# Calendar Family Pro - Web App

## Guide and Code for Web App Family Calendar

### One-Line Download & Execute:

After running, you'll have a fully functional Family Calendar Pro ready for production use!

### Enhanced Backend (FastAPI improvements)

**File: backend/app/models.py** (New - Better model organization)

```
from datetime import datetime, timezone
from sqlalchemy import Column, Integer, String, Boolean, DateTime, Text, ForeignKey, JSON,
UniqueConstraint
from sqlalchemy.orm import declarative_base, relationship

Base = declarative_base()

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True)
    username = Column(String(50), unique=True, index=True, nullable=False)
    email = Column(String(200), unique=True, index=True, nullable=False)
    hashed_password = Column(String(255), nullable=False)
    is_admin = Column(Boolean, default=False)
    is_active = Column(Boolean, default=True)
    telegram_chat_id = Column(String(64), nullable=True)
    ntfy_topic = Column(String(128), nullable=True)
    created_at = Column(DateTime(timezone=True), default=lambda: datetime.now(timezone.utc))
    last_login = Column(DateTime(timezone=True), nullable=True)

    settings = relationship("UserSettings", back_populates="user", uselist=False,
cascade="all, delete-orphan")
    events = relationship("Event", back_populates="owner", cascade="all, delete-orphan")

class UserSettings(Base):
```

```
    __tablename__ = "user_settings"
    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("users.id"), unique=True)
    theme = Column(String(10), default="day")
    default_notifiers = Column(JSON, default=lambda: ["email"])
    default_offset_min = Column(Integer, default=60)
    timezone = Column(String(50), default="UTC")
    language = Column(String(10), default="en")

    user = relationship("User", back_populates="settings")

class Event(Base):
    __tablename__ = "events"
    id = Column(Integer, primary_key=True)
    owner_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    title = Column(String(200), nullable=False)
    description = Column(Text, nullable=True)
    location = Column(String(255), nullable=True)
    start_utc = Column(DateTime(timezone=True), nullable=False)
    end_utc = Column(DateTime(timezone=True), nullable=True)
    all_day = Column(Boolean, default=False)
    color = Column(String(7), default="#1a73e8")
    category = Column(String(50), nullable=True)
    notify_offsets_min = Column(JSON, default=lambda: [])
    notify_channels = Column(JSON, nullable=True)
    is_recurring = Column(Boolean, default=False)
    recurrence_rule = Column(JSON, nullable=True)
    created_at = Column(DateTime(timezone=True), default=lambda: datetime.now(timezone.utc))
    updated_at = Column(DateTime(timezone=True), default=lambda: datetime.now(timezone.utc),
onupdate=lambda: datetime.now(timezone.utc))

    owner = relationship("User", back_populates="events")

class AppSettings(Base):
    __tablename__ = "app_settings"
    id = Column(Integer, primary_key=True, default=1)
    app_name = Column(String(100), default="Family Calendar")
    ntfy_server = Column(String(255), nullable=True)
    smtp_host = Column(String(255), nullable=True)
    smtp_port = Column(Integer, nullable=True)
```

```
    smtp_use_tls = Column(Boolean, default=True)
    smtp_user = Column(String(255), nullable=True)
    smtp_password = Column(String(255), nullable=True)
    smtp_from = Column(String(255), nullable=True)
    telegram_bot_token = Column(String(255), nullable=True)
    max_users = Column(Integer, default=10)
    registration_enabled = Column(Boolean, default=False)

class NotificationLog(Base):
    __tablename__ = "notification_logs"
    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("users.id"))
    event_id = Column(Integer, ForeignKey("events.id"))
    channel = Column(String(20), nullable=False)
    status = Column(String(20), nullable=False)  # sent, failed, pending
    sent_at = Column(DateTime(timezone=True), default=lambda: datetime.now(timezone.utc))
    error_message = Column(Text, nullable=True)

class UserSession(Base):
    __tablename__ = "user_sessions"
    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("users.id"))
    session_token = Column(String(255), unique=True, nullable=False)
    created_at = Column(DateTime(timezone=True), default=lambda: datetime.now(timezone.utc))
    expires_at = Column(DateTime(timezone=True), nullable=False)
    is_active = Column(Boolean, default=True)
```

