Low-Level Design (LLD) Document: Breakdown of how each component is implemented

1. **Data Preprocessing:** Missing values, scaling, outlier handling

```
df.isnull().sum()
```
✓ 0.0s

```
coin          0
symbol        0
price         0
1h            7
24h           7
7d            8
24h_volume    7
mkt_cap       0
date          0
dtype: int64
```

```
missing_value_features = ['1h', '24h', '7d', '24h_volume']
for i in missing_value_features:
    df[i] = df[i].fillna(df[i].median())
```
✓ 0.0s

```
df.isnull().sum()
```
✓ 0.0s

```
coin          0
symbol        0
price         0
1h            0
24h           0
7d            0
24h_volume    0
mkt_cap       0
date          0
dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler
num_cols  = df.select_dtypes(include = 'float64').columns
scaler = MinMaxScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])
```
✓ 0.3s

```
df
```
✓ 0.0s

	coin	symbol	price	1h	24h	7d	24h_volume	mkt_cap	date
0	Bitcoin	BTC	9.913189e-01	0.908636	0.552739	0.118660	6.108705e-01	0.993446	2022-03-16
1	Ethereum	ETH	6.658398e-02	0.911139	0.556010	0.120596	3.408769e-01	0.421435	2022-03-16
2	Tether	USDT	2.426167e-05	0.879850	0.527392	0.108014	1.000000e+00	0.102962	2022-03-16
3	BNB	BNB	9.302654e-03	0.903630	0.551104	0.108788	2.409346e-02	0.082445	2022-03-16
4	USD Coin	USDC	2.425862e-05	0.879850	0.528209	0.108014	6.683828e-02	0.067211	2022-03-16
...	...	...	...	...	...	...	...	...	...
995	IRISnet	IRIS	1.344718e-06	0.901126	0.525756	0.090979	5.138242e-05	0.000003	2022-03-17
996	Circuits of Value	COVAL	9.209975e-07	0.883605	0.518397	0.097561	6.331012e-06	0.000003	2022-03-17
997	ARPA Chain	ARPA	1.674137e-06	0.881101	0.534751	0.100852	2.353287e-04	0.000003	2022-03-17
998	SuperRare	RARE	1.127229e-05	0.877347	0.539657	0.111692	1.622201e-04	0.000002	2022-03-17
999	Verus Coin	VRSC	2.424680e-05	0.947434	0.578087	0.084398	3.499626e-07	0.000002	2022-03-17

1000 rows × 9 columns

```python
def cap_outliers(df, cols, lower_percentile=0.01, upper_percentile=0.99):
    capped_df = df.copy()
    for col in cols:
        if col in capped_df.columns:
            lower_limit = capped_df[col].quantile(lower_percentile)
            upper_limit = capped_df[col].quantile(upper_percentile)
            capped_df[col] = np.where(
                capped_df[col] < lower_limit, lower_limit,
                np.where(capped_df[col] > upper_limit, upper_limit, capped_df[col])
            )
            print(f"{col}: capped at {lower_percentile*100}% and {upper_percentile*100}% percentiles")
    return capped_df

numeric_cols = ['price', '1h', '24h', '7d', '24h_volume', 'mkt_cap']
df_capped = cap_outliers(df, numeric_cols)

for col in ['price', '24h_volume', 'mkt_cap']:
    if col in df_capped.columns:
        df_capped[col] = np.log1p(df_capped[col])  # log(1 + x)
        print(f"Applied log transformation on {col}")

print("\n Outlier handling complete (Capping + Log Transform).")
print("Original shape:", df.shape)
print("New shape:", df_capped.shape)
```

[22] ✓ 0.0s

```
price: capped at 1.0% and 99.0% percentiles
1h: capped at 1.0% and 99.0% percentiles
24h: capped at 1.0% and 99.0% percentiles
7d: capped at 1.0% and 99.0% percentiles
24h_volume: capped at 1.0% and 99.0% percentiles
mkt_cap: capped at 1.0% and 99.0% percentiles
Applied log transformation on price
Applied log transformation on 24h_volume
Applied log transformation on mkt_cap

 Outlier handling complete (Capping + Log Transform).
Original shape: (1000, 9)
New shape: (1000, 9)
```

