Source Code:

1) Index js

backend > index.js > ...

```javascript
import dotenv from "dotenv";
dotenv.config();

import express from "express";
import cors from "cors";
import connectDB from "./lib/db.js";
import authRoutes from "./routes/auth.route.js";
import blogRoutes from "./routes/blog.route.js";
import userRoutes from "./routes/user.route.js";
import commentRoutes from "./routes/comment.route.js";
import aiRoutes from "./routes/apiAi.route.js";
import cookieParser from "cookie-parser";
import rateLimit from "express-rate-limit";
import path from "path";
import { fileURLToPath } from "url";

const app = express();

// Required for Render to trust proxy headers (for secure cookies)
app.set("trust proxy", 1);

// Basic CORS setup (not needed if frontend and backend on same domain, but safe fallback)
app.use(cors({
  origin: true,
  credentials: true,
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 60 * 1000,
  max: 200,
  message: {
    status: 429,
    error: "Too many requests. Please try again after a minute.",
  },
});
app.use(limiter);
```

backend > index.js > ...

```
// Body parser and cookies
app.use(express.json({ limit: "10mb" }));
app.use(express.urlencoded({ extended: true, limit: "10mb" }));
app.use(cookieParser());

// Routes
app.use("/api/auth", authRoutes);
app.use("/api/blogs", blogRoutes);
app.use("/api/user", userRoutes);
app.use("/api/comments", commentRoutes);
app.use("/api/ai", aiRoutes);


//  Serve React Frontend

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

 //Serve static files from frontend build
app.use(express.static(path.resolve(__dirname, "client", "dist")));

// Avoid matching /api or any backend paths
app.get(/^\/(?!api).*/, (req, res) => {
  res.sendFile(path.resolve(__dirname, "client", "dist", "index.html"));
});


// Error Handler

app.use((err, req, res, next) => {
  console.error("SERVER ERROR:", err.stack || err.message);
  res.status(500).json({ message: "Internal server error" });
});


// Start Server
```

```
const PORT = process.env.PORT || 10000;
app.listen(PORT, "0.0.0.0", () => {
  console.log(`🚀 Server running on port ${PORT}`);
  connectDB();
});
```

2) Controller

a) apiAi.controller.js

backend > controller > apiAi.controller.js > ...

```
// Inside your controller (apiAi.controller.js)

import Groq from "groq-sdk";
const groq = new Groq({
  apiKey: process.env.GROQ_API_KEY, // make sure this is set in your .env file
});


export const generateBlogSuggestions = async (req, res) => {
  try {
    const { idea } = req.body;

    const completion = await groq.chat.completions.create({
      model: "llama3-70b-8192",
      messages: [
        {
          role: "system",
          content: `You are a blog writing assistant. Given a short idea, generate:
- A blog title (3-6 words)
- 3 bullet points (each 2-3 lines)

Respond ONLY in strict JSON format like:
{
  "title": "Your Blog Title",
  "points": [
    "Bullet point 1",
    "Bullet point 2",
    "Bullet point 3"
  ]
}`,
        },
        {
          role: "user",
          content: idea,
        },
      ],
      temperature: 0.7,
```

```
    });

    const raw = completion.choices[0]?.message?.content;

    // ✅ Extract valid JSON using RegEx
    const jsonMatch = raw.match(/{[\s\S]+}/);
    if (!jsonMatch) throw new Error("No valid JSON in AI response");

    const parsed = JSON.parse(jsonMatch[0]);

    return res.status(200).json(parsed);
  } catch (error) {
    console.error("🔥 Error generating blog suggestion:", error);
    return res.status(500).json({ error: "Something went wrong while generating suggestion" });
  }
};