**File: backend/app/auth.py** (Enhanced authentication)

```
from datetime import datetime, timedelta, timezone
from typing import Optional
from fastapi import Depends, HTTPException, status, Request
from fastapi.security import OAuth2PasswordBearer
from jose import JWTError, jwt
from passlib.context import CryptContext
from sqlalchemy.orm import Session
from .models import User, UserSession
from .database import get_db
import secrets
import os
```

```
SECRET_KEY = os.getenv("SECRET_KEY", "CHANGE_ME_PRODUCTION")
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60 * 24  # 24 hours

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/token")

class AuthManager:
    @staticmethod
    def verify_password(plain_password: str, hashed_password: str) -> bool:
        return pwd_context.verify(plain_password, hashed_password)

    @staticmethod
    def get_password_hash(password: str) -> str:
        return pwd_context.hash(password)

    @staticmethod
    def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
        to_encode = data.copy()
        if expires_delta:
            expire = datetime.now(timezone.utc) + expires_delta
        else:
            expire = datetime.now(timezone.utc) +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
        to_encode.update({"exp": expire})
        return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

    @staticmethod
    def create_session_token() -> str:
        return secrets.token_urlsafe(32)

async def get_current_user(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db))
-> User:
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
```

```
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception

    user = db.query(User).filter(User.username == username, User.is_active == True).first()
    if user is None:
        raise credentials_exception

    # Update last login
    user.last_login = datetime.now(timezone.utc)
    db.commit()

    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)) -> User:
    if not current_user.is_active:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

async def get_admin_user(current_user: User = Depends(get_current_active_user)) -> User:
    if not current_user.is_admin:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Admin access required"
        )
    return current_user

def authenticate_user(db: Session, username: str, password: str) -> Optional[User]:
    user = db.query(User).filter(User.username == username, User.is_active == True).first()
    if not user or not AuthManager.verify_password(password, user.hashed_password):
        return None
    return user
```

**File: backend/app/notifications.py** (Enhanced notification system)

```
import asyncio
import smtplib
```

```
from datetime import datetime, timezone, timedelta
from email.message import EmailMessage
from typing import List, Optional
import httpx
from sqlalchemy.orm import Session
from .models import Event, User, AppSettings, NotificationLog
from .database import SessionLocal
import logging

logger = logging.getLogger(__name__)

class NotificationService:
    def __init__(self):
        self.session_timeout = 30

    async def send_ntfy_notification(self, server_url: str, topic: str, title: str, message:
str) -> bool:
        """Send notification via self-hosted ntfy"""
        try:
            url = f"{server_url.rstrip('/')}/{topic}"
            headers = {
                "Title": title,
                "Priority": "default",
                "Tags": "calendar,reminder"
            }

            async with httpx.AsyncClient(timeout=self.session_timeout) as client:
                response = await client.post(url, content=message, headers=headers)
                return response.status_code == 200
        except Exception as e:
            logger.error(f"NTFY notification failed: {e}")
            return False

    def send_email_notification(self, smtp_config: dict, to_email: str, subject: str, body:
str) -> bool:
        """Send email notification via SMTP"""
        try:
            msg = EmailMessage()
            msg["Subject"] = subject
            msg["From"] = smtp_config["from"]
```

```
            msg["To"] = to_email
