Technisches Konzept: FUR-Kalender v2

1. Architekturprinzipien

Das Kalendermodul folgt Clean Architecture: Trennung von UI (Cogs), Logik (Services) und Daten (Models/Repos). Asynchrone Verarbeitung mit asyncio wird durchgängig genutzt.

2. Modulstruktur

- Cogs: calendar_cog.py (Slash-Kommandos)
- Services: calendar_service.py (Google Sync, Event-Logik)
- Models: user_model.py, event_model.py
- Utils: timezone.py, oauth_utils.py

3. Datenfluss

- OAuth mit PKCE für User-Login
- Google Events mit Delta-Sync (syncToken)
- Datenbank (MongoDB) speichert Events und Tokens
- Täglicher/Wöchentlicher DM-Versand via Discord Tasks

4. Sicherheit

- PKCE-basierter OAuth2-Flow
- Refresh Tokens verschlüsselt speichern
- Zugriff über HTTPS
- Discord-Ratelimits und Fehlerhandling bei DMs beachten

5. Erweiterbarkeit

- Kalender-Provider Interface definieren

- GoogleCalendarProvider als Basis

- Später: Outlook, iCloud, ICS-Dateien

6. Risiken & Bottlenecks

- Google API-Quotas

- Token-Ablauf (HTTP 410) -> Neusync

- Discord DM-Blockierungen

- Railway Storage flüchtig -> MongoDB Atlas

7. Empfohlene Libraries

- discord.py / pycord

- google-api-python-client, google-auth-oauthlib

- motor (MongoDB)

- zoneinfo / pytz für Zeitzonen

8. Beispielcode

Slash-Command:

```
@commands.slash_command(description="Zeige Events für heute")
async def calendar_today(self, ctx):
    ...
```

DM-Button:

```
view = discord.ui.View()
view.add_item(discord.ui.Button(label="Google-Konto verknüpfen", url=auth_url))
await ctx.respond("Verknüpfe deinen Kalender:", view=view)
```

9. Integration

- Cog kann per Extension geladen werden

- Rollenprüfung per @commands.has_role

- Nutzerkonfiguration über DB (Zeitzone, DM-Frequenz)

Ziel: Ein skalierbares, wartbares Kalender-Modul mit Google-Sync und Discord-DM-Reminder.