# stylgen_v0 Architecture Diagrams

## 1. High-Level System Overview

```
graph TB
    Client[Client/Frontend]

    subgraph "FastAPI Application"
        API[API Routes<br/>main.py]

        subgraph "Core Components"
            Pipeline[Pipeline<br/>Orchestrator]
            Embedder[Embedder<br/>Text→Vector]
            LLM[LLM Provider<br/>Ollama/Dummy]
            VStore[Vector Store<br/>Per-user Index]
            Persona[Persona Builder]
        end

        subgraph "Storage Layer"
            Memory[Memory Store<br/>In-memory]
        end
    end

    subgraph "External Services"
        Ollama[Ollama Server<br/>localhost:11434]
    end

    Client -->|HTTP/SSE| API
    API --> Pipeline
    Pipeline --> Embedder
    Pipeline --> LLM
    Pipeline --> VStore
    API --> Memory
    Persona --> Embedder
    LLM -->|API calls| Ollama
```

## 2. Persona Creation Flow

```
sequenceDiagram
    participant C as Client
    participant A as API
    participant P as Persona Builder
    participant E as Embedder
    participant V as Vector Store
    participant M as Memory Store

    C->>A: POST /persona<br/>{user_id, samples[], preferences}
    A->>E: embed(samples)
    E-->>A: embeddings[]

    A->>V: replace(user_id, vec_items[])
    Note over V: Clears old samples<br/>Stores new ones

    A->>P: build_persona(samples, preferences)
    P->>E: embed(samples)
    E-->>P: embeddings[]
    P-->>P: Calculate centroid<br/>Select exemplar_ids
    P-->>A: PersonaCard

    A->>M: store.personas[user_id] = PersonaCard
    A->>M: store.samples[user_id] = WritingSample[]

    A-->>C: PersonaCreateResponse<br/>{user_id, num_samples, persona}
```

## 3. Generation Pipeline (Detailed)

```
flowchart TB
    Start([Generation Request])

    subgraph "1. Retrieval Phase"
        GetPersona[Load PersonaCard<br/>from Memory]
        CheckExemplars{Has<br/>exemplar_ids?}
        UseExemplars[Fetch exemplars<br/>by ID]
        QueryVectors[Query vectors with<br/>centroid embedding]
        GetSamples[Get top-k similar<br/>samples via cosine]
    end

    subgraph "2. Prompt Construction"
        BuildSystem[Build System Prompt<br/>- Tone descriptors<br/>- Formality
level<br/>- Emoji/hashtag rules<br/>- Banned phrases<br/>- Structure preference]
        BuildUser[Build User Prompt<br/>- Brief keywords<br/>- Goal & audience<br/>-
Length hint<br/>- CTA & link<br/>- Exemplar snippets]
    end

    subgraph "3. LLM Generation"
        SetTemp{Custom<br/>temperature?}
        UseCustom[Use provided<br/>temperature]
        Alternate[Alternate between<br/>0.6 and 0.8]
        CallLLM[Call LLM Provider<br/>with prompts]
        HandleError{LLM<br/>available?}
        UseDummy[Use Dummy<br/>Provider]
    end

    subgraph "4. Critique & Polish"
        RemoveBanned[Remove banned phrases<br/>via regex]
        NormalizeWS[Normalize whitespace<br/>preserve line breaks]
        CheckLength[Check length<br/>60-140% of hint]
    end
    subgraph "5. Scoring"
        StyleSim[Calculate style similarity<br/>cosine(text, centroid)]
        Novelty[Calculate novelty<br/>1 - max_sim(text, samples)]
        Structure[Check structure<br/>requirements]
        CompositeScore[Composite scoring<br/>style + novelty + constraints]
    end

    subgraph "6. Selection"
        SortVariants[Sort variants<br/>by score]
        SelectBest[Select top as<br/>'chosen']
        Package[Package response<br/>with generation_id]
    end

    Start --> GetPersona
    GetPersona --> CheckExemplars
    CheckExemplars -->|Yes| UseExemplars
    CheckExemplars -->|No| QueryVectors
    QueryVectors --> GetSamples
    UseExemplars --> BuildSystem
    GetSamples --> BuildSystem

    BuildSystem --> BuildUser
    BuildUser --> SetTemp
    SetTemp -->|Yes| UseCustom
    SetTemp -->|No| Alternate
    UseCustom --> CallLLM
    Alternate --> CallLLM
    CallLLM --> HandleError
    HandleError -->|Success| RemoveBanned
    HandleError -->|Fail| UseDummy
    UseDummy --> RemoveBanned

    RemoveBanned --> NormalizeWS
    NormalizeWS --> CheckLength
    CheckLength --> StyleSim
    StyleSim --> Novelty
    Novelty --> Structure
    Structure --> CompositeScore
    CompositeScore --> SortVariants
    SortVariants --> SelectBest
    SelectBest --> Package
    Package --> End([Return Response])
```