            msg.set_content(body)

            with smtplib.SMTP(smtp_config["host"], smtp_config["port"]) as smtp:
                if smtp_config.get("use_tls", True):
                    smtp.starttls()
                if smtp_config.get("user") and smtp_config.get("password"):
                    smtp.login(smtp_config["user"], smtp_config["password"])
                smtp.send_message(msg)
            return True
        except Exception as e:
            logger.error(f"Email notification failed: {e}")
            return False

    async def send_telegram_notification(self, bot_token: str, chat_id: str, message: str) ->
bool:
        """Send notification via Telegram Bot API"""
        try:
            url = f"https://api.telegram.org/bot{bot_token}/sendMessage"
            payload = {
                "chat_id": chat_id,
                "text": message,
                "parse_mode": "Markdown",
                "disable_web_page_preview": True
            }

            async with httpx.AsyncClient(timeout=self.session_timeout) as client:
                response = await client.post(url, json=payload)
                return response.status_code == 200
        except Exception as e:
            logger.error(f"Telegram notification failed: {e}")
            return False

    async def send_event_notification(self, db: Session, event_id: int, offset_minutes: int =
0):
        """Send notifications for a specific event"""
        event = db.query(Event).filter(Event.id == event_id).first()
        if not event:
            return
```

```
        user = db.query(User).filter(User.id == event.owner_id).first()
        if not user:
            return

        app_settings = db.query(AppSettings).first()
        if not app_settings:
            return

        # Determine which channels to use
        channels = event.notify_channels or (user.settings.default_notifiers if user.settings
else ["email"])

        # Format notification content
        event_time = event.start_utc.strftime("%Y-%m-%d %H:%M UTC")
        if offset_minutes > 0:
            title = f"Reminder: {event.title} in {offset_minutes} minutes"
        else:
            title = f"Event starting now: {event.title}"

        body = f"""
**{event.title}**
□□{event_time}
□□{event.location or "No location specified"}

{event.description or ""}
        """.strip()

        # Send notifications
        for channel in channels:
            success = False
            error_msg = None

            try:
                if channel == "ntfy" and app_settings.ntfy_server and user.ntfy_topic:
                    success = await self.send_ntfy_notification(
                        app_settings.ntfy_server, user.ntfy_topic, title, body
                    )
                elif channel == "email" and user.email and app_settings.smtp_host:
                    smtp_config = {
                        "host": app_settings.smtp_host,
```

```
                        "port": app_settings.smtp_port or 587,
                        "use_tls": app_settings.smtp_use_tls,
                        "user": app_settings.smtp_user,
                        "password": app_settings.smtp_password,
                        "from": app_settings.smtp_from
                    }
                    success = self.send_email_notification(smtp_config, user.email, title,
body)
                elif channel == "telegram" and app_settings.telegram_bot_token and
user.telegram_chat_id:
                    success = await self.send_telegram_notification(
                        app_settings.telegram_bot_token, user.telegram_chat_id, body
                    )
            except Exception as e:
                error_msg = str(e)
                logger.error(f"Notification failed for channel {channel}: {e}")

            # Log the notification attempt
            log_entry = NotificationLog(
                user_id=user.id,
                event_id=event.id,
                channel=channel,
                status="sent" if success else "failed",
                error_message=error_msg
            )
            db.add(log_entry)

        db.commit()

notification_service = NotificationService()
```

