API og Database Integrasjonsguide

PM Simulator - Nye Hædda Barneskole

Dato: 14. desember 2025 **Versjon:** 1.0 **Formål:** Detaljert guide for integrasjon av Gemini AI API og Supabase database

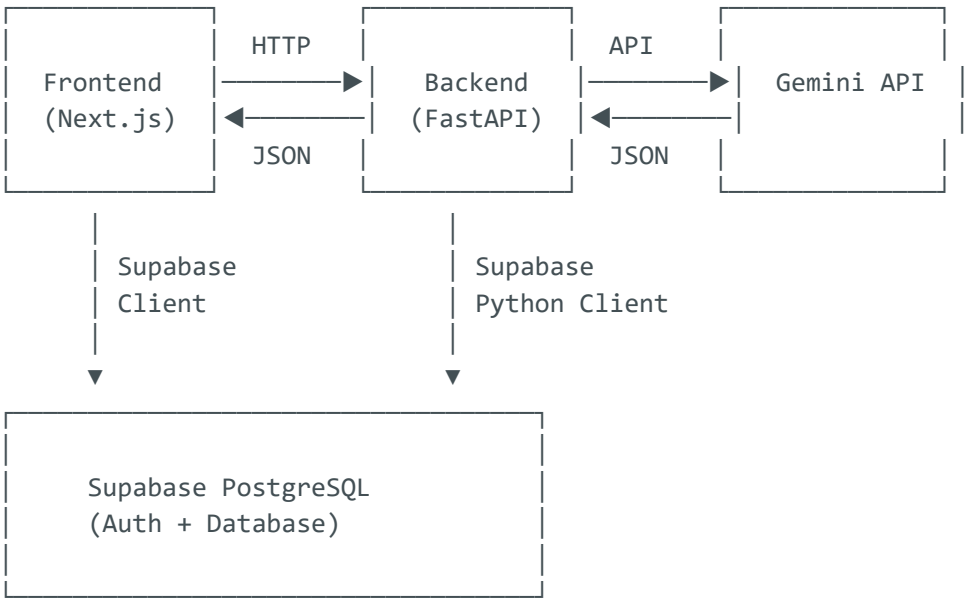
---

📋 Innholdsfortegnelse

---

1. Oversikt over arkitekturen

Dataflyt



Nåværende Status

- ✅ **Frontend:** Next.js 14+ med App Router, chat UI klar
- ✅ **Backend:** FastAPI med Supabase JWT autentisering
- ✅ **Auth:** Komplett JWT-basert autentisering med middleware
- 🔄 **Gemini API:** Konfigurert men ikke integrert i endpoints

- 🔄 **Database:** Supabase PostgreSQL tilkoblet, men tabeller ikke opprettet
- 🔄 **API Endpoints:** Kun `/` og `/me` implementert

---

2. Gemini API Integrasjon

2.1 Forutsetninger

Nødvendige komponenter:

- Google Cloud Project med Gemini API aktivert
- API-nøkkel fra Google AI Studio
- `google-generativeai` Python-pakke (allerede installert i backend)

2.2 Backend Konfigurasjon

Steg 1: Legg til API-nøkkel i miljøvariabler

Fil: `backend/.env.local`

```
# Eksisterende Supabase-config...
SUPABASE_URL=https://cmntglldaqrekloixxoc.supabase.co
SUPABASE_KEY=<din_key>
SUPABASE_JWT_SECRET=<din_secret>

# Legg til Gemini API-nøkkel
GEMINI_API_KEY=AIzaSy...  # Hent fra Google AI Studio
GEMINI_MODEL=gemini-1.5-pro  # Eller gemini-1.5-flash for raskere respons
```

Steg 2: Oppdater config.py for Gemini

Fil: `backend/config.py`

```
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    supabase_url: str
    supabase_key: str
    supabase_jwt_secret: str

    # Legg til Gemini-konfigurasjon
    gemini_api_key: str
```

```
    gemini_model: str = "gemini-1.5-pro"

    # Gemini parametere (valgfritt å overstyre)
    gemini_temperature: float = 0.7
    gemini_max_tokens: int = 2048
    gemini_top_p: float = 0.95

    class Config:
        env_file = ".env.local"

settings = Settings()
```

Steg 3: Opprett Gemini Service

Ny fil: backend/services/gemini_service.py

```
import google.generativeai as genai
from typing import List, Dict, Optional
from config import settings
import logging

logger = logging.getLogger(__name__)

# Konfigurer Gemini API
genai.configure(api_key=settings.gemini_api_key)

class GeminiService:
    """Service for å kommunisere med Gemini AI API"""

    def __init__(self):
        self.model = genai.GenerativeModel(
            model_name=settings.gemini_model,
            generation_config={
                "temperature": settings.gemini_temperature,
                "top_p": settings.gemini_top_p,
                "max_output_tokens": settings.gemini_max_tokens,
            }
        )

    async def chat_with_agent(
        self,
        agent_id: str,
        system_prompt: str,
        conversation_history: List[Dict[str, str]],
        user_message: str,
        game_context: Optional[Dict] = None
    ) -> str:
        """
        Send melding til Gemini med agent-spesifikt system prompt

        Args:
            agent_id: ID til agenten (f.eks. 'ole_hansen')
            system_prompt: System prompt for agenten
            conversation_history: Liste med tidligere meldinger
            user_message: Brukerens nye melding
```

```
            game_context: Spillstatus (budsjett, WBS, etc.)

        Returns:
            Gemini AI respons som tekst
        """
        try:
            # Bygg full prompt med kontekst
            full_prompt = self._build_full_prompt(
                system_prompt=system_prompt,
                conversation_history=conversation_history,
                user_message=user_message,
                game_context=game_context
            )

            # Send til Gemini
            logger.info(f"Sending request to Gemini for agent: {agent_id}")
            response = self.model.generate_content(full_prompt)

            # Logg for telemetri (se .logging/process-api-requests.py)
            logger.info(f"Gemini response received for {agent_id}")

            return response.text

        except Exception as e:
            logger.error(f"Gemini API error for agent {agent_id}: {str(e)}")
            raise

    def _build_full_prompt(
        self,
        system_prompt: str,
        conversation_history: List[Dict[str, str]],
        user_message: str,
        game_context: Optional[Dict] = None
    ) -> str:
        """Bygg komplett prompt med all kontekst"""

        prompt_parts = [
            "# System Prompt (Din rolle)",
            system_prompt,
            ""
        ]

        # Legg til spillkontekst hvis tilgjengelig
        if game_context:
            prompt_parts.extend([
                "# Nåværende spillstatus",
                f"Budsjett brukt: {game_context.get('budget_used', 0)} /
{game_context.get('total_budget', 10000000)} NOK",
                f"WBS områder: {', '.join(game_context.get('wbs_areas', []))}",
                ""
            ])

        # Legg til samtalehistorikk
        if conversation_history:
            prompt_parts.append("# Tidligere samtale")
            for msg in conversation_history[-10:]:  # Siste 10 meldinger
                role = "Bruker" if msg["role"] == "user" else "Deg"
                prompt_parts.append(f"{role}: {msg['content']}")
            prompt_parts.append("")
```

```
        # Legg til ny melding
        prompt_parts.extend([
            "# Ny melding fra prosjektlederen",
            f"Bruker: {user_message}",
            "",
            "Svar naturlig som {agent_navn} basert på din rolle og spillkonteksten
over:"
        ])

        return "\n".join(prompt_parts)

    async def validate_commitment(
        self,
        wbs_id: str,
        supplier_quote: float,
        user_reasoning: str
    ) -> Dict[str, any]:
        """
        Bruk Gemini til å validere om en forpliktelse er fornuftig
        (Valgfri funksjon for ekstra validering)
        """
        prompt = f"""
        Analyser om følgende forpliktelse virker fornuftig:

        WBS område: {wbs_id}
        Tilbudt pris: {supplier_quote} NOK
        Begrunnelse: {user_reasoning}

        Svar med JSON:
        {{
            "is_reasonable": true/false,
            "risk_level": "low/medium/high",
            "feedback": "kort tilbakemelding"
        }}
        """

        response = self.model.generate_content(prompt)
        # Parse JSON fra respons (krever ekstra validering)
        return {"is_reasonable": True, "risk_level": "low", "feedback": "OK"}

# Global instans
gemini_service = GeminiService()
```

