)).
14            append("\r\n");
15    }
16
17    // Alarm-Unterst tzung nach RFC 5545
18    if (alarmMinutesBefore != null && alarmMinutesBefore > 0) {
19        ical.append("BEGIN:VALARM\r\n");
20        ical.append("TRIGGER:-PT").append(alarmMinutesBefore).
21            append("M\r\n");
22        ical.append("ACTION:DISPLAY\r\n");
23        ical.append("DESCRIPTION:Reminder\r\n");
24        ical.append("END:VALARM\r\n");
25    }
26
27    ical.append("END:VEVENT\r\n");
28    return ical.toString();
29 }
30
31 private String escapeText(String text) {
32     if (text == null) return "";
33 }

```

```

30     return text.replace("\\\\", "\\\\\\\\")
31         .replace(", ", "\\\", ")
32         .replace(";;", "\\\"; ")
33         .replace("\n", "\\\\n")
34         .replace("\r", "");
35 }

```

### 2.4.3 MyEventBuilder - Builder Pattern

Der MyEventBuilder implementiert das Builder Pattern für flexible Event-Erstellung:

```

1 public class MyEventBuilder {
2     // Package-private fields matching MyEvent
3     String uid;
4     String summary;
5     LocalDateTime start;
6     LocalDateTime end;
7     List<String> attendees = new ArrayList<>();
8     // ... alle weiteren Felder
9
10    // Fluent API methods
11    public MyEventBuilder setSummary(String summary) {
12        this.summary = summary;
13        return this;
14    }
15
16    public MyEventBuilder setStart(LocalDateTime start) {
17        this.start = start;
18        return this;
19    }
20
21    // Convenience methods for status
22    public MyEventBuilder setStatusConfirmed() {
23        this.status = "CONFIRMED";
24        return this;
25    }
26
27    public MyEventBuilder setStatusTentative() {
28        this.status = "TENTATIVE";
29        return this;
30    }
31
32    // Convenience methods for alarms
33    public MyEventBuilder setAlarm15MinutesBefore() {
34        this.alarmMinutesBefore = 15;
35        return this;
36    }
37
38    public MyEventBuilder setAlarm1DayBefore() {
39        this.alarmMinutesBefore = 1440;
40        return this;

```

```

41 }
42
43     // Build method creates MyEvent instance
44     public MyEvent build() {
45         return new MyEvent(this);
46     }
47 }
```

### Warum das Builder Pattern hier verwendet wird:

Das Builder Pattern löst das Problem der *Constructor Overloading Hell* bei der MyEvent-Klasse. Mit 23 optionalen iCal-Feldern wären theoretisch  $2^{23}$  verschiedene Konstruktoren nötig. Das *Bu*

- **Lesbarkeit:** Methodennamen beschreiben explizit, welche Eigenschaft gesetzt wird
- **Flexibilität:** Nur benötigte Felder müssen gesetzt werden
- **Unveränderlichkeit:** Das finale MyEvent-Objekt kann nach Erstellung unveränderlich sein
- **Validation:** Builder kann Konsistenzprüfungen vor der Objekterstellung durchführen
- **Convenience:** Vordefinierte Methoden wie `setAlarm15MinutesBefore()` reduzieren Entwicklungsaufwand

### Builder Pattern Vorteile:

- Fluent API für lesbare Event-Erstellung
- Optionale Parameter ohne Konstruktor-Überladung
- Validation vor Object-Erstellung möglich
- Convenience-Methoden für häufige Werte

#### 2.4.4 EventFactory - Factory Pattern

Die EventFactory kapselt verschiedene Event-Erstellungsstrategien:

```

1 public class EventFactory {
2     // Factory method für Builder-Zugriff
3     public static MyEventBuilder builder() {
4         return new MyEventBuilder();
5     }
6
7     // Simple Event-Erstellung
8     public static CalendarEvent createSimple(String summary,
9             LocalDateTime start, LocalDateTime end) {
10        return new MyEventBuilder()
11            .setSummary(summary)
12            .setStart(start)
13            .setEnd(end)
14            .build();
15    }
16
17    // Complex Event mit allen Parametern
```

```

18     public static CalendarEvent createFromData(String uid, String
19         summary,
20             String description, String location, LocalDateTime
21                 start,
22                 LocalDateTime end, String organizer, List<String>
23                     attendees) {
24             return new MyEventBuilder()
25                 .setUid(uid)
26                 .setSummary(summary)
27                 .setDescription(description)
28                 .setLocation(location)
29                 .setStart(start)
30                 .setEnd(end)
31                 .setOrganizer(organizer)
32                 .setAttendees(attendees)
33                 .build();
34
35
36     // Google Calendar Integration
37     public static CalendarEvent createFromGoogle(Event
38         googleEvent) {
39         return EventConverter.convert(googleEvent);
40     }
41 }
```

### Warum das Factory Pattern hier verwendet wird:

Das Factory Pattern abstrahiert die komplexe Event-Erstellung und bietet verschiedene Erstellungsstrategien für unterschiedliche Anwendungsfälle:

- **Abstraktion der Komplexität:** Versteckt die Details des Builder-Patterns vor einfachen Anwendungsfällen
- **Verschiedene Datenquellen:** Unterstützt Events aus Google Calendar, manueller Eingabe oder einfachen Parametern
- **Konsistenz:** Stellt sicher, dass alle Events nach den gleichen Regeln erstellt werden
- **Erweiterbarkeit:** Neue Event-Quellen können einfach als weitere Factory-Methoden hinzugefügt werden
- **API-Vereinfachung:** Bietet sowohl einfache als auch komplexe Event-Erstellung über eine einheitliche Schnittstelle

#### 2.4.5 EventConverter - Datenkonvertierung

Der EventConverter konvertiert zwischen Google Calendar und internem Format:

```

1 public class EventConverter {
2     public static CalendarEvent convert(Event googleEvent) {
3         if (googleEvent == null) return null;
4
5         MyEventBuilder builder = new MyEventBuilder();
6 }
```

```

7   // Basic mapping
8   builder.setUid(googleEvent.getId())
9     .setSummary(googleEvent.getSummary())
10    .setDescription(googleEvent.getDescription())
11    .setLocation(googleEvent.getLocation());
12
13   // DateTime conversion with timezone handling
14   if (googleEvent.getStart() != null) {
15     LocalDateTime startTime = convertEventDateTime(
16       googleEvent.getStart());
17     builder.setStart(startTime);
18   }
19
20   // Organizer mit iCal-Format
21   if (googleEvent.getOrganizer() != null) {
22     String organizerEmail = googleEvent.getOrganizer().
23       getEmail();
24     String organizerName = googleEvent.getOrganizer().
25       getDisplayName();
26     if (organizerEmail != null) {
27       String organizer = organizerName != null ?
28         "CN=" + organizerName + ":MAILTO:" +
29         organizerEmail :
30         "MAILTO:" + organizerEmail;
31     }
32     builder.setOrganizer(organizer);
33   }
34
35   // Attendees mit Response Status nach iCal-Standard
36   if (googleEvent.getAttendees() != null) {
37     List<String> attendees = new ArrayList<>();
38     for (EventAttendee attendee : googleEvent.
39       getAttendees()) {
40       if (attendee.getEmail() != null) {
41         String attendeeStr = attendee.getDisplayName()
42           () != null ?
43             "CN=" + attendee.getDisplayName() + ":" +
44             "MAILTO:" + attendee.getEmail() :
45             "MAILTO:" + attendee.getEmail();
46
47         if (attendee.getResponseStatus() != null) {
48           attendeeStr += ";PARTSTAT=" +
49             convertResponseStatus(attendee.
50               getResponseStatus());
51         }
52         attendees.add(attendeeStr);
53       }
54     }
55     builder.setAttendees(attendees);
56   }
57
58 }
```

```

49     // Status conversion mit Standard-Fallback
50     if (googleEvent.getStatus() != null) {
51         builder.setStatus(convertStatus(googleEvent.getStatus
52             ()));
53     }
54
55     return builder.build();
56 }
57
58 // Helper methods f r Type-Conversion
59 private static LocalDateTime convertEventDateTime(
60     EventDateTime eventDateTime) {
61     DateTime dateTime = eventDateTime.getDateTime();
62     if (dateTime == null) dateTime = eventDateTime.getDate();
63
64     return LocalDateTime.ofInstant(
65         java.time.Instant.ofEpochMilli(dateTime.getValue()),
66         ZoneId.systemDefault()
67     );
68 }
69
70 private static String convertStatus(String googleStatus) {
71     return switch (googleStatus.toLowerCase()) {
72         case "confirmed" -> "CONFIRMED";
73         case "tentative" -> "TENTATIVE";
74         case "cancelled" -> "CANCELLED";
75         default -> "CONFIRMED";
76     };
77 }
78
79 private static String convertResponseStatus(String
80     responseStatus) {
81     return switch (responseStatus.toLowerCase()) {
82         case "accepted" -> "ACCEPTED";
83         case "declined" -> "DECLINED";
84         case "tentative" -> "TENTATIVE";
85         case "needsaction" -> "NEEDS-ACTION";
86         default -> "NEEDS-ACTION";
87     };
88 }
89
90 }

```

### Warum der Converter hier verwendet wird:

Der EventConverter implementiert das *Adapter Pattern* und löst das Problem der Inkompatibilität zwischen Google Calendar API und iCal-Standard:

- **API-Unabhängigkeit:** Trennt die interne Event-Repräsentation von externen APIs
- **Datenformat-Mapping:** Konvertiert zwischen unterschiedlichen Datenstrukturen und Formaten
- **Timezone-Handling:** Behandelt komplexe Zeitzonenkonvertierungen einheitlich

- **Standard-Compliance:** Stellt sicher, dass alle konvertierten Events iCal-konform sind
- **Robustheit:** Bietet Fallback-Werte für fehlende oder ungültige Daten
- **Erweiterbarkeit:** Neue APIs können durch weitere Converter-Methoden unterstützt werden

#### 2.4.6 Design Pattern Zusammenspiel

Verwendungsbeispiel der kombinierten Patterns:

```

1 // 1. Simple Event creation via Factory
2 CalendarEvent simpleEvent = EventFactory.createSimple(
3     "Meeting",
4     LocalDateTime.now(),
5     LocalDateTime.now().plusHours(1)
6 );
7
8 // 2. Complex Event mit Builder Pattern via Factory
9 CalendarEvent complexEvent = EventFactory.builder()
10    .setSummary("Quarterly Review")
11    .setDescription("Team quarterly review meeting")
12    .setLocation("Conference Room A")
13    .setStart(LocalDateTime.of(2024, 3, 15, 14, 0))
14    .setEnd(LocalDateTime.of(2024, 3, 15, 16, 0))
15    .addAttendee("MAILTO:colleague@company.com")
16    .setStatusConfirmed()
17    .setAlarm15MinutesBefore()
18    .setPriorityHigh()
19    .build();
20
21 // 3. Google Calendar conversion via Converter
22 Event googleEvent = /* from API */;
23 CalendarEvent convertedEvent = EventFactory.createFromGoogle(
24     googleEvent);
25
26 // 4. iCal export via Interface
27 String icalContent = complexEvent.toICal();

```

Pattern-Synergien im Zusammenspiel:

- **Interface + Implementation:** Garantiert einheitliche API trotz verschiedener Erstellungswege
- **Builder + Factory:** Factory abstrahiert Builder-Komplexität für einfache Fälle
- **Converter + Builder:** Converter nutzt Builder für saubere Objekterstellung
- **Alle Patterns + iCal:** Gemeinsames Ziel der RFC-konformen iCal-Ausgabe
- **Testbarkeit:** Interface ermöglicht Mock-Objects, Builder ermöglicht präzise Testobjekte

Architektureller Nutzen:

- **Saubere Trennung:** Jedes Pattern hat eine spezifische Verantwortlichkeit
- **Wartbarkeit:** Änderungen in einem Bereich beeinflussen andere minimal
- **Erweiterbarkeit:** Neue Event-Quellen oder -Formate sind einfach integrierbar
- **Code-Qualität:** Patterns fördern konsistente und lesbare Implementierung

## 2.5 ICal - Zentrale Container-Klasse für Kalenderdaten

Die ICal-Klasse ist die zentrale Container-Klasse, welche die Referenzen auf die Events für einen Export hält. Im Gegensatz zu Utility-Klassen wie `CalendarApiConnector` oder `InputValidator` repräsentiert sie einen konkreten Geschäftsobjekt-Typ mit eigenem Zustand und Lebenszyklus.

### 2.5.1 Abgrenzung zu Utility-Klassen

**Warum ICal keine Utility-Klasse ist:**

- **Zustandsbehaftet:** Die Klasse verwaltet eine interne Liste von Events als Instanzvariable
- **Geschäftsobjekt:** Repräsentiert eine komplette iCal-Datei als fachliches Konzept
- **Lebenszyklus:** Objekte werden erstellt, befüllt, modifiziert und exportiert
- **Instanzmethoden:** Operationen arbeiten auf dem spezifischen Zustand der Instanz
- **Multiple Instanzen:** Verschiedene ICal-Objekte können parallel verschiedene Kalender repräsentieren

**Unterschied zu Utility-Klassen:**

- **CalendarApiConnector:** Singleton, verwaltet API-Verbindung (Service-Klasse)
- **InputValidator:** Statische Methoden, zustandslos (reine Utility-Klasse)
- **ICal:** Zustandsbehaftete Container-Klasse für Geschäftsdaten

### 2.5.2 Klassenarchitektur

Die ICal implementiert eine *Container Klasse* zur Verwaltung von myEvent Objekten:

```

1 public class ICal {
2     // Core composition: ICal contains multiple CalendarEvents
3     private List<CalendarEvent> events;
4
5     // Default constructor for empty calendar
6     public ICal() {
7         this.events = new ArrayList<>();
8     }
9
10    // Constructor with Google Calendar Events integration
11    public ICal(Events events) {
12        this.events = new ArrayList<>();
13        if (events != null && events.getItems() != null) {

```

```

14         for (Event event : events.getItems()) {
15             addGoogleEvent(event); // Convert and add each
16             Google Event
17         }
18     }
19 }
```

### 2.5.3 Event-Management und Integration

Event-Hinzufügung mit Factory-Integration:

```

1 public void addGoogleEvent(Event event) {
2     CalendarEvent calendarEvent = EventFactory.createFromGoogle(
3         event);
4     this.events.add(calendarEvent);
5 }
```

Diese Methode demonstriert die Integration des **Factory Pattern**: Nutzung von EventFactory für Event-Erstellung.

### 2.5.4 iCal-Formatierung und RFC-Compliance

Generierung des iCal-Standards nach RFC 5545:

```

1 public String getICalString() {
2     StringBuilder icalStringBuilder = new StringBuilder();
3
4     // iCal header according to RFC 5545
5     icalStringBuilder.append("BEGIN:VCALENDAR\r\n");
6     icalStringBuilder.append("VERSION:2.0\r\n");
7     icalStringBuilder.append("PRODID:-//G2iCal - iCal Exporter//"
8         "EN\r\n");
9
10    // Add all events
11    for (CalendarEvent event : events) {
12        icalStringBuilder.append(event.toICal());
13    }
14
15    // iCal footer
16    icalStringBuilder.append("END:VCALENDAR\r\n");
17
18    return foldLines(icalStringBuilder.toString());
19 }
```

RFC 5545 Compliance-Features:

- **VCALENDAR-Block**: Korrekte Kapselung aller Events
- **VERSION 2.0**: Standard iCal-Version für maximale Kompatibilität
- **PRODID**: Eindeutige Produktidentifikation für die G2iCal-Anwendung

- **CRLF-Terminierung:** Korrekte Zeilenendigungen (  
r  
n)
- **Polymorphe Event-Integration:** Verschiedene Event-Typen über Interface

### 2.5.5 RFC-konforme Zeilenumbruch-Behandlung

Die `foldLines()`-Methode implementiert die iCal-Spezifikation für Zeilenlängen:

```

1 private String foldLines(String icalString) {
2     StringBuilder result = new StringBuilder();
3     String[] lines = icalString.split("\r\n");
4
5     for (String line : lines) {
6         if (line.length() <= 75) {
7             result.append(line).append("\r\n");
8         } else {
9             // Fold long lines according to RFC 5545
10            result.append(line.substring(0, 75)).append("\r\n");
11            String remaining = line.substring(75);
12
13            while (remaining.length() > 74) { // 74 + 1 space =
14                75 chars
15                result.append(" ").append(remaining.substring(0,
16                    74)).append("\r\n");
17                remaining = remaining.substring(74);
18            }
19
20            if (!remaining.isEmpty()) {
21                result.append(" ").append(remaining).append("\r\n");
22            }
23        }
24    }
25
26    return result.toString();
27}

```

#### RFC 5545 Line Folding Regeln:

- **75-Zeichen-Limit:** Maximale Zeilenlänge für Kalender-Clients
- **Space-Continuation:** Folgezeilen beginnen mit Leerzeichen
- **74-Zeichen-Folgezeilen:** Berücksichtigung des führenden Leerzeichens
- **Preservation:** Beibehaltung des ursprünglichen Inhalts

### 2.5.6 Datei-Export mit Exception-Handling

Robuster Datei-Export mit strukturierter Fehlerbehandlung:

```

1 public boolean exportICalToFile(String filePath, String fileName)

```

```

2     throws ICalExportException {
3     try {
4         Files.write(
5             Paths.get(filePath, fileName),
6             getICalString().getBytes(),
7             java.nio.file.StandardOpenOption.WRITE,
8             java.nio.file.StandardOpenOption.CREATE,
9             java.nio.file.StandardOpenOption.TRUNCATE_EXISTING
10        );
11        return true;
12    } catch (IOException e) {
13        throw new ICalExportException(
14            "Failed to export iCal to file: " + filePath + "/" +
15            fileName, e);
16    }
}

```

### Export-Konfiguration und Fehlerbehandlung:

- **Pfad-Kombination:** Sichere Verbindung von Verzeichnis und Dateiname
- **Atomare Schreiboperation:** `Files.write()` für konsistente Dateierstellung
- **Standard-Optionen:** CREATE, WRITE, TRUNCATE\_EXISTING für zuverlässige Überschreibung
- **Exception-Wrapping:** Konvertierung von `IOException` zu spezifischer `ICalExportException`
- **Detaillierte Fehlermeldungen:** Inklusive Pfadinformationen für Debugging

### 2.5.7 Integration in die Gesamtarchitektur

#### Rolle der ICal-Klasse im Systemkontext:

```

1 // Typical usage pattern in G2iCal application
2 public static void main(String[] args) {
3     // 1. Get events from Google Calendar via API Connector
4     CalendarApiConnector connector = CalendarApiConnector.
5         getInstance();
6     Events googleEvents = connector.getCalendarEventsByCalendarId
7         (calendarId, start, end);
8
9     // 2. Create ICal container with events
10    ICal calendar = new ICal(googleEvents);
11
12    // 3. Export to file system
13    boolean success = calendar.exportICalToFile(filePath,
14        fileName);
15
16}

```

#### Architekturelle Verantwortlichkeiten:

- **Datenmodell:** Zentrale Repräsentation von Kalenderdaten

- **Format-Transformation:** Konvertierung zwischen API-Format und iCal-Standard
- **Persistierung:** Datei-Export für externe Kalender-Anwendungen
- **Aggregation:** Sammlung und Verwaltung multipler Events

## 2.6 InputValidator - Utility-Klasse für Eingabeverifikation

Die `InputValidator`-Klasse ist eine reine Utility-Klasse mit statischen Methoden für die Validierung von Benutzereingaben. Als Utility-Klasse enthält sie keine Instanzvariablen und alle Methoden sind statisch.

### 2.6.1 Beispiel: Datumsbereich-Validierung

Die `validateDateRange`-Methode zeigt die typische Arbeitsweise der Validierungslogik:

```

1 public static ValidationResult validateDateRange(DateTime
2     startDateDT, String endDateStr) {
3     // Validate end date first using existing validation method
4     ValidationResult endValidation = validateDate(endDateStr, "
5         End date");
6     if (!endValidation.isValid()) {
7         return endValidation;
8     }
9
10    try {
11        // Convert Google DateTime to LocalDate for comparison
12        String startDateStr = startDateDT.toStringRfc3339().
13            substring(0, 10);
14        LocalDate startDate = LocalDate.parse(startDateStr,
15            DATE_FORMATTER);
16        LocalDate endDate = LocalDate.parse(endDateStr.trim(),
17            DATE_FORMATTER);
18
19        // Check logical date order
20        if (startDate.isAfter(endDate)) {
21            return new ValidationResult(false,
22                "Start date cannot be after end date. Start: " +
23                startDate.format(DATE_FORMATTER) + ", End: " +
24                endDate.format(DATE_FORMATTER));
25        }
26
27        // Check for reasonable date range (max 2 years)
28        if (startDate.plusYears(2).isBefore(endDate)) {
29            return new ValidationResult(false,
                "Date range is too large (maximum 2 years).
                    Please select a shorter period.");
        }
    }
}

```

```

30         return new ValidationResult(false, "Error parsing date
31             range");
32     }

```

### Validierungslogik:

1. **End-Date-Validierung:** Wiederverwendung der `validateDate()`-Methode
2. **Format-Konvertierung:** Google DateTime zu LocalDate für Vergleiche
3. **Logische Prüfung:** Start-Datum darf nicht nach End-Datum liegen
4. **Bereichsprüfung:** Maximaler Zeitraum von 2 Jahren für Performance
5. **Exception-Handling:** Sichere Behandlung von Parse-Fehlern

### 2.6.2 ValidationResult - Ergebnis-Container

Die innere `ValidationResult`-Klasse kapselt Validierungsergebnisse:

```

1 public static class ValidationResult {
2     private final boolean valid;
3     private final String errorMessage;
4
5     public ValidationResult(boolean valid, String errorMessage) {
6         this.valid = valid;
7         this.errorMessage = errorMessage;
8     }
9
10    public boolean isValid() { return valid; }
11    public String getErrorMessage() { return errorMessage; }
12 }

```

Diese Klasse bietet strukturierte Rückgabewerte mit Validierungsstatus und detaillierter Fehlermeldung für eine benutzerfreundliche Fehlerbehandlung.

## 2.7 G2iCal - Hauptanwendungsklasse

Die `G2iCal`-Klasse ist die zentrale Steuerungseinheit der Anwendung und orchestriert den gesamten Workflow vom Benutzereingabe bis zum iCal-Export. Sie implementiert sowohl einen Command-Line-Interface als auch einen interaktiven Modus.

### 2.7.1 Anwendungsworkflow

Die `main`-Methode koordiniert den folgenden Ablauf:

