PROGRAM :

FRONTEND/

src/api/client.js :

```
import axios from 'axios';
const api = axios.create({
  baseURL: '/api',
  withCredentials: true,
  headers: {
    'Content-Type': 'application/json'
  }
});
export default api;
```

src/components/ProtectedRoute.jsx:

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
export default function ProtectedRoute({ children }) {
  const { user, loading } = useAuth();
  if (loading) return <div style={{ padding: 24 }}>Loading...</div>;
  if (!user) return <Navigate to="/login" replace />;
  return children;
}
```

Src/context/AuthContext.jsx :

```
import { createContext, useContext, useEffect, useState, useCallback } from 'react';
import api from '../api/client';
const AuthContext = createContext(null);
export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const fetchMe = useCallback(async () => {
    try {
      const { data } = await api.get('/auth/me');
      setUser(data);
    } catch (e) {
      setUser(null);
    } finally {
      setLoading(false);
    }  }, []);
  useEffect(() => {
    fetchMe();
  }, [fetchMe]);
  const login = async (username, password) => {
    await api.post('/auth/login', { username, password });
    await fetchMe();  };
  const register = async (username, email, password) => {
    console.log('Registering with:', { username, email, password });
    try {
      const response = await api.post('/auth/register', { username, email, password });
      console.log('Register response:', response.data);
    } catch (err) {
      console.error('Register error:', err.response?.data || err.message);
      throw err;
    }  };
  const logout = async () => {
    try {
      await api.post('/auth/logout');
    } finally {
      setUser(null);
    }  };
  return (
    <AuthContext.Provider value={{ user, loading, login, logout, register }}>
      {children}
    </AuthContext.Provider>
  );}
export function useAuth() {
  return useContext(AuthContext);
}
```

<u>Src/pages/Login.jsx :</u>

```
import { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { useAuth } from '../context/AuthContext.jsx';
export default function Login() {
  const { login } = useAuth();
  const navigate = useNavigate();
  const [form, setForm] = useState({ username: '', password: '' });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const onChange = (e) => setForm((f) => ({ ...f, [e.target.name]: e.target.value }));
  const onSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setLoading(true);
    try {
      await login(form.username.trim(), form.password);
      navigate('/');
    } catch (err) {
      setError(err?.response?.data?.message || 'Login failed');
    } finally {
      setLoading(false);
    }  };
  return (
    <div style={{ maxWidth: 420, margin: '24px auto' }}>
      <h2>Login</h2>
      <form onSubmit={onSubmit} style={{ display: 'grid', gap: 12 }}>
        <label>
          <div>Username</div>
          <input name="username" value={form.username} onChange={onChange} required />
        </label>
        <label>
          <div>Password</div>
          <input type="password" name="password" value={form.password} onChange={onChange} required />
        </label>
        {error && <div style={{ color: 'crimson' }}>{error}</div>}
        <button type="submit" disabled={loading}>{loading ? 'Logging in...' : 'Login'}</button>
      </form>
      <p style={{ marginTop: 12 }}> No account? <Link to="/register">Register</Link> </p>
    </div>  );}
```

Src/pages/Register.jsx :

```
import { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { useAuth } from '../context/AuthContext.jsx';
export default function Register() {
  const { register } = useAuth();
  const navigate = useNavigate();
  const [form, setForm] = useState({ username: '', email: '', password: '' });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const onChange = (e) => setForm((f) => ({ ...f, [e.target.name]: e.target.value }));
  const onSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setLoading(true);
    try {
      await register(form.username.trim(), form.email.trim(), form.password);
      // Back to login after successful registration
      navigate('/login');
    } catch (err) {
      setError(err?.response?.data?.message || 'Registration failed');
    } finally {
      setLoading(false);
    }  };
  return (
    <div style={{ maxWidth: 420, margin: '24px auto' }}>
      <h2>Register</h2>
      <form onSubmit={onSubmit} style={{ display: 'grid', gap: 12 }}>
        <label>
          <div>Username</div>
          <input name="username" value={form.username} onChange={onChange} required />
        </label>
        <label>
          <div>Email</div>
          <input type="email" name="email" value={form.email} onChange={onChange} required />
        </label>
        <label>
          <div>Password</div>
          <input type="password" name="password" value={form.password} onChange={onChange} required />
        </label>
        {error && <div style={{ color: 'crimson' }}>{error}</div>}
        <button type="submit" disabled={loading}>{loading ? 'Registering...' : 'Register'}</button>
      </form>
      <p style={{ marginTop: 12 }}>  Already have an account? <Link to="/login">Login</Link></p>  </div>);}
```

Src/pages/Todo.jsx :