Steg 4: Opprett Chat Endpoint

Fil: `backend/main.py` (legg til nytt endpoint)

```
from fastapi import FastAPI, Depends, HTTPException, status
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from typing import List, Dict, Optional
import logging

# Importer Gemini service
```

```python
from services.gemini_service import gemini_service

# ... eksisterende imports og setup ...

# Request/Response models
class ChatMessage(BaseModel):
    role: str  # "user" eller "assistant"
    content: str

class ChatRequest(BaseModel):
    agent_id: str
    message: str
    conversation_history: List[ChatMessage] = []
    game_context: Optional[Dict] = None

class ChatResponse(BaseModel):
    agent_id: str
    response: str
    timestamp: str

# Nytt endpoint for chat
@app.post("/api/chat", response_model=ChatResponse)
async def chat_with_agent(
    request: ChatRequest,
    current_user: dict = Depends(get_current_user)
):
    """
    Send melding til en AI-agent og få respons fra Gemini

    Krever autentisering (JWT token)
    """
    from datetime import datetime

    # Valider agent_id (må matche agents i AI_AGENT_SYSTEM_PROMPTS.md)
    valid_agents = ["ole_hansen", "kari_nilsen", "per_olsen", "lise_berg"]
    if request.agent_id not in valid_agents:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Invalid agent_id. Must be one of: {valid_agents}"
        )

    # Hent system prompt for agenten
    # TODO: Last fra database eller cache fra AI_AGENT_SYSTEM_PROMPTS.md
    system_prompts = {
        "ole_hansen": "Du er Ole Hansen, leder for elektro og varme...",
        "kari_nilsen": "Du er Kari Nilsen, ansvarlig for ventilasjon...",
        "per_olsen": "Du er Per Olsen, byggeleder...",
        "lise_berg": "Du er Lise Berg, arkitekt..."
    }
    system_prompt = system_prompts.get(request.agent_id, "")

    try:
        # Konverter Pydantic modeller til dict
        history = [msg.dict() for msg in request.conversation_history]

        # Send til Gemini
        ai_response = await gemini_service.chat_with_agent(
            agent_id=request.agent_id,
            system_prompt=system_prompt,
```

```
            conversation_history=history,
            user_message=request.message,
            game_context=request.game_context
        )

        # TODO: Lagre samtale til database (negotiation_history tabell)

        return ChatResponse(
            agent_id=request.agent_id,
            response=ai_response,
            timestamp=datetime.utcnow().isoformat()
        )

    except Exception as e:
        logger.error(f"Error in chat endpoint: {str(e)}")
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail="Failed to get AI response"
        )
```

2.3 Frontend Integrasjon med Gemini

Steg 1: Opprett API Client

Ny fil: `frontend/lib/api/chat.ts`

```
import { createBrowserClient } from '@/lib/supabase/client';

export interface ChatMessage {
  role: 'user' | 'assistant';
  content: string;
}

export interface ChatRequest {
  agent_id: string;
  message: string;
  conversation_history: ChatMessage[];
  game_context?: {
    budget_used: number;
    total_budget: number;
    wbs_areas: string[];
  };
}

export interface ChatResponse {
  agent_id: string;
  response: string;
  timestamp: string;
}

/**
 * Send melding til backend chat endpoint
 * Håndterer automatisk JWT token via Supabase
```

```
 */
export async function sendChatMessage(request: ChatRequest): Promise<ChatResponse> {
  const supabase = createBrowserClient();

  // Hent Supabase session token
  const { data: { session } } = await supabase.auth.getSession();

  if (!session?.access_token) {
    throw new Error('Not authenticated');
  }

  // Send til backend med JWT
  const response = await fetch('http://localhost:8000/api/chat', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${session.access_token}`,
    },
    body: JSON.stringify(request),
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.detail || 'Failed to send message');
  }

  return response.json();
}
```

Steg 2: Oppdater Chat Component

Fil: `frontend/components/chat-interface.tsx`

Erstatt den falske `handleSendMessage` funksjonen med:

```
import { sendChatMessage, ChatMessage } from '@/lib/api/chat';
import { useState } from 'react';

// ... inne i ChatInterface component ...

const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState<string | null>(null);

const handleSendMessage = async () => {
  if (!input.trim() || !selectedAgent) return;

  const userMessage: ChatMessage = {
    role: 'user',
    content: input.trim(),
  };

  // Legg til brukermelding i UI
  setMessages((prev) => [...prev, userMessage]);
  setInput('');
  setIsLoading(true);
```

```
    setError(null);

    try {
      // Send til backend
      const response = await sendChatMessage({
        agent_id: selectedAgent,
        message: userMessage.content,
        conversation_history: messages,
        game_context: {
          budget_used: 2500000,
          total_budget: 10000000,
          wbs_areas: ['Elektro', 'Ventilasjon', 'Bygg'],
        },
      });

      // Legg til AI-respons
      setMessages((prev) => [
        ...prev,
        {
          role: 'assistant',
          content: response.response,
        },
      ]);
    } catch (err) {
      console.error('Chat error:', err);
      setError(err instanceof Error ? err.message : 'Kunne ikke sende melding');
    } finally {
      setIsLoading(false);
    }
  };
```

2.4 Logging og Telemetri

Prosjektet har allerede et telemetrisystem for Gemini API:

Fil: `.logging/process-api-requests.py`

Dette scriptet prosesserer logger fra backend for å spore:

- Total antall API-kall
- Token-bruk (input/output)
- Kostnader
- Feilrate

Bruk:

```
# Kjør backend med logging
cd backend
python main.py > ../.logging/api-requests.log 2>&1
```

```
# Prosesser logger
cd .logging
python process-api-requests.py
```

---

3. Database Konfigurasjon

3.1 Supabase Oppsett

Tilkoblingsdetaljer (allerede konfigurert):

```
SUPABASE_URL=https://cmntglldaqrekloixxoc.supabase.co
SUPABASE_KEY=sb_publishable_ChXJKjeBDhzccsllf_4d8A_MoOP3S6l
SUPABASE_JWT_SECRET=<secret_key>
```

3.2 Opprett Database-tabeller

Metode 1: Via Supabase Dashboard (Anbefalt for POC)

1. Gå til https://supabase.com/dashboard
2. Velg prosjektet "cmntglldaqrekloixxoc"
3. Naviger til **SQL Editor**
4. Kjør SQL-scriptene under

Metode 2: Via Supabase CLI

```
# Installer Supabase CLI
npm install -g supabase

# Koble til prosjekt
supabase login
supabase link --project-ref cmntglldaqrekloixxoc

# Kjør migrasjoner (lag fil først)
supabase db push
```