```

1 public static void main(String... args) {
2     // 1. Welcome and help handling
3     displayWelcomeMessage();
4     displayInfoIfPorted(args);
5
6     // 2. API initialization

```

```

7   CalendarApiConnector apiConnector = CalendarApiConnector.
8       getInstance();
9
10  // 3. User input collection
11  DateTime startTime = getUserInputStartDate(args);
12  DateTime endTime = getUserInputEndDate(startTime, args);
13  String fileName = getUserInputFileName(args);
14  String selectedCalendar = getUserInputCalendar(args);
15
16  // 4. Event fetching and processing
17  Events events = apiConnector.getCalendarEventsByCalendarId(
18      selectedCalendar, startTime, endTime);
19  ICal ical = new ICal(events);
20
21  // 5. Export to file
22  ical.exportICalToFile(EXPORT_FILE_PATH, fileName);
}

```

### Workflow-Phasen:

1. **Initialisierung:** Begrüßung und Help-Handling
2. **API-Setup:** Singleton-basierte Connector-Initialisierung
3. **Input-Sammlung:** Validierte Benutzereingaben (CLI oder interaktiv)
4. **Datenverarbeitung:** Event-Abruf und ICal-Container-Erstellung
5. **Export:** Dateierstellung im Downloads-Verzeichnis

#### 2.7.2 Eingabe-Management mit Fallback-Mechanismus

Die Anwendung unterstützt sowohl Command-Line-Parameter als auch interaktive Eingaben:

```

1 private static DateTime getUserInputStartDate(String... args) {
2     Scanner scanner = new Scanner(System.in);
3
4     // Try command line argument first
5     if (args.length > 0) {
6         String startDate = args[0].trim();
7         ValidationResult result = InputValidator.validateDate(
8             startDate, "start");
9         if (result.isValid()) {
10             return InputValidator.convertToDateTIme(startDate,
11                 false);
12         } else {
13             System.out.println("Error: " + result.getErrorMessage
14                 ());
15         }
16     }
17
18     // Fallback to interactive input
19     return InputValidator.convertToDateTIme(

```

```

17     promptForValidInput(scanner, "Enter start date (YYYY-MM-
18         DD): ",
19         dateStr -> InputValidator.validateDate(dateStr, "Start date")), false);
}

```

### Input-Strategie:

- **CLI-First:** Versucht zuerst Command-Line-Argumente zu verwenden
- **Validation:** Alle Eingaben werden durch InputValidator geprüft
- **Interactive-Fallback:** Bei fehlenden/ungültigen CLI-Args erfolgt interaktive Eingabe
- **Retry-Loop:** promptForValidInput wiederholt bis gültige Eingabe erfolgt

### 2.7.3 Kalender-Auswahl mit dynamischer Liste

Die Kalenderauswahl zeigt verfügbare Kalender und ermöglicht Index-basierte Selektion:

```

1 private static void displayCalendarList(List<CalendarListEntry>
2     calendars) {
3     System.out.println("\nAvailable Calendars:");
4     for (int i = 0; i < calendars.size(); i++) {
5         CalendarListEntry calendar = calendars.get(i);
6         System.out.printf(" [%d] %s\n", i, calendar.getSummary()
7             );
8         System.out.printf(" ID: %s\n", calendar.getId());
9     }
8 }

```

### 2.7.4 Help-System und Dokumentation

Das integrierte Help-System bietet nötige Nutzungsinformationen:

```

1 private static void displayInfoIfPorted(String... args) {
2     if (args.length > 0 && (args[0].equals("--info") || args[0].
3         equals("-i"))
4         || args[0].equals("--help") || args[0].equals("-h"))) {
5
6         System.out.println("USAGE:");
7         System.out.println("  java G2iCal [start-date] [end-date]
8             [filename] [calendar-index]");
9         System.out.println("EXAMPLES:");
10        System.out.println("  java G2iCal 2025-07-29 2025-08-05
11            my_events 0");
12        exit(0);
11    }
11 }

```

## 2.7.5 Fehlerbehandlung und Benutzerführung

Die Anwendung implementiert robuste Fehlerbehandlung:

```
1 try {
2     ical.exportICalToFile(EXPORT_FILE_PATH, fileName);
3     System.out.println("\nSuccess! iCal file generated in " +
4         EXPORT_FILE_PATH);
5     exit(0);
6 } catch (ICalExportException e) {
7     System.err.println("Export Error: " + e.getMessage());
8     exit(1);
}
```

Error-Handling-Prinzipien:

- **Strukturierte Exceptions:** Spezifische Exception-Typen für verschiedene Fehlerarten
- **Benutzerfreundliche Meldungen:** Klare Fehlerbeschreibungen ohne technische Details
- **Proper Exit Codes:** 0 für Erfolg, 1 für Fehler (Unix-Konvention)
- **Retry-Mechanismus:** Eingabefehler führen zu erneuter Eingabeaufforderung

## 2.7.6 Konfiguration und Pfad-Management

```
1 static String EXPORT_FILE_PATH = System.getProperty("user.home")
+ "/Downloads";
```

Die Anwendung verwendet das Downloads-Verzeichnis des Benutzers als Standard-Export-Pfad, was plattformunabhängig und benutzerfreundlich ist.

## 2.8 Ablaufdarstellung als UML Sequenz-Diagramm

Das Sequenz-Diagramm in Abbildung 2 zeigt dein klassischen Ablauf der G2iCal-Anwendung. Hier wird das Programm mit einfachen CLI Argumenten ausgeführt. Dabei wird zunächst eine Verbindung zu Google API aufgebaut und die Credentials werden gespeichert. Danach werden die User Inputs aus den CLI Arguments geholt. Darauf erfolgt der API Zugriff mit der erforderlichen Timerange. Daraus werden dann Events und das iCal Objekt erstellt. Vor allem das Zusammenspiel der Event Factory, Converter und Builder kann hier gut erkannt werden. Zuletzt wird die iCal Datei erstellt.

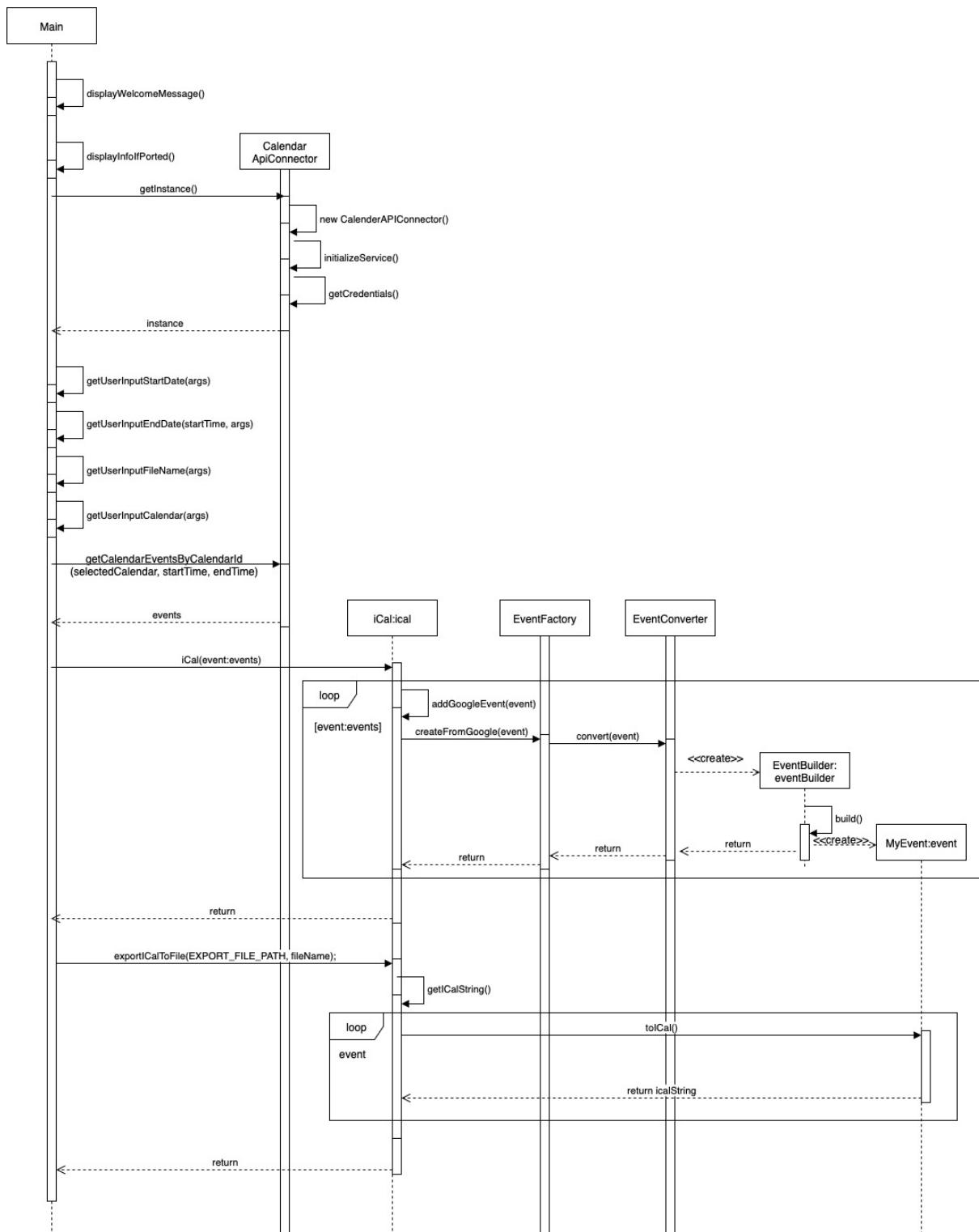


Abbildung 2: Sequenz Diagramm

## 3 Fragen

### 3.1 Welche Funktionen bietet das Tool Gradle? Wofür kann es eingesetzt werden?

#### 3.1.1 Was ist Gradle?

Gradle ist ein Build-Automation-Tool. Es automatisiert den gesamten Build-Prozess von der Kompilierung des Quellcodes, Einbindung von Bibliotheken bis zur Erstellung ausführbarer Anwendungen.

#### 3.1.2 Funktionen von Gradle

Gradle bietet umfassende Dependency-Management-Funktionen, die in diesem Projekt besonders wichtig sind. Das Tool verwaltet automatisch alle externen Bibliotheken wie die Google Calendar API, OAuth-Authentifizierung und weitere Dependencies. So werden die Abhängigkeiten in der build.gradle-Datei festgelegt und dann installiert Gradle sie automatisch aus Maven-Repositories herunter.

Die Build-Automatisierung ist eine weitere Funktion von Gradle. Es kompiliert den Java-Quellcode aus dem src/main/java-Verzeichnis und erstellt ausführbare JAR-Dateien.

#### 3.1.3 Einsatzgebiete in G2iCal

In diesem Projekt ermöglicht Gradle die Integration der Google Calendar API. So werden auch Abhängigkeiten zwischen verschiedenen Google-Bibliotheken automatisch aufgelöst. Gradle stellt sicher, dass alle notwendigen Java-Bibliotheken zur Laufzeit verfügbar sind.

In größeren Projekten die in Teams umgesetzt werden, vereinfacht Gradle auch die Entwicklung im Team indem jeder Entwickler das Projekt mit einem einfachen Gradle build-Befehl kompilieren, ohne sich um die manuelle Konfiguration von Dependencies kümmern zu müssen.

## **3.2 Was bedeuten die Schritte „API aktivieren“ und „OAuth-Zustimmungsbildschirm konfigurieren“?**

### **3.2.1 API aktivieren**

Bevor eine Anwendung auf eine Google API zugreifen kann, muss die jeweilige Programmierschnittstelle explizit im zugehörigen Google-Cloud-Projekt aktiviert werden. Durch diesen Schritt wird die API für das Projekt freigeschaltet, sodass Anfragen entgegengenommen und verarbeitet werden können. Darüber hinaus ermöglicht die Aktivierung eine zentrale Überwachung, Abrechnung sowie Zugriffskontrolle durch Google. Im Kontext dieses Projekts muss die Google Calendar API aktiviert werden, um auf Kalenderdaten zugreifen und diese programmatisch verarbeiten zu können.

### **3.2.2 OAuth-Zustimmungsbildschirm konfigurieren**

Wenn eine Anwendung auf personenbezogene Daten eines Google-Nutzers zugreifen möchte – beispielsweise Kalendereinträge – ist eine vorherige Autorisierung durch den Nutzer erforderlich. Hierzu wird in der Google Cloud Console ein sogenannter OAuth-Zustimmungsbildschirm konfiguriert. Dieser zeigt dem Nutzer transparent an: welche Anwendung den Zugriff anfordert, welche Daten (Scopes) angefragt werden und wer der verantwortliche Entwickler bzw. Anbieter der App ist. Die Konfiguration umfasst unter anderem den Namen der Anwendung, eine Support-E-Mail-Adresse, sowie ggf. eine Liste von Testnutzern (sofern die Anwendung sich noch im Testmodus befindet). Dieser Schritt ist notwendig, um den Sicherheits- und Datenschutzanforderungen von Google gerecht zu werden und eine kontrollierte Autorisierung zu ermöglichen.

### 3.3 Wie kann man in einer IDE die Google Calendar API verfügbar machen?

Um die Google Calendar API in einer IDE verfügbar zu machen, gibt es verschiedene Ansätze. Hier gilt das Gradle: In der build.gradle-Datei sind die notwendigen Dependencies definiert, die automatisch heruntergeladen und in den Classpath eingebunden werden. Gradle lädt dabei nicht nur die Haupt-API-Bibliothek herunter, sondern auch alle transitiven Abhängigkeiten wie HTTP-Transport-Layer, JSON-Parser und OAuth2-Bibliotheken.

Wenn das Projekt in IntelliJ IDEA geöffnet wird, erkennt die IDE automatisch die Gradle-Konfiguration und importiert alle Dependencies. Dann werden Imports wie `import com.google.api.services.calendar.Calendar` und `import com.google.api.client.googleapis.javanet.GoogleNetHttpTransport` korrekt aufgelöst.

Ähnlich wie Gradle ist Maven, da werden die entsprechenden Dependencies in der pom.xml definieren. Der Ansatz beruht auf dem gleichen Konzept, nur die Syntax ist unterschiedlich.

Alternativ können die Google Calendar API-JARs manuell heruntergeladen werden und in den Projekt-Classpath eingebunden werden. Dabei müssen jedoch alle Abhängigkeiten manuell verwaltet werden, was fehleranfällig ist. In IntelliJ IDEA würden das über "Project Structure -> Modules -> Dependencies" gehen.

#### Credentials-Setup:

Zusätzlich wird die credentials.json-Datei im src/main/resources-Verzeichnis für die API benötigt. Diese Datei erhält man über die Google Cloud Console, wo das Projekt erstellt und die Calendar API aktiviert ist. Ohne diese Credentials-Datei funktioniert die Authentifizierung in `getCredentials()` nicht, auch wenn die API-Bibliotheken verfügbar sind.

## 3.4 Erkläre die Code-Zeilen mit GoogleNetHttpTransport und Calendar.Builder

Die beiden Zeilen sind zwei zentrale Komponenten der Google Calendar API Connection im Projekt:

### 3.4.1 GoogleNetHttpTransport

