Meeting Brief — Abschlussprojekt Reiseinfo

Kurzleitfaden für das Meeting (1 Stunde): Zusammenfassung, Demo-Schritte und kurze Antworten auf erwartete Fragen.

1) Kurz (30–60s) — Was wir haben

- Full-Stack MVP: Backend (Node.js + TypeScript + Express + Prisma) und Frontend (React + TypeScript + Vite).
- Datenmodell: `Destination`, `Tour`, `Booking` (Prisma + SQLite dev DB: `backend/prisma/dev.db`, Seed-Daten vorhanden).
- Dev-Setup: Backend läuft auf `http://localhost:4000`; Frontend served von Vite (Local URL in Terminal, typ. `http://localhost:5173` oder `5175`). Vite proxy leitet `/api` an das Backend weiter.
- Repo: top-level Hilfsdateien vorhanden: `.env.example`, `docker-compose.yml`, `LICENSE`, `.gitignore`, sowie einfache `Dockerfile`s für `backend` und `frontend`.

2) Demo-Ablauf (was du zeigen solltest)

1. Lokales Starten (schnell):

```
# Backend
cd backend
npm install
npm run dev

# Frontend (neues Terminal)
cd frontend
npm install
npm run dev
```

2. Browser öffnen: Local URL aus dem Vite-Terminal (z. B. `http://localhost:5175/`).

3. Im Browser zeigen:

- `Home` — kurze Projektübersicht.
- `Destinations` — Liste der Destinationen (geladen über `GET /api/destinations`).
- `TourDetails` — Detailansicht einer Tour; zeige das Booking-Formular und lege eine Test-Buchung an (POST `/api/bookings`).

4. Optional: Terminal/Logs zeigen (z. B. Vite-Log oder Backend-Log), um zu belegen, dass Requests ankommen.

3) Wichtige Befehle (für Fragen oder schnelles Debugging)

Start/Stop/Checks

```
# Prüfen, ob Backend antwortet
curl -sS http://localhost:4000/api/destinations | jq

# Prüfen über Vite-Proxy
```

```
curl -sS http://localhost:5175/api/destinations | jq

# Docker Compose (falls verwendet)
docker compose up --build
```

Port-Konflikt (häufige Frage)

- Wenn `docker compose` scheitert wegen `address already in use` für Port `5173`, lief bereits ein Vite-Server. Beende ihn oder mappe einen anderen Host-Port im `docker-compose.yml`.

4) Kurze Antworten auf erwartete Fragen

- Wer kann den Code sehen? — Das GitHub-Repo ist aktuell öffentlich; bestimmte Collaborators (`Abdullah-Jlilati`, `Maik-Protze`) haben Admin-Rechte.
- Welche Technologien? — Backend: Node.js, TypeScript, Express, Prisma (SQLite). Frontend: React, TypeScript, Vite, React Router.
- Welche Endpunkte? — z. B. `GET /api/destinations`, `GET /api/tours/:id`, `POST /api/bookings`, `GET /api/bookings`.
- Ist das produktionsbereit? — Noch nicht vollständig: Dockerfiles sind dev-orientiert; fehlen Tests, Produktions-Build/Deployment, und Secret-Management.

5) Nächste Schritte (kurz vorschlagen)

- Kurzfristig (1–2 Tage): README erweitern, `.dockerignore` hinzufügen, produktionsfähige Dockerfiles (Build → Serve), einfache Integrationstest(e) für Buchungs-Flow.
- Mittelfristig: CI/CD Pipeline erweitern (Deploy-Stage), Deployment (Azure/Heroku), UX-Verbesserungen, Validierung & Fehlerhandling.

6) Ein Satz, den du im Meeting sagen kannst

„Wir haben ein Full-Stack-MVP fertiggestellt mit einer Express/Prisma-API und einer React/Vite-UI. Im Moment zeigen wir live die Destinations-Liste und erstellen eine Test-Buchung — als nächste Schritte planen wir produktionsfähige Docker-Images, Tests und eine kurze README mit Deploy-Anleitung.“

Datei: `MEETING_BRIEF.md` (im Repo-Root). Wenn du willst, kann ich daraus sofort eine PDF generieren oder eine kurze `README.md` erstellen, die du im GitHub-Repo anzeigst.