Brainstorm: AI-Project

1. Project Name

1.1. Origin

- Part of the Poppy Corporation, along with
  - Poppy Stays → Web Fundamentals
    (P.O.P.P.Y = Poppy Offers Perfect Places for You)
  - Poppy Space → Project Lab
    (P.O.P.P.Y. = Poppy's Observation & Pioneering of Planets for You)

1.2. Poppy Universe

- AI Astronomy / Recommendation Engine
- P.O.P.P.Y = Precise Observations & Planetary Predictions for You

2. Goal / Objective

- personalized astronomical recommendations based on user location, time, and preferences

3. Concept :

- Users log in and give location, time, etc.
- Optional filters (skill level, telescope <-> naked eye, interests, etc.)
- Start with approximate visibility (east, north, etc.)
- Upgrade later to nearly exact to exact positions (altitude ranges)
- Recommendations personalized by user preferences/history

4. Users/Target Audience

- skill levels: beginner → advanced
- big location (northern/southern hemisphere)
- interests: planets, stars, galaxies, constellations

5. Data:

-

5.1. Approximate Locations / MVP

- Objects stored with rough directions (east, north, etc.)
- Visible months stored instead of exact days
- Hemisphere included (north, south, both)
- Difficulty/popularity stored for ranking
- Model filters objects by month, hemisphere, direction, and user prefs
- Interactions tracked for personalization
- No external API needed, works for any year approximately

PROBLEM:

➔ Over time, location data can become wrong. For example, Jupiter was seen in March and September in 2025, but in 2045, it might only be visible in March. So, old data will become inaccurate.

5.2. Exact / upgraded version → ONLY WHEN EXTRA TIME!!!

- No need to add new columns for altitude range and azimuth
- Exact positions are calculated dynamically at request time using
    - User's latitude & longitude
    - Datetime of recommendation request
    - Either via C# calculations or an API (API → Astronomy API)
- Model uses these calculated values as extra features for recommendations
- Provides "nearly" exact position for the user's datetime
- Approximate columns can stay as a fallback
- Optional, done only if time allows.

PROBLEM:

➔ Orbital calculations are very hard to program. External Api calls are always a risk of both security and Space Complexity.
➔ Idea: Use API first, then try to code it myself!

5.3. Minimal Database Tables

- Users
  - ID → 1
  - Name → “Rynn”
  - Password → “ABC”
  - Big Location → “Northern Hemisphere”
  - Skill level → “Beginner”

➔ Note: Exact location not stored → Passed dynamically for recommendations (GDPR)!

- Objects Table
  - ID → 1
  - Name → “Saturn”
  - Type → “Planet”
  - Hemisphere → “Northern”
  - Visible Months → “Aug, Sep, Oct”
  - Direction → “East”
  - Difficulty → “Beginner”
  - Popularity → “8.5”

➔ Note: Altitude Range will only be used for the exact location when time allows it.

- Interaction Table
  - ID → 1
  - User_ID → 1
  - Object_ID → 1
  - Timestamp → 2025-09-27-21:00
  - Rating → 5

6. AI/Recommendation logic

- Filtering by month, hemisphere, direction, and user preferences
- Ranking by altitude, magnitude, popularity, difficulty
- Exact positions used as extra features if time allows

7. API/ External integrations

- Optional: AstronomyAPI for exact positions
- Input: Latitude, longitude & Datetime → Output: altitude, azimuth, magnitude, phase
- Fallback: approximate DB values

8. ML & Forecasting (optional)

- Analyze user interactions to predict preferred objects
- Track popularity trends per skill level, hemisphere

9. Frontend/presentation

Approach	Estimated Time	Components Included	Pros	Cons
Little Time Left	~ 10 – 15 hrs.	- Basic HTML / CSS / JS frontend - Node.js / Python backend - SQL Database - C# Model Call - Optional Simple Sky map	- Fast MVP - Easy to debug - Minimal setup	- Basic visuals - Limited UX
More Time Left	~ 30 – 50 hrs.	- Full Vue + Vite Frontend - Node.js / Python backend - SQL Database - C# Model call - Interactive sky map - Multiple pages - Advanced visuals	- Polished UX - Interactive - Expandable	- Higher Learning curve - More debugging

NOTES:

- Time Estimated includes all listed components, including optional visualizations
- C# AI model call: ~1 – 2 hrs in both cases
- Database setup same as Poppy Stays (users, objects, interactions, etc.)

9.1. Little time left

- Simple web page or simple app to show recommendations
  - Minimal & Simple Html/Css/JS
- Backend handles user login & database access
  - Node.js or Python handles basic logic & database access
- Calls the C# AI model for recommendations dynamically

9.2. More time left

- Full Website with front & backend
  - Different pages for user profile, recommendations, etc.
  - Frontend: Vue + Vite (interactive & visually appealing)
    - Like Poppy Stays → Web Fundamentals
  - Backend: Node.js (or Python) handles user management, API calls, and DB
  - Backend calls the C# AI model whenever recommendations are needed. Security / Hacking AI
- API key protection <-> Own calculations if time allows!
- Exact user location is not stored
- Password hashing

10. AI-Ethics

- Privacy → exact location never stored
- Transparency: explain how recommendations are generated!!!!!

11. Challenges/Risks

- Orbital Calculations are tough to implement from scratch
  - BUT payoff will be very high (full control security-wise + just super cool)
- API request fails or is down
  - API is easier but has external drawbacks that can't be managed by me.
- Time constraints for exact positioning

12. Stretch Goals/Extras

- Sky chart visualization
- Advanced ML-Based Personalization
- Upgrade to exact positions if time allows

END