```

b) auth.controller.js

backend > controller > auth.controller.js > ...

```javascript
import User from "../models/user.model.js";
import bcrypt from "bcryptjs"
import generateWebToken from "../lib/util.js";

export const signup = async (req , res) => {

    const {username , email , password} = req.body;
    try {

        if(!username || !email || !password){
            return res.status(400).json({message : "All fields are mandatory for signup"})
        }

        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(email) || password.length < 5) {
            return res.status(400).json({ message: "Please enter valid credentials." });
        }

        // check if user already exists
        const user = await User.findOne({username});
        if(user){
            return res.status(400).json({message : "username already exists please try another one"})
        }

        // check if email already exists
        const user2 = await User.findOne({email});
        if(user2){
            return res.status(400).json({message : "email already exists please try another one"})
        }

        // check if password is valid
        if(password.length < 5){
            return res.status(400).json({message : "Password must be at least 6 characters long"})
        }

        const salt = await bcrypt.genSalt(10)  // basically encrypt the password 10 times
        const hashedPassword = await bcrypt.hash(password , salt)
```

```
        const newUser = new User({
            username,
            email,
            password : hashedPassword
        })

        const token = generateWebToken(newUser._id , res)

        if(newUser){
            await newUser.save();
            return res.status(201).json({
                token : token,
                id : newUser._id,
                username : username,
                email : email
            })
        }else{
           return  res.status(400).json({message : "Invalid credentials"})
        }
    } catch (error) {
        console.log(error.message)
        return res.status(500).json({message : "Internal server error" })
    }
}

export const signin = async (req , res) => {
    const {username , password} = req.body;
    try {
        const existingUser = await User.findOne({username})
        if(!existingUser){
            return res.status(400).json({message : "User does not exist please try to signup"})
        }

        const passwordCompare = await bcrypt.compare(password , existingUser.password)

        if(!passwordCompare){
```

```
            return res.status(404).json({message : "email or password is incorrect"})
        }

        // if everything is fine then generate the token
        // and send the user data
        const token = generateWebToken(existingUser._id , res)
        res.status(200).json({
            token : token,
            _id: existingUser._id,
            firstname: existingUser.firstname,
            lastname : existingUser.lastname,
            username : existingUser.username,
            email: existingUser.email,
          });
    } catch (error) {
        return res.status(500).json({message : "Internal server error"})
    }
}

export async function logout(req, res) {
  try {
    res.clearCookie("token", {
    httpOnly: true,
    secure: true,
    sameSite: "None",
  });
  return res.status(200).json({ message: "Logged out successfully" });
  } catch (error) {
    console.log("Logout error:", error.message);
    res.status(500).json({ message: "Internal server error" });
  }
}

export async function authCheck(req, res) {
```

```
    try {
      res.status(200).json(req.user);
    } catch (error) {
      console.error("authCheck error:", error.message);
      res.status(500).json({ message: "Internal server error" });
    }
  }

```

c) blog.controller.js

backend > controller > blog.controller.js > ...

```
import Blog from "../models/blog.model.js";
import User from "../models/user.model.js";
import cloudinary from "../lib/cloudinary.js";

// give all the blogs of all the users
export const getAllBlogs = async (req, res) => {
  try {
    const blogs = await Blog.find()
      .sort({ createdAt: -1 })
      .populate("owner", "username");
    return res.status(200).json(blogs);
  } catch (error) {
    return res.status(500).json({ message: "Internal server error" });
  }
};

// give all the blogs of a particular user
export const getBlogById = async (req, res) => {
  const blogId = req.params.id;
  try {
    const blog = await Blog.findById(blogId).populate("owner", "username");
    if (!blog) {
      return res.status(404).json({ message: "Blog not found" });
    }
    return res.status(200).json(blog);
  } catch (error) {
    return res.status(500).json({ message: "Internal server error" });
  }
};

// create a blog (only authenticated users)
export const createBlog = async (req, res) => {
  const { title, content, topics } = req.body;
  const owner = req.user._id;

  try {
    if (!title || !content || !topics || !owner) {
```

```
    }

    const user = await User.findById(owner);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    let imageUrl = "";
    if (req.file && req.file.path) {
      imageUrl = req.file.path; // Cloudinary gives us secure URL
    }

    const newBlog = new Blog({
      title,
      content,
      topics,
      owner,
      image: imageUrl,
    });

    await newBlog.save();
    await User.findByIdAndUpdate(owner, { $push: { blogs: newBlog._id } });

    return res.status(201).json(newBlog);
  } catch (error) {
    console.error(error);
    return res.status(500).json(error);
  }
};

// delete the blogs (only owner can do)
export const deleteBlog = async (req, res) => {
  const blogId = req.params.id;
  const userId = req.user._id;
  try {
    const blog = await Blog.findById(blogId);
```

```
      return res.status(404).json({ message: "Blog not found" });
    }

    if (blog.owner.toString() !== userId.toString()) {
      return res
        .status(403)
        .json({ message: "Not authorized to delete this blog" });
    }

    await Blog.findByIdAndDelete(blogId);
    return res.status(200).json({
      message: "Blog deleted successfully",
      title: blog.title,
    });
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ message: "Internal server error" });
  }
};

// update the blog (only owner can do)
export const updateBlog = async (req, res) => {
  try {
    const { title, content, topics, image } = req.body;
    const blogId = req.params.id;
    const userId = req.user._id;

    // Check blog exists
    const blog = await Blog.findById(blogId);
    if (!blog) return res.status(404).json({ message: "Blog not found" });

    // Check ownership
    if (blog.owner.toString() !== userId.toString()) {
      return res.status(403).json({ message: "Unauthorized to update" });
    }

    let imageUrl = blog.image;
    if (image && image.startsWith("data:image")) {
      const uploadResponse = await cloudinary.uploader.upload(image);
      imageUrl = uploadResponse.secure_url;
    }

    // Update the blog
    blog.title = title;
    blog.content = content;
    blog.topics = topics;
    blog.image = imageUrl;

    await blog.save();
    res.status(200).json(blog);
  } catch (error) {
    console.error("Error updating blog:", error);
    res.status(500).json({ message: "Internal Server Error", error: error.message });
  }
};
```

