ExoHabitAI

Milestone 3 - Frontend in Next.js

Name: Satya Sri Dheeraj M
Date: 09 February 2026

1. Architecture

The frontend is built with Next.js 14 using the App Router architecture. The application follows a modular structure with dedicated directories for components, hooks, utilities, and types. React Three Fiber handles 3D rendering are used. State management uses React hooks for local state and custom hooks for API integration.

2. Implementation Components

a. Core Structure

The application uses Next.js file-based routing with three primary routes: dashboard (/) for visualization, prediction form (/predict) for parameter input, and 404 handler. TypeScript interfaces in types.ts define strict contracts for API responses, ensuring type safety across components. The API client in lib/api.ts provides a centralized abstraction layer for all backend communication with error handling and response normalization.

b. 3D Visualization System

The visualization layer uses React Three Fiber to render an orbital ring interface displaying planets organized by habitability probability. Three concentric rings represent high (≥70%), moderate (50-70%), and low (<50%) confidence candidates. Each ring rotates at different speeds while maintaining fixed billboard orientation to ensure labels always face the camera.

Planet components render as glowing spheres with emissive materials, point lights, and HTML overlays for labels. Hover interactions scale planets 50% larger, display targeting rings, and show click prompts. The camera controller enables orbital navigation with configurable zoom, pan, and auto-rotation disabled by default to prevent disorientation.

Critical implementation detail: HTML labels from drei/Html render outside React's normal DOM hierarchy and require special z-index handling. When prediction modals open, the entire 3D scene is hidden using visibility:hidden to prevent label bleed-through, as standard overlay techniques cannot block these labels.

c. UI Components

The dashboard features a left sidebar with system health indicators and statistical metrics including total habitable worlds, discovery rate, average habitability score, and diversity index. The prediction form implements multi-step navigation with validation at each stage, displaying real-time feedback for invalid inputs and parameter constraints.

Results display combines 3D planet visualization with detailed prediction metrics in a modal overlay. The modal uses pure black (#000000) background with solid card

backgrounds (#0a0e1a) to ensure complete opacity. A summary popup provides executive analysis and key scientific factors with color-coded impact indicators (green for positive, red for negative, gray for neutral).

3. API Integration

The frontend communicates with the Flask backend through a centralized API client that abstracts endpoint complexity. Custom React hooks (useRanking, usePrediction) manage async state, loading indicators, and error conditions. The ranking hook fetches pre-computed habitability rankings with configurable count and threshold parameters, while the prediction hook handles form submission and response formatting.

Error handling distinguishes between network failures (connection errors, timeouts), client errors (400 invalid input), and server errors (500 prediction failures). User-facing error messages provide actionable feedback without exposing technical implementation details. Loading states prevent duplicate submissions and provide visual feedback during API calls.

4. Route Structure and Data Flow

The application implements three primary user flows corresponding to distinct routes:

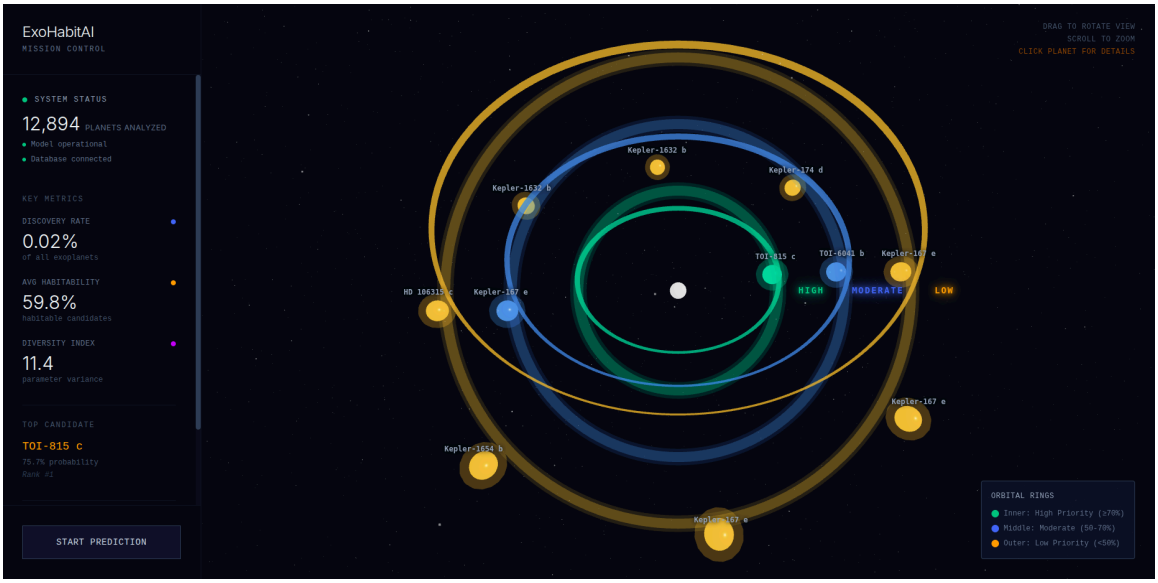
Route	Component	Backend Endpoint
/	MissionControl	GET /rank?top=20&threshold=0.0
/predict	PredictionForm	POST /predict (on form submit)
/predict (results)	ResultsDisplay	Uses cached prediction response