3.3 SQL Migrations

Fil: `database/migrations/001_initial_schema.sql` (opprett denne)

```
-- ============================================================
-- PM Simulator Database Schema
-- Versjon: 1.0
-- Dato: 2025-12-14
-- ============================================================

-- Enable UUID extension
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- ============================================================
-- 1. GAME SESSIONS TABLE
-- Lagrer hoved-spillsesjoner for hver bruker
-- ============================================================
CREATE TABLE public.game_sessions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,

    -- Spillstatus
    status VARCHAR(20) NOT NULL DEFAULT 'in_progress'
        CHECK (status IN ('in_progress', 'completed', 'paused', 'abandoned')),

    -- Budsjett-tracking
    total_budget NUMERIC(12, 2) NOT NULL DEFAULT 10000000.00, -- 10 millioner NOK
    current_budget_used NUMERIC(12, 2) NOT NULL DEFAULT 0.00,
    budget_remaining NUMERIC(12, 2) GENERATED ALWAYS AS (total_budget -
current_budget_used) STORED,

    -- Spilldata (JSON for fleksibilitet)
    game_state JSONB DEFAULT '{}'::jsonb,
    -- Eksempel game_state struktur:
    -- {
    --   "wbs_areas": ["Elektro", "Ventilasjon", "Bygg"],
    --   "current_phase": "negotiation",
    --   "decisions_made": 5,
    --   "active_agent": "ole_hansen"
    -- }

    -- Agent offers (kan også være egen tabell)
    agent_offers JSONB DEFAULT '[]'::jsonb,
    -- Eksempel agent_offers struktur:
    -- [
    --   {
    --     "agent_id": "ole_hansen",
    --     "wbs_id": "WBS-001",
    --     "initial_price": 1500000,
    --     "current_price": 1350000,
    --     "status": "negotiating"
    --   }
    -- ]

    -- Metadata
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    completed_at TIMESTAMP WITH TIME ZONE,
```

```
    -- Performance metrics
    duration_seconds INTEGER GENERATED ALWAYS AS
        (EXTRACT(EPOCH FROM (COALESCE(completed_at, NOW()) - created_at))::INTEGER)
STORED
);

-- Indexes for performance
CREATE INDEX idx_game_sessions_user_id ON public.game_sessions(user_id);
CREATE INDEX idx_game_sessions_status ON public.game_sessions(status);
CREATE INDEX idx_game_sessions_created_at ON public.game_sessions(created_at DESC);

-- Row Level Security (RLS)
ALTER TABLE public.game_sessions ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can view their own sessions"
    ON public.game_sessions FOR SELECT
    USING (auth.uid() = user_id);

CREATE POLICY "Users can create their own sessions"
    ON public.game_sessions FOR INSERT
    WITH CHECK (auth.uid() = user_id);

CREATE POLICY "Users can update their own sessions"
    ON public.game_sessions FOR UPDATE
    USING (auth.uid() = user_id);

-- ================================================
-- 2. WBS COMMITMENTS TABLE
-- Lagrer forpliktelser til leverandører for WBS-områder
-- ================================================
CREATE TABLE public.wbs_commitments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    session_id UUID NOT NULL REFERENCES public.game_sessions(id) ON DELETE CASCADE,

    -- WBS identifikasjon
    wbs_id VARCHAR(50) NOT NULL, -- f.eks. "WBS-001-ELEKTRO"
    wbs_name VARCHAR(200) NOT NULL,
    wbs_category VARCHAR(100), -- "Elektro", "Ventilasjon", etc.

    -- Leverandør/Agent info
    agent_id VARCHAR(50) NOT NULL, -- "ole_hansen", "kari_nilsen", etc.
    agent_name VARCHAR(100) NOT NULL,

    -- Pris-informasjon
    initial_price NUMERIC(12, 2) NOT NULL,
    negotiated_price NUMERIC(12, 2) NOT NULL,
    committed_price NUMERIC(12, 2) NOT NULL,
    savings NUMERIC(12, 2) GENERATED ALWAYS AS (initial_price - committed_price)
STORED,
    savings_percentage NUMERIC(5, 2) GENERATED ALWAYS AS
        (CASE WHEN initial_price > 0
         THEN ((initial_price - committed_price) / initial_price * 100)
         ELSE 0 END) STORED,

    -- Status
    status VARCHAR(20) NOT NULL DEFAULT 'pending'
        CHECK (status IN ('pending', 'committed', 'rejected', 'renegotiating')),

    -- Metadata
```

```
    user_reasoning TEXT, -- Brukerens begrunnelse for forpliktelsen
    ai_validation JSONB, -- Gemini validering (valgfritt)

    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    committed_at TIMESTAMP WITH TIME ZONE,

    -- Constraints
    CONSTRAINT positive_prices CHECK (
        initial_price >= 0 AND
        negotiated_price >= 0 AND
        committed_price >= 0
    )
);

-- Indexes
CREATE INDEX idx_wbs_commitments_session_id ON public.wbs_commitments(session_id);
CREATE INDEX idx_wbs_commitments_agent_id ON public.wbs_commitments(agent_id);
CREATE INDEX idx_wbs_commitments_status ON public.wbs_commitments(status);

-- RLS
ALTER TABLE public.wbs_commitments ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can view commitments for their sessions"
    ON public.wbs_commitments FOR SELECT
    USING (
        EXISTS (
            SELECT 1 FROM public.game_sessions
            WHERE id = wbs_commitments.session_id
            AND user_id = auth.uid()
        )
    );

CREATE POLICY "Users can manage commitments for their sessions"
    ON public.wbs_commitments FOR ALL
    USING (
        EXISTS (
            SELECT 1 FROM public.game_sessions
            WHERE id = wbs_commitments.session_id
            AND user_id = auth.uid()
        )
    );

-- ======================================================
-- 3. NEGOTIATION HISTORY TABLE
-- Lagrer alle chat-meldinger mellom bruker og AI-agenter
-- ======================================================
CREATE TABLE public.negotiation_history (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    session_id UUID NOT NULL REFERENCES public.game_sessions(id) ON DELETE CASCADE,

    -- Agent info
    agent_id VARCHAR(50) NOT NULL,
    agent_name VARCHAR(100) NOT NULL,

    -- Meldings-innhold
    user_message TEXT NOT NULL,
    agent_response TEXT NOT NULL,

    -- Kontekst (valgfritt - for analyse)
```

```
    context_snapshot JSONB DEFAULT '{}'::jsonb,
    -- Eksempel:
    -- {
    --   "budget_used": 2500000,
    --   "wbs_id": "WBS-001",
    --   "current_price": 1350000
    -- }

    -- Metadata
    timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    response_time_ms INTEGER, -- Hvor lang tid Gemini brukte

    -- Sentiment analysis (fremtidig bruk)
    sentiment VARCHAR(20) CHECK (sentiment IN ('positive', 'neutral', 'negative',
NULL))
);

-- Indexes
CREATE INDEX idx_negotiation_history_session_id ON
public.negotiation_history(session_id);
CREATE INDEX idx_negotiation_history_agent_id ON public.negotiation_history(agent_id);
CREATE INDEX idx_negotiation_history_timestamp ON public.negotiation_history(timestamp
DESC);

-- RLS
ALTER TABLE public.negotiation_history ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can view negotiation history for their sessions"
    ON public.negotiation_history FOR SELECT
    USING (
        EXISTS (
            SELECT 1 FROM public.game_sessions
            WHERE id = negotiation_history.session_id
            AND user_id = auth.uid()
        )
    );

CREATE POLICY "Users can create negotiation history for their sessions"
    ON public.negotiation_history FOR INSERT
    WITH CHECK (
        EXISTS (
            SELECT 1 FROM public.game_sessions
            WHERE id = negotiation_history.session_id
            AND user_id = auth.uid()
        )
    );

-- ======================================================
-- 4. USER ANALYTICS TABLE (Fremtidig bruk)
-- Lagrer aggregert brukerdata for leaderboard og progresjon
-- ======================================================
CREATE TABLE public.user_analytics (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,

    -- Statistikk
    total_sessions INTEGER DEFAULT 0,
    completed_sessions INTEGER DEFAULT 0,
    total_budget_saved NUMERIC(12, 2) DEFAULT 0.00,
```

```
    average_savings_percentage NUMERIC(5, 2) DEFAULT 0.00,

    -- Beste prestasjon
    best_session_id UUID REFERENCES public.game_sessions(id) ON DELETE SET NULL,
    best_savings_percentage NUMERIC(5, 2) DEFAULT 0.00,

    -- Tidsbruk
    average_session_duration_seconds INTEGER DEFAULT 0,
    total_play_time_seconds INTEGER DEFAULT 0,

    -- Metadata
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    -- Unique constraint
    CONSTRAINT unique_user_analytics UNIQUE (user_id)
);

-- Index
CREATE INDEX idx_user_analytics_user_id ON public.user_analytics(user_id);

-- RLS
ALTER TABLE public.user_analytics ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can view their own analytics"
    ON public.user_analytics FOR SELECT
    USING (auth.uid() = user_id);

CREATE POLICY "Users can update their own analytics"
    ON public.user_analytics FOR ALL
    USING (auth.uid() = user_id);

-- ====================================================
-- 5. FUNCTIONS & TRIGGERS
-- ====================================================

-- Funksjon: Oppdater updated_at timestamp
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger for game_sessions
CREATE TRIGGER update_game_sessions_updated_at
    BEFORE UPDATE ON public.game_sessions
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();

-- Trigger for user_analytics
CREATE TRIGGER update_user_analytics_updated_at
    BEFORE UPDATE ON public.user_analytics
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();

-- ====================================================
-- 6. INITIAL DATA (Valgfritt)
```

```
-- ====================================================

-- Seed data kan legges til her hvis nødvendig
-- Eksempel: Standard WBS-kategorier, agent-informasjon, etc.

-- ====================================================
-- MIGRATION COMPLETE
-- ====================================================
```