**File: backend/app/main.py** (Enhanced main application)

```
import os
from datetime import datetime, timedelta, timezone
from typing import List, Optional
from fastapi import FastAPI, Depends, HTTPException, status, BackgroundTasks
from fastapi.security import OAuth2PasswordRequestForm
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
```

```
from sqlalchemy.orm import Session
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from apscheduler.triggers.date import DateTrigger

from .models import Base, User, UserSettings, Event, AppSettings, NotificationLog
from .database import engine, get_db
from .auth import AuthManager, get_current_active_user, get_admin_user, authenticate_user
from .notifications import notification_service
from .schemas import *

# Create tables
Base.metadata.create_all(bind=engine)

app = FastAPI(
    title="Family Calendar Pro",
    description="Advanced multi-user calendar with notifications",
    version="2.0.0"
)

# CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Configure appropriately for production
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Static files
app.mount("/static", StaticFiles(directory="frontend/static"), name="static")

# Scheduler for notifications
scheduler = AsyncIOScheduler()

def schedule_event_notifications(event: Event, db: Session):
    """Schedule all notifications for an event"""
    now = datetime.now(timezone.utc)

    # Clear existing jobs for this event
    for job in scheduler.get_jobs():
```

```
        if job.id.startswith(f"event-{event.id}-"):
            job.remove()

    # Schedule notifications for each offset
    offsets = event.notify_offsets_min or []
    if 0 not in offsets:  # Always include notification at event time
        offsets.append(0)

    for offset in offsets:
        run_time = event.start_utc - timedelta(minutes=offset)
        if run_time > now:
            scheduler.add_job(
                notification_service.send_event_notification,
                trigger=DateTrigger(run_date=run_time),
                args=[SessionLocal(), event.id, offset],
                id=f"event-{event.id}-{offset}",
                replace_existing=True
            )

@app.on_event("startup")
async def startup_event():
    """Initialize application on startup"""
    db = SessionLocal()

    # Ensure app settings exist
    if not db.query(AppSettings).first():
        settings = AppSettings()
        db.add(settings)
        db.commit()

    # Start scheduler
    scheduler.start()

    # Reschedule all future events
    now = datetime.now(timezone.utc)
    future_events = db.query(Event).filter(Event.start_utc > now).all()
    for event in future_events:
        schedule_event_notifications(event, db)

    db.close()
```

```
# Authentication endpoints
@app.post("/auth/token", response_model=Token)
async def login_for_access_token(form_data: OAuth2PasswordRequestForm = Depends(), db: Session
= Depends(get_db)):
    user = authenticate_user(db, form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )

    access_token_expires = timedelta(minutes=AuthManager.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = AuthManager.create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}

@app.post("/auth/bootstrap", response_model=UserResponse)
async def bootstrap_admin(user_data: UserCreate, db: Session = Depends(get_db)):
    """Create first admin user if no users exist"""
    if db.query(User).count() > 0:
        raise HTTPException(status_code=400, detail="Users already exist")

    hashed_password = AuthManager.get_password_hash(user_data.password)
    user = User(
        username=user_data.username,
        email=user_data.email,
        hashed_password=hashed_password,
        is_admin=True,
        telegram_chat_id=user_data.telegram_chat_id,
        ntfy_topic=user_data.ntfy_topic
    )
    db.add(user)
    db.flush()

    # Create default settings
    settings = UserSettings(user_id=user.id)
    db.add(settings)
```

```
    db.commit()
    db.refresh(user)

    return user

# User management endpoints
@app.get("/users/me", response_model=UserResponse)
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user

@app.get("/users", response_model=List[UserResponse])
async def list_users(current_user: User = Depends(get_admin_user), db: Session =
Depends(get_db)):
    return db.query(User).all()

@app.post("/users", response_model=UserResponse)
async def create_user(user_data: UserCreate, current_user: User = Depends(get_admin_user), db:
Session = Depends(get_db)):
    # Check if username or email already exists
    if db.query(User).filter(User.username == user_data.username).first():
        raise HTTPException(status_code=400, detail="Username already registered")
    if db.query(User).filter(User.email == user_data.email).first():
        raise HTTPException(status_code=400, detail="Email already registered")

    hashed_password = AuthManager.get_password_hash(user_data.password)
    user = User(
        username=user_data.username,
        email=user_data.email,
        hashed_password=hashed_password,
        is_admin=user_data.is_admin,
        telegram_chat_id=user_data.telegram_chat_id,
        ntfy_topic=user_data.ntfy_topic
    )
    db.add(user)
    db.flush()

    # Create default settings
    settings = UserSettings(user_id=user.id)
    db.add(settings)
    db.commit()
```

```
    db.refresh(user)

    return user

@app.put("/users/{user_id}", response_model=UserResponse)
async def update_user(user_id: int, user_data: UserUpdate, current_user: User =
Depends(get_admin_user), db: Session = Depends(get_db)):
    user = db.query(User).filter(User.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")

    # Update fields
    if user_data.email:
        user.email = user_data.email
    if user_data.is_admin is not None:
        user.is_admin = user_data.is_admin
    if user_data.is_active is not None:
        user.is_active = user_data.is_active
    if user_data.telegram_chat_id:
        user.telegram_chat_id = user_data.telegram_chat_id
    if user_data.ntfy_topic:
        user.ntfy_topic = user_data.ntfy_topic
    if user_data.password:
        user.hashed_password = AuthManager.get_password_hash(user_data.password)