## 4. Streaming Generation Flow

```
sequenceDiagram
    participant C as Client
    participant A as API
    participant P as Pipeline
    participant L as LLM Provider
    participant O as Ollama

    C->>A: POST /generate/stream
    A->>P: prepare_generation_context()
    P-->>A: exemplars, system_prompt

    A->>C: SSE: event: meta<br/>data: {exemplars, goal, keywords}

    A->>L: stream_generate()
    L->>O: POST /api/generate<br/>{stream: true}

    loop Token Streaming
        O-->>L: {response: "token"}
        L-->>A: yield "token"
        A->>C: SSE: data: token
    end

    O-->>L: {done: true, eval_count, eval_duration}
    L-->>A: Log throughput metrics
    A->>C: SSE: event: done<br/>data: end
```

## 5. Data Model Relationships

```
erDiagram
    USER ||--|| PERSONA-CARD : has
    USER ||--o{ WRITING-SAMPLE : owns
    USER ||--o{ GENERATION-RECORD : creates
    PERSONA-CARD ||--o{ EXEMPLAR-ID : references
    GENERATION-RECORD ||--o{ GENERATION-VARIANT : contains

    PERSONA-CARD {
        string user_id PK
        preferences object
        list exemplar_ids
        list centroid
    }

    WRITING-SAMPLE {
        string id PK
        string user_id FK
        string text
        list embedding
    }

    GENERATION-RECORD {
        string generation_id PK
        string user_id FK
        timestamp created_at
        brief object
        list variants
    }

    GENERATION-VARIANT {
        string text
        score object
        float style_similarity
        float novelty
        bool structure_ok
        bool length_ok
    }
```

## 6. Embedding & Retrieval Flow

```
flowchart LR
    subgraph "Text Processing"
        Input[Raw Text]
        Token[Tokenize<br/>lowercase, split]
        Hash[Hash tokens<br/>BLAKE2b]
        Vector[Build sparse vector<br/>dim=384]
        Norm[L2 Normalize]
    end

    subgraph "Storage"
        VecItem[VecItem<br/>{id, text, vec}]
        UserIndex[(Per-user Index)]
    end

    subgraph "Retrieval"
        Query[Query Vector]
        Cosine[Cosine Similarity<br/>dot(a,b) / ||a||·||b||]
        TopK[Select top-k<br/>by similarity]
    end

    Input --> Token
    Token --> Hash
    Hash --> Vector
    Vector --> Norm
    Norm --> VecItem
    VecItem --> UserIndex

    Query --> Cosine
    UserIndex --> Cosine
    Cosine --> TopK
```

