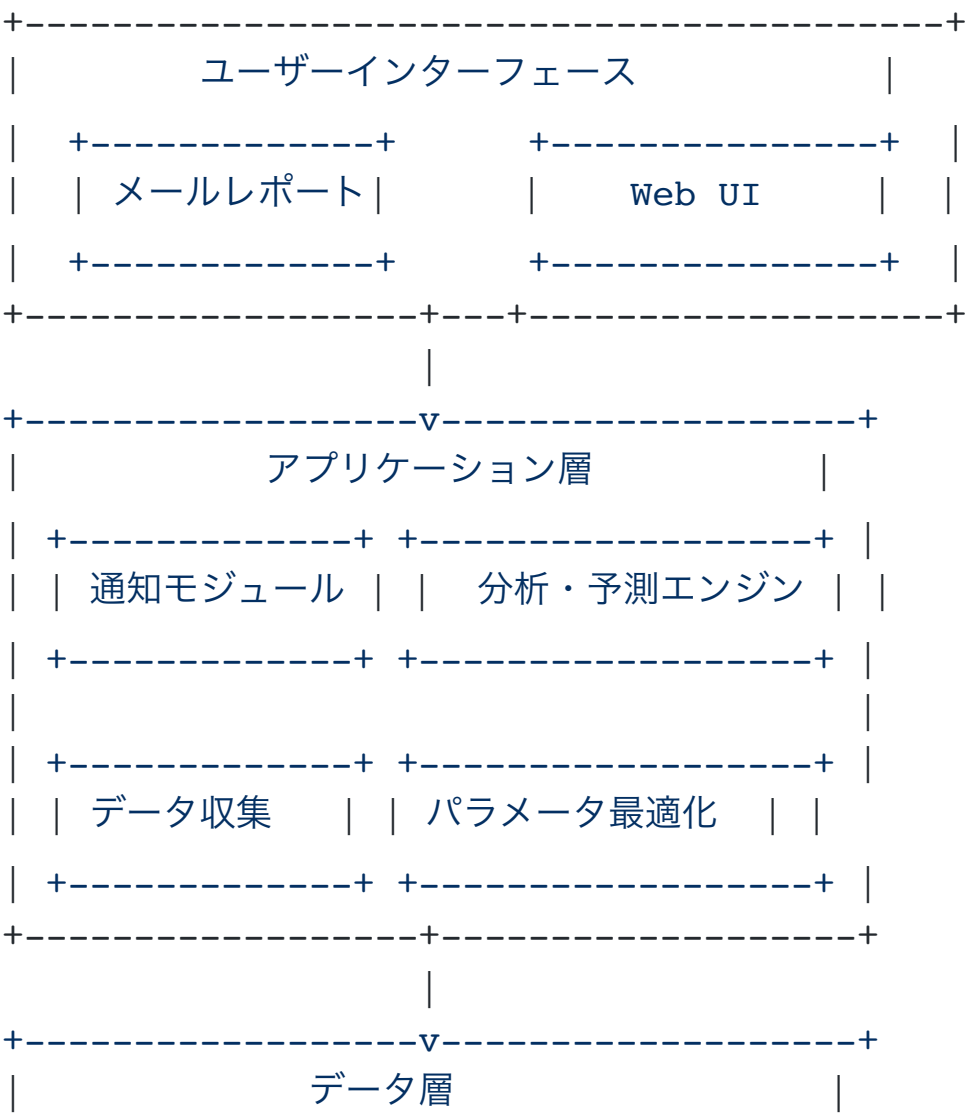
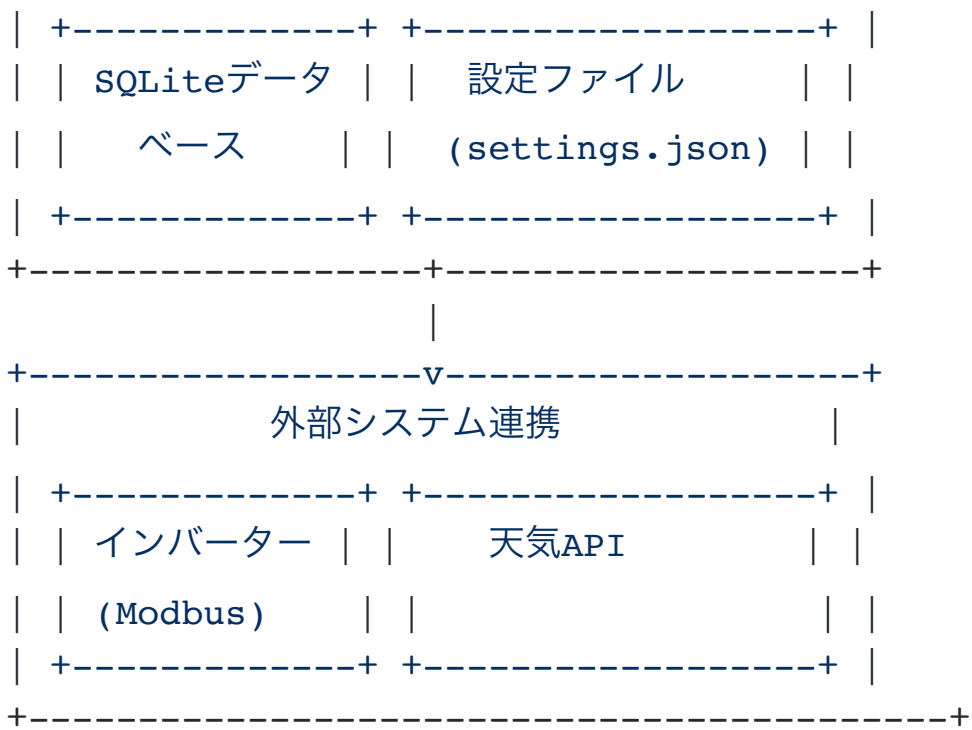
HANAZONOシステム自動最適化プロジェクト - 補足資料パック

