# System Architecture – Vedic Astro App

## 1. Overview

This document describes the high-level architecture of the Vedic Astro application using:

1. **Vercel** for hosting the React/Vite frontend.
2. **Render.com** for hosting the Node/Express backend API.
3. **MongoDB Atlas** for the managed MongoDB database.
4. **GitHub** as the source-of-truth repository that triggers deployments.

The goal is a simple, cloud-native architecture that is easy to manage for a single developer.

---

## 2. Components

### 2.1 Client (User Browser)

1. Renders the React/Vite single-page application served from Vercel.
2. Communicates with the backend API over HTTPS using the base URL provided by `VITE_API_URL`.
3. Handles user interactions (login/guest flow, chart requests, etc.).

### 2.2 Frontend – Vercel

1. Hosts the static assets produced by `npm run build` in the `frontend` directory.
2. Uses the environment variable `VITE_API_URL` to know which backend URL to call.
3. Communicates directly with the backend on Render via HTTPS.
4. Automatically builds and deploys whenever code is pushed to the configured GitHub branch (typically `main`).

### 2.3 Backend API – Render Web Service

1. Node/Express application located under the `backend` directory.
2. Entry point is `server.js`.
3. Uses environment variables:
    1. `PORT` – port assigned by Render (e.g. `10000`).
    2. `MONGODB_URI` – MongoDB Atlas connection string.
    3. `JWT_SECRET` – secret key for signing JWT tokens.
    4. `EMAIL_USER`, `EMAIL_PASSWORD` – SMTP credentials (optional but required for email features).
4. Provides REST endpoints that the frontend calls for authentication and chart-related operations.
5. Connects to MongoDB Atlas using the `MONGODB_URI` connection string.

### 2.4 Database – MongoDB Atlas

1. Cloud-hosted MongoDB cluster running on MongoDB Atlas.

2. Exposes a connection string of the form:

```
mongodb+srv://<username>:<password>@<cluster-host>/?
retryWrites=true&w=majority&appName=<ClusterName>
```

3. Stores all persistent data (e.g. user accounts, saved charts, etc.).
4. Access is controlled via database users and IP allowlist (typically `0.0.0.0/0` for public cloud services).

### 2.5 GitHub Repository

1. Stores the full application source under a single repository with the structure:
    1. `backend/` – Node/Express backend.
    2. `frontend/` – React/Vite frontend.
2. Acts as the integration point for both Render and Vercel:
    1. A push to `main` triggers a new deployment on Render (backend).
    2. The same push triggers a new deployment on Vercel (frontend).

---

## 3. Data and Request Flow

### 3.1 High-Level Flow

1. A user opens the Vercel URL in a browser.
2. Vercel serves the pre-built frontend assets.
3. The frontend reads `VITE_API_URL` and sends API requests to the Render backend.
4. The Render backend reads `MONGODB_URI`, connects to MongoDB Atlas, and performs CRUD operations.
5. Responses are returned to the frontend, which updates the UI.

### 3.2 Mermaid Architecture Diagram

The following Mermaid diagram can be rendered by any Markdown viewer that supports Mermaid (such as GitHub, some documentation tools, or VS Code extensions):

```
flowchart LR
    subgraph Client["User's Browser"]
        UI[React UI]
    end

    subgraph Vercel["Vercel - Frontend"]
        FE[Static Assets
(Vite Build)]
    end

    subgraph Render["Render.com - Backend API"]
        BE[Node/Express
Backend]
    end
    subgraph Atlas["MongoDB Atlas"]
        DB[(MongoDB Cluster)]
    end

    subgraph GitHub["GitHub Repo"]
        REPO[vedic-astro-app]
    end

    UI <-- HTTPS --> FE
    FE <-- HTTPS (VITE_API_URL) --> BE
    BE <-- SRV Connection (MONGODB_URI) --> DB

    REPO -- "git push" --> Render
    REPO -- "git push" --> Vercel
```

---

## 4. Environments

Currently the architecture uses a single environment (production), but it can be extended to staging and development:

1. **Production**
    1. Vercel production deployment with its own `VITE_API_URL`.
    2. Render production backend with production `MONGODB_URI` and secrets.
    3. MongoDB Atlas production cluster.
2. **Staging (optional)**
    1. Create a second Render service and Vercel project.
    2. Use a separate Atlas database or cluster.
    3. Configure different environment variables (`MONGODB_URI`, `JWT_SECRET`, `VITE_API_URL`).

---

## 5. Operational Notes

1. **Scaling**
    1. Vercel automatically scales frontend delivery via its edge network.
    2. Render scales the backend service within the constraints of the chosen plan.
    3. MongoDB Atlas scales storage and performance as needed via cluster tier upgrades.
2. **Security**
    1. Secrets are stored only in:
        1. Render environment variables.
        2. Vercel environment variables.
        3. Local `.env` files that are never committed.
    2. Atlas access is restricted to authenticated users and configured IP ranges.
3. **Deployment Automation**

1. Commits to the tracked GitHub branch automatically trigger builds on both Render and Vercel.
2. No manual file uploads are necessary; everything flows from Git.

---

## 6. Summary

1. Frontend is hosted on **Vercel**, built from the `frontend` directory.
2. Backend API is hosted on **Render.com**, running from the `backend` directory.
3. Persistent data is stored in **MongoDB Atlas**, accessed via `MONGODB_URI`.
4. **GitHub** is the single source of truth and deployment trigger for both frontend and backend.