d) user.controller.js

backend > controller > user.controller.js > ...

```javascript
import User from "../models/user.model.js";

export const addDetails = async (req, res) => {
  const userId = req.user._id;
  const { background, education }= req.body;
  console.log(background , education)
  if (!background || !education) {
    return res.status(400).json({message : 'Please fill all the details'});
  }

  try {
    const user = await User.findById(userId);
    if (!user) {
      return res.status(400).json({ message: "User not found" });
    }
    user.background = background;
    user.education = education;
    await user.save();
    return res.status(200).json({user});
  } catch (error) {
    return res.status(500).json({ message: "Internal server error" });
  }
};

export const getProfile = async (req, res) => {
  const userId = req.user._id;
  try {
    const user = await User.findById(userId);
    if (!user) {
      return res.status(400).json({ message: "User not found" });
    }
    return res.status(200).json({ user });
  } catch (error) {
    return res.status(500).json({ message: "Internal server error" });
  }
};
```

3)Routes

a) apiAi.route.js

backend > routes > apiAi.route.js > ...

```
import express from "express";
import { generateBlogSuggestions } from "../controller/apiAi.controller.js";
import { verifyToken } from "../middleware/auth.middleware.js";

const router = express.Router();

router.post("/suggest", verifyToken, generateBlogSuggestions);

export default router;
```

b) auth.route.js

backend > routes > auth.route.js > ...

```
import express from "express";
import {
  signup,
  signin,
  logout,
  authCheck,
} from "../controller/auth.controller.js";
import { verifyToken } from "../middleware/auth.middleware.js";

const router = express.Router();

router.post("/register", signup);
router.post("/login", signin);
router.post("/logout", verifyToken, logout);
router.get("/check-auth", verifyToken, authCheck);

export default router;
```

c) blog.route.js

backend > routes > blog.route.js > ...

```javascript
import express from "express";
import {  getAllBlogs, getBlogById, createBlog  , deleteBlog , updateBlog } from "../controller/blog.controller.js";
import { verifyToken } from "../middleware/auth.middleware.js";
import {upload} from '../middleware/multer.js';

const router = express.Router();

// GET /api/blogs -> get all blogs
router.get("/getBlogs", getAllBlogs);

// GET /api/blogs/:id -> get a specific blog by ID
router.get("/:id", getBlogById);

// POST /api/blogs -> create a new blog
router.post('/create', verifyToken, upload.single('image'), createBlog);

// Delete /api/blogs/deleteBlog/:id -> delete a specific blog by ID
router.delete("/delete/:id", verifyToken , deleteBlog);

// PUT /api/blogs/update/:id -> update a specific blog by ID
router.put("/update/:id", verifyToken, updateBlog);

export default router;
```

d) comment.route.js

backend > routes > comment.route.js > ...

```javascript
import express from "express";
import {
  addComment,
  getCommentsByBlogId,
  likePost
} from "../controller/comment.controller.js";
import { verifyToken } from "../middleware/auth.middleware.js";

const router = express.Router();

// POST /api/comments/:id -> add comment to a specific blog
router.post("/:id", verifyToken, addComment);

// GET /api/comments/:id -> get all comments for a specific blog
router.get("/:id", getCommentsByBlogId);

// // DELETE /api/comments/:id -> delete a specific comment
// router.delete("/:commentId", verifyToken, deleteComment);

// PUT /api/comments/:id -> like a specific blog
router.put("/:id" , verifyToken , likePost)

export default router;
```