本資料は「HANAZONOシステム自動最適化プロジェクト - 統合ロードマップ+」を補完するための追加情報です。主要な資料だけでもプロジェクトの全体像はほぼ把握できますが、ここではより詳細な実装例やアーキテクチャ図、環境構築手順などを提供します。

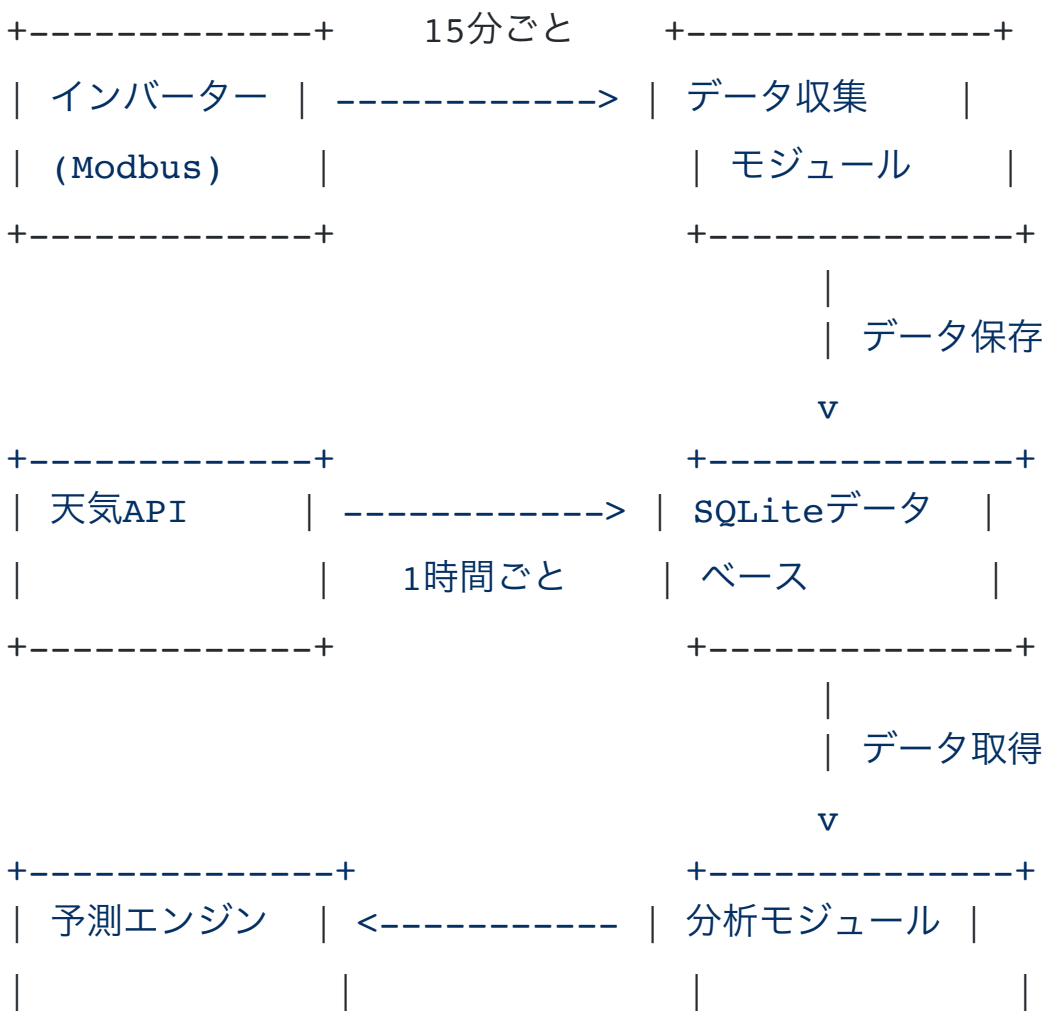
1. システムアーキテクチャと設計図

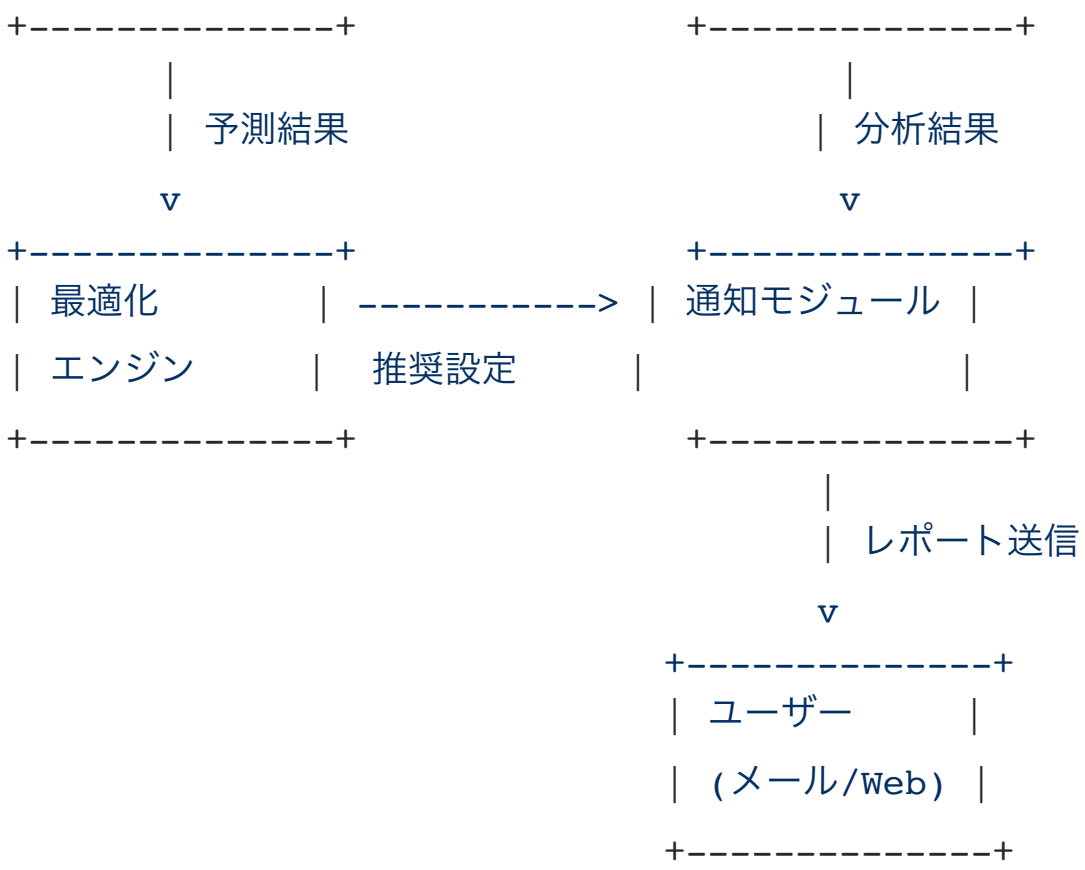
1.1 全体アーキテクチャ図




SQLiteデータベース
設定ファイル (settings.json)
外部システム連携
インバーター (Modbus)
天気API

1.2 データフロー図


インバーター (Modbus)
15分ごと
データ収集モジュール
データ保存
SQLiteデータベース
天気API
1時間ごと
データ取得
分析モジュール
予測エンジン



2. 環境構築と開発セットアップ

2.1 開発環境セットアップ手順

Copy

```
# Raspberry Pi Zero 2 W初期設定
# 1. OSインストール
# 2. 基本設定

# 必要なパッケージのインストール
sudo apt update
sudo apt install -y python3-pip python3-venv git
sqlite3

# プロジェクトディレクトリの作成
mkdir -p ~/lvyuan_solar_control
cd ~/lvyuan_solar_control
```

```
# Python仮想環境の作成とアクティベーション
python3 -m venv venv
source venv/bin/activate

# 必要なPythonパッケージのインストール
pip install pysolarmanv5 pymodbus requests matplotlib
pandas scikit-learn flask flask-restful APScheduler

# プロジェクト構造の作成
mkdir -p modules utils data logs charts templates web
mkdir -p data/db

# 設定ファイルの初期化
cat > settings.json << EOF
{
  "inverter": {
    "ip": "192.168.0.202",
    "serial_number": "3528830226",
    "port": 8899,
    "mb_slave_id": 1
  },
  "email": {
    "smtp_server": "smtp.example.com",
    "smtp_port": 587,
    "smtp_user": "your_username",
    "smtp_password": "your_password",
    "from_address": "sender@example.com",
    "to_addresses": ["recipient@example.com"]
  },
  "weather_api": {
    "provider": "openweathermap",
    "api_key": "your_api_key",
    "location": "kagawa,jp"
  },
```

```
  "system": {
    "data_retention": {
      "detailed_days": 30,
      "daily_days": 365,
      "monthly_limit": 0
    },
    "log_level": "INFO",
    "backup_enabled": true
  }
}
EOF

# Gitリポジトリの初期化
git init
echo "venv/" > .gitignore
echo "*.pyc" >> .gitignore
echo "__pycache__/" >> .gitignore
echo "settings.json" >> .gitignore
git add .
git commit -m "Initial project setup"

# cronジョブの設定
(crontab -l 2>/dev/null; echo "*/15 * * * * cd ~/lvyuan_solar_control && source venv/bin/activate && python main.py --collect") | crontab -