```
1 httpTransport = GoogleNetHttpTransport.newTrustedTransport();
```

*GoogleNetHttpTransport.newTrustedTransport()* erstellt einen HTTP-Transport-Client, der für die Kommunikation mit den Google APIs verwendet wird. Diese Methode initialisiert einen vertrauenswürdigen HTTP-Transport mit vordefinierten SSL-Zertifikaten und Sicherheitseinstellungen. Der Transport fungiert als niedrigste Ebene der Netzwerkkommunikation und ermöglicht API-Anfragen über verschlüsselte HTTPS-Verbindungen. Gradle lädt automatisch die notwendigen Abhängigkeiten wie google-http-client herunter, die für diese Funktionalität erforderlich sind.

### 3.4.2 Calendar.Builder

```
1 calendarService = new Calendar.Builder(httpTransport,
 2   JSON_FACTORY, getCredentials())
 3   .setApplicationName(APPLICATION_NAME)
 4   .build();
```

Der *Calendar.Builder* konstruiert den Google Calendar Service-Client. Er benötigt drei wesentliche Komponenten: den zuvor erstellten HTTP-Transport für die Netzwerkkommunikation, die JSON-Factory (GsonFactory) für die Serialisierung und Deserialisierung von API-Responses, und die OAuth2-Credentials für die Authentifizierung.

Die *setApplicationName()*-Methode identifiziert Ihre Anwendung gegenüber den Google-Servern, was für Monitoring und Rate-Limiting wichtig ist. Der Builder konfiguriert intern alle notwendigen Komponenten wie Request-Interceptors, Error-Handler und Retry-Mechanismen.

Das resultierende Calendar-Objekt ist Ihr Hauptzugang zur Google Calendar API und ermöglicht Operationen wie *calendarService.events().list()* oder *calendarService.calendarList().list()*, die später in den Methoden *getCalendarEventsByCalendarId()* und *getUserCalendars()* verwendet werden.

## Teil II

# Aufgabe 2 - GUI-Erweiterung mit Swing

## 4 Programmbeschreibung

Das Programm basiert auf Aufgabenteil 1 und erweitert diesen um ein grafisches Benutzerinterface (GUI).

Es bietet eine selbsterklärende Benutzerführung. Beim ersten Programmstart oder beim Starten ohne authentifizierten Google-Account erscheint zunächst ein Willkommensfenster (siehe Abbildung 4e). Dieses fordert den Benutzer auf, sich mit seinem Google-Account zu authentifizieren. Beim Klicken auf den Login-Button öffnet sich ein Login-Dialog (siehe Abbildung 4a), in dem die Authentifizierung mit einem Google-Account via OAuth erläutert wird. Zudem wird der OAuth-Flow gestartet. Da das Programm bei bestimmten Systemeinstellungen den Browser nicht automatisch öffnen kann, wird zusätzlich ein Link für das manuelle Öffnen angeboten. Nach erfolgreicher Anmeldung im Browser wird das Hauptfenster geöffnet (siehe Abbildung 3b).

Das Hauptfenster ist in drei Bereiche unterteilt:

1. **Load-Data-Form:** Hier kann der Benutzer Kalenderdaten aus einem seiner Kalender für einen bestimmten Zeitraum laden. Beim Wählen eines Datums wird dieses direkt auf Korrektheit geprüft, alternativ kann es über den integrierten Kalender ausgewählt werden. Beim Klick auf *Load Calendar Data* werden die Eingaben validiert. Bei einem Fehler erhält der Benutzer eine Rückmeldung (siehe Abbildung 4f). Sind alle Felder korrekt ausgefüllt, werden die Kalenderdaten über die Google API geladen.

2. **Datenliste:** Im zweiten Bereich der Hauptseite werden die geladenen Daten in einer Liste angezeigt. Hierbei werden nur die wichtigsten Informationen wie Titel, Start- und Enddatum, Ort sowie Beschreibung dargestellt. Das Programm exportiert jedoch deutlich mehr Daten, was durch die Spalte „...“ dargestellt ist.

3. **Statusanzeige und Export:** Im dritten Bereich befindet sich eine Statusanzeige sowie der *Export*-Button. Beim Klick auf diesen Button werden die geladenen Events in einer iCal-Datei im in den Einstellungen festgelegten Verzeichnis gespeichert. Der Export ist nur möglich, wenn zuvor erfolgreich Kalenderdaten geladen wurden.

Die Einstellungen können über das Menü oder den Shortcut *Command + ,* geöffnet werden (siehe Abbildung 4c). Dort lassen sich sowohl der Dateiname als auch der Speicherordner für die iCal-Datei festlegen. Standardmäßig speichert das Programm in den *Downloads*-Ordner. Beim Klick auf „Browse...“ öffnet sich ein Dialog, in dem der gewünschte Speicherort ausgewählt werden kann. Vor dem Speichern der Einstellungen werden die Eingaben validiert: Es wird überprüft, ob der Dateiname nur erlaubte Zeichen enthält und ob auf das angegebene Verzeichnis Zugriffsrechte bestehen. Außerdem können die Einstellungen auf die Standardwerte zurückgesetzt werden (siehe Abbildung 4c).

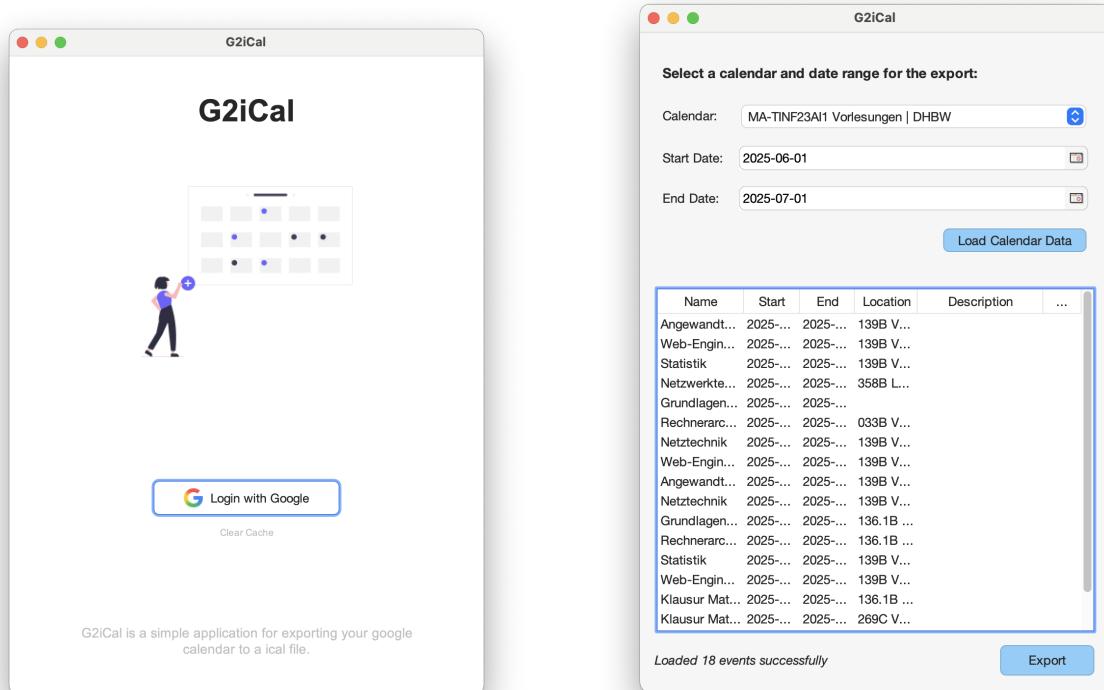
Das Menü enthält weitere Funktionen: Unter *Account* kann der aktuell angemeldete Account angezeigt und sich abgemeldet werden. Beim Abmelden öffnet sich wieder der Wel-

come Screen. Unter *File* kann über *Save* (Command + S) die Export-Funktion ausgeführt werden, und über *New* (Command + N) werden die geladenen Daten zurückgesetzt.

Die Fenster des Programms sind responsiv aufgebaut. Der Benutzer kann die Fenster beliebig verschieben und in der Größe anpassen, wobei sich die Benutzeroberfläche automatisch anpasst (siehe z. B. Abbildung 3c im Vergleich zu Abbildung 3b).

## 4.1 UI-Screenshots

Die hier gezeigten Screenshots stammen aus der macOS-Version. Auf anderen Betriebssystemen wie Windows kann die Anwendung ein anderes Erscheinungsbild haben. Auf macOS wird die Menüleiste in der globalen Menübar dargestellt, während sie unter Windows innerhalb des Programmfensters angezeigt wird. Außerdem unterscheidet sich der Menüaufbau: macOS fügt automatisch ein Menü mit dem Anwendungsnamen hinzu, in dem sich Punkte wie *About* und *Quit* befinden. Unter Windows ist *Quit* im Menü *File* zu finden, und es gibt ein zusätzliches *Help*-Menü mit dem Unterpunkt *About*.



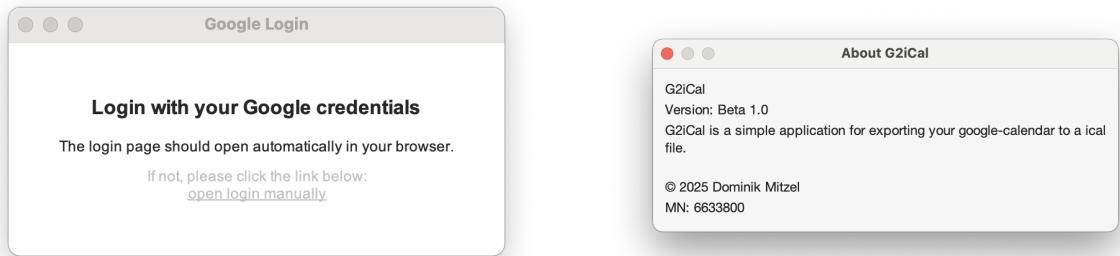
(a) Start Screen

(b) Main Screen

Name	Start	End	Location	Description	...
Angewandte Mathematik	2025-06-02 09:30	2025-06-02 12:00	139B VL-Raum Unterrichtsraum		
Web-Engineering 2	2025-06-02 13:00	2025-06-02 15:30	139B VL-Raum Unterrichtsraum		
Statistik	2025-06-03 09:30	2025-06-03 12:00	139B VL-Raum Unterrichtsraum		
Netzwerktechnik_Netzwerklabor mit Hr. Kaiser	2025-06-03 13:00	2025-06-03 16:15	358B Labor Unterrichtsraum		
Grundlagen des Software Engineering	2025-06-04 09:00	2025-06-04 12:35			
Rechnerarchitekturen 1	2025-06-04 13:15	2025-06-04 15:45	033B VL-Raum Unterrichtsraum, 13...		
Netztechnik	2025-06-05 09:00	2025-06-05 12:35	139B VL-Raum Unterrichtsraum		
Web-Engineering 2	2025-06-06 09:00	2025-06-06 12:15	139B VL-Raum Unterrichtsraum		
Angewandte Mathematik	2025-06-10 09:30	2025-06-10 12:00	139B VL-Raum Unterrichtsraum		
Netztechnik	2025-06-10 13:00	2025-06-10 16:35	139B VL-Raum Unterrichtsraum		
Grundlagen des Software Engineering	2025-06-11 09:00	2025-06-11 12:35	136.1B VL-Raum Unterrichtsraum		
Rechnerarchitekturen 1	2025-06-11 13:15	2025-06-11 15:45	136.1B VL-Raum Unterrichtsraum		
Statistik	2025-06-12 13:00	2025-06-12 15:30	139B VL-Raum Unterrichtsraum		
Web-Engineering 2	2025-06-13 09:00	2025-06-13 12:15	139B VL-Raum Unterrichtsraum		
Klausur Mathematik II (Angewandte Mathematik)	2025-06-16 10:30	2025-06-16 12:00	136.1B VL-Raum Unterrichtsraum		
Klausur Mathematik II (Statistik) 90 min	2025-06-17 11:00	2025-06-17 12:30	269C VL-Raum Unterrichtsraum		
Klausur Kommunikations- und Netztechnik (90...	2025-06-18 14:30	2025-06-18 16:00	269C VL-Raum Unterrichtsraum		
Klausur Technische Informatik II (Rechnerarchi...	2025-06-20 11:30	2025-06-20 14:00	178C VL-Raum Unterrichtsraum		

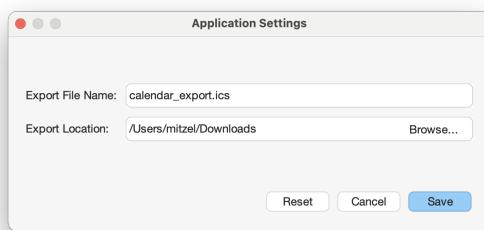
(c) Fullscreen with mac menu bar

Abbildung 3: UI Screenshots – Screens

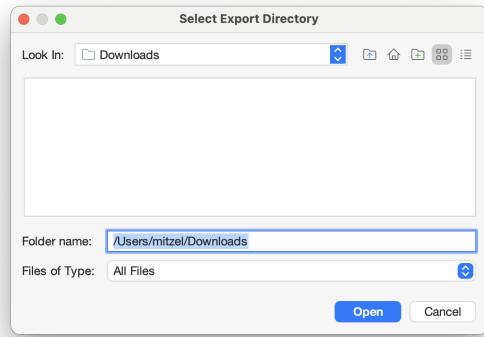


(a) Login Dialog

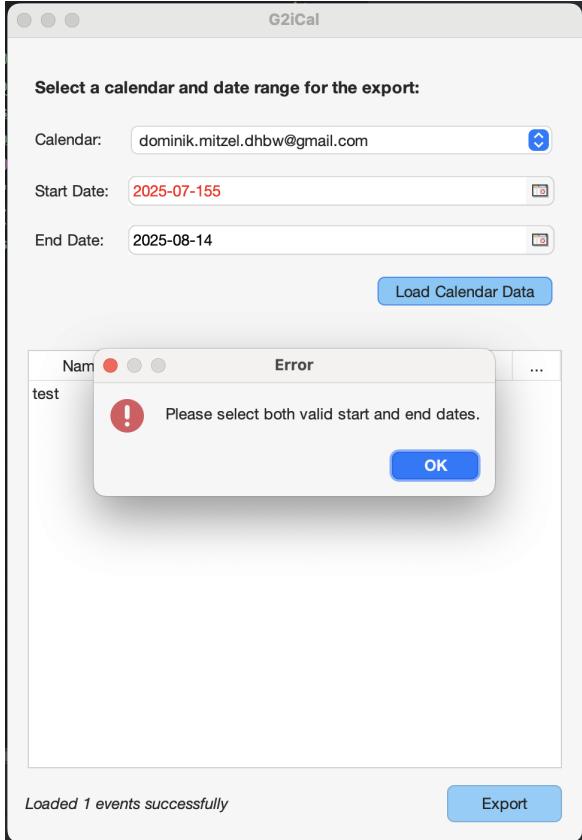
(b) About Dialog



(c) Setting Dialog



(d) Browse for storage directory



(e) Reset Settings

(f) Input validation

Abbildung 4: UI Screenshots – Dialoge und Popups 2

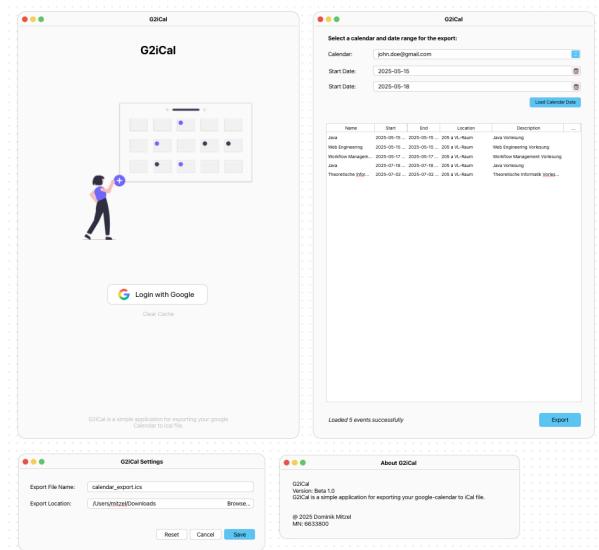


Abbildung 5: Planung der UI

## 5 Dokumentation

### 5.1 UI/UX-Design

Um eine gute UX und UI zu ermöglichen, wird die GUI im Vorfeld geplant. Im ersten Schritt wird hierzu zunächst ein Freihand-Sketch erstellt; in diesem Projekt wurde dafür Freeform verwendet (siehe Abbildung 5). Die endgültige Umsetzung lässt sich in Abbildung 3 und Abbildung 4 nachvollziehen.

### 5.2 Projektverzeichnis

```
google_calendar_iCal/
|-- .gradle/
|-- build/                                # Gradle Build-Ausgabe
|-- docs/                                   # Documentation
|-- gradle/                                 # Gradle Wrapper-Dateien

|-- src/main/
|   |-- java/
|       |-- G2iCalGUI.java                  # Main GUI Version
|       |-- G2iCal.java                    # Main CLI Version
|       |-- calendar/
|           |-- CalendarEvent.java        # Event Class Interface
|           |-- ICal.java                # ICal Class
|           |-- MyEvent.java            # MyEvent Class
|           +-+ MyEventBuilder.java     # MyEvent Builder Class
|           |-- exceptions/
|               +-+ ICalExportException.java # Exception-Handling for ICal export
|           |-- ui/
|               |-- main
|                   |-- Contorller.java    # Controller for Main View
```

```

|   |   |   |   +- View.java           # Main View
|   |   |   +- start
|   |   |       |-- Contorller.java    # Controller for Start View
|   |   |       +- View.java          # Start View
|   |   +- utils/
|   |       |-- CalendarApiConnector.java # Connection to the Google API
|   |       |-- EventConverter.java     # Convert Google event into MyEvent
|   |       |-- EventFactory.java      # Eventfactory MyEvent
|   |       |-- InputValidator.java    # Input Validator for Date, Path and Name
|   |       +- Settings.java          # Haldes the Settings storage
|   +- resources/
|       |-- UI/                      # Images for the UI
|       |-- credentials.json         # Google Cloud Credentials
|       +- settings.json             # Settings Storage
|-- tokens/                           # Google API Token-Speicher

