Technical Implementation Guide

AI-Powered Personal Productivity System

3.1 Local LLM Integration

Model Selection Criteria

Model	Size	RAM Required	Task Parsing	Speed	Quality
Mistral-7B-Instruct	4.1GB	8GB	Excellent	Fast	High
Llama-2-7B	3.8GB	8GB	Good	Fast	Medium
Phi-2	1.7GB	4GB	Good	Very Fast	Medium
TinyLlama-1.1B	638MB	2GB	Basic	Instant	Low

Prompt Engineering Specifications

yaml

```yaml
Task_Parsing_Prompt:
  system: |
    You are a task parsing assistant. Extract structured data from natural language.
    Output valid JSON only. No explanations.

  template: |
    Parse this task: "{input}"

    Extract:
    - title: main task description
    - due_date: ISO 8601 format or null
    - priority: high/medium/low
    - tags: relevant categories
    - recurrence: pattern if mentioned
    - duration: estimated time in minutes

    Output JSON:

Priority_Scoring_Prompt:
  system: |
    Score task priority from 0-100 based on:
    - Urgency (due date proximity)
    - Importance (impact/consequences)
    - Dependencies (blocking other tasks)
    - User patterns (historical behavior)

  variables:
    - current_date
    - user_preferences
    - task_history
    - dependency_graph

Schedule_Optimization_Prompt:
  system: |
    Optimize daily schedule considering:
    - Task priorities and deadlines
    - Estimated durations
    - Energy levels (morning/afternoon)
    - Context switching costs
    - Buffer time requirements
```

Performance Optimization

```javascript
// Web Worker Configuration for LLM
{
  "worker": {
    "memory": "2GB",
    "threads": 4,
    "cache": "aggressive",
    "quantization": "int8",
    "batchSize": 1,
    "contextLength": 2048
  },
  "caching": {
    "embeddings": true,
    "commonPrompts": true,
    "ttl": 3600
  },
  "fallback": {
    "timeout": 5000,
    "retries": 2,
    "degradedMode": true
  }
}
```

3.2 Claude Connector Specification

Integration Protocol

```typescript
```

```
interface ClaudeConnectorAPI {
  // Export Methods
  exportState(): SystemState;
  exportTasks(filter?: TaskFilter): Task[];
  exportSchedule(range: DateRange): ScheduleEntry[];
  exportAnalytics(): AnalyticsData;

  // Import Methods
  importCommands(commands: ClaudeCommand[]): ExecutionResult[];
  importTasks(tasks: Task[]): ImportResult;
  importAutomation(rules: AutomationRule[]): void;

  // Real-time Bridge
  enableBridge(): void;
  disableBridge(): void;
  executeCommand(command: string): any;

  // Formatting
  formatForClaude(data: any): string;
  parseFromClaude(input: string): any;
}
```

Command Language Specification

```bnf
command ::= action target [conditions] [options]

action ::= CREATE | UPDATE | DELETE | SCHEDULE | ANALYZE | OPTIMIZE

target ::= TASK | PROJECT | AUTOMATION | SCHEDULE | ALL

conditions ::= WHERE field operator value [AND|OR conditions]

options ::= WITH { key: value, ... }

Examples:
- CREATE TASK "Review documentation" WITH {priority: "high", due: "tomorrow"}
- UPDATE TASK WHERE status="pending" WITH {tag: "urgent"}
- SCHEDULE ALL WHERE priority>7 WITH {method: "time_blocking"}
- ANALYZE TASKS WHERE created_date>"2024-01-01"
```

Data Exchange Format

```json
{
  "version": "1.0",
  "timestamp": "2024-01-15T10:00:00Z",
  "context": {
    "user_timezone": "America/New_York",
    "current_view": "weekly",
    "active_filters": ["incomplete", "this_week"]
  },
  "data": {
    "tasks": [...],
    "schedule": [...],
    "patterns": {...}
  },
  "metrics": {
    "completion_rate": 0.75,
    "avg_task_duration": 45,
    "overdue_count": 3
  },
  "suggestions": {
    "from_ai": [...],
    "from_patterns": [...]
  }
}
```

3.3 Sync Engine Specification

Conflict Resolution Algorithm

```python
```

```
# CRDT-based Merge Strategy Pseudocode

def merge_conflicts(local, remote, base):
  """Three-way merge with CRDT principles"""

  # 1. Last-Write-Wins for simple fields
  if local.updated_at > remote.updated_at:
    result.title = local.title
  else:
    result.title = remote.title

  # 2. Union for collections
  result.tags = unique(local.tags + remote.tags)

  # 3. Max for counters
  result.version = max(local.version, remote.version) + 1

  # 4. Custom merge for complex fields
  result.content = merge_rich_text(
    local.content,
    remote.content,
    base.content
  )

  # 5. Preserve both for conflicts
  if has_semantic_conflict(local, remote):
    result.conflicts = {
      'local': local,
      'remote': remote,
      'resolved': False
    }

  return result
```