(crontab -l 2>/dev/null; echo "0 8 * * * cd ~/lvyuan_solar_control && source venv/bin/activate && python main.py --daily-report") | crontab -
```

2.2 必要なパッケージとバージョン

パッケージ名	バージョン	用途
pysolarmanv5	2.0.1+	インバーターModbus通信

pymodbus	3.0.0+	Modbusプロトコル処理
requests	2.27.0+	HTTP API通信
matplotlib	3.5.0+	グラフ生成
pandas	1.3.5+	データ分析
numpy	1.21.0+	数値計算
scikit-learn	1.0.2+	機械学習/予測モデル
flask	2.0.0+	Webインターフェース
APScheduler	3.9.0+	ジョブスケジューリング
SQLAlchemy	1.4.0+	ORM（オプション）

3. 実装コード詳細サンプル

3.1 メインエントリーポイント (main.py)

Copy

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
HANAZONOソーラー蓄電システム自動最適化プロジェクト
メインエントリーポイント
"""

import os
import sys
import argparse
import logging
from datetime import datetime
```

```
# モジュールのインポート
from utils.logger import setup_logging
from utils.config import load_settings
from modules.collector import LvyuanCollector
from modules.notifier import EmailNotifier
from modules.analyzer import DataAnalyzer

def main():
    """メイン関数"""
    # コマンドライン引数のパース
    parser =
argparse.ArgumentParser(description='LVYUAN Solar
Control System')
    parser.add_argument('--collect',
action='store_true', help='データ収集を実行')
    parser.add_argument('--daily-report',
action='store_true', help='日次レポートを送信')
    parser.add_argument('--analyze',
action='store_true', help='データ分析を実行')
    parser.add_argument('--optimize',
action='store_true', help='最適化計算を実行')
    parser.add_argument('--date', type=str, help='対象
日付（YYYYMMDD形式）')
    args = parser.parse_args()

    # ロギング設定
    logger = setup_logging()
    logger.info("HANAZONOソーラー蓄電システム自動最適化シス
テム起動")

    # 設定ファイルの読み込み
```

```
    try:
        settings = load_settings()
    except Exception as e:
        logger.error(f"設定ファイル読み込みエラー:
{str(e)}")
        sys.exit(1)

    # データ収集実行
    if args.collect:
        try:
            collector =
LvyuanCollector(settings['inverter'])
            success = collector.collect_all_data()
            if success:
                logger.info("データ収集完了")
            else:
                logger.warning("データ収集に問題がありました")
        except Exception as e:
            logger.error(f"データ収集エラー: {str(e)}")
            import traceback
            logger.error(traceback.format_exc())