-- .gitignore
-- build.gradle                         # Gradle Build-Konfiguration
-- gradlew                               # Gradle Wrapper (Unix)
-- gradlew.bat                            # Gradle Wrapper (Windows)
-- settings.gradle.kts                  # Gradle Einstellungen

```

### 5.2.1 Im Vergleich zu Teil 1

In Teil 2 sind hauptsächlich die folgenden Elemente hinzugekommen:

- **ui package:** Enthält zwei Packages für die beiden Haupt-Views: **ui/main/** und **ui/start**. Diese beinhalten jeweils eine View und einen Controller.
- **resources/UI/:** Beinhaltet Bilder wie das Google-Logo oder das Willkommensbild.
- **utils/Settings.java:** Enthält die Logik, um Einstellungen wie den Speicherpfad oder den Export-Dateinamen zu verwalten und in der Datei **resources/settings.json** zu speichern. Dadurch bleiben die Einstellungen auch nach einem Programmneustart erhalten.

## 5.3 Umsetzung des Model-View-Controller-Patterns

Die G2iCal-Anwendung implementiert das **Model-View-Controller-Pattern** (MVC) als Architekturmuster zur Trennung von Datenhaltung, Darstellung und Geschäftslogik.

### View-Schicht

Die **View-Klassen** sind ausschließlich für die Darstellung der Benutzeroberfläche zuständig. Sie enthalten UI-Komponenten wie **JFrame**, **JPanel**, **JButton** und **JTable**, aber keine Geschäftslogik. Die Views kommunizieren ausschließlich über ihre zugewiesenen Controller und verwenden package-private Methoden, um den

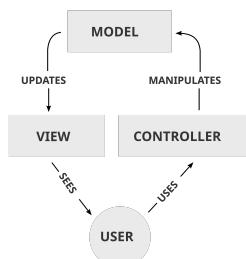


Abbildung 6: MVC Pattern  
(Bildquelle: Wikipedia)

Zugriff zu beschränken. Benutzereingaben werden über ActionListener an den Controller weitergeleitet, ohne dass die View die Verarbeitung übernimmt.

### Controller-Schicht

Die **Controller-Klassen** verwalten die Geschäftslogik und koordinieren zwischen View und Model. Sie verarbeiten Benutzereingaben, validieren Daten über **InputValidator**, kommunizieren mit externen APIs über **CalendarApiConnector** und aktualisieren die View entsprechend. Die Controller enthalten keine UI-Komponenten und arbeiten ausschließlich über definierte Schnittstellen.

### Model-Schicht

Das **Model** wird durch verschiedene Datenklassen repräsentiert. Die **ICal-Klasse** kapselt Kalenderdaten von **MyEvent**-Objekten und deren Verarbeitung, **Settings** verwaltet Anwendungseinstellungen persistent, und **CalendarApiConnector** abstrahiert die Google Calendar API-Kommunikation. Die Models sind unabhängig von der Darstellungsschicht und stellen reine Datenoperationen bereit.

### Kommunikationsfluss

Der **Datenfluss** folgt dem MVC-Paradigma strikt: Views leiten Benutzeraktionen an Controller weiter, Controller verarbeiten die Logik und aktualisieren Models, Models benachrichtigen indirekt über Controller-Methoden die Views. Die Trennung wird durch package-private Konstruktoren und Methoden verstärkt, die direkten Zugriff zwischen den Schichten verhindern.

### Vorteile der Implementierung

Diese Architektur ermöglicht **lose Kopplung** zwischen den Komponenten, erleichtert Wartung und Testing, und unterstützt die Wiederverwendbarkeit von Code. Die klare Trennung der Verantwortlichkeiten macht das System erweiterbar und robust gegenüber Änderungen in einzelnen Schichten.

## 5.4 Klassendiagramm

Änderung im Klassendiagramm umfassen folgende Klassen:

### 5.4.1 Klasse start.View

Die **start.View** Klasse repräsentiert die Startansicht der G2iCal-Anwendung und implementiert die View-Komponente des MVC-Patterns für den Anmeldeprozess.

### Architektur und Verantwortlichkeiten

Die Klasse erbt von **JFrame** und ist ausschließlich für die Darstellung der Benutzeroberfläche zuständig. Sie enthält keine Geschäftslogik und kommuniziert ausschließlich über den zugewiesenen **Controller**. Der package-private Konstruktor **View(Controller controller)** stellt sicher, dass nur Klassen innerhalb des **ui.start** Packages eine Instanz erstellen können, was die Kapselung verstärkt.

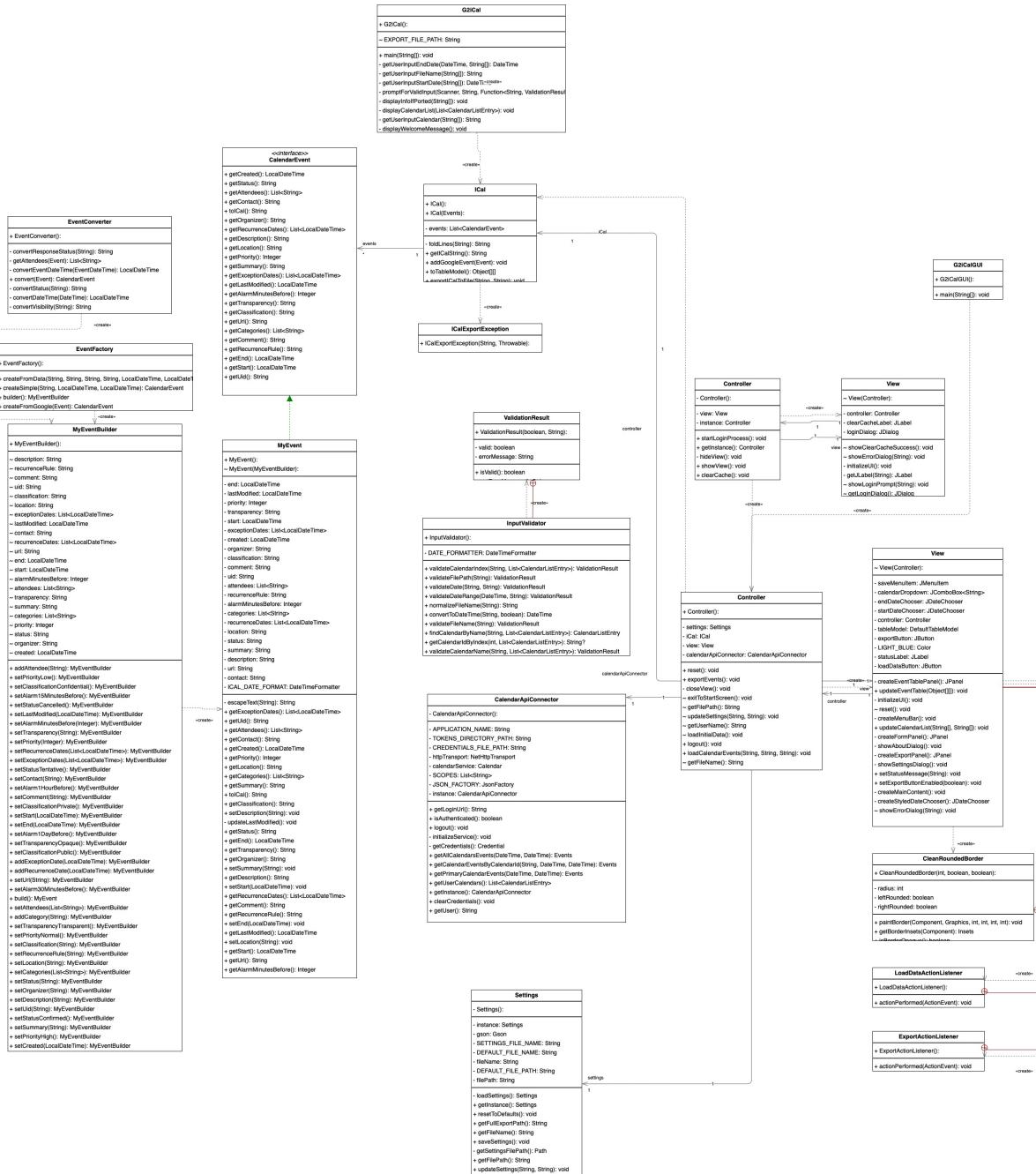


Abbildung 7: Klassendiagramm Teil 2

## UI-Komponenten und Layout

Das **BoxLayout** wird als Hauptlayout verwendet, um die Komponenten vertikal anzuordnen. Die Startansicht enthält einen Titel mit großer Schrift (`Font.BOLD, 32`), ein skaliertes Willkommensbild mit Seitenverhältnis-Erhaltung, einen Login-Button mit Google-Icon und eine Anwendungsbeschreibung. Die Bildverarbeitung erfolgt über `Image.getScaledInstance()` mit `SCALE_SMOOTH` für optimale Qualität.

## Benutzerinteraktionen

Der **Login-Button** verwendet einen ActionListener, der die Methode `controller.startLoginProcess()` aufruft. Das **Clear Cache Label** implementiert einen MouseListener für die Funktionalität `controller.clearCache()` und ändert visuelles Feedback durch Cursor- und Farbänderungen. Alle Benutzeraktionen werden direkt an den Controller delegiert, ohne lokale Verarbeitung.

## Dialog-Management

Die Methode `showLoginPrompt(String link)` öffnet ein Dialog mit Google-Login-Anweisungen. Der Dialog enthält einen klickbaren Link, der über `Desktop.getDesktop().browse()` den Standard-Browser öffnet. Bei Fehlern wird ein Fallback-Dialog mit kopierbarem Link angezeigt. Die Instanzvariable `loginDialog` ermöglicht externen Zugriff über `getLoginDialog()`.

## Feedback-Mechanismen

Die Methode `showClearCacheSuccess()` implementiert visuelles Feedback durch temporäre Farbänderung des Cache-Labels auf Grün und Deaktivierung der Klick-Funktionalität. Ein `javax.swing.Timer` stellt nach 5 Sekunden den ursprünglichen Zustand wieder her. Fehlermeldungen werden über `showErrorDialog()` in standardisierten JOptionPane-Dialogen angezeigt.

## Ressourcen-Management

Die Klasse lädt externe Ressourcen wie Bilder mit Exception-Handling und Fallback-Verhalten. Die Pfade `src/main/resources/UI/calendar_welcome.png` und `src/main/resources/UI/` werden relativ referenziert. Bei fehlenden Ressourcen wird die Anwendung funktionsfähig gehalten, ohne visuellen Inhalt zu verlieren.

### 5.4.2 Klasse start.Controller

Die `start.Controller` Klasse implementiert das Singleton-Pattern und steuert die Geschäftslogik der Startansicht der G2iCal-Anwendung im Rahmen des MVC-Patterns für die `start.View`.

## Architektur und Singleton-Pattern

Die Klasse verwendet das **Singleton-Pattern** mit der statischen Instanzvariable `instance` und der Methode `getInstance()`. Der private Konstruktor stellt sicher, dass nur eine Instanz der Startansicht existiert. Dies verhindert mehrfache Initialisierung der Benutzeroberfläche und gewährleistet konsistente Zustandsverwaltung während des Anwendungslbenszyklus.

## **View-Integration und MVC-Kommunikation**

Der Controller instanziert die zugehörige `View` im Konstruktor und hält eine Referenz darauf. Die Kommunikation erfolgt ausschließlich über definierte Methoden, wobei die View Benutzeraktionen über `ActionListener` und `MouseListener` an den Controller weiterleitet. Die package-private Sichtbarkeit der Controller-Methoden beschränkt den Zugriff auf das `ui.start` Package.

## **OAuth-Login-Prozess**

Die Methode `startLoginProcess()` koordiniert den Google OAuth-Flow. Sie ruft zunächst `CalendarApiConnector.getLoginUrl()` auf und zeigt den Login-Dialog über `view.showLoginPrompt()`. Der eigentliche OAuth-Prozess wird in einem separaten Thread mit `SwingUtilities.invokeLater()` und einem 500ms Delay ausgeführt, um vollständiges UI-Rendering zu gewährleisten. Bei erfolgreichem Login wird die Startansicht ausgeblendet und der `ui.main.Controller` initialisiert.

## **Thread-Management und Fehlerbehandlung**

Der Login-Prozess verwendet explizites Thread-Management mit `SwingUtilities.invokeLater()` für UI-Updates und separaten Worker-Threads für API-Aufrufe. Exception-Handling erfolgt für `IOException`, `GeneralSecurityException` und `InterruptedException`. Bei Fehlern werden Benutzer über `view.showErrorDialog()` informiert und Fehlerdetails in die Konsole geschrieben.

## **Cache-Verwaltung**

Die Methode `clearCache()` delegiert die Credential-Lösung an `CalendarApiConnector.clearCreden` und liefert visuelles Feedback über `view.showClearCacheSuccess()`. Diese Funktionalität ermöglicht es Benutzern, gespeicherte Authentifizierungsdaten zu entfernen, was bei Debugging nützlich ist.

## **View-Zustandsverwaltung**

Die private Methode `hideView()` verwaltet den Übergang zwischen Ansichten. Sie schließt aktive Login-Dialöge über `getLoginDialog().dispose()` und versteckt die Hauptansicht mit `setVisible(false)`. Die öffentliche Methode `showView()` ermöglicht die Rückkehr zur Startansicht nach Logout-Operationen. Diese Zustandsverwaltung unterstützt nahtlose Navigation zwischen verschiedenen Anwendungsbereichen.

### **5.4.3 Klasse main.View**

Die `main.View` Klasse repräsentiert die Hauptbenutzeroberfläche der G2iCal-Anwendung und implementiert eine responsive, benutzerfreundliche Oberfläche für die Kalenderauswahl, Datenvisualisierung und Export-Funktionalität.

## **Architektur und MVC-Integration**

Die Klasse erbt von `JFrame` und implementiert die View-Komponente des MVC-Patterns. Der package-private Konstruktor `View(Controller controller)` gewährleistet kontrollierte Instanziierung durch den zugehörigen Controller. Die Klasse enthält ausschließlich UI-Logik und delegiert alle Geschäftsoperationen an den `Controller`.

## UI-Komponenten und Responsive Design

Die Benutzeroberfläche verwendet ein **BorderLayout** als Hauptlayout mit drei Hauptbereichen: Formular (NORTH), Tabelle (CENTER) und Export-Bereich (SOUTH). Die Mindestgröße von 480x300 Pixeln und die Verwendung von `GridBagLayout` im Formularbereich ermöglichen responsive Anpassung an verschiedene Bildschirmgrößen. Stil-Konstanten wie `LIGHT_BLUE` gewährleisten konsistente Farbgebung.