Sync State Machine

```
States:
- IDLE: No sync needed
- QUEUED: Changes pending sync
- SYNCING: Active synchronization
- CONFLICT: Manual resolution required
- ERROR: Sync failed, retry pending
```

```
Transitions:
IDLE → QUEUED: Local change detected
QUEUED → SYNCING: Network available & sync triggered
SYNCING → IDLE: Sync successful
SYNCING → CONFLICT: Merge conflict detected
SYNCING → ERROR: Network/server error
CONFLICT → SYNCING: Conflict resolved
ERROR → QUEUED: Retry scheduled
```

3.4 API Implementation

REST API Endpoints

yaml

```
# Task Operations
GET /api/tasks
  query:
    - status: pending|completed|archived
    - priority: 0-10
    - due_before: ISO8601
    - due_after: ISO8601
    - tags: comma-separated
    - search: string
    - limit: number
    - offset: number
  response: Task[]

POST /api/tasks
  body: CreateTaskDTO
  response: Task

PUT /api/tasks/:id
  body: UpdateTaskDTO
  response: Task

DELETE /api/tasks/:id
  response: { success: boolean }

# Bulk Operations
POST /api/tasks/bulk
  body: {
    operation: create|update|delete,
    tasks: Task[]
  }
  response: BulkResult

# AI Operations
POST /api/ai/parse
  body: { text: string }
  response: ParsedTask

POST /api/ai/schedule
  body: { tasks: Task[] }
  response: Schedule

# Sync Operations
POST /api/sync/push
```

```
  body: {
    device_id: string,
    changes: Delta[]
  }
  response: SyncResult

GET /api/sync/pull
  query:
    - device_id: string
    - since: timestamp
  response: Delta[]
```

WebSocket Events

```typescript
// Client → Server Events
interface ClientEvents {
  'task:create': { task: Task };
  'task:update': { id: string; changes: Partial<Task> };
  'task:delete': { id: string };
  'sync:request': { device_id: string };
  'presence:update': { status: 'active' | 'idle' };
}

// Server → Client Events
interface ServerEvents {
  'task:created': { task: Task; device_id: string };
  'task:updated': { id: string; changes: Partial<Task>; device_id: string };
  'task:deleted': { id: string; device_id: string };
  'sync:delta': { changes: Delta[] };
  'presence:others': { devices: Device[] };
}
```

3.5 Security Implementation

Encryption Implementation

```javascript
```

```
// AES-256-GCM Encryption Implementation
class EncryptionService {
  async generateKey(password, salt) {
    const encoder = new TextEncoder();
    const keyMaterial = await crypto.subtle.importKey(
      'raw',
      encoder.encode(password),
      { name: 'PBKDF2' },
      false,
      ['deriveBits', 'deriveKey']
    );

    return crypto.subtle.deriveKey(
      {
        name: 'PBKDF2',
        salt: encoder.encode(salt),
        iterations: 1750000,
        hash: 'SHA-256'
      },
      keyMaterial,
      { name: 'AES-GCM', length: 256 },
      true,
      ['encrypt', 'decrypt']
    );
  }

  async encrypt(data, key) {
    const iv = crypto.getRandomValues(new Uint8Array(16));
    const encrypted = await crypto.subtle.encrypt(
      { name: 'AES-GCM', iv },
      key,
      new TextEncoder().encode(JSON.stringify(data))
    );

    return {
      iv: Array.from(iv),
      data: Array.from(new Uint8Array(encrypted))
    };
  }

  async decrypt(encryptedData, key) {
    const decrypted = await crypto.subtle.decrypt(
      { name: 'AES-GCM', iv: new Uint8Array(encryptedData.iv) },
      key,
      new Uint8Array(encryptedData.data)
    );

    return JSON.parse(new TextDecoder().decode(decrypted));
  }
}
```

Authentication Flow

```javascript
```

```
// JWT Token Management
class AuthService {
  constructor() {
    this.accessToken = null;
    this.refreshToken = null;
    this.tokenRefreshTimer = null;
  }

  async login(email, password) {
    const response = await fetch('/api/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password })
    });

    const { access_token, refresh_token, expires_in } = await response.json();

    this.accessToken = access_token;
    this.refreshToken = refresh_token;
    this.scheduleTokenRefresh(expires_in);

    return { success: true };
  }

  scheduleTokenRefresh(expiresIn) {
    // Refresh 5 minutes before expiry
    const refreshTime = (expiresIn - 300) * 1000;

    this.tokenRefreshTimer = setTimeout(() => {
      this.refreshAccessToken();
    }, refreshTime);
  }

  async refreshAccessToken() {
    const response = await fetch('/api/auth/refresh', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ refresh_token: this.refreshToken })
    });

    const { access_token, expires_in } = await response.json();

    this.accessToken = access_token;
    this.scheduleTokenRefresh(expires_in);
  }
}
```

Document Version: 1.0.0

Last Updated: January 2024

Next Review: February 2024