```
import { useEffect, useMemo, useState } from 'react';
import api from '../api/client';
export default function Todo() {
  const [tasks, setTasks] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [editingId, setEditingId] = useState(null);
  const [filter, setFilter] = useState('all'); // all | active | completed
  const [q, setQ] = useState('');
  const loadTasks = async () => {
    setLoading(true);
    setError('');
    try {
      const { data } = await api.get('/tasks', { params: { status: filter, q } });
      setTasks(data);
    } catch (err) {
      setError(err?.response?.data?.message || 'Failed to load tasks');
    } finally {
      setLoading(false);  }  };
  useEffect(() => {
    loadTasks();
  }, [filter, q]);
  const onAddOrUpdate = async (e) => {
    e.preventDefault();
    setError('');
    try {
      if (editingId) {
        const { data } = await api.put(`/tasks/${editingId}`, { title: title.trim(), description });
        setTasks((prev) => prev.map((t) => (t._id === editingId ? data : t)));
        setEditingId(null);
      } else {
        const { data } = await api.post('/tasks', { title: title.trim(), description });
        setTasks((prev) => [data, ...prev]);
      }
      setTitle('');
      setDescription('');
    } catch (err) {
      setError(err?.response?.data?.message || 'Failed to save task');
    }  };
  const onEdit = (task) => {
    setEditingId(task._id);
    setTitle(task.title);
    setDescription(task.description || '');
  };
  const onCancelEdit = () => {
    setEditingId(null);
    setTitle('');
    setDescription('');
  };
  const onToggle = async (task) => {
    try {
      const { data } = await api.put(`/tasks/${task._id}`, { completed: !task.completed });
      setTasks((prev) => prev.map((t) => (t._id === task._id ? data : t)));
    } catch (err) {
      setError(err?.response?.data?.message || 'Failed to update task');
    }  };
  const onDelete = async (task) => {
    if (!confirm('Delete this task?')) return;
    try {
      await api.delete(`/tasks/${task._id}`);
      setTasks((prev) => prev.filter((t) => t._id !== task._id));
    } catch (err) {
      setError(err?.response?.data?.message || 'Failed to delete task');
    }  };
  const counts = useMemo(() => ({
    total: tasks.length,
    active: tasks.filter(t => !t.completed).length,
    completed: tasks.filter(t => t.completed).length,
  }), [tasks]);
  return (
    <div>
      <h2>My Tasks</h2>
      <div style={{ display: 'flex', gap: 8, alignItems: 'center', marginBottom: 12 }}>
        <button onClick={() => setFilter('all')} disabled={filter==='all'}>All ({counts.total})</button>
        <button onClick={() => setFilter('active')} disabled={filter==='active'}>Active ({counts.active})</button>
        <button onClick={() => setFilter('completed')} disabled={filter==='completed'}>Completed
({counts.completed})</button>
        <input
          placeholder="Search..."
          value={q}
          onChange={(e) => setQ(e.target.value)}
          style={{ marginLeft: 'auto' }}    />
      </div>
      <form onSubmit={onAddOrUpdate} style={{ display: 'grid', gap: 8, marginBottom: 16 }}>
        <input
          placeholder="Task title"
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          required  />
        <textarea
          placeholder="Description (optional)"
          value={description}
          onChange={(e) => setDescription(e.target.value)}
          rows={3}     />
        <div style={{ display: 'flex', gap: 8 }}>
          <button type="submit" disabled={!title.trim()}>
            {editingId ? 'Update Task' : 'Add Task'}
          </button>
          {editingId && (
            <button type="button" onClick={onCancelEdit}>Cancel</button>
          )}
        </div>
      </form>
      {error && <div style={{ color: 'crimson', marginBottom: 12 }}>{error}</div>}
      {loading ? (
        <div>Loading tasks...</div>
      ) : tasks.length === 0 ? (
        <div>No tasks found.</div>
      ) : (
        <ul style={{ listStyle: 'none', padding: 0, display: 'grid', gap: 8 }}>
          {tasks.map((t) => (
        <li key={t._id} style={{ border: '1px solid #eee', borderRadius: 6, padding: 12, display: 'flex', gap: 12, alignItems:
'flex-start' }}>
          <input type="checkbox" checked={t.completed} onChange={() => onToggle(t)} />
          <div style={{ flex: 1 }}>
            <div style={{ fontWeight: 600, textDecoration: t.completed ? 'line-through' : 'none' }}>{t.title}</div>
            {t.description && <div style={{ color: '#555', marginTop: 4, whiteSpace: 'pre-wrap' }}>{t.description}</div>}
            <div style={{ color: '#888', marginTop: 6, fontSize: 12 }}>Created {new
Date(t.createdAt).toLocaleString()}</div>
          </div>
          <div style={{ display: 'flex', gap: 6 }}>
            <button onClick={() => onEdit(t)}>Edit</button>
            <button onClick={() => onDelete(t)} style={{ color: 'crimson' }}>Delete</button>
          </div>
        </li>))}</ul>)}  </div>);}
```

<u>Src/App.css :</u>