Dashboard workflow: Component mounts → useRanking hook calls /rank → Backend returns top 20 candidates → 3D scene renders orbital rings with planets → User clicks planet → Modal displays prediction details from ranking data.

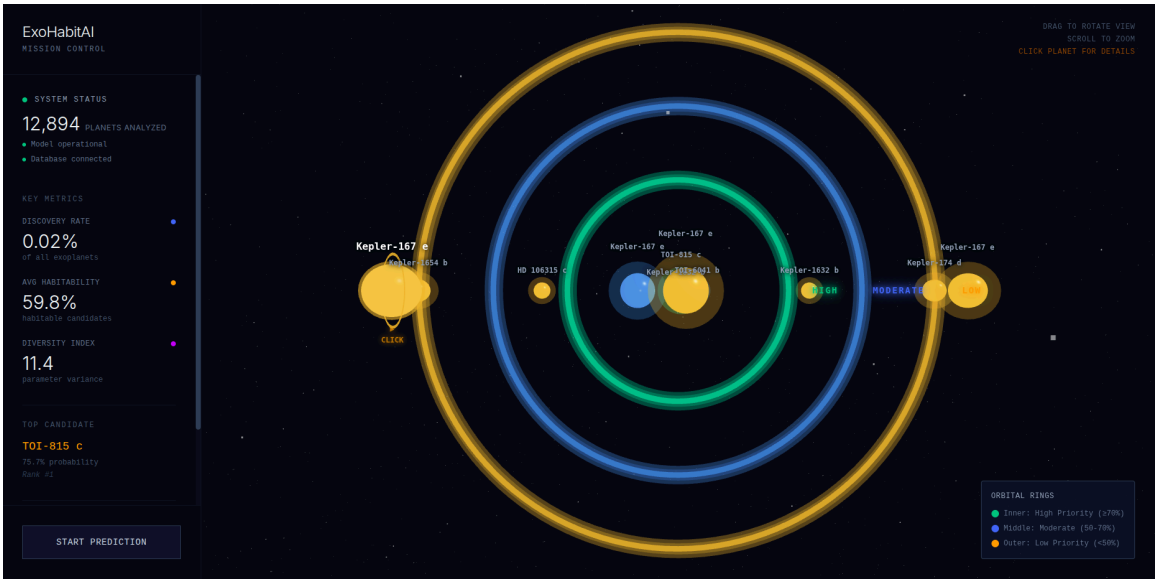
Prediction workflow: User navigates to /predict → Form validates inputs client-side → Submit triggers POST /predict → Backend returns prediction → Results page displays with 3D visualization → Summary button reveals scientific analysis.

