# Project Documentation: Solar Energy Forecasting System

## 1. Overview

This project involves developing an energy forecasting system using deep learning techniques and integrating it into a scalable software architecture. The goal is to enable real-time solar energy production forecasting and data monitoring.

## 2. Deep Learning Challenge: Solar Energy Forecasting

### 2.1 Dataset

The dataset includes:
- Historical power production data (hourly intervals).
- Weather data: Temperature, irradiance, and cloud cover.
- System configuration details and maintenance logs.

### 2.2 Data Analysis and Preprocessing

1. Data Quality Check:
- Missing values handled using interpolation.
- Outliers managed with statistical thresholds (e.g., z-score).

2. Feature Engineering:
- Normalized numeric columns for temperature, irradiance, and cloud cover.
- Created lagged features to capture temporal patterns for forecasting.
- Encoded categorical variables using one-hot encoding.

3. Preprocessing Pipeline:
- Automated handling of missing data, scaling, and feature generation.

### 2.3 Model Development

1. Models Used:
- LSTM: Captures long-term dependencies in time-series data.
- Transformer with Attention Mechanisms: Adds interpretability and handles longer sequences.

2. Implementation:
- Configured an LSTM model with dropout regularization to avoid overfitting.
- Integrated attention layers in the Transformer to highlight critical features.

3. Training Pipeline:
- Used 80-10-10 split for training, validation, and testing.
- Early stopping and learning rate scheduling for performance optimization.

4. Metrics:
- MSE, MAE, and MAPE computed for evaluation.

## 3. Software Engineering Challenge: System Integration

### 3.1 Architecture Design

1. APIs:
- /forecast: Generates solar energy predictions based on input data.
- /forecasts/recent: Retrieves the most recent forecast data from the database.
- /health: Provides system status.

2. Data Flow:
- User inputs -> Preprocessing -> Model inference -> Database storage -> User response.

3. Database Schema:
- forecast_results table includes:
  - id: Unique identifier.
  - timestamp: Time of forecast.
  - input_data: JSON of input features.
  - forecast_values: JSON of predictions.
  - model_metrics: JSON of evaluation metrics.

### 3.2 Implementation

1. Used FastAPI for API creation.
2. SQLite database for lightweight, local storage.
3. Logging integrated for monitoring and debugging.

### 3.3 Scaling Strategy

1. Deployment with Uvicorn in asynchronous mode for handling concurrent requests.
2. Horizontal scaling with containerization using Docker (future scope).

## 4. Dependencies

The following Python libraries and tools are required:
- tensorflow
- fastapi
- uvicorn
- numpy
- pandas
- sqlalchemy
- pydantic
- torch

## 5. Technical Implementation

### 5.1 Model and Scripts

1. Trained model: solar_forecaster.h5
2. Jupyter notebook for preprocessing, training, and evaluation.
3. Python scripts for API and database integration.

### 5.2 System Deployment

1. Install dependencies(In the backend directory):
   python -m venv backend

   pip install -r requirements.txt
2. Start the application:
   uvicorn backend:app --host 0.0.0.0 --port 8000 --reload