    db.commit()
    db.refresh(user)
    return user

@app.delete("/users/{user_id}")
async def delete_user(user_id: int, current_user: User = Depends(get_admin_user), db: Session
= Depends(get_db)):
    if user_id == current_user.id:
        raise HTTPException(status_code=400, detail="Cannot delete yourself")

    user = db.query(User).filter(User.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")

    db.delete(user)
```

```
    db.commit()
    return {"message": "User deleted successfully"}

# Event endpoints
@app.get("/events", response_model=List[EventResponse])
async def get_events(current_user: User = Depends(get_current_active_user), db: Session =
Depends(get_db)):
    events = db.query(Event).filter(Event.owner_id == current_user.id).all()
    return events

@app.post("/events", response_model=EventResponse)
async def create_event(event_data: EventCreate, current_user: User =
Depends(get_current_active_user), db: Session = Depends(get_db)):
    event = Event(
        owner_id=current_user.id,
        title=event_data.title,
        description=event_data.description,
        location=event_data.location,
        start_utc=event_data.start_utc,
        end_utc=event_data.end_utc,
        all_day=event_data.all_day,
        color=event_data.color,
        category=event_data.category,
        notify_offsets_min=event_data.notify_offsets_min,
        notify_channels=event_data.notify_channels
    )
    db.add(event)
    db.commit()
    db.refresh(event)

    # Schedule notifications
    schedule_event_notifications(event, db)

    return event

@app.put("/events/{event_id}", response_model=EventResponse)
async def update_event(event_id: int, event_data: EventUpdate, current_user: User =
Depends(get_current_active_user), db: Session = Depends(get_db)):
    event = db.query(Event).filter(Event.id == event_id, Event.owner_id ==
current_user.id).first()
```

```
    if not event:
        raise HTTPException(status_code=404, detail="Event not found")

    # Update fields
    for field, value in event_data.dict(exclude_unset=True).items():
        setattr(event, field, value)

    event.updated_at = datetime.now(timezone.utc)
    db.commit()
    db.refresh(event)

    # Reschedule notifications
    schedule_event_notifications(event, db)

    return event

@app.delete("/events/{event_id}")
async def delete_event(event_id: int, current_user: User = Depends(get_current_active_user),
db: Session = Depends(get_db)):
    event = db.query(Event).filter(Event.id == event_id, Event.owner_id ==
current_user.id).first()
    if not event:
        raise HTTPException(status_code=404, detail="Event not found")

    # Remove scheduled jobs
    for job in scheduler.get_jobs():
        if job.id.startswith(f"event-{event.id}-"):
            job.remove()

    db.delete(event)
    db.commit()
    return {"message": "Event deleted successfully"}

@app.get("/events/search", response_model=List[EventResponse])
async def search_events(
    q: Optional[str] = None,
    category: Optional[str] = None,
    start_date: Optional[datetime] = None,
    end_date: Optional[datetime] = None,
    current_user: User = Depends(get_current_active_user),
```

```
    db: Session = Depends(get_db)
):
    query = db.query(Event).filter(Event.owner_id == current_user.id)

    if q:
        search_term = f"%{q}%"
        query = query.filter(
            (Event.title.like(search_term)) |
            (Event.description.like(search_term)) |
            (Event.location.like(search_term))
        )

    if category:
        query = query.filter(Event.category == category)

    if start_date:
        query = query.filter(Event.start_utc >= start_date)

    if end_date:
        query = query.filter(Event.start_utc <= end_date)

    events = query.order_by(Event.start_utc).limit(100).all()
    return events

# Settings endpoints
@app.get("/settings/user", response_model=UserSettingsResponse)
async def get_user_settings(current_user: User = Depends(get_current_active_user), db: Session
= Depends(get_db)):
    settings = db.query(UserSettings).filter(UserSettings.user_id == current_user.id).first()
    if not settings:
        settings = UserSettings(user_id=current_user.id)
        db.add(settings)
        db.commit()
        db.refresh(settings)
    return settings

@app.put("/settings/user", response_model=UserSettingsResponse)
async def update_user_settings(settings_data: UserSettingsUpdate, current_user: User =
Depends(get_current_active_user), db: Session = Depends(get_db)):
    settings = db.query(UserSettings).filter(UserSettings.user_id == current_user.id).first()
```

```
    if not settings:
        settings = UserSettings(user_id=current_user.id)
        db.add(settings)

    for field, value in settings_data.dict(exclude_unset=True).items():
        setattr(settings, field, value)

    db.commit()
    db.refresh(settings)
    return settings

@app.get("/settings/app", response_model=AppSettingsResponse)
async def get_app_settings(current_user: User = Depends(get_admin_user), db: Session =
Depends(get_db)):
    settings = db.query(AppSettings).first()
    if not settings:
        settings = AppSettings()
        db.add(settings)
        db.commit()
        db.refresh(settings)
    return settings

@app.put("/settings/app", response_model=AppSettingsResponse)
async def update_app_settings(settings_data: AppSettingsUpdate, current_user: User =
Depends(get_admin_user), db: Session = Depends(get_db)):
    settings = db.query(AppSettings).first()
    if not settings:
        settings = AppSettings()
        db.add(settings)

    for field, value in settings_data.dict(exclude_unset=True).items():
        setattr(settings, field, value)

    db.commit()
    db.refresh(settings)
    return settings

# Notification logs endpoint
@app.get("/notifications/logs", response_model=List[NotificationLogResponse])
async def get_notification_logs(current_user: User = Depends(get_current_active_user), db:
```

```
Session = Depends(get_db)):
    logs = db.query(NotificationLog).filter(NotificationLog.user_id ==
current_user.id).order_by(NotificationLog.sent_at.desc()).limit(50).all()
    return logs
```