2. **Feature Engineering:** Created columns — trading volume, social media index, etc.

```python
# Trading Volume – captures how active the coin is
df_capped['trading_volume_feature'] = df_capped['24h_volume'] / (df_capped['mkt_cap'] + 1e-9)

# Transaction Patterns – captures price change momentum
df_capped['transaction_pattern_feature'] = df_capped['1h'] + df_capped['24h'] + df_capped['7d']

# Exchange Listings – proxy: how often a coin appears in dataset
coin_freq = df_capped['coin'].value_counts(normalize=True)
df_capped['exchange_listing_feature'] = df_capped['coin'].map(coin_freq)

# Social Media Activity – proxy: absolute short-term volatility
df_capped['social_media_activity_feature'] = abs(df_capped['1h']) + abs(df_capped['24h'])
```

[24] ✓ 0.0s

```python
feature_cols = [
    'trading_volume_feature',
    'transaction_pattern_feature',
    'exchange_listing_feature',
    'social_media_activity_feature'
]
df_capped['liquidity_ratio'] = df_capped['24h_volume'] / (df_capped['mkt_cap'] + 1e-9)

target = 'liquidity_ratio'

print("\n Final Simplified Feature Set:")
print(feature_cols)
print("\nSample preview:")
print(df_capped[feature_cols + [target]].head())
```

[25] ✓ 0.0s

```
 Final Simplified Feature Set:
['trading_volume_feature', 'transaction_pattern_feature', 'exchange_listing_feature', 'social_media_activity_feature']

Sample preview:
   trading_volume_feature  transaction_pattern_feature  \
0                1.262355                     1.580035
1                1.262355                     1.587745
2                1.262355                     1.515255
3                0.492738                     1.563522
4                1.262355                     1.516073

   exchange_listing_feature  social_media_activity_feature  liquidity_ratio
0                     0.002                       1.461375         1.262355
1                     0.002                       1.467149         1.262355
2                     0.002                       1.407241         1.262355
3                     0.002                       1.454733         0.492738
4                     0.002                       1.408059         1.262355
```

3. **Model Training:** Random Forest Regressor + GridSearchCV

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

rf = RandomForestRegressor(random_state=42)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

grid = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring='r2',
    cv=5,
    verbose=2,
    n_jobs=-1
)

grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print(" Best Cross-Validation R² Score:", grid.best_score_)
```

```
[32] ✓ 1m 17.5s
... Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best Parameters: {'max_depth': 30, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
 Best Cross-Validation R² Score: 0.9291192528903448
```

4. **Model Saving:** joblib.dump(model, 'liquidity_model.pkl')

```python
import joblib

joblib.dump(best_ridge, "liquidity_model.pkl")
print("Tuned Ridge model saved as tuned_ridge_liquidity_model.pkl")
```

```
[37] ✓ 0.0s
... Tuned Ridge model saved as tuned_ridge_liquidity_model.pkl
```

5. **API Logic (Flask):** Route /predict → reads input → returns prediction.

app.py > ...

```python
from flask import Flask, render_template, request
import numpy as np
import joblib

app = Flask(__name__)

model = joblib.load("liquidity_model.pkl")

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        trading_volume = float(request.form['trading_volume'])
        transaction_pattern = float(request.form['transaction_pattern'])
        exchange_listing = float(request.form['exchange_listing'])
        social_media_activity = float(request.form['social_media_activity'])

        input_data = np.array([[trading_volume, transaction_pattern, exchange_listing, social_media_activity]])

        prediction = model.predict(input_data)[0]

        return render_template(
            'index.html',
            prediction_text=f"Predicted Liquidity Ratio: {prediction:.6f}"
        )

    except Exception as e:
        return render_template('index.html', prediction_text=f"Error: {str(e)}")

if __name__ == '__main__':
    app.run(debug=True)
```