3.4 Kjør Migration

Via Supabase Dashboard:

1. Kopier hele SQL-scriptet over
2. Lim inn i SQL Editor
3. Klikk "Run"

Verifiser at tabellene er opprettet:

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public'
ORDER BY table_name;
```

Du skal se:

- `game_sessions`
- `wbs_commitments`
- `negotiation_history`
- `user_analytics`

---

4. Database Schema og Datafelter

4.1 Fullstendig Oversikt over Datafelter

Tabell: `game_sessions`

Formål: Hovedtabell for spillsesjoner - én rad per spill-session

Felt	Type	Beskrivelse	Eksempel
`id`	UUID	Primærnøkkel	`550e8400-e29b-41d4-a716-446655440000`
`user_id`	UUID	FK til auth.users	`7c9e6679-7425-40de-944b-e07fc1f90ae7`
`status`	VARCHAR(20)	Spillstatus	`in_progress`, `completed`, `paused`
`total_budget`	NUMERIC(12,2)	Total budsjett for prosjektet	`10000000.00` (10M NOK)
`current_budget_used`	NUMERIC(12,2)	Budsjett brukt så langt	`2500000.00`
`budget_remaining`	NUMERIC(12,2)	Auto-kalkulert gjenværende	`7500000.00`
`game_state`	JSONB	Fleksibel spilldata	`{"wbs_areas": ["Elektro"], "phase": "negotiation"}`
`agent_offers`	JSONB	Nåværende tilbud fra agenter	`[{"agent_id": "ole_hansen", "price": 1350000}]`
`created_at`	TIMESTAMP	Når sesjonen ble opprettet	`2025-12-14T10:30:00Z`
`updated_at`	TIMESTAMP	Siste oppdatering	`2025-12-14T11:45:00Z`
`completed_at`	TIMESTAMP	Når spillet ble fullført	`2025-12-14T12:00:00Z`
`duration_seconds`	INTEGER	Auto-kalkulert varighet	`5400` (90 min)

Viktige constraints:

- `user_id` må eksistere i auth.users
- `status` må være én av: `in_progress`, `completed`, `paused`, `abandoned`
- `current_budget_used` <= `total_budget` (valideres i backend)

Tabell: `wbs_commitments`

Formål: Lagrer forpliktelser til leverandører for hvert WBS-område

Felt	Type	Beskrivelse	Eksempel
`id`	UUID	Primærnøkkel	`...`
`session_id`	UUID	FK til game_sessions	`550e8400-...`
`wbs_id`	VARCHAR(50)	WBS identifikator	`WBS-001-ELEKTRO`
`wbs_name`	VARCHAR(200)	Beskrivende navn	`Elektrisk installasjon`
`wbs_category`	VARCHAR(100)	Kategori	`Elektro`, `Ventilasjon`
`agent_id`	VARCHAR(50)	Agent-ID	`ole_hansen`
`agent_name`	VARCHAR(100)	Agent-navn	`Ole Hansen`
`initial_price`	NUMERIC(12,2)	Opprinnelig tilbud	`1500000.00`
`negotiated_price`	NUMERIC(12,2)	Pris etter forhandling	`1350000.00`
`committed_price`	NUMERIC(12,2)	Endelig forpliktet pris	`1350000.00`
`savings`	NUMERIC(12,2)	Auto-kalkulert besparelse	`150000.00`
`savings_percentage`	NUMERIC(5,2)	Besparelse i %	`10.00`
`status`	VARCHAR(20)	Forpliktelsesstatus	`committed`, `pending`
`user_reasoning`	TEXT	Brukerens begrunnelse	"Godt tilbud, innenfor budsjett"
`ai_validation`	JSONB	Gemini validering (valgfritt)	`{"risk_level": "low"}`
`created_at`	TIMESTAMP	Opprettelsestidspunkt	`...`
`committed_at`	TIMESTAMP	Når forpliktelsen ble akseptert	`...`

Viktige constraints:

- Alle priser må være >= 0

- `session_id` må eksistere

Tabell: `negotiation_history`

Formål: Logger alle chat-meldinger mellom bruker og AI-agenter

Felt	Type	Beskrivelse	Eksempel
`id`	UUID	Primærnøkkel	`...`
`session_id`	UUID	FK til game_sessions	`550e8400-...`
`agent_id`	VARCHAR(50)	Hvilken agent	`ole_hansen`
`agent_name`	VARCHAR(100)	Agent-navn	`Ole Hansen`
`user_message`	TEXT	Brukerens melding	"Kan du gå ned til 1.3M?"
`agent_response`	TEXT	AI-respons	"Det er vanskelig, men jeg kan tilby 1.35M"
`context_snapshot`	JSONB	Kontekst på tidspunktet	`{"budget_used": 2500000}`
`timestamp`	TIMESTAMP	Når meldingen ble sendt	`2025-12-14T11:30:00Z`
`response_time_ms`	INTEGER	Gemini responstid	`1200` (1.2 sekunder)
`sentiment`	VARCHAR(20)	Sentiment (fremtidig)	`positive`, `neutral`

Tabell: `user_analytics`

Formål: Aggregert statistikk per bruker (for leaderboard, progresjon)

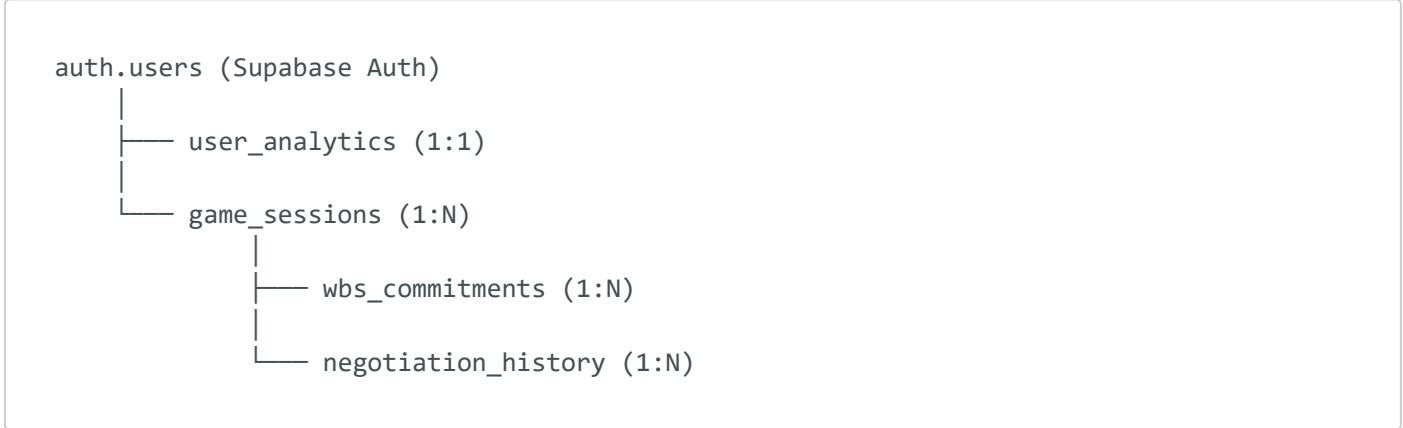
Felt	Type	Beskrivelse	Eksempel
`id`	UUID	Primærnøkkel	`...`
`user_id`	UUID	FK til auth.users (UNIQUE)	`7c9e6679-...`
`total_sessions`	INTEGER	Totalt antall sesjoner	`15`

Felt	Type	Beskrivelse	Eksempel
`completed_sessions`	INTEGER	Fullførte sesjoner	`12`
`total_budget_saved`	NUMERIC(12,2)	Total besparelse	`1800000.00`
`average_savings_percentage`	NUMERIC(5,2)	Gjennomsnittlig besparelse	`12.50` %
`best_session_id`	UUID	Beste sesjon (FK)	`550e8400-...`
`best_savings_percentage`	NUMERIC(5,2)	Beste besparelse	`18.00` %
`average_session_duration_seconds`	INTEGER	Gjennomsnittlig varighet	`3600` (60 min)
`total_play_time_seconds`	INTEGER	Total spilletid	`54000` (15 timer)
`created_at`	TIMESTAMP	Første gang opprettet	`...`
`updated_at`	TIMESTAMP	Siste oppdatering	`...`

Viktige noter:

- Én rad per bruker (enforces av UNIQUE constraint)
- Oppdateres automatisk via trigger/backend når sesjoner fullføres

4.2 Relasjonsskjema



4.3 Eksempel Data Flow

Scenario: Bruker forhandler med Ole Hansen om elektro-arbeid

1. **Start ny sesjon:**