# Enhanced Frontend

**File: frontend/static/index.html** (Improved UI with modals)

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Family Calendar Pro</title>
  <link rel="stylesheet" href="/static/vendor/fullcalendar/index.global.min.css">
  <link rel="stylesheet" href="/static/styles.css">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
</head>
<body class="theme-day">
  <!-- Top Navigation -->
  <header class="topbar">
    <div class="nav-left">
      <h1 class="app-title"><i class="fas fa-calendar-alt"></i> Family Calendar</h1>
      <div class="nav-controls">
        <button id="todayBtn" class="btn btn-secondary">Today</button>
        <div class="view-buttons">
          <button id="monthBtn" class="btn btn-outline active">Month</button>
          <button id="weekBtn" class="btn btn-outline">Week</button>
          <button id="dayBtn" class="btn btn-outline">Day</button>
        </div>
      </div>
    </div>
    <div class="nav-right">
      <div class="search-box">
        <input id="searchInput" type="text" placeholder="Search events..." />
        <button id="searchBtn" class="btn btn-icon"><i class="fas fa-search"></i></button>
      </div>
      <div class="user-menu" id="userMenu">
```

```
        <button id="userMenuBtn" class="btn btn-icon">
          <i class="fas fa-user"></i>
          <span id="currentUsername"></span>
        </button>
        <div class="dropdown-menu" id="userDropdown">
          <a href="#" id="profileBtn"><i class="fas fa-user-cog"></i> Profile</a>
          <a href="#" id="settingsBtn"><i class="fas fa-cog"></i> Settings</a>
          <a href="#" id="adminBtn" class="admin-only"><i class="fas fa-shield-alt"></i> Admin
Panel</a>
          <hr>
          <a href="#" id="logoutBtn"><i class="fas fa-sign-out-alt"></i> Logout</a>
        </div>
      </div>
    </div>
  </header>

  <!-- Main Content -->
  <main class="main-content">
    <!-- Login Form -->
    <div id="loginContainer" class="login-container">
      <div class="login-card">
        <h2>Welcome Back</h2>
        <form id="loginForm">
          <div class="form-group">
            <label for="username">Username</label>
            <input id="username" type="text" required />
          </div>
          <div class="form-group">
            <label for="password">Password</label>
            <input id="password" type="password" required />
          </div>
          <button type="submit" class="btn btn-primary btn-block">Sign In</button>
        </form>
        <div class="login-error" id="loginError"></div>
      </div>
    </div>

    <!-- Sidebar -->
    <aside class="sidebar" id="sidebar">
      <div class="sidebar-section">
```

```
      <h3>Quick Actions</h3>
      <button id="newEventBtn" class="btn btn-primary btn-block">
        <i class="fas fa-plus"></i> New Event
      </button>
    </div>

    <!-- Mini Calendar -->
    <div class="sidebar-section">
      <h3>Navigate</h3>
      <div id="miniCalendar"></div>
    </div>

    <!-- Categories -->
    <div class="sidebar-section">
      <h3>Categories</h3>
      <div id="categoriesList" class="categories-list"></div>
    </div>

    <!-- Admin Panel -->
    <div class="sidebar-section admin-only" id="adminSection">
      <h3>Administration</h3>
      <button id="manageUsersBtn" class="btn btn-outline btn-block">
        <i class="fas fa-users"></i> Manage Users
      </button>
      <button id="appSettingsBtn" class="btn btn-outline btn-block">
        <i class="fas fa-server"></i> App Settings
      </button>
    </div>
  </aside>

  <!-- Calendar -->
  <section class="calendar-container">
    <div id="calendar"></div>
  </section>
</main>

<!-- Event Modal -->
<div id="eventModal" class="modal">
  <div class="modal-dialog modal-large">
    <div class="modal-content">
```

```
<div class="modal-header">
  <h3 id="eventModalTitle">New Event</h3>
  <button id="closeEventModal" class="btn btn-icon">
    <i class="fas fa-times"></i>
  </button>
</div>
<div class="modal-body">
  <form id="eventForm">
    <div class="form-row">
      <div class="form-group">
        <label for="eventTitle">Title *</label>
        <input id="eventTitle" type="text" required />
      </div>
      <div class="form-group">
        <label for="eventCategory">Category</label>
        <select id="eventCategory">
          <option value="">Select Category</option>
          <option value="work">Work</option>
          <option value="personal">Personal</option>
          <option value="family">Family</option>
          <option value="health">Health</option>
          <option value="other">Other</option>
        </select>
      </div>
    </div>

    <div class="form-group">
      <label for="eventDescription">Description</label>
      <textarea id="eventDescription" rows="3"></textarea>
    </div>

    <div class="form-group">
      <label for="eventLocation">Location</label>
      <input id="eventLocation" type="text" />
    </div>

    <div class="form-row">
      <div class="form-group">
        <label for="eventStart">Start Date & Time *</label>
        <input id="eventStart" type="datetime-local" required />
```

```
            </div>
            <div class="form-group">
              <label for="eventEnd">End Date & Time</label>
              <input id="eventEnd" type="datetime-local" />
            </div>
          </div>

          <div class="form-row">
            <div class="form-group">
              <label>
                <input id="eventAllDay" type="checkbox" />
                All Day Event
              </label>
            </div>
            <div class="form-group">
              <label for="eventColor">Color</label>
              <input id="eventColor" type="color" value="#1a73e8" />
            </div>
          </div>

          <div class="form-group">
            <label>Notification Channels</label>
            <div class="checkbox-group">
              <label><input id="notifyEmail" type="checkbox" value="email" /> Email</label>
              <label><input id="notifyNtfy" type="checkbox" value="ntfy" /> NTFY</label>
              <label><input id="notifyTelegram" type="checkbox" value="telegram" />
Telegram</label>
            </div>
          </div>