e) user.route.js

backend > routes > user.route.js > default

```
import express from "express";
import {  getProfile, addDetails } from "../controller/user.controller.js";
import { verifyToken } from "../middleware/auth.middleware.js";

const router = express.Router();

// GET /api/user/profile -> get user profile
router.get("/profile", verifyToken, getProfile);

// POST /api/user/addDetails -> add user details
router.post("/addDetails", verifyToken, addDetails);


export default router;
```

Output:

1) Main page

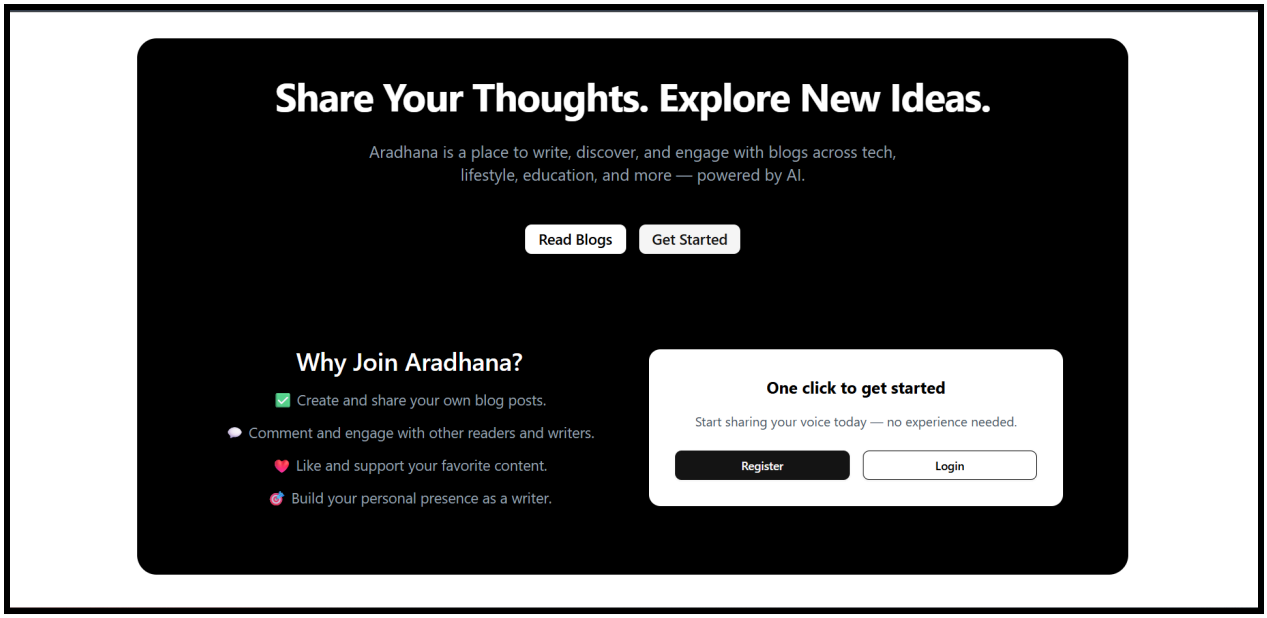

Figure 1: Main page

2) Registration Page:

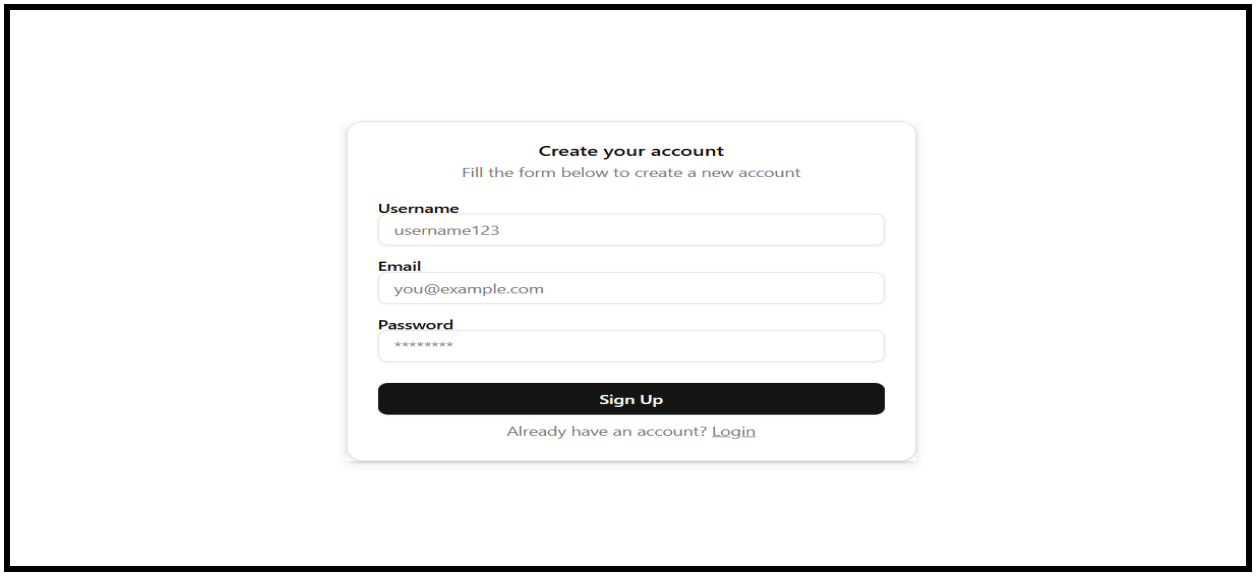

Figure 2 : Registration Page

3) Home page:

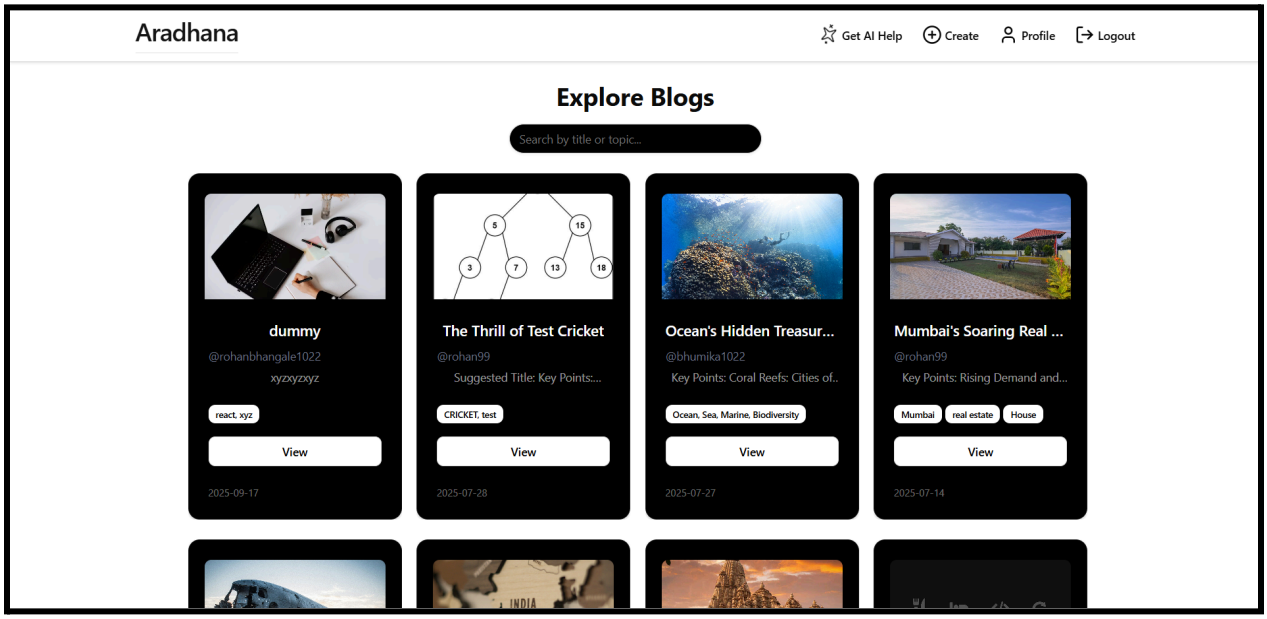

Figure 3 : Blog page showing blogs

4) Profile Page :

