AI-Assisted Full-Stack Development: Building a Booking App - Concise Student Guide

Generated on December 03, 2025

Table of Contents

Overview {#overview}

Hands-on course: Build a booking app using AI tools (Antigravity, QWEN, Grok, Google Stitch) + Linux/Docker/Git. Reverse engineering flow: WHAT → WHERE → HOW → WHY → WHEN.

Objectives: Full-stack app dev, AI proficiency, pro workflows, deployment.

Prerequisites: Basic coding; WSL2 (Windows); GitHub; tool accounts.

Structure:

	Session		Focus	Duration	Outputs
1	Prototyping	1h	Visual prototype		
2	Environment	2h	Setup + scaffold		
3	Development	2h	Full-stack code		
4	Enhancement	2h	Refined UI/code		
5	Deployment	2h	Live app + repo		

AI Tools Matrix:

Task	Tool	Why
Prototyping	Antigravity	Visual speed
Code Gen	QWEN CLI	Structured
Logic	Grok	Deep convo
UI/UX	Google Stitch	Pro designs
Deploy	Manual + AI	Control

Session 1: Rapid Prototyping with Antigravity {#session-1}

Objective: Visual booking app prototype.

Breakdown: Intro (10m), Setup/UI (15m), Functionality (15m), Testing (15m), Wrap (5m).

Activities

1. **Intro**: Antigravity features (drag-drop, data modeling, preview). Tour interface.

2. **Setup/UI**:
    - New project: "Booking App" template.
    - Structure: `pages/`, `components/`, `data-models/`, `assets/`.
    - Landing: Header (logo, nav, avatar); Hero (msg, search, CTA); Grid (4 cards: img, name, price, "Book").
3. **Functionality**:
    - Detail page: Img, desc, price, calendar, form (name/email/phone/date/time/notes).
    - Logic: Validation, submit → modal.
    - My Bookings: Table (ID, service, date/time, status); Cancel action.
    - Models:

```text
Service
id: auto
name: string
description: text
price: number
duration: number
category: string
image_url: string
```

```text
Booking
id: auto
service_id: reference
user_email: string
user_name: string
booking_date: date
booking_time: time
status: enum[confirmed, pending, cancelled]
created_at: timestamp
```

4. **Testing**: Flow (card → form → list); Validation (empty/invalid/past). Enh: Filters (cat/price/date), loading/notifs, responsive.
5. **Wrap**: Built: Pages/forms/models. Limits: Mock data, no auth.
   Homework: Export JSON for Session 2.

Session 2: Linux/Docker & Code Generation {#session-2}

Objective: Dev env + initial code. Duration: 2h.

Part A: Linux/WSL (45m)

1. Setup: WSL2/Ubuntu; cmds (`ls/cd/mkdir/nano/sudo`); `apt update/install` .
2. Stack: Python 3.11, Node/npm, Git, VS Code WSL.
3. Structure:

```
Bash

mkdir booking-app && cd $_ && mkdir backend frontend database
```

Part B: Docker (30m)

1. Concepts: Containers vs VMs; Images/containers/registries; Dockerfile.
2. Practical: Docker Desktop/WSL; `docker run hello-world` .

```
dockerfile

FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

3. Compose: `docker-compose.yml` for app+DB.

Part C: QWEN (45m)

1. Install: `pip install qwen-cli`; `qwen config --api-key KEY`.
2. Translate: Export Antigravity; Gen code:

```
Bash

qwen generate "FastAPI backend for booking app w/ SQLite"
qwen generate "React frontend w/ booking form"
```

3. Structure:

```
text

booking-app/
├── backend/ (app.py, models.py, requirements.txt, Dockerfile)
├── frontend/ (package.json, src/, Dockerfile)
└── docker-compose.yml
```

Homework: Docker run services; Seed data.

Session 3: Deep Development with Grok AI {#session-3}

Objective: Full-stack build. Duration: 2h.

Part A: Schema (30m)

1. Analyze: Visual → SQL relations/constraints.
2. Schema:

```SQL
CREATE TABLE services (
  id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  description TEXT,
  price DECIMAL(10,2),
  category TEXT,
  duration_minutes INTEGER,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE bookings (
  id INTEGER PRIMARY KEY,
  service_id INTEGER,
  user_email TEXT NOT NULL,
  user_name TEXT NOT NULL,
  booking_date DATE NOT NULL,
  booking_time TIME NOT NULL,
  status TEXT DEFAULT 'pending',
  FOREIGN KEY (service_id) REFERENCES services(id)
);
```

Part B: Backend (45m)

1. FastAPI: Pydantic models, CRUD.