          <div class="form-group">
            <label for="reminderOffsets">Reminder Times (minutes before)</label>
            <div class="reminder-options">
              <label><input type="checkbox" value="5" /> 5 min</label>
              <label><input type="checkbox" value="15" /> 15 min</label>
              <label><input type="checkbox" value="30" /> 30 min</label>
              <label><input type="checkbox" value="60" checked /> 1 hour</label>
              <label><input type="checkbox" value="1440" /> 1 day</label>
            </div>
            <input id="customReminder" type="text" placeholder="Custom (comma-separated
```

```
minutes)" />
            </div>
          </form>
        </div>
        <div class="modal-footer">
          <button id="deleteEventBtn" class="btn btn-danger" style="display: none;">
            <i class="fas fa-trash"></i> Delete
          </button>
          <div class="modal-actions">
            <button id="cancelEventBtn" class="btn btn-secondary">Cancel</button>
            <button id="saveEventBtn" class="btn btn-primary">
              <i class="fas fa-save"></i> Save Event
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>

  <!-- User Management Modal -->
  <div id="userManagementModal" class="modal">
    <div class="modal-dialog modal-large">
      <div class="modal-content">
        <div class="modal-header">
          <h3>User Management</h3>
          <button id="closeUserModal" class="btn btn-icon">
            <i class="fas fa-times"></i>
          </button>
        </div>
        <div class="modal-body">
          <div class="tab-container">
            <div class="tabs">
              <button class="tab-button active" data-tab="usersList">Users List</button>
              <button class="tab-button" data-tab="addUser">Add User</button>
            </div>

            <div id="usersList" class="tab-content active">
              <div class="table-container">
                <table id="usersTable" class="data-table">
                  <thead>
```

```
          <tr>
            <th>Username</th>
            <th>Email</th>
            <th>Role</th>
            <th>Status</th>
            <th>Last Login</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody></tbody>
      </table>
    </div>
  </div>

  <div id="addUser" class="tab-content">
    <form id="addUserForm">
      <div class="form-row">
        <div class="form-group">
          <label for="newUsername">Username *</label>
          <input id="newUsername" type="text" required />
        </div>
        <div class="form-group">
          <label for="newEmail">Email *</label>
          <input id="newEmail" type="email" required />
        </div>
      </div>
      <div class="form-group">
        <label for="newPassword">Password *</label>
        <input id="newPassword" type="password" required />
      </div>
      <div class="form-row">
        <div class="form-group">
          <label for="newTelegramChat">Telegram Chat ID</label>
          <input id="newTelegramChat" type="text" />
        </div>
        <div class="form-group">
          <label for="newNtfyTopic">NTFY Topic</label>
          <input id="newNtfyTopic" type="text" />
        </div>
      </div>
```

```
              <div class="form-group">
                <label>
                  <input id="newIsAdmin" type="checkbox" />
                  Administrator privileges
                </label>
              </div>
              <button type="submit" class="btn btn-primary">
                <i class="fas fa-plus"></i> Create User
              </button>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

<!-- Settings Modal -->
<div id="settingsModal" class="modal">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h3>Settings</h3>
        <button id="closeSettingsModal" class="btn btn-icon">
          <i class="fas fa-times"></i>
        </button>
      </div>
      <div class="modal-body">
        <form id="settingsForm">
          <div class="form-group">
            <label for="settingsTheme">Theme</label>
            <select id="settingsTheme">
              <option value="day">Day</option>
              <option value="night">Night</option>
            </select>
          </div>

          <div class="form-group">
            <label>Default Notifications</label>
            <div class="checkbox-group">
```

```
                <label><input id="defaultEmail" type="checkbox" value="email" /> Email</label>
                <label><input id="defaultNtfy" type="checkbox" value="ntfy" /> NTFY</label>
                <label><input id="defaultTelegram" type="checkbox" value="telegram" />
Telegram</label>
              </div>
            </div>

            <div class="form-group">
              <label for="defaultReminder">Default Reminder Time</label>
              <select id="defaultReminder">
                <option value="5">5 minutes</option>
                <option value="15">15 minutes</option>
                <option value="30">30 minutes</option>
                <option value="60" selected>1 hour</option>
                <option value="1440">1 day</option>
              </select>
            </div>