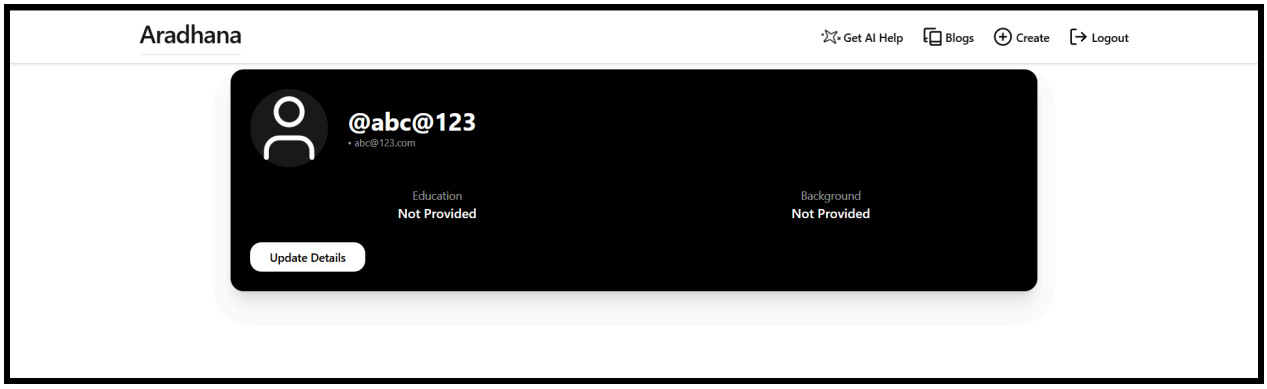

Figure 4 : Profile page

5) Create Blog Page:

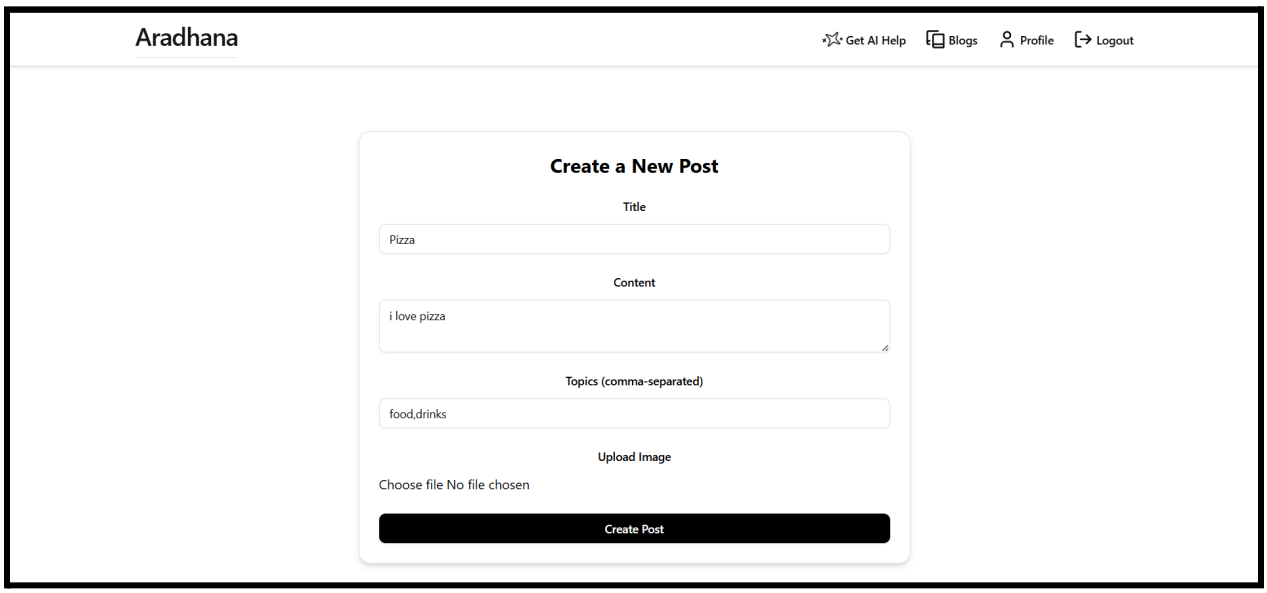

Figure 5 : Blogs Creations Page

6) Search Functionality:

Figure 6 : Search Functionality which helps search blogs based on topics entered