5. User Interface Screenshots

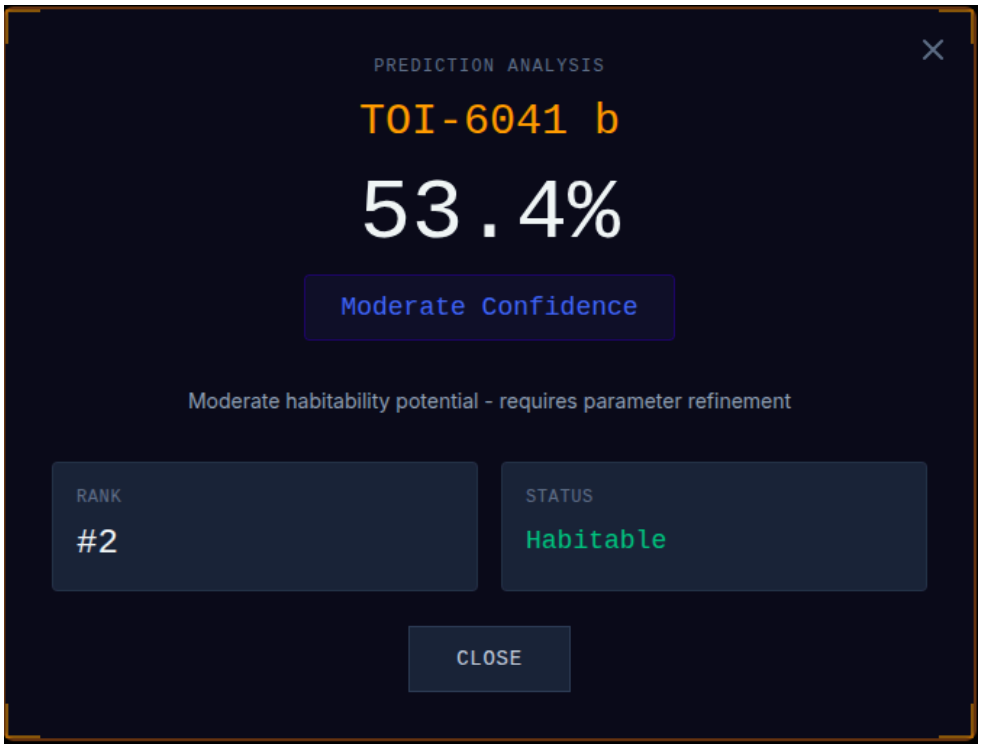
a) Dashboard - Orbital Rings Visualization