```
// POST /api/sessions
{
  "user_id": "auth_user_123",
  "total_budget": 10000000
}
→ Opprett rad i game_sessions med status="in_progress"
```

2. **Send melding til agent:**

```
// POST /api/chat
{
  "agent_id": "ole_hansen",
  "message": "Hva er ditt beste tilbud for elektro?",
  "conversation_history": []
}
→ Opprett rad i negotiation_history
→ Send til Gemini API
→ Returner respons
```

3. **Forplikt til tilbud:**

```
// POST /api/sessions/{id}/commitments
{
  "wbs_id": "WBS-001-ELEKTRO",
  "agent_id": "ole_hansen",
  "committed_price": 1350000,
  "user_reasoning": "Godt tilbud"
}
→ Opprett rad i wbs_commitments
→ Oppdater game_sessions.current_budget_used += 1350000
```

4. **Fullfør sesjon:**

```
// PUT /api/sessions/{id}
{
  "status": "completed"
}
→ Oppdater game_sessions.status, completed_at
→ Oppdater user_analytics statistikk
```

5. Backend API Endpoints

5.1 Oversikt over Nødvendige Endpoints

Endpoint	Metode	Beskrivelse	Auth
`/api/sessions`	POST	Opprett ny spillsesjon	✅
`/api/sessions/{id}`	GET	Hent sesjondata	✅
`/api/sessions/{id}`	PUT	Oppdater sesjon (status, budget)	✅
`/api/sessions/{id}`	DELETE	Slett sesjon	✅
`/api/sessions/{id}/commitments`	POST	Legg til WBS-forpliktelse	✅
`/api/sessions/{id}/commitments`	GET	Hent alle forpliktelser	✅
`/api/chat`	POST	Send melding til AI-agent	✅
`/api/sessions/{id}/history`	GET	Hent samtalehistorikk	✅
`/api/sessions/{id}/export`	GET	Eksporter sesjondata (JSON/CSV)	✅
`/api/leaderboard`	GET	Hent leaderboard (top 10)	✅
`/api/analytics/me`	GET	Hent brukerens statistikk	✅

5.2 Detaljerte Endpoint-spesifikasjoner

5.2.1 Session Management

POST /api/sessions - Opprett ny sesjon

```python
@app.post("/api/sessions")
async def create_session(
    total_budget: float = 10000000.00,
    current_user: dict = Depends(get_current_user)
):
    """Opprett ny spillsesjon"""
    # Sett inn i database
    data = {
```

```
        "user_id": current_user["id"],
        "total_budget": total_budget,
        "status": "in_progress",
        "game_state": {},
        "agent_offers": []
    }

    response = supabase.table("game_sessions").insert(data).execute()
    return response.data[0]
```

GET /api/sessions/{id} - Hent sesjon

```
@app.get("/api/sessions/{session_id}")
async def get_session(
    session_id: str,
    current_user: dict = Depends(get_current_user)
):
    """Hent spillsesjon med alle detaljer"""
    # Hent sesjon
    session = supabase.table("game_sessions")\
        .select("*, wbs_commitments(*)")\
        .eq("id", session_id)\
        .eq("user_id", current_user["id"])\
        .single()\
        .execute()

    if not session.data:
        raise HTTPException(status_code=404, detail="Session not found")

    return session.data
```

PUT /api/sessions/{id} - Oppdater sesjon

```
class SessionUpdate(BaseModel):
    status: Optional[str] = None
    current_budget_used: Optional[float] = None
    game_state: Optional[dict] = None

@app.put("/api/sessions/{session_id}")
async def update_session(
    session_id: str,
    update: SessionUpdate,
    current_user: dict = Depends(get_current_user)
):
    """Oppdater sesjondata"""
    update_data = update.dict(exclude_none=True)