## Menüsystem und Plattform-Integration

Das **Menüsystem** erkennt automatisch das Betriebssystem und passt sich entsprechend an. Auf macOS werden spezielle Desktop-Handler für About- und Quit-Funktionen verwendet, während andere Systeme traditionelle Menüstrukturen erhalten. Tastenkürzel werden plattformspezifisch zugewiesen (Cmd auf macOS, Ctrl auf anderen Systemen).

## Formular-Panel und Dateneingabe

Das `createFormPanel()` implementiert ein strukturiertes Eingabeformular mit `JComboBox` für Kalenderauswahl und `JDateChooser` für Datumseingaben. Die Standard-Zeitspanne von 30 Tagen wird automatisch gesetzt. Custom-Styling erfolgt über die `CleanRoundedBorder` Klasse für verbundene UI-Elemente.

## Event-Tabelle und Datenvisualisierung

Die **Event-Tabelle** verwendet ein `DefaultTableModel` mit konfigurierten Spaltenbreiten für optimale Darstellung. Die Tabelle ist schreibgeschützt (`isCellEditable` returns false) und unterstützt horizontales sowie vertikales Scrolling. Das `AUTO_RESIZE_ALL_COLUMNS` Verhalten gewährleistet responsive Spaltenanpassung.

## Action Listener und Eingabevalidierung

Die `LoadDataActionListener` Klasse implementiert umfassende Validierung: Kalenderauswahl-Prüfung, Datumsvalidierung, Chronologie-Kontrolle und Zeitspannen-Begrenzung (maximal 1 Jahr). Bei erfolgreicher Validierung werden formatierte Daten an `controller.loadCalendarEvent` übertragen. Der `ExportActionListener` koordiniert den Export-Prozess über `controller.exportEvent`.

## Dialog-Management und Benutzerinteraktion

Die Klasse implementiert drei spezialisierte Dialoge: **About-Dialog** mit Anwendungs-informationen, **Settings-Dialog** mit Dateiname- und Pfad-Konfiguration sowie **Error-Dialoge** für Fehlermeldungen. Der Settings-Dialog enthält einen integrierten Dateibrowser mit `JFileChooser` und Input-Validierung über `InputValidator`.

## Custom Border Implementation

Die innere Klasse `CleanRoundedBorder` implementiert das `Border` Interface für moderne UI-Gestaltung. Sie unterstützt selektive Rundung (links, rechts oder vollständig) für verbundene UI-Elemente und verwendet Anti-Aliasing für glatte Darstellung. Die Implementierung berücksichtigt `Graphics2D` für erweiterte Rendering-Funktionen.

#### 5.4.4 DateChooser-Implementierung und Styling

Die DateChooser-Funktionalität der G2iCal-Anwendung implementiert eine benutzerdefinierte Lösung für Datumsauswahl mit modernem Design und nahtloser Integration in die Benutzeroberfläche.

#### JDateChooser-Integration und Konfiguration

Die Anwendung verwendet die **toedter JDateChooser-Bibliothek** als Basis für die Datumsauswahl. Die Methode `createStyledDateChooser()` erstellt Instanzen mit dem Datumsformat "yyyy-MM-dd" und einer Standardgröße von 120x25 Pixeln. Die Konfiguration erfolgt über `setDateFormatString()` für konsistente Datumsdarstellung und `setPreferredSize()` für einheitliche UI-Dimensionen.

#### Custom Styling und Connected Design

Das Styling implementiert ein **Connected Design** zwischen Textfeld und Kalender-Button durch asymmetrische Rundung. Die Methode verwendet `SwingUtilities.invokeLater()` für verzögerte Styling-Anwendung nach vollständiger Komponenten-Initialisierung. Das Textfeld erhält linksseitige Rundung (`CleanRoundedBorder(12, true, false)`) während der Kalender-Button rechtsseitige Rundung (`CleanRoundedBorder(12, false, true)`) erhält.

#### Formatierung und Controller-Integration

Die Datumsformatierung für Controller-Kommunikation erfolgt über `SimpleDateFormat` mit "yyyy-MM-dd" Pattern. Die `format()` Methode konvertiert Date-Objekte zu String-Repräsentationen für API-Aufrufe. Diese Trennung gewährleistet saubere Datenübertragung zwischen View- und Controller-Schichten ohne direkte Date-Objekt-Abhängigkeiten.

#### 5.4.5 Klasse main.Controller

Die `main.Controller` Klasse fungiert als zentrale Steuerungseinheit der Hauptanwendung und implementiert die Controller-Komponente des MVC-Patterns für die Kalender-Export-Funktionalität.

#### Architektur und Dependency Management

Die Klasse verwaltet vier kritische Abhängigkeiten: `View` für die Benutzeroberfläche, `CalendarApiConnector` für Google Calendar API-Kommunikation, `Settings` für persistente Konfiguration und `ICal` für Kalenderdaten-Verarbeitung. Der Konstruktor initialisiert diese Komponenten und ruft `loadInitialData()` auf.

#### Initialisierung und Datenladung

Die Methode `loadInitialData()` führt den ersten API-Aufruf aus, um verfügbare Kalender zu laden. Diese Liste von Kalendern wird dann an die View gegeben für die Kalenderauswahl im Form. Exception-Handling erfolgt über `IOException` mit Benutzerbenachrichtigung durch die View.

## Benutzer- und Einstellungsmanagement

Drei package-private Methoden verwalten Anwendungseinstellungen: `getUserName()` ruft Benutzerdaten von der Google API ab, `getFileName()` und `getFilePath()` delegieren an das Settings-Singleton. Die Methode `updateSettings()` aktualisiert sowohl Dateiname als auch Pfad.

## Session- und Navigationskontrolle

Die Anwendungsnavigation wird durch drei Methoden gesteuert: `exitToStartScreen()` schließt die aktuelle View und reaktiviert den Start-Controller, `logout()` löscht gespeicherte Credentials über `CalendarApiConnector.logout()` und `closeView()` verwaltet View-Ressourcen durch `setVisible(false)` und `dispose()`.

## Event-Datenverarbeitung

Die öffentliche Methode `loadCalendarEvents()` koordiniert komplexe Datenvalidierung und API-Kommunikation. Sie verwendet `InputValidator` für Datumsvalidierung, konvertiert String-Eingaben zu `DateTime`-Objekten, führt API-Aufrufe aus und erstellt `ICal`-Instanzen. Bei erfolgreicher Verarbeitung wird die View mit Tabellendaten aktualisiert und der Export-Button aktiviert.

## Export-Funktionalität und Validierung

Die Methode `exportEvents()` implementiert umfassende Validierung vor dem Export: Überprüfung der `ICal`-Instanz, Dateinamen-Normalisierung über `InputValidator.normalizeFileName()` und Pfad-Validierung. Bei erfolgreicher Validierung delegiert sie an `iCal.exportICalToFile()` und liefert Statusfeedback. Exception-Handling erfolgt spezifisch für `ICalExportException`.

## Zustandsverwaltung und Reset-Funktionalität

Die `reset()` Methode stellt den Controller-Zustand zurück, indem sie die `ICal`-Instanz auf null setzt und `view.reset()` aufruft. Dies ermöglicht Benutzern, neue Datenoperationen zu starten, ohne die Anwendung neu zu initialisieren. Die Kapselung der `closeView()`-Methode als private gewährleistet kontrollierte Ressourcenverwaltung.

### 5.4.6 Settings-Verwaltung und Persistierung

Die Settings-Funktionalität der G2iCal-Anwendung implementiert ein System zur persistenten Speicherung von Anwendungseinstellungen mittels JSON-Serialisierung und dem Singleton-Pattern.

## Singleton-Pattern und Initialisierung

Die `Settings` Klasse verwendet das **Singleton-Pattern** mit lazy initialization über `getInstance()`. Die statische Variable `instance` wird erst bei Bedarf initialisiert, wobei `loadSettings()` automatisch aufgerufen wird. Der private Konstruktor initialisiert Standardwerte: `DEFAULT_FILE_NAME` als "calendar\_export.ics" und `DEFAULT_FILE_PATH` als System-Downloads-Ordner über `System.getProperty("user.home")`.

## JSON-Persistierung mit Gson

Die Persistierung erfolgt über die **Google Gson-Bibliothek** mit einem statischen `GsonBuilder` der Pretty-Printing aktiviert hat. Die Methode `saveSettings()` serialisiert die aktuelle Instanz zu JSON und speichert sie in `src/main/resources/settings.json`. Bei der Deserialisierung in `loadSettings()` werden geladene Werte validiert und bei null- oder leeren Werten durch Standardwerte ersetzt. Json biete sich in deisem Projekt an da die Gson-Bibliothek schon an anderer stelle gebraucht wird und so nicht extra dafür geladen werden muss.

## Dateipfad-Management und Fallback-Strategien

Die `getSettingsFilePath()` Methode konstruiert den Ressourcen-Pfad dynamisch über `System.getProperty("user.dir")`. Bei Fehlern beim Zugriff auf den Resources-Ordner wird ein Fallback auf das aktuelle Arbeitsverzeichnis implementiert. Die Methode `Files.createDirectory()` gewährleistet, dass das übergeordnete Verzeichnis vor dem Schreibvorgang existiert.

## Validierung und Exception-Handling

Das System implementiert mehrschichtige Validierung: In `loadSettings()` werden `IOException` und `JsonSyntaxException` abgefangen, wobei bei Fehlern eine neue Instanz mit Standardwerten erstellt wird. Die `updateSettings()` Methode validiert Eingaben auf null- und Leer-Werte vor der Zuweisung und führt automatisches Trimming durch.

## Settings-Update und Transaktionalität

Die `updateSettings(String fileName, String filePath)` Methode implementiert atomare Updates beider Werte mit sofortiger Persistierung. Die `resetToDefaults()` Methode stellt Standardwerte wieder her und speichert diese automatisch. Die `getFullExportPath()` Methode kombiniert Pfad und Dateiname über `Paths.get()` für plattformunabhängige Pfad-Konstruktion.

## Integration in das MVC-Pattern

Die Settings-Klasse fungiert als **Model-Komponente** im MVC-Pattern und wird vom `main.Controller` über Getter-Methoden `getFileName()` und `getFilePath()` abgerufen. Der Controller delegiert Settings-Updates über `updateSettings()` und gewährleistet damit die Trennung von Geschäftslogik und Datenhaltung. Die View-Schicht hat keinen direkten Zugriff auf Settings-Objekte.

## Beispiel-Konfiguration

Die JSON-Struktur in `src/main/resources/settings.json` ist minimal und benutzerfreundlich:

```
{  
  "fileName": "calendar_export.ics",  
  "filePath": "/Users/mitzel/Downloads"  
}
```

Diese Struktur ermöglicht manuelle Bearbeitung durch technische Benutzer und automatische Verwaltung durch die Anwendung. s

## 6 Fragen

### 6.1 Wie definiert man Zeilen/Spalten für ein JTable-Objekt

Die Definition von Zeilen und Spalten für ein `JTable`-Objekt erfolgt primär über das zugrunde liegende `TableModel`. Dieses Modell bestimmt sowohl die Struktur als auch den Inhalt der Tabelle.

#### 6.1.1 Spaltendefinition über TableModel

Die **Spaltenanzahl und -namen** werden durch die Methoden `getColumnCount()` und `getColumnName(int columnIndex)` des `TableModel` definiert. Bei Verwendung des `DefaultTableModel` können Spalten über den Konstruktor als String-Array übergeben werden: `new DefaultTableModel(columnNames, 0)`. Die Null als zweiter Parameter definiert die initiale Zeilenzahl.

#### 6.1.2 Zeilenverwaltung

**Zeilen** werden dynamisch über das `TableModel` verwaltet. Das `DefaultTableModel` bietet Methoden wie `addRow(Object[] rowData)` zum Hinzufügen neuer Zeilen und `setRowCount(int rowCount)` zum Festlegen der Gesamtzeilenzahl. Die Methode `removeRow(int row)` ermöglicht das Entfernen einzelner Zeilen.

#### 6.1.3 Datentyp-Definition

Die **Datentypen** der Spalten werden durch `getColumnClass(int columnIndex)` definiert. Diese Methode bestimmt, welcher Renderer und Editor für die jeweilige Spalte verwendet wird. Standardmäßig werden alle Werte als `Object` behandelt und als String dargestellt.

#### 6.1.4 Editierbarkeit

Die **Editierbarkeit** einzelner Zellen wird über `isCellEditable(int row, int column)` gesteuert. Durch Überschreiben dieser Methode kann gezielt festgelegt werden, welche Zellen bearbeitbar sind. Im Beispiel wird durch `return false` die gesamte Tabelle als schreibgeschützt definiert.

#### 6.1.5 Spaltenbreiten-Konfiguration

Die **Spaltenbreiten** werden über das  `TableColumnModel` konfiguriert. Methoden wie `setPreferredWidth()`, `setMinWidth()` und `setMaxWidth()` ermöglichen die präzise Kontrolle der Spaltenbreiten. Die automatische Größenanpassung wird durch `setAutoResizeMode()` gesteuert.

Die Kombination aus `TableModel` für Datenstruktur und `TableColumnModel` für Darstellungsattribute ermöglicht eine flexible und umfassende Konfiguration von Tabellen in Java Swing.

## 6.2 Was muss beachtet werden, wenn bei einem JTable-Objekt die Spaltenüberschriften angezeigt werden sollen?

Für die korrekte Anzeige von Spaltenüberschriften bei einem `JTable`-Objekt müssen verschiedene Aspekte beachtet werden.

### 6.2.1 JScrollPane-Integration

Das `JScrollPane` ist essentiell für die Anzeige der Spaltenüberschriften. Die Tabelle muss in ein `JScrollPane` eingebettet werden, da nur dieses die `JTableHeader` automatisch im Viewport-Header positioniert. Ohne `JScrollPane` werden die Spaltenüberschriften nicht angezeigt, selbst wenn sie im `TableModel` definiert sind.

### 6.2.2 TableModel-Konfiguration

Das verwendete `TableModel` muss die Methode `getColumnName(int columnIndex)` korrekt implementieren. Diese Methode definiert den Text, der in den Spaltenüberschriften angezeigt wird. Bei `DefaultTableModel` können die Spaltenüberschriften entweder im Konstruktor als Array übergeben oder nachträglich mit `setColumnIdentifiers()` gesetzt werden.

### 6.2.3 Header-Sichtbarkeit

Die `JTableHeader` kann explizit über `table.getTableHeader().setVisible(true)` gesteuert werden. Standardmäßig ist sie sichtbar, kann aber programmatisch ausgeblendet werden. Für spezielle Anforderungen kann eine eigene `TableCellRenderer` für den Header implementiert werden.

### 6.2.4 Layout-Konfiguration

Bei manueller Layout-Verwaltung ohne `JScrollPane` muss die `JTableHeader` explizit zum Container hinzugefügt werden. Dies erfolgt über `table.getTableHeader()` und die anschließende Positionierung im gewünschten Layout-Bereich, typischerweise im NORTH-Bereich eines `BorderLayout`.

### 6.2.5 Spaltenbreiten-Management

Die Spaltenbreiten können über das `TableColumnModel` angepasst werden. Methoden wie `setPreferredWidth()`, `setMinWidth()` und `setMaxWidth()` ermöglichen die Kontrolle über die Darstellung. Die automatische Größenanpassung kann mit `setAutoResizeMode()` konfiguriert werden.

Die korrekte Integration in ein `JScrollPane` ist der wichtigste Faktor für die Anzeige von Spaltenüberschriften, da dies die standardkonforme und robuste Lösung darstellt.

## 6.3 Welche Layoutmanager haben Sie verwendet? Begründen Sie Ihre Wahl.

Es wurden verschiedene Layoutmanager eingesetzt, um eine responsive und benutzerfreundliche Oberfläche zu schaffen.

### 6.3.1 BorderLayout

Das **BorderLayout** wird als Hauptlayout für die zentrale Inhaltsstruktur verwendet. Es teilt den verfügbaren Platz in fünf Bereiche auf: NORTH, SOUTH, EAST, WEST und CENTER. In der Hauptansicht wird das Formular im NORTH-Bereich, die Ereignistabelle im CENTER-Bereich und die Export-Kontrollen im SOUTH-Bereich platziert. Diese Wahl ermöglicht eine klare Hierarchie der UI-Elemente und sorgt dafür, dass der CENTER-Bereich automatisch den verfügbaren Platz ausfüllt, was die responsive Tabellendarstellung ermöglicht.

### 6.3.2 GridBagLayout

Das **GridBagLayout** kommt im Formularbereich zum Einsatz und bietet maximale Flexibilität bei der Positionierung von Komponenten. Mit **GridBagConstraints** können präzise Gewichtungen, Abstände und Ausrichtungen definiert werden. Dies ermöglicht eine professionelle Anordnung der Labels, Dropdown-Menüs, Datumswähler und Buttons mit konsistenten Abständen und optimaler Platznutzung.

### 6.3.3 BoxLayout

Das **BoxLayout** wird für vertikale und horizontale Anordnungen verwendet, insbesondere in der Startansicht für die vertikale Staplung der Elemente. Es bietet einfache Kontrolle über die Reihenfolge und Abstände zwischen Komponenten und eignet sich gut für lineare Layouts ohne komplexe Positionierungsanforderungen.

### 6.3.4 FlowLayout

Das **FlowLayout** wird für Button-Panels verwendet, um Buttons horizontal anzuordnen und automatisch umzubrechen, falls der Platz nicht ausreicht. Dies ist besonders nützlich für Aktions-Buttons am unteren Rand von Dialogen, da es eine natürliche und plattformkonforme Anordnung ermöglicht.

### 6.3.5 Begründung der Wahl:

Jeder Layoutmanager wird dort eingesetzt, wo seine Stärken am besten zur Geltung kommen: BorderLayout für die Hauptstruktur, GridBagLayout für Formulare, BoxLayout für einfache Stapelungen und FlowLayout für flexible Button-Anordnungen. Diese Strategie resultiert in einer responsiven Benutzeroberfläche, die sich automatisch an verschiedene Fenstergrößen anpasst.

## 6.4 Wie definiert man, welche Aktion bei der Auswahl eines Menüs ausgeführt werden soll?

### 6.4.1 Definition von Menü-Aktionen

Die Definition von Menü-Aktionen in Java Swing erfolgt über das **ActionListener-Interface**. Jedes JMenuItem kann einen ActionListener erhalten, der die Methode `actionPerformed(ActionEvent e)` implementiert.

### 6.4.2 Implementierungsarten

**Lambda-Ausdrücke** bieten eine moderne und kompakte Syntax für einfache Aktionen. Die Aktion wird direkt beim Hinzufügen zum MenuItem definiert. Beispielsweise bei Exit to Start Screen:

```
1 JMenuItem exitItem = new JMenuItem("Exit to Start Screen");
2 exitItem.addActionListener(e -> controller.exitToStartScreen())
  ;
```

**Separate ActionListener-Klassen** werden als innere Klassen der View implementiert und sind wiederverwendbar sowie gut strukturiert. Diese Variante ist ideal für umfangreiche Aktionslogik und ermöglicht eine bessere Code-Organisation. Zum Beispiel bei save:

```
1 loadDataButton.addActionListener(new LoadDataActionListener());
2
3 private class LoadDataActionListener implements ActionListener {
4     @Override
5     public void actionPerformed(ActionEvent e) {
6         // Aktionscode hier
7         controller.loadCalendarEvents(selectedCalendarId,
8             fromDateStr, toDateStr);
9     }
}
```

### 6.4.3 Zusätzliche Konfiguration

**Tastenkürzel** können mit der Methode `setAccelerator()` definiert werden, um Menüs auch über die Tastatur zugänglich zu machen. Dabei wird zwischen macOS (`meta`) und anderen Systemen (`ctrl`) unterschieden.

Die **Aktivierung und Deaktivierung** von Menüelementen erfolgt über `setEnabled()`, um die Benutzerführung zu verbessern und nicht verfügbare Funktionen zu kennzeichnen.

Das ActionListener-Pattern ermöglicht eine saubere Trennung zwischen UI-Komponenten und Logik durch Delegation an Controller-Methoden. Dies unterstützt das Model-View-Controller-Architekturmuster und verbessert die Wartbarkeit des Codes.