            <div class="form-group">
              <label for="settingsTimezone">Timezone</label>
              <select id="settingsTimezone">
                <option value="UTC">UTC</option>
                <option value="Europe/Athens">Europe/Athens</option>
                <option value="America/New_York">America/New_York</option>
                <option value="America/Los_Angeles">America/Los_Angeles</option>
              </select>
            </div>
          </form>
        </div>
        <div class="modal-footer">
          <button id="cancelSettingsBtn" class="btn btn-secondary">Cancel</button>
          <button id="saveSettingsBtn" class="btn btn-primary">
            <i class="fas fa-save"></i> Save Settings
          </button>
        </div>
      </div>
    </div>
  </div>

  <!-- App Settings Modal (Admin Only) -->
```

```
<div id="appSettingsModal" class="modal">
  <div class="modal-dialog modal-large">
    <div class="modal-content">
      <div class="modal-header">
        <h3>Application Settings</h3>
        <button id="closeAppSettingsModal" class="btn btn-icon">
          <i class="fas fa-times"></i>
        </button>
      </div>
      <div class="modal-body">
        <form id="appSettingsForm">
          <div class="settings-section">
            <h4><i class="fas fa-bell"></i> NTFY Settings</h4>
            <div class="form-group">
              <label for="ntfyServer">NTFY Server URL</label>
              <input id="ntfyServer" type="url" placeholder="https://ntfy.yourdomain.com" />
            </div>
          </div>

          <div class="settings-section">
            <h4><i class="fas fa-envelope"></i> Email Settings</h4>
            <div class="form-row">
              <div class="form-group">
                <label for="smtpHost">SMTP Host</label>
                <input id="smtpHost" type="text" placeholder="smtp.example.com" />
              </div>
              <div class="form-group">
                <label for="smtpPort">SMTP Port</label>
                <input id="smtpPort" type="number" placeholder="587" />
              </div>
            </div>
            <div class="form-row">
              <div class="form-group">
                <label for="smtpUser">SMTP Username</label>
                <input id="smtpUser" type="text" />
              </div>
              <div class="form-group">
                <label for="smtpPassword">SMTP Password</label>
                <input id="smtpPassword" type="password" />
              </div>
```

```
  </div>
  <div class="form-group">
    <label for="smtpFrom">From Address</label>
    <input id="smtpFrom" type="email" placeholder="noreply@example.com" />
  </div>
  <div class="form-group">
    <label>
      <input id="smtpUseTls" type="checkbox" checked />
      Use TLS/STARTTLS
    </label>
  </div>
</div>

<div class="settings-section">
  <h4><i class="fab fa-telegram"></i> Telegram Settings</h4>
  <div class="form-group">
    <label for="telegramToken">Bot Token</label>
    <input id="telegramToken" type="text" placeholder="123456789:ABCDEF..." />
    <small>Get this from @BotFather on Telegram</small>
  </div>
</div>

<div class="settings-section">
  <h4><i class="fas fa-cog"></i> General Settings</h4>
  <div class="form-row">
    <div class="form-group">
      <label for="appName">Application Name</label>
      <input id="appName" type="text" value="Family Calendar" />
    </div>
    <div class="form-group">
      <label for="maxUsers">Maximum Users</label>
      <input id="maxUsers" type="number" value="10" min="1" max="50" />
    </div>
  </div>
  <div class="form-group">
    <label>
      <input id="registrationEnabled" type="checkbox" />
      Allow user registration
    </label>
  </div>
```

```
            </div>
          </form>
        </div>
        <div class="modal-footer">
          <button id="testNotificationsBtn" class="btn btn-outline">
            <i class="fas fa-paper-plane"></i> Test Notifications
          </button>
          <div class="modal-actions">
            <button id="cancelAppSettingsBtn" class="btn btn-secondary">Cancel</button>
            <button id="saveAppSettingsBtn" class="btn btn-primary">
              <i class="fas fa-save"></i> Save Settings
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>

  <!-- Toast Notifications -->
  <div id="toastContainer" class="toast-container"></div>

  <script src="/static/vendor/fullcalendar/index.global.min.js"></script>
  <script src="/static/app.js"></script>
</body>
</html>
```

## This enhanced version provides:

1. **Improved User Management**: Proper JWT authentication, user creation/editing/deletion, role-based access control
2. **Enhanced Admin Panel**: Full-featured admin interface with user management and system settings
3. **Advanced Notifications**: Robust notification system with logging, multiple channels, and error handling
4. **Professional UI**: Modern interface with modals, responsive design, better forms and navigation
5. **Better Security**: Input validation, proper error handling, session management
6. **Enhanced Features**: Event categories, colors, locations, advanced search, timezone support

The UI now includes professional modals instead of basic prompts, comprehensive admin panels, and a much more polished user experience that rivals commercial calendar applications.

# Ins0mniA

Revision #3
Created 2025-09-03 01:01:56 EEST by Green
Updated 2025-09-04 02:12:34 EEST by Green