    # Hvis status endres til "completed", sett completed_at
    if update.status == "completed":
        update_data["completed_at"] = "now()"

    response = supabase.table("game_sessions")\
        .update(update_data)\
```

```
        .eq("id", session_id)\
        .eq("user_id", current_user["id"])\
        .execute()

    # Oppdater user_analytics hvis fullført
    if update.status == "completed":
        # TODO: Oppdater statistikk
        pass

    return response.data[0]
```

5.2.2 WBS Commitments

POST /api/sessions/{id}/commitments - Forplikt til WBS

```
class CommitmentCreate(BaseModel):
    wbs_id: str
    wbs_name: str
    wbs_category: str
    agent_id: str
    agent_name: str
    initial_price: float
    negotiated_price: float
    committed_price: float
    user_reasoning: Optional[str] = None

@app.post("/api/sessions/{session_id}/commitments")
async def create_commitment(
    session_id: str,
    commitment: CommitmentCreate,
    current_user: dict = Depends(get_current_user)
):
    """Opprett WBS-forpliktelse og oppdater budsjett"""

    # Valider at sesjonen tilhører brukeren
    session = supabase.table("game_sessions")\
        .select("current_budget_used, total_budget")\
        .eq("id", session_id)\
        .eq("user_id", current_user["id"])\
        .single()\
        .execute()

    if not session.data:
        raise HTTPException(status_code=404)

    # Sjekk budsjett
    new_total = session.data["current_budget_used"] + commitment.committed_price
    if new_total > session.data["total_budget"]:
        raise HTTPException(
            status_code=400,
            detail=f"Budget exceeded: {new_total} > {session.data['total_budget']}"
        )

    # Opprett forpliktelse
    commitment_data = commitment.dict()
```

```
    commitment_data["session_id"] = session_id
    commitment_data["status"] = "committed"
    commitment_data["committed_at"] = "now()"

    response = supabase.table("wbs_commitments").insert(commitment_data).execute()

    # Oppdater budsjett
    supabase.table("game_sessions")\
        .update({"current_budget_used": new_total})\
        .eq("id", session_id)\
        .execute()

    return response.data[0]
```

GET /api/sessions/{id}/commitments - Hent forpliktelser

```
@app.get("/api/sessions/{session_id}/commitments")
async def get_commitments(
    session_id: str,
    current_user: dict = Depends(get_current_user)
):
    """Hent alle WBS-forpliktelser for sesjonen"""
    response = supabase.table("wbs_commitments")\
        .select("*")\
        .eq("session_id", session_id)\
        .order("created_at", desc=False)\
        .execute()

    return response.data
```

5.2.3 Chat & Negotiation

/api/chat - Se seksjon 2.4 for fullstendig implementasjon

GET /api/sessions/{id}/history - Hent samtalehistorikk

```
@app.get("/api/sessions/{session_id}/history")
async def get_negotiation_history(
    session_id: str,
    agent_id: Optional[str] = None,
    current_user: dict = Depends(get_current_user)
):
    """Hent samtalehistorikk, eventuelt filtrert på agent"""
    query = supabase.table("negotiation_history")\
        .select("*")\
        .eq("session_id", session_id)\
        .order("timestamp", desc=False)

    if agent_id:
        query = query.eq("agent_id", agent_id)
```

```
    response = query.execute()
    return response.data
```

5.2.4 Analytics & Leaderboard

GET /api/analytics/me - Brukerstatistikk

```
@app.get("/api/analytics/me")
async def get_my_analytics(current_user: dict = Depends(get_current_user)):
    """Hent innlogget brukers statistikk"""
    response = supabase.table("user_analytics")\
        .select("*")\
        .eq("user_id", current_user["id"])\
        .single()\
        .execute()

    # Hvis ikke eksisterer, opprett
    if not response.data:
        # Opprett initial record
        pass

    return response.data
```

GET /api/leaderboard - Leaderboard

```
@app.get("/api/leaderboard")
async def get_leaderboard(limit: int = 10):
    """Hent top spillere basert på besparelser"""
    response = supabase.table("user_analytics")\
        .select("*, auth.users(email)")\  # Join for å få brukerinfo
        .order("average_savings_percentage", desc=True)\
        .limit(limit)\
        .execute()

    return response.data
```

5.3 Error Handling

Alle endpoints bør håndtere følgende feil:

```
from fastapi import HTTPException, status

# 401 Unauthorized - Mangler eller ugyldig JWT
if not current_user:
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
```

```
            detail="Not authenticated"
        )

    # 403 Forbidden - Prøver å aksessere andres data
    if session.user_id != current_user["id"]:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Access denied"
        )

    # 404 Not Found - Ressurs finnes ikke
    if not session:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Session not found"
        )

    # 400 Bad Request - Valideringsfeil
    if new_budget > total_budget:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Budget exceeded"
        )

    # 500 Internal Server Error - Database/API feil
    try:
        # ... database operasjon
    except Exception as e:
        logger.error(f"Database error: {str(e)}")
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail="Internal server error"
        )
```

---

6. Frontend Integrasjon

6.1 Supabase Client Setup (Allerede implementert)

Frontend bruker Supabase SSR library for autentisering:

Filer:

- `frontend/lib/supabase/client.ts` - Browser client
- `frontend/lib/supabase/server.ts` - Server client
- `frontend/proxy.ts` - Middleware for session management

Viktig: JWT token håndteres automatisk av Supabase client.

6.2 API Client Library

Ny fil: `frontend/lib/api/sessions.ts`

```
import { createBrowserClient } from '@/lib/supabase/client';

const API_BASE_URL = 'http://localhost:8000';

/**
 * Helper for å lage autentiserte requests
 */
async function fetchWithAuth(endpoint: string, options: RequestInit = {}) {
  const supabase = createBrowserClient();
  const { data: { session } } = await supabase.auth.getSession();

  if (!session?.access_token) {
    throw new Error('Not authenticated');
  }

  const response = await fetch(`${API_BASE_URL}${endpoint}`, {
    ...options,
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${session.access_token}`,
      ...options.headers,
    },
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.detail || 'Request failed');
  }

  return response.json();
}

// ===== SESSION API =====

export interface GameSession {
  id: string;
  user_id: string;
  status: 'in_progress' | 'completed' | 'paused';
  total_budget: number;
  current_budget_used: number;
  budget_remaining: number;
  game_state: Record<string, any>;
  agent_offers: any[];
  created_at: string;
  updated_at: string;
  completed_at?: string;
}

export async function createSession(totalBudget: number = 10000000):
Promise<GameSession> {
  return fetchWithAuth('/api/sessions', {
    method: 'POST',
```

```
    body: JSON.stringify({ total_budget: totalBudget }),
  });
}

export async function getSession(sessionId: string): Promise<GameSession> {
  return fetchWithAuth(`/api/sessions/${sessionId}`);
}

export async function updateSession(
  sessionId: string,
  updates: Partial<GameSession>
): Promise<GameSession> {
  return fetchWithAuth(`/api/sessions/${sessionId}`, {
    method: 'PUT',
    body: JSON.stringify(updates),
  });
}

export async function completeSession(sessionId: string): Promise<GameSession> {
  return updateSession(sessionId, { status: 'completed' });
}