    # 日次レポート送信
    if args.daily_report:
        try:
            target_date = args.date
            if not target_date:
                # デフォルトは前日
                yesterday = datetime.now().date() -
timedelta(days=1)
                target_date =
yesterday.strftime("%Y%m%d")
```

```
            notifier =
EmailNotifier(settings['email'])
            success =
notifier.send_daily_report(target_date)
            if success:
                logger.info(f"日次レポート送信完了:
{target_date}")
            else:
                logger.warning(f"日次レポート送信に問題が
ありました: {target_date}")
        except Exception as e:
            logger.error(f"レポート送信エラー:
{str(e)}")
            import traceback
            logger.error(traceback.format_exc())

    # データ分析実行
    if args.analyze:
        try:
            target_date = args.date
            analyzer = DataAnalyzer(settings)
            results = analyzer.analyze(target_date)
            logger.info(f"データ分析完了: {results}")
        except Exception as e:
            logger.error(f"データ分析エラー: {str(e)}")

    # 何も指定されていない場合のヘルプ
    if not (args.collect or args.daily_report or
args.analyze or args.optimize):
        parser.print_help()

if __name__ == "__main__":
```

```
    main()
```

3.2 データベースユーティリティ (utils/database.py)

Copy

```
# -*- coding: utf-8 -*-

"""
データベース操作ユーティリティ
"""

import os
import sqlite3
import logging
from datetime import datetime, timedelta
import json
import csv

logger = logging.getLogger(__name__)

class Database:
    """SQLiteデータベース操作クラス"""

    def __init__(self, db_path='data/db/
solar_data.db'):
        """初期化

        Args:
            db_path: データベースファイルパス
        """
        self.db_path = db_path
        # データベースディレクトリの作成
        os.makedirs(os.path.dirname(db_path),
exist_ok=True)
        self.setup()
```

```
    def get_connection(self):
        """データベース接続を取得"""
        conn = sqlite3.connect(self.db_path)
        conn.row_factory = sqlite3.Row  # 列名でアクセス
可能に
        return conn

    def setup(self):
        """データベースの初期セットアップ"""
        conn = self.get_connection()
        cursor = conn.cursor()

        # テーブル作成
        # 計測データテーブル
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS measurements (
            timestamp TEXT PRIMARY KEY,
            battery_soc INTEGER,
            battery_voltage REAL,
            battery_current REAL,
            pv_voltage REAL,
            pv_current REAL,
            pv_power REAL,
            load_power REAL,
            grid_power REAL,
            temperature REAL
        )
        ''')

        # パラメータ履歴テーブル
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS parameter_history
```

```
(
        timestamp TEXT PRIMARY KEY,
        charge_current INTEGER,
        charge_time INTEGER,
        output_soc INTEGER,
        change_reason TEXT,
        weather TEXT,
        season TEXT
    )
    ''')

    # 天気データテーブル
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS weather_data (
        date TEXT PRIMARY KEY,
        weather TEXT,
        temp_high REAL,
        temp_low REAL,
        precipitation REAL
    )
    ''')

    # 日次サマリーテーブル
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS daily_summary (
        date TEXT PRIMARY KEY,
        total_generation REAL,
        total_consumption REAL,
        grid_purchase REAL,
        grid_feed_in REAL,
        self_consumption_rate REAL,
        average_soc REAL,
        min_soc INTEGER,
        max_soc INTEGER
    )
```

```
        ''')

        conn.commit()
        conn.close()

    def save_measurement(self, data):
        """計測データを保存

        Args:
            data: 計測データ辞書（timestamp,
battery_soc, ...）

        Returns:
            成功した場合はTrue
        """
        try:
            conn = self.get_connection()
            cursor = conn.cursor()

            # タイムスタンプが既に存在するか確認
            cursor.execute("SELECT timestamp FROM
measurements WHERE timestamp = ?",
(data['timestamp'],))
            exists = cursor.fetchone()

            if exists:
                # 更新
                placeholders = ", ".join([f"{k} = ?"
for k in data.keys() if k != 'timestamp'])
                values = [data[k] for k in
data.keys() if k != 'timestamp']
                values.append(data['timestamp'])
```

```
                cursor.execute(f"UPDATE measurements
SET {placeholders} WHERE timestamp = ?", values)
            else:
                # 挿入
                placeholders = ", ".join(["?"] *
len(data))
                columns = ", ".join(data.keys())
                values = list(data.values())

                cursor.execute(f"INSERT INTO
measurements ({columns}) VALUES ({placeholders})",
values)

            conn.commit()
            conn.close()
            return True
        except Exception as e:
            logger.error(f"計測データ保存エラー:
{str(e)}")
            return False

    def import_legacy_data(self, data_dir='data'):
        """従来のJSONデータをインポート

        Args:
            data_dir: 従来のデータファイルディレクトリ

        Returns:
            インポートされたレコード数
        """
        imported_count = 0

        try:
```

```
            # ディレクトリ内のJSONファイルを検索
            for file in os.listdir(data_dir):
                if file.startswith("lvyuan_data_")
and file.endswith(".json"):
                    file_path =
os.path.join(data_dir, file)

                    with open(file_path, 'r') as f:
                        data_list = json.load(f)

                        for record in data_list:
                            # タイムスタンプ形式の変換（必
要に応じて）
                            record['timestamp'] =
datetime.fromisoformat(record['timestamp']).isoformat
()

                            # データベースに保存
                            success =
self.save_measurement(record)
                            if success:
                                imported_count += 1

            logger.info(f"{imported_count}件のレガシーデ
ータをインポートしました")
            return imported_count

        except Exception as e:
            logger.error(f"レガシーデータインポートエラー:
{str(e)}")
            return imported_count

    def get_daily_data(self, date):
```

```
        """特定の日のデータを取得

        Args:
            date: YYYYMMDD形式の日付文字列

        Returns:
            データリスト
        """
        try:
            conn = self.get_connection()
            cursor = conn.cursor()