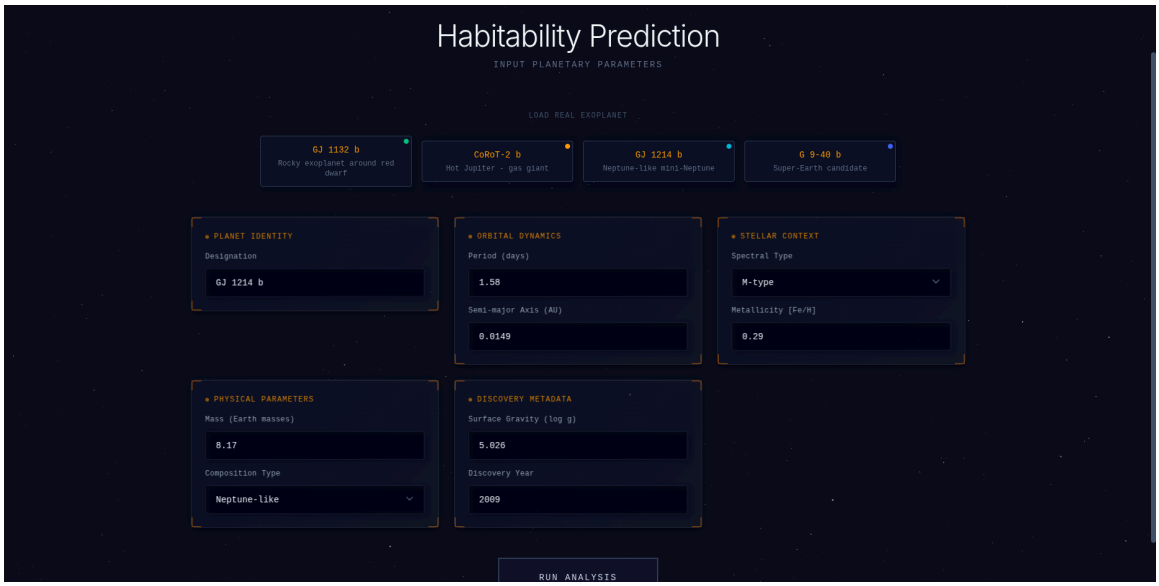
b) Planet Hover Interaction



c) Prediction Modal

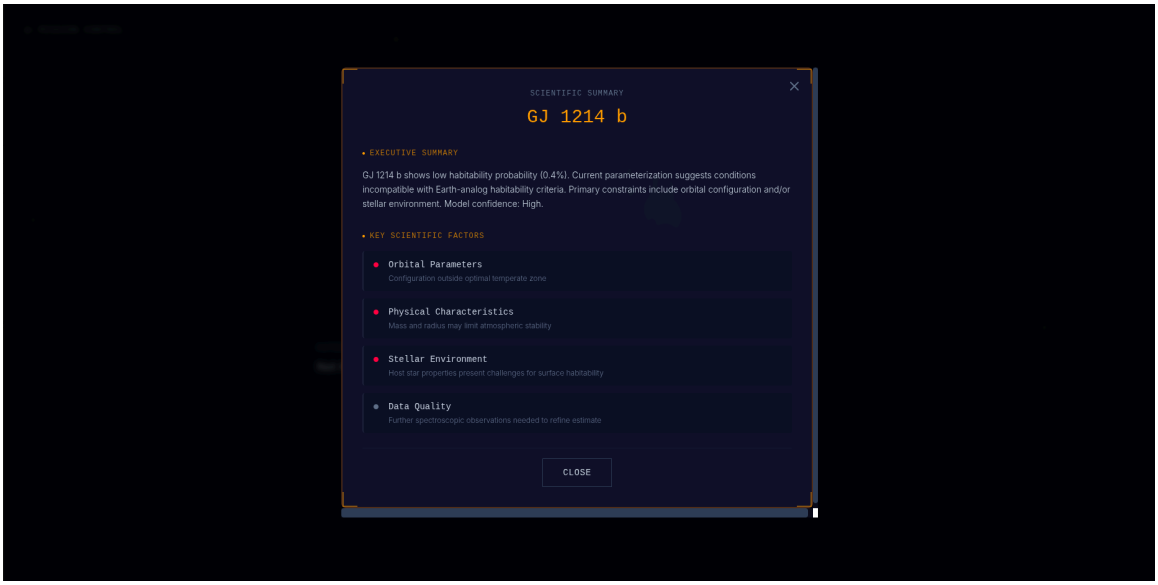


d) Prediction Form



e) Results Page with Summary

f) View Summary



6. Styling and Design

The application uses Tailwind CSS for utility-first styling with a dark theme based on slate colors (#050810 background, #0f172a cards, #1e293b accents). Typography uses monospace fonts (font-mono class) for scientific authenticity with uppercase tracking for headers. Color coding follows scientific conventions: emerald (#10b981) for high

confidence, blue (#3b82f6) for moderate, amber (#f59e0b) for low, and red (#ef4444) for negative indicators.

3D materials use emissive properties for self-illumination effects, creating the appearance of glowing celestial objects. Ring geometries use MeshBasicMaterial with transparency and DoubleSide rendering to ensure visibility from all angles. Planet spheres combine MeshStandardMaterial for realistic shading with high metalness (0.8) and low roughness (0.2) for a polished appearance.

Animation uses Framer Motion for modal transitions and page transitions. Modal overlays fade in with 0.3s duration while modal cards scale from 0.9 to 1.0 with vertical translation for depth perception. Loading states use pulse animations on skeleton elements to indicate async operations.

7. Performance Optimizations

Three.js components are dynamically imported with ssr:false to prevent server-side rendering errors. The Scene component loads with a fallback loading message while JavaScript initializes. Planet geometries are reused through component memoization rather than creating new instances per render.

The ranking endpoint response is cached in component state to prevent unnecessary API calls when clicking between planets. The 3D scene visibility toggling (visibility:hidden when modal opens) prevents expensive re-renders while maintaining scene state. Auto-rotation is disabled by default to reduce GPU usage during idle states.

HTML labels use drei/Html for DOM-based rendering, which is more performant than Three.js text rendering for dynamic content. Point lights use limited distance parameters (4-8 units) to prevent unnecessary shadow calculations beyond their effective range.

8. Conclusion

The ExoHabitAI frontend provides an immersive, scientifically accurate interface for exploring exoplanet habitability predictions. The 3D orbital visualization makes abstract probability rankings immediately understandable through spatial organization and visual encoding. The modular architecture separates concerns effectively, enabling independent development and testing of visualization, API integration, and UI components. Performance optimizations ensure smooth 60fps rendering even with dozens of animated planets, while comprehensive error handling provides resilient operation across network conditions and edge cases. The application successfully bridges the gap between complex machine learning predictions and accessible scientific communication.