📁 Roadmap

```
image_recommender/
├── main.py
├── feature_extraction/
│   ├── color_histogram.py
│   ├── cnn_embedding.py
│   ├── third_metric.py        # z. B. Gabor oder Autoencoder
│   └── generator.py           # Speicherfreundlicher Bild-Loader
├── database/
│   └── image_db.py
├── similarity/
│   ├── color_similarity.py
│   ├── embedding_similarity.py
│   ├── third_similarity.py
│   └── metric_combination.py  # Kombinierte Ähnlichkeitsberechnung
├── ann_search/
│   └── faiss_index.py
├── visualization/
│   └── umap_projection.py
├── ui/
│   └── streamlit_app.py
├── evaluation/
│   ├── evaluate_metrics.py    # Präzision, Vergleich
│   └── runtime_profiling.py   # cProfile + Analyse
├── data/
│   ├── raw_images/
│   ├── embeddings.npy
│   ├── combined_scores.json
│   └── metadata.sqlite
├── utils/
│   └── image_utils.py
├── tests/
│   ├── test_similarity.py
```

```
│   ├── test_db.py
│   └── test_generator.py
└── README.md
```

🛠️ Entwicklungsschritte (verbessert + erweitert)

✅ Phase 1: Feature-Extraktion (3 Metriken)

> Ziel: Farbbasierte, neuronale und semantische Features

- `color_histogram.py`: HSV-Histogramm (OpenCV)
- `cnn_embedding.py`: ResNet50-Features (Keras)
- `third_metric.py`:
  - Option 1: Gabor-Filter + PCA
  - Option 2: Autoencoder-Bottleneck-Vektor
  - Option 3: CLIP-Embedding (OpenAI, nur Text-Bild optional)

➡️ Output: `.npy`-Dateien mit Feature-Vektoren pro Bild

✅ Phase 2: Relationale Datenbank (Metadata & Pfade)

> Ziel: SQLite-DB für Pfad, Größe, Auflösung, Vektorpfad

- `image_db.py`: `create_table()`, `insert()`, `query_by_id()`
- Nutzung von `sqlite3`
- Zusatz: Einbindung von EXIF-Daten (Kameramodell etc.)

✅ Phase 3: ANN-Suche mit Faiss (Skalierung)

> Ziel: Ähnliche Bilder schnell finden (top-k Suche in Sekunden)

- `faiss_index.py`: Index trainieren, laden, durchsuchen

- Optimierung: Auf Embedding-Vektoren reduziert

✅ Phase 4: Kombination von Metriken

> Ziel: Ähnlichkeit auf Basis mehrerer Metriken berechnen

- `metric_combination.py`:

```
def combine_scores(score1, score2, alpha=0.5):
    return alpha * score1 + (1 - alpha) * score2
```

- Unterstützung für:
  - Kombination zweier Bilder (multi-query)
  - Kombination mehrerer Metriken (z. B. CNN + Histogramm)

✅ Phase 5: Dimensionalität & Visualisierung

> Ziel: Position aller Bilder im 2D/3D-Raum darstellen

- `umap_projection.py`:
  - UMAP, t-SNE, optional T-MAP
  - interaktive Plotly-Grafik mit Hover-Image
- Diskussion in Report: Cluster sinnvoll? Ausreißer?

✅ Phase 6: Streamlit-GUI mit Useroptionen

> Ziel: Upload → Anzeige Top-5 ähnlicher Bilder

- `streamlit_app.py`:
  - Upload-Button
  - Dropdown: Auswahl der Metrik(en)
  - Vorschau ähnliche Bilder

- Optional: zwei Bilder gleichzeitig hochladen (kombinierte Suche)

✅ Phase 7: Bewertung & Optimierung

> Ziel: Laufzeitanalyse + Qualität der Metriken

- `evaluate_metrics.py`:
  - Precision@5, Recall, ggf. Manuelles Rating
- `runtime_profiling.py`:
  - Nutzung von `cProfile`, Visualisierung mit Snakeviz

✅ Phase 8: Tests + Clean Code + Dokumentation

- Unit-Tests:
  - `pytest` für alle Funktionen (Metriken, DB, Generator)
- `README.md` + PDF:
  - Motivation, Designskizze, Beschreibung Metriken & Ergebnisse
- Optional:
  - Dockerfile für Deployment
  - GitHub Actions CI-Test-Workflow