            # 日付文字列を整形
            date_obj = datetime.strptime(date,
"%Y%m%d")
            start_date = date_obj.strftime("%Y-%m-
%d")
            end_date = (date_obj +
timedelta(days=1)).strftime("%Y-%m-%d")

            cursor.execute(
                "SELECT * FROM measurements WHERE
timestamp >= ? AND timestamp < ? ORDER BY timestamp",
                (start_date, end_date)
            )

            # dictに変換
            rows = cursor.fetchall()
            result = [dict(row) for row in rows]

            conn.close()
            return result
        except Exception as e:
```

```
            logger.error(f"日次データ取得エラー:
{str(e)}")
            return []

    def generate_daily_summary(self, date):
        """日次サマリーの生成と保存

        Args:
            date: YYYYMMDD形式の日付文字列

        Returns:
            サマリー辞書またはNone
        """
        try:
            # その日のデータを取得
            daily_data = self.get_daily_data(date)

            if not daily_data:
                logger.warning(f"日付 {date} のデータが
ありません")
                return None

            # 計算用の変数初期化
            total_generation = 0
            total_consumption = 0
            grid_purchase = 0
            grid_feed_in = 0
            soc_values = []

            # 各レコードを処理
            for record in daily_data:
                # PV発電量が正の場合のみ合計
```

```
                pv_power = record.get('pv_power', 0)
                if pv_power > 0:
                    # 15分ごとのデータなので、kWhに変換（W
* 1/4h / 1000）
                    total_generation += pv_power *
0.25 / 1000

                # 負荷電力
                load_power = record.get('load_power',
0)
                if load_power > 0:
                    total_consumption += load_power *
0.25 / 1000

                # グリッド電力（正なら購入、負なら売電）
                grid_power = record.get('grid_power',
0)
                if grid_power > 0:
                    grid_purchase += grid_power *
0.25 / 1000
                else:
                    grid_feed_in += abs(grid_power) *
0.25 / 1000

                # SOC値を記録
                soc = record.get('battery_soc', 0)
                if soc > 0:
                    soc_values.append(soc)

            # 自家消費率計算
            self_consumption = total_generation -
grid_feed_in
            self_consumption_rate =
```

```
(self_consumption / total_generation * 100) if
total_generation > 0 else 0

            # SOC統計
            avg_soc = sum(soc_values) /
len(soc_values) if soc_values else 0
            min_soc = min(soc_values) if soc_values
else 0
            max_soc = max(soc_values) if soc_values
else 0

            # サマリーデータの作成
            date_str = datetime.strptime(date,
"%Y%m%d").strftime("%Y-%m-%d")
            summary = {
                'date': date_str,
                'total_generation':
round(total_generation, 2),
                'total_consumption':
round(total_consumption, 2),
                'grid_purchase': round(grid_purchase,
2),
                'grid_feed_in': round(grid_feed_in,
2),
                'self_consumption_rate':
round(self_consumption_rate, 1),
                'average_soc': round(avg_soc, 1),
                'min_soc': int(min_soc),
                'max_soc': int(max_soc)
            }

            # データベースに保存
            conn = self.get_connection()
            cursor = conn.cursor()
```

```
            # 既存データがあれば更新、なければ挿入
            cursor.execute("SELECT date FROM
daily_summary WHERE date = ?", (date_str,))
            exists = cursor.fetchone()

            if exists:
                placeholders = ", ".join([f"{k} = ?"
for k in summary.keys() if k != 'date'])
                values = [summary[k] for k in
summary.keys() if k != 'date']
                values.append(date_str)

                cursor.execute(f"UPDATE daily_summary
SET {placeholders} WHERE date = ?", values)
            else:
                placeholders = ", ".join(["?"] *
len(summary))
                columns = ", ".join(summary.keys())
                values = list(summary.values())

                cursor.execute(f"INSERT INTO
daily_summary ({columns}) VALUES ({placeholders})",
values)

            conn.commit()
            conn.close()

            logger.info(f"日付 {date} の日次サマリー生成
完了")

            return summary

        except Exception as e:
            logger.error(f"日次サマリー生成エラー:
```

```
{str(e)}")
                return None
```

4. 詳細トラブルシューティングガイド

4.1 一般的な問題と解決法

4.1.1 インバーター通信エラー

エラー症状	考えられる原因	対処法
接続タイムアウト	ネットワーク問題	Wi-Fi信号強度確認、インバーターIPアドレス確認
	インバーターオフライン	インバーター電源の確認、WiFiモジュールの再起動
Modbus例外	レジスタアドレス誤り	レジスタマップの見直し、アドレスのダブルチェック
	読取権限なし	読取専用レジスタであることを確認
CRC不一致	通信パラメーター不正	ボーレート、データビット、停止ビットの確認
	信号ノイズ	RS485ケーブルのシールド確認、電力線との距離確保

インバーター通信障害の復旧手順：

1. 基本確認：

   ```
   Copy
   ```

   ```
   ping [インバーターIPアドレス]
   ```

2.

3. WiFiモジュールの再起動：
    - LSW-5A8153-RS485の電源を10秒間オフにし、再度オン
4. ネットワークスキャンで検出：

```
Copy
sudo nmap -sP 192.168.0.0/24
```

5.

```

```

6. Raspberry Piネットワーク設定確認：

```
Copy
ifconfig
```

7. `iwconfig`
8.