```
Python

from fastapi import FastAPI
from pydantic import BaseModel
app = FastAPI()
class BookingCreate(BaseModel):
  service_id: int
  user_email: str
  user_name: str
  booking_date: str
  booking_time: str
@app.post("/bookings/")
async def create_booking(booking: BookingCreate):
  pass  # DB insert
```

2. Endpoints: GET/POST services/bookings; User bookings; Update/cancel.
3. Integrate: SQLAlchemy, pooling, errors.

Full Code: `requirements.txt` (fastapi/uvicorn/sqlalchemy/pydantic); `models.py/database.py/schemas.py/main.py`. Run: `uvicorn main:app --reload`.

Part C: Frontend (45m)

1. React: ServiceCard.

```
JavaScript

const ServiceCard = ({service, onBook}) => (
  <div className="service-card">
    <h3>{service.name}</h3>
    <p>{service.description}</p>
    <button onClick={() => onBook(service.id)}>
      Book Now - ${service.price}
    </button>
  </div>
);
```

2. Pages: Listing/form/bookings; Router; API layer.
3. Test: Connect, flow, errors.

   Homework: Add auth/payment sim; E2E test.

Session 4: UI/UX Enhancement & Tool Limits {#session-4}

Objective: Refine + critique AI. Duration: 2h.

Part A: Analysis (30m)

1. Grok: Inconsistent code, no context/sec/perf.
2. QWEN: Pattern-limited, no arch/errors.
3. Antigravity: Generic UI, poor responsive/custom/prod perf.

Part B: UI/UX (45m)

1. Stitch/FigJam: Design system, components, responsive.
2. Redesign: Flows, wires, palette/typo, mobile-first.
3. Impl:

```css
:root {
  --primary: #3b82f6;
  --secondary: #10b981;
  --radius: 12px;
  --shadow: 0 4px 6px -1px rgba(0,0,0,0.1);
}
.booking-card {
  border-radius: var(--radius);
  box-shadow: var(--shadow);
  transition: transform 0.2s;
}
```

Part C: Refactor (45m)

1. Antigravity: Layout gen.
2. QWEN: Logic: `qwen generate "Refactor validation w/ errors"`.
3. Grok: Features (real-time check/conflict/email).
4. Manual: Perf/accessibility/browser tests.

   Homework: Code redesigned UI; Doc AI changes.

Session 5: Deployment & Workflow {#session-5}

Objective: Deploy + collab. Duration: 2h.

Part A: Docker (45m)

1. Multi-stage:

```
dockerfile

FROM python:3.11-slim AS builder
COPY requirements.txt .
RUN pip install --user -r requirements.txt
FROM python:3.11-slim
COPY --from=builder /root/.local /root/.local
COPY . /app
WORKDIR /app
CMD ["uvicorn", "app:app", "--host", "0.0.0.0"]
```

2. Compose:

```
YAML

version: '3.8'
services:
  backend:
    build: ./backend
    ports: ["8000:8000"]
    environment:
      - DATABASE_URL=sqlite:///./bookings.db
  frontend:
    build: ./frontend
    ports: ["3000:3000"]
    depends_on: [backend]
```

3. Volumes: DB/logs/env.

Part B: Git (45m)

1. Setup:

```
Bash

git init && git add . && git commit -m "v1.0" && git branch -M main
git remote add origin https://github.com/user/booking-app.git && git pus
```

2. Strategy: main/develop; Feat branches (feat/form); PRs.
3. Review: AI code audit, sec/perf/docs.

Part C: Deploy (30m)

1. Options: Render (easy), AWS EB, DO App.
2. GitHub Actions:

```
YAML

name: Deploy
on: [push]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - run: curl -X POST ${{ secrets.RENDER_HOOK }}
```

3. Monitor: Logs (Sentry), perf, backups.

Showcase: Repo, live app, diagram, AI lessons.

Resources & Assessment {#resources}

Criteria:

1. Tech (40%): App/Docker/Git.
2. AI (30%): Prompts/eval/tool use.
3. Practices (20%): Quality/docs/deploy.
4. Reflection (10%): Limits/extend/solve.

Resources: Templates, cheat sheets (Linux/Docker/Git), prompts, troubleshooting, rubric.

Tips: Specific prompts; Test AI output; Ext: Stripe/JWT/Postgres. Share repo for feedback!

Markdown for PDF: Use Typora/Obsidian export, or Pandoc (`pandoc guide.md -o guide.pdf` *). Customize as needed.*