## 7. Component Dependencies

```
graph TD
    subgraph "Entry Points"
        Main[main.py]
    end

    subgraph "Request/Response Models"
        Schemas[schemas.py]
    end

    subgraph "Core Logic"
        Pipeline[pipeline.py]
        LLM[llm.py]
        Embeddings[embeddings.py]
        VectorStore[vector_store.py]
        Persona[persona.py]
    end

    subgraph "Storage"
        Memory[memory.py]
    end

    subgraph "External"
        Ollama[Ollama API]
        ST[sentence-transformers<br/>optional]
    end

    Main --> Schemas
    Main --> Pipeline
    Main --> Memory
    Pipeline --> LLM
    Pipeline --> Embeddings
    Pipeline --> VectorStore
    Pipeline --> Persona
    Persona --> Embeddings
    LLM --> Ollama
    Embeddings -.-> ST

    style ST stroke-dasharray: 5 5
```

## 8. Critique & Scoring Detail

```
flowchart TB
    Text[Generated Text]

    subgraph "Critique Phase"
        BannedList[Global Banned +<br/>User Taboo Phrases]
        Regex[Word-boundary<br/>case-insensitive regex]
        Remove[Remove all matches]
        Normalize[Normalize whitespace<br/>s+→s, \\t→space]
    end

    subgraph "Length Check"
        CalcLen[Calculate length]
        Compare[Compare to hint]
        InRange{60-140%<br/>of hint?}
        LengthOK[length_ok = true]
        LengthFail[length_ok = false]
    end

    subgraph "Style Scoring"
        EmbedText[Embed text]
        GetCentroid[Get persona<br/>centroid]
        CosineSim[cosine(text_emb,<br/>centroid)]
        StyleScore[style_similarity<br/>0.0-1.0]
    end

    subgraph "Novelty Scoring"
        AllSamples[Get all user<br/>samples]
        MaxSim[Find max similarity<br/>to any sample]
        NoveltyCalc[novelty =<br/>1 - max_sim]
        NoveltyScore[novelty<br/>0.0-1.0]
    end

    subgraph "Final Score"
        Composite[weighted_score =<br/>style * 0.5 +<br/>novelty * 0.3 +
<br/>constraints * 0.2]
    end

    Text --> BannedList
    BannedList --> Regex
    Regex --> Remove
    Remove --> Normalize

    Normalize --> CalcLen
    CalcLen --> Compare
    Compare --> InRange
    InRange -->|Yes| LengthOK
    InRange -->|No| LengthFail

    Normalize --> EmbedText
    EmbedText --> CosineSim
    GetCentroid --> CosineSim
    CosineSim --> StyleScore

    EmbedText --> MaxSim
    AllSamples --> MaxSim
    MaxSim --> NoveltyCalc
    NoveltyCalc --> NoveltyScore

    StyleScore --> Composite
    NoveltyScore --> Composite
    LengthOK --> Composite
    LengthFail --> Composite
```

## 9. API Request Flow

```
stateDiagram-v2
    [*] --> HealthCheck: GET /health
    [*] --> CreatePersona: POST /persona
    [*] --> Generate: POST /generate
    [*] --> Stream: POST /generate/stream
    [*] --> Feedback: POST /feedback

    CreatePersona --> PersonaStored
    PersonaStored --> Generate
    PersonaStored --> Stream

    Generate --> VariantsCreated
    Stream --> TokensStreamed

    VariantsCreated --> Feedback
    Feedback --> FeedbackStored

    HealthCheck --> [*]
    FeedbackStored --> [*]
    VariantsCreated --> [*]
    TokensStreamed --> [*]

    note right of Generate
        Requires persona
        Returns scored variants
    end note

    note right of Stream
        SSE format
        No critique/scoring
    end note
```

## Usage Notes

These diagrams show:

1. **System Overview**: How components connect
2. **Persona Creation**: Step-by-step user profile building
3. **Generation Pipeline**: Detailed 6-phase generation process
4. **Streaming**: Real-time token generation flow
5. **Data Models**: Entity relationships
6. **Embedding/Retrieval**: Vector search mechanics
7. **Dependencies**: Module interconnections
8. **Critique/Scoring**: Post-processing detail

9. **API Flow**: Request state transitions

Each diagram focuses on a specific aspect to help understand the granular workflow.