```

```

9. インバーターへの手動接続テスト：

```
Copy
from pysolarmanv5 import PySolarmanV5
```

10.
```
    modbus =
PySolarmanV5(address="192.168.0.202",
serial=3528830226, port=8899, mb_slave_id=1,
verbose=True)
```

11.
```
    result =
```

```
    modbus.read_holding_registers(0x0100, 1)
    print(f"Battery SOC: {result[0]}%")

```

4.1.2 データベースエラー

エラー症状	考えられる原因	対処法
データベース接続不能	ファイルアクセス権限	ファイル/ディレクトリのパーミッション確認
	破損したDBファイル	バックアップから復元、新規DB作成
クエリエラー	SQL構文エラー	クエリのデバッグ、パラメーター確認
	テーブルスキーマ不一致	スキーマ定義の確認、マイグレーション実行
データ整合性エラー	重複キー	一意制約のチェック、データクリーニング
	トランザクション失敗	コミット/ロールバック処理の見直し

データベース修復手順：

1. データベース整合性チェック：

```
Copy

sqlite3 data/db/solar_data.db "PRAGMA
```

```
integrity_check;"
```

2.

3. バックアップ作成：

```
Copy
sqlite3 data/db/solar_data.db ".backup data/db/
backup_$(date +%Y%m%d).db"
```

4.

5. スキーマ検証：

```
Copy
sqlite3 data/db/solar_data.db ".schema"
```

6.

7. データベース再構築（最終手段）：

```
Copy
mv data/db/solar_data.db data/db/
solar_data.db.old
```

8. `python -c "from utils.database import Database; db = Database(); print('データベース再構築完了')"`

9.

4.2 システム管理とメンテナンス

4.2.1 ログローテーション設定

Copy

```
# /etc/logrotate.d/lvyuan_solar にコピーするファイル内容
/home/pi/lvyuan_solar_control/logs/*.log {
  weekly
  rotate 12
  compress
  delaycompress
  missingok
  notifempty
  create 0640 pi pi
}
```

4.2.2 システム自動バックアップスクリプト

Copy

```
#!/bin/bash
# backup_system.sh

# バックアップ先ディレクトリ
BACKUP_DIR="/home/pi/backups"
DATE=$(date +%Y%m%d)

# ディレクトリ作成
mkdir -p $BACKUP_DIR

# データベースバックアップ
sqlite3 /home/pi/lvyuan_solar_control/data/db/
solar_data.db ".backup $BACKUP_DIR/
solar_data_$DATE.db"

# 設定ファイルバックアップ
```

```
cp /home/pi/lvyuan_solar_control/settings.json
$BACKUP_DIR/settings_$DATE.json

# ログファイルバックアップ
tar -czf $BACKUP_DIR/logs_$DATE.tar.gz /home/pi/
lvyuan_solar_control/logs/

# 古いバックアップの削除（30日以上前）
find $BACKUP_DIR -name "*.db" -mtime +30 -delete
find $BACKUP_DIR -name "*.json" -mtime +30 -delete
find $BACKUP_DIR -name "*.tar.gz" -mtime +30 -delete

echo "Backup completed: $DATE"
```

4.2.3 システム健全性チェック

Copy

```
#!/usr/bin/env python3
# check_system_health.py

import os
import psutil
import subprocess
import smtplib
from email.mime.text import MIMEText
import sqlite3
import json

def check_disk_space():
    """ディスク使用量をチェック"""
    disk = psutil.disk_usage('/')
    free_percent = disk.free / disk.total * 100
    return {
        'total_gb': disk.total / (1024 * 1024 *
1024),
        'used_gb': disk.used / (1024 * 1024 * 1024),
```

```
        'free_gb': disk.free / (1024 * 1024 * 1024),
        'free_percent': free_percent,
        'status': 'OK' if free_percent > 20 else
'WARNING'
    }

def check_memory():
    """メモリ使用量をチェック"""
    mem = psutil.virtual_memory()
    free_percent = mem.available / mem.total * 100
    return {
        'total_mb': mem.total / (1024 * 1024),
        'used_mb': mem.used / (1024 * 1024),
        'available_mb': mem.available / (1024 *
1024),
        'free_percent': free_percent,
        'status': 'OK' if free_percent > 20 else
'WARNING'
    }

def check_cpu():
    """CPU使用率をチェック"""
    cpu_percent = psutil.cpu_percent(interval=1)
    return {
        'usage_percent': cpu_percent,
        'status': 'OK' if cpu_percent < 80 else
'WARNING'
    }

def check_temperature():
    """CPU温度をチェック"""
    try:
        temp = subprocess.check_output(['vcgencmd',
'measure_temp'])
```

```
            temp_str =
temp.decode('utf-8').strip().replace('temp=',
'').replace('\'C', '')
            temp_value = float(temp_str)
            return {
                'cpu_temp_c': temp_value,
                'status': 'OK' if temp_value < 70 else
'WARNING'
            }
        except:
            return {'cpu_temp_c': -1, 'status': 'ERROR'}

def check_processes():
    """重要なプロセスを確認"""
    result = {}
    pids = {}
    for proc in psutil.process_iter(['pid', 'name']):
        process = proc.info
        if 'python' in process['name']:
            pids[process['pid']] = process['name']

    result['python_processes'] = len(pids)
    result['status'] = 'OK' if len(pids) > 0 else
'WARNING'
    return result

def
```

コピー

おわりですか？

申し訳ありません、最後の部分が途中で切れてしまいました。続きをお伝えします：

Copy

```
def check_database():
    """データベース接続をチェック"""
    try:
        conn = sqlite3.connect('/home/pi/
lvyuan_solar_control/data/db/solar_data.db')
        cursor = conn.cursor()
        cursor.execute('PRAGMA integrity_check')
        result = cursor.fetchone()

        # データポイント数の確認
        cursor.execute('SELECT COUNT(*) FROM
measurements')
        count = cursor.fetchone()[0]

        # 最新データの確認
        cursor.execute('SELECT MAX(timestamp) FROM
measurements')
        last_timestamp = cursor.fetchone()[0]

        conn.close()

        return {
            'integrity': result[0],
            'record_count': count,
            'last_update': last_timestamp,
            'status': 'OK' if result[0] == 'ok' else
'ERROR'
        }
    except Exception as e:
        return {
            'integrity': 'ERROR',
            'error_message': str(e),
            'status': 'ERROR'
        }
```