// ===== COMMITMENT API =====

export interface WBSCommitment {
  id: string;
  session_id: string;
  wbs_id: string;
  wbs_name: string;
  agent_id: string;
  agent_name: string;
  initial_price: number;
  negotiated_price: number;
  committed_price: number;
  savings: number;
  savings_percentage: number;
  status: string;
  user_reasoning?: string;
}

export async function createCommitment(
  sessionId: string,
  commitment: Omit<WBSCommitment, 'id' | 'session_id' | 'savings' |
'savings_percentage'>
): Promise<WBSCommitment> {
  return fetchWithAuth(`/api/sessions/${sessionId}/commitments`, {
    method: 'POST',
    body: JSON.stringify(commitment),
  });
}

export async function getCommitments(sessionId: string): Promise<WBSCommitment[]> {
  return fetchWithAuth(`/api/sessions/${sessionId}/commitments`);
}

// ===== NEGOTIATION HISTORY API =====

export interface NegotiationMessage {
  id: string;
```

```
  agent_id: string;
  agent_name: string;
  user_message: string;
  agent_response: string;
  timestamp: string;
}

export async function getNegotiationHistory(
  sessionId: string,
  agentId?: string
): Promise<NegotiationMessage[]> {
  const params = agentId ? `?agent_id=${agentId}` : '';
  return fetchWithAuth(`/api/sessions/${sessionId}/history${params}`);
}

// ===== ANALYTICS API =====

export interface UserAnalytics {
  total_sessions: number;
  completed_sessions: number;
  total_budget_saved: number;
  average_savings_percentage: number;
  best_savings_percentage: number;
  average_session_duration_seconds: number;
}

export async function getMyAnalytics(): Promise<UserAnalytics> {
  return fetchWithAuth('/api/analytics/me');
}

export async function getLeaderboard(limit: number = 10): Promise<UserAnalytics[]> {
  return fetchWithAuth(`/api/leaderboard?limit=${limit}`);
}
```

6.3 Integrasjon i Chat Component

Oppdater: `frontend/components/chat-interface.tsx`

```
'use client';

import { useState, useEffect } from 'react';
import { sendChatMessage, ChatMessage } from '@/lib/api/chat';
import { createSession, getSession, GameSession } from '@/lib/api/sessions';

export function ChatInterface({ agents }: { agents: Agent[] }) {
  const [session, setSession] = useState<GameSession | null>(null);
  const [selectedAgent, setSelectedAgent] = useState<string>('');
  const [messages, setMessages] = useState<ChatMessage[]>([]);
  const [input, setInput] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  // Opprett sesjon ved mount
  useEffect(() => {
    const initSession = async () => {
```

```
      try {
        const newSession = await createSession();
        setSession(newSession);
      } catch (error) {
        console.error('Failed to create session:', error);
      }
    };

    initSession();
  }, []);

  const handleSendMessage = async () => {
    if (!input.trim() || !selectedAgent || !session) return;

    const userMessage: ChatMessage = {
      role: 'user',
      content: input.trim(),
    };

    setMessages((prev) => [...prev, userMessage]);
    setInput('');
    setIsLoading(true);

    try {
      const response = await sendChatMessage({
        agent_id: selectedAgent,
        message: userMessage.content,
        conversation_history: messages,
        game_context: {
          budget_used: session.current_budget_used,
          total_budget: session.total_budget,
          wbs_areas: [], // Hent fra session.game_state
        },
      });

      setMessages((prev) => [
        ...prev,
        {
          role: 'assistant',
          content: response.response,
        },
      ]);

      // Refresh session data
      const updatedSession = await getSession(session.id);
      setSession(updatedSession);

    } catch (error) {
      console.error('Chat error:', error);
      // Vis feilmelding til bruker
    } finally {
      setIsLoading(false);
    }
  };

  // ... resten av UI komponenten
}
```

6.4 Nye Sider for Sesjonsstyring

Ny fil: `frontend/app/game/page.tsx`

```
'use client';

import { useState, useEffect } from 'react';
import { createSession, getCommitments } from '@/lib/api/sessions';
import { ChatInterface } from '@/components/chat-interface';
import { BudgetDisplay } from '@/components/budget-display';
import { CommitmentList } from '@/components/commitment-list';

export default function GamePage() {
  const [sessionId, setSessionId] = useState<string | null>(null);
  const [commitments, setCommitments] = useState([]);

  useEffect(() => {
    const init = async () => {
      const session = await createSession();
      setSessionId(session.id);
    };
    init();
  }, []);

  useEffect(() => {
    if (sessionId) {
      const loadCommitments = async () => {
        const data = await getCommitments(sessionId);
        setCommitments(data);
      };
      loadCommitments();
    }
  }, [sessionId]);

  if (!sessionId) return <div>Loading...</div>;

  return (
    <div className="grid grid-cols-3 gap-4">
      <div className="col-span-2">
        <ChatInterface sessionId={sessionId} />
      </div>
      <div>
        <BudgetDisplay sessionId={sessionId} />
        <CommitmentList commitments={commitments} />
      </div>
    </div>
  );
}
```

---

7. Sikkerhet og Best Practices

7.1 Autentisering og Autorisering

JWT Token Flyt:

1. Bruker logger inn via frontend → Supabase Auth
2. Supabase returnerer JWT access token
3. Frontend lagrer token i httpOnly cookie (via Supabase SSR)
4. Alle requests til backend inkluderer: `Authorization: Bearer <token>`
5. Backend validerer JWT mot Supabase JWKS
6. Backend ekstraher `user_id` fra JWT claims
7. Database queries bruker `user_id` for å enforce RLS policies

RLS (Row Level Security):

Alle tabeller har RLS aktivert:

- Brukere kan KUN se egne sesjoner
- Brukere kan KUN opprette sesjoner for seg selv
- Brukere kan IKKE se andre brukeres data (selv om de får session_id)

7.2 API Rate Limiting

Anbefaling: Legg til rate limiting på Gemini endpoints

```
from slowapi import Limiter
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)

@app.post("/api/chat")
@limiter.limit("10/minute")  # Maks 10 requests per minutt
async def chat_with_agent(...):
    # ...
```

7.3 Input Validering

Alltid valider brukerinput:

```
from pydantic import BaseModel, validator

class ChatRequest(BaseModel):
```

```
    agent_id: str
    message: str

    @validator('message')
    def message_not_empty(cls, v):
        if not v.strip():
            raise ValueError('Message cannot be empty')
        if len(v) > 5000:
            raise ValueError('Message too long (max 5000 chars)')
        return v

    @validator('agent_id')
    def valid_agent(cls, v):
        valid_agents = ["ole_hansen", "kari_nilsen", "per_olsen", "lise_berg"]
        if v not in valid_agents:
            raise ValueError(f'Invalid agent_id: {v}')
        return v
```

7.4 Miljøvariabler

ALDRI commit API-nøkler til git:

```
# .gitignore
.env.local
.env
*.env
```

Bruk environment variables:

```
# Production
GEMINI_API_KEY=<production_key>
SUPABASE_URL=<production_url>

# Development
GEMINI_API_KEY=<dev_key>
SUPABASE_URL=<dev_url>
```

7.5 CORS Konfigurasjon

Backend: Kun tillat frontend domain

```
# Development
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],
```

```
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Production
app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://your-domain.com"],
    allow_credentials=True,
    allow_methods=["GET", "POST", "PUT", "DELETE"],
    allow_headers=["Content-Type", "Authorization"],
)
```

7.6 Logging og Monitoring

Logg ALDRI sensitive data:

```
import logging

logger = logging.getLogger(__name__)

# ALDRI logg JWT tokens, API keys, passord
logger.info(f"User {user_id} sent message to {agent_id}")  # OK
logger.debug(f"Token: {jwt_token}")  # IKKE OK!