```
def send_alert_email(status_report):
    """アラートメールを送信"""
    try:
        # 設定読み込み
        with open('/home/pi/lvyuan_solar_control/
settings.json', 'r') as f:
            settings = json.load(f)

        email_config = settings.get('email', {})

        # メール内容構築
        subject = f"ALERT: HANAZONO System Health
Check - {status_report.get('overall_status')}"
        body = "System Health Report\n\n"

        for category, data in status_report.items():
            if category != 'overall_status':
                body += f"\n== {category.upper()}
==\n"
                for key, value in data.items():
                    body += f"{key}: {value}\n"

        # メール送信
        msg = MIMEText(body)
        msg['Subject'] = subject
        msg['From'] =
email_config.get('from_address')
        msg['To'] = ',
'.join(email_config.get('to_addresses', []))

        smtp =
smtplib.SMTP(email_config.get('smtp_server'),
email_config.get('smtp_port'))
```

```
            smtp.starttls()
            smtp.login(email_config.get('smtp_user'),
email_config.get('smtp_password'))
            smtp.send_message(msg)
            smtp.quit()

            return True
        except Exception as e:
            print(f"メール送信エラー: {str(e)}")
            return False

def main():
    """メインチェック関数"""
    status_report = {}

    # 各種チェックを実行
    status_report['disk'] = check_disk_space()
    status_report['memory'] = check_memory()
    status_report['cpu'] = check_cpu()
    status_report['temperature'] =
check_temperature()
    status_report['processes'] = check_processes()
    status_report['database'] = check_database()

    # 全体ステータス判定
    overall_status = 'OK'
    for category, data in status_report.items():
        if data.get('status') == 'ERROR':
            overall_status = 'ERROR'
            break
        elif data.get('status') == 'WARNING' and
overall_status != 'ERROR':
            overall_status = 'WARNING'
```

```
    status_report['overall_status'] = overall_status

    # 問題があればアラートメール送信
    if overall_status != 'OK':
        send_alert_email(status_report)

    # 結果表示
    print(f"System Health Check: {overall_status}")
    for category, data in status_report.items():
        if category != 'overall_status':
            print(f"- {category}:
{data.get('status')}")

    return status_report

if __name__ == "__main__":
    main()
```

5. テストケースと検証手順

5.1 単体テスト例

5.1.1 データ収集モジュールのテスト (tests/test_collector.py)

Copy

```
import unittest
from unittest.mock import patch, MagicMock
import sys
import os
import json

# テスト対象モジュールのインポート
sys.path.append(os.path.dirname(os.path.dirname(os.pa
th.abspath(__file__))))
from modules.collector import LvyuanCollector

class TestLvyuanCollector(unittest.TestCase):
```

```
    """LvyuanCollectorクラスのテスト"""

    def setUp(self):
        """テスト前の準備"""
        self.test_settings = {
            'ip': '192.168.0.202',
            'serial_number': '3528830226',
            'port': 8899,
            'mb_slave_id': 1
        }

    @patch('modules.collector.PySolarmanV5')
    def test_connection(self, mock_pysolarman):
        """接続処理のテスト"""
        # モックオブジェクト設定
        mock_instance = MagicMock()
        mock_pysolarman.return_value = mock_instance

        # テスト実行
        collector =
LvyuanCollector(self.test_settings)
        result = collector._create_connection()

        # アサーション
        self.assertTrue(result)
        mock_pysolarman.assert_called_once_with(
            address=self.test_settings['ip'],

serial=self.test_settings['serial_number'],
            port=self.test_settings['port'],

mb_slave_id=self.test_settings['mb_slave_id'],
            verbose=False
```

```
        )

    @patch('modules.collector.PySolarmanV5')
    def test_read_registers(self, mock_pysolarman):
        """レジスタ読み取りのテスト"""
        # モックオブジェクト設定
        mock_instance = MagicMock()

mock_instance.read_holding_registers.return_value =
[50, 540, 12]  # SOC 50%, 電圧54.0V, 電流1.2A
        mock_pysolarman.return_value = mock_instance

        # テスト実行
        collector =
LvyuanCollector(self.test_settings)
        result = collector.read_registers(0x0100, 3)

        # アサーション
        self.assertEqual(result, [50, 540, 12])

mock_instance.read_holding_registers.assert_called_on
ce_with(0x0100, 3)

    @patch('modules.collector.PySolarmanV5')
    def test_get_battery_status(self,
mock_pysolarman):
        """バッテリー状態取得のテスト"""
        # モックオブジェクト設定
        mock_instance = MagicMock()

mock_instance.read_holding_registers.return_value =
[50, 540, 12]  # SOC 50%, 電圧54.0V, 電流1.2A
        mock_pysolarman.return_value = mock_instance
```

```
        # テスト実行
        collector =
LvyuanCollector(self.test_settings)
        result = collector.get_battery_status()

        # アサーション
        self.assertEqual(result['soc'], 50)
        self.assertEqual(result['voltage'], 54.0)
        self.assertEqual(result['current'], 1.2)

    @patch('modules.collector.PySolarmanV5')
    @patch('modules.collector.datetime')
    @patch('builtins.open',
new_callable=unittest.mock.mock_open)
    @patch('json.dump')
    def test_collect_all_data(self, mock_json_dump,
mock_open, mock_datetime, mock_pysolarman):
        """データ収集全体テスト"""
        # モックオブジェクト設定
        mock_instance = MagicMock()

mock_instance.read_holding_registers.side_effect = [
            [50, 540, 12],  # バッテリー状態
            [350, 10, 3500],  # PVデータ
            [60, 45, 45]  # 充電設定
        ]
        mock_pysolarman.return_value = mock_instance

        mock_now = MagicMock()
        mock_now.strftime.return_value = "20250502"
        mock_datetime.now.return_value = mock_now
```

```
        # テスト実行
        collector =
LvyuanCollector(self.test_settings)
        result = collector.collect_all_data()

        # アサーション
        self.assertTrue(result)
        mock_open.assert_called_once()
        mock_json_dump.assert_called_once()

if __name__ == '__main__':
    unittest.main()
```

5.2 統合テスト計画

5.2.1 End-to-Endテストシナリオ

1. **基本データフロー検証**:
   - データ収集 → データベース保存 → レポート生成 → メール送信
   - 各ステップの成否確認
   - データの整合性検証
2. **フォールバックシナリオ**:
   - 前日データ欠損時のフォールバック動作
   - モックデータでの挙動確認
3. **エラー処理検証**:
   - インバーター通信切断時の動作
   - データベース障害時の対応
   - メール送信失敗時の再試行
4. **データベース移行テスト**:
   - 既存データからの正常移行確認

- データ整合性検証
- 古いデータフォーマットの互換性

6. セキュリティ強化ガイド

6.1 認証セキュリティ

6.1.1 Webインターフェース認証

Copy

```
# Flask Webインターフェース用セキュリティ実装例

from flask import Flask, render_template, request,
redirect, url_for, session
from functools import wraps
import os
import secrets

app = Flask(__name__)
app.secret_key = secrets.token_hex(16)  # セッション用の
安全な秘密鍵

# アクセス制御用デコレータ
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'logged_in' not in session:
            return redirect(url_for('login',
next=request.url))
        return f(*args, **kwargs)
    return decorated_function

# ログインページ
@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
```

```
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # ハードコーディングは本番環境では避け、設定ファイル等から読み込む
        # もしくはパスワードハッシュを使用する
        if username == 'admin' and password == os.environ.get('ADMIN_PASSWORD', 'default_password'):
            session['logged_in'] = True
            session['username'] = username
            return redirect(url_for('index'))
        else:
            error = '無効なユーザー名またはパスワードです'

    return render_template('login.html', error=error)

# ログアウト
@app.route('/logout')
def logout():
    session.pop('logged_in', None)
    session.pop('username', None)
    return redirect(url_for('login'))

# ダッシュボード（保護されたページ）
@app.route('/')
@login_required
def index():
    return render_template('index.html')

# 設定ページ（保護されたページ）
@app.route('/settings')
@login_required
```

```
def settings():
    return render_template('settings.html')
```

6.2 通信セキュリティ

6.2.1 APIアクセス制限

Copy

```
# API制限の実装例

from functools import wraps
from flask import request, jsonify
import time
import hashlib
import hmac

# API呼び出し制限（レート制限）
def rate_limit(max_calls=100, time_frame=3600):
    """
    レート制限デコレータ

    Args:
        max_calls: 時間枠あたりの最大呼び出し回数
        time_frame: 時間枠（秒）
    """
    calls = {}

    def decorator(f):
        @wraps(f)
        def wrapped(*args, **kwargs):
            # クライアントIPの取得
            client_ip = request.remote_addr

            # 現在の呼び出し情報
            current_time = time.time()
```

```
            calls.setdefault(client_ip, [])

            # 時間枠外の古い呼び出しを削除
            calls[client_ip] = [call_time for
call_time in calls[client_ip]
                                if call_time >
current_time - time_frame]

            # 呼び出し回数チェック
            if len(calls[client_ip]) >= max_calls:
                return jsonify({'error': '呼び出し回数制
限を超えました。しばらく経ってから再試行してください。'}), 429

            # 呼び出し記録を追加
            calls[client_ip].append(current_time)

            return f(*args, **kwargs)
        return wrapped
    return decorator

# API認証
def api_auth_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        # リクエストからAPIキーとシグネチャを取得
        api_key = request.headers.get('X-API-Key')
        timestamp = request.headers.get('X-
Timestamp')
        signature = request.headers.get('X-
Signature')

        # APIキー確認
        if not api_key or api_key !=
```

```
os.environ.get('API_KEY'):
            return jsonify({'error': 'APIキーが無効で
す'}), 401

        # タイムスタンプ確認（リプレイアタック対策）
        if not timestamp or abs(int(timestamp) -
int(time.time())) > 300:  # 5分以内
            return jsonify({'error': 'タイムスタンプが無
効です'}), 401

        # シグネチャ確認
        if not signature:
            return jsonify({'error': 'シグネチャが必要で
す'}), 401

        # シグネチャ検証
        expected_signature = hmac.new(
            os.environ.get('API_SECRET').encode(),
            f"{api_key}{timestamp}
{request.path}".encode(),
            hashlib.sha256
        ).hexdigest()

        if not hmac.compare_digest(signature,
expected_signature):
            return jsonify({'error': 'シグネチャが無効で
す'}), 401

        return f(*args, **kwargs)
    return decorated_function

@app.route('/api/solar/data')
```

```
@api_auth_required
@rate_limit(max_calls=100)
def get_solar_data():
    # 実際のデータ取得処理
    # ...
    return jsonify({'data': 'solar_data'})
```

7. パフォーマンス最適化ヒント

7.1 Raspberry Pi Zero 2 W向けリソース最適化

1. **データベース最適化**:
   - インデックス追加（頻繁に検索される列）
   - 定期的なVACUUMによるデータベース最適化
   - クエリキャッシュの活用
2. **メモリ使用量削減**:
   - 大きなデータ操作時のジェネレーターパターン使用
   - 不要なプロセス/サービスの無効化
   - zramによるスワップ最適化
3. **ディスクI/O最適化**:
   - 書き込みバッファリング
   - ログローテーション設定
   - tmpfs活用（一時データ用）
4. **プロセス管理**:
   - 複数の重いプロセスの同時実行を避ける
   - スケジューリング調整（優先度低下など）
   - cron設定の分散

これらの補足資料が「HANAZONOシステム自動最適化プロジェクト - 統合ロードマップ+」をさらに完璧なものにし、初めてプロジ

ェクトに触れる方にとっても詳細な実装方法や問題解決のヒントとなることを願っています。主要な資料と合わせて、プロジェクトの全体像を完全に把握できるようになったと思います。