# Logg feil med kontekst
try:
    response = gemini_service.chat(...)
except Exception as e:
    logger.error(f"Gemini API error for user {user_id}: {str(e)}")
    # Send til Sentry/monitoring system
```

---

8. Testing og Feilsøking

8.1 Test Database Connection

Script: `backend/test_db.py`

```
from config import settings
from supabase import create_client

supabase = create_client(settings.supabase_url, settings.supabase_key)

# Test 1: List tables
```

```
try:
    response = supabase.table("game_sessions").select("*").limit(1).execute()
    print("✓ Database connection successful")
    print(f"  Tables accessible: game_sessions")
except Exception as e:
    print(f"✗ Database connection failed: {str(e)}")

# Test 2: Create test session
try:
    test_data = {
        "user_id": "test-user-id",  # Bruk en real user_id fra auth.users
        "total_budget": 10000000,
        "status": "in_progress"
    }
    response = supabase.table("game_sessions").insert(test_data).execute()
    print("✓ Insert test successful")
    print(f"  Created session: {response.data[0]['id']}")
except Exception as e:
    print(f"✗ Insert test failed: {str(e)}")
```

Kjør:

```
cd backend
python test_db.py
```

8.2 Test Gemini API

Script: `backend/test_gemini.py`

```
import google.generativeai as genai
from config import settings

genai.configure(api_key=settings.gemini_api_key)
model = genai.GenerativeModel(model_name=settings.gemini_model)

try:
    response = model.generate_content("Hei, svar kort på norsk: hvem er du?")
    print("✓ Gemini API connection successful")
    print(f"  Response: {response.text}")
except Exception as e:
    print(f"✗ Gemini API connection failed: {str(e)}")
```

Kjør:

```
cd backend
python test_gemini.py
```

8.3 Test Full Flow (End-to-End)

Bruk Postman/cURL:

1. **Få JWT token fra Supabase:**

```
# Login via Supabase (eller bruk frontend)
# Kopier access_token fra response
```

2. **Test chat endpoint:**

```
curl -X POST http://localhost:8000/api/chat \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <your_jwt_token>" \
  -d '{
    "agent_id": "ole_hansen",
    "message": "Hva er ditt beste tilbud?",
    "conversation_history": []
  }'
```

3. **Test session creation:**

```
curl -X POST http://localhost:8000/api/sessions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <your_jwt_token>" \
  -d '{
    "total_budget": 10000000
  }'
```

8.4 Common Issues

Problem: `401 Unauthorized` når du kaller backend

Løsning:

- Sjekk at JWT token er gyldig (ikke utløpt)
- Sjekk at token inkluderes i Authorization header
- Sjekk at `SUPABASE_JWT_SECRET` er korrekt i backend `.env.local`

Problem: `CORS error` i browser console

Løsning:

- Sjekk at backend kjører på port 8000
- Sjekk at CORS middleware inkluderer `http://localhost:3000`
- Sjekk at `allow_credentials=True` er satt

Problem: Gemini API returnerer feil

Løsning:

- Sjekk at `GEMINI_API_KEY` er gyldig
- Sjekk at API-nøkkelen har tilgang til Gemini modellen
- Sjekk kvote/rate limits i Google Cloud Console

Problem: Database RLS blokkerer queries

Løsning:

- Sjekk at JWT token inneholder korrekt `user_id`
- Sjekk at RLS policies tillater operasjonen
- Test uten RLS (midlertidig) for å isolere problemet:

```
ALTER TABLE game_sessions DISABLE ROW LEVEL SECURITY;
```

8.5 Logging for Debugging

Backend:

```
import logging

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

# I endpoints:
logger.debug(f"Received request: {request}")
logger.info(f"User {user_id} created session {session_id}")
logger.error(f"Database error: {str(e)}")
```

Frontend:

```
// I browser console
console.log('Session created:', session);
console.error('API error:', error);
```

9. Deployment Checklist

Når POC skal deployes:

Backend (f.eks. Railway, Render, Fly.io)

- ☐ Sett alle miljøvariabler i produksjon
- ☐ Oppdater CORS til production domain
- ☐ Aktiver HTTPS (required for Supabase JWT)
- ☐ Sett opp logging/monitoring (Sentry)
- ☐ Test alle endpoints med production database
- ☐ Dokumenter API med Swagger/OpenAPI

Frontend (f.eks. Vercel, Netlify)

- ☐ Oppdater `API_BASE_URL` til production backend
- ☐ Konfigurer environment variables
- ☐ Test autentiseringsflyt
- ☐ Test alle sider fungerer med production API
- ☐ Optimaliser build (minify, tree-shaking)

Database (Supabase)

- ☐ Verifiser at alle tabeller er opprettet
- ☐ Test RLS policies
- ☐ Sett opp backups
- ☐ Konfigurer database limits (connections, storage)
- ☐ Overvåk query performance

---

10. Neste Steg

Umiddelbare oppgaver:

1. **Opprett database-tabeller** (Seksjon 3.3)

- Kjør SQL migration i Supabase Dashboard
- Verifiser at tabellene er opprettet

2. **Implementer Gemini Service** (Seksjon 2.2)
   - Legg til `gemini_service.py` i backend
   - Oppdater `config.py` med Gemini-variabler
3. **Implementer Chat Endpoint** (Seksjon 2.2)
   - Legg til `/api/chat` i `main.py`
   - Test med Postman/cURL
4. **Integrer frontend med backend** (Seksjon 6)
   - Opprett `lib/api/chat.ts` og `lib/api/sessions.ts`
   - Oppdater `chat-interface.tsx` til å bruke real API
5. **Implementer Session Management Endpoints** (Seksjon 5)
   - POST/GET/PUT `/api/sessions`
   - POST/GET `/api/sessions/{id}/commitments`
6. **Test full flow**
   - Opprett bruker → login → start sesjon → chat → commit → fullfør
   - Verifiser data i database

Fremtidige forbedringer:

- Implementer leaderboard UI
- Legg til analytics dashboard
- Implementer export-funksjonalitet (JSON/CSV)
- Legge til mer avansert Gemini validering
- Implementere WebSocket for real-time updates
- A/B testing av forskjellige prompts

Vedlegg A: Miljøvariabel Mal

Backend `.env.local`:

```
# Supabase
SUPABASE_URL=https://cmntglldaqrekloixxoc.supabase.co
SUPABASE_KEY=<anon_key>
SUPABASE_JWT_SECRET=<jwt_secret>

# Gemini AI
GEMINI_API_KEY=AIzaSy...
GEMINI_MODEL=gemini-1.5-pro
GEMINI_TEMPERATURE=0.7
GEMINI_MAX_TOKENS=2048

# App Config
ENVIRONMENT=development
LOG_LEVEL=INFO
```

Frontend `.env.local`:

```
# Supabase
NEXT_PUBLIC_SUPABASE_URL=https://cmntglldaqrekloixxoc.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=<anon_key>

# Backend API
NEXT_PUBLIC_API_URL=http://localhost:8000

# App Config
NEXT_PUBLIC_APP_NAME=PM Simulator
```

Vedlegg B: Quick Reference Commands

```
# Backend
cd backend
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
pip install -r requirements.txt
python main.py

# Frontend
cd frontend
npm install
npm run dev

# Database
# Via Supabase Dashboard → SQL Editor
# Eller via CLI:
supabase db push

# Testing
cd backend
```

```
python test_db.py
python test_gemini.py

# Logging
cd .logging
python process-api-requests.py
```

Kontakt for spørsmål: [Ditt navn/team] **Sist oppdatert:** 14. desember 2025