```
#root {
  max-width: 1100px;
  margin: 0 auto;
  padding: 0 1rem 2rem 1rem;
  text-align: left;
}
.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);}
@keyframes logo-spin {
  from {
    transform: rotate(0deg);  }
  to {
    transform: rotate(360deg);
  }}
@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }}
.card {
  padding: 2em;}
.read-the-docs {
  color: #888;}
```

<u>src/main.jsx :</u>

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import { BrowserRouter } from 'react-router-dom'
import { AuthProvider } from './context/AuthContext.jsx'
createRoot(document.getElementById('root')).render(
  <StrictMode>
```

```
    <BrowserRouter>
      <AuthProvider>
        <App />
      </AuthProvider>
    </BrowserRouter>
  </StrictMode>,
)
```
BACKEND /

Src/config/db.js :
```
import mongoose from 'mongoose';
import dotenv from 'dotenv';
dotenv.config();
const MONGO_URI = process.env.MONGO_URI || 'mongodb://127.0.0.1:27017/todo-list';
mongoose.set('strictQuery', true);
mongoose
  .connect(MONGO_URI, {
    autoIndex: true
  })
  .then(() => console.log('Connected to MongoDB'))
  .catch((err) => {
    console.error('MongoDB connection error:', err.message);
    process.exit(1);
  });
```
Src/controllers/taskController.js :
```
import Task from '../models/Task.js';
export async function createTask(req, res) {
  try {
    const { title, description = '' } = req.body;
    if (!title) return res.status(400).json({ message: 'title is required' });
    const task = await Task.create({ user: req.userId, title, description });
    return res.status(201).json(task);
  } catch (err) {
    console.error('Create task error:', err.message);
    return res.status(500).json({ message: 'Server error' });
  }}
export async function getTasks(req, res) {
  try {
    const { status = 'all', q = '' } = req.query;
    const filter = { user: req.userId };
    if (status === 'active') filter.completed = false;
    else if (status === 'completed') filter.completed = true;
    if (q) {
      filter.$or = [
        { title: { $regex: q, $options: 'i' } },
        { description: { $regex: q, $options: 'i' } }
      ];  }
    const tasks = await Task.find(filter).sort({ createdAt: -1 });
    return res.json(tasks);
  } catch (err) {
    console.error('Get tasks error:', err.message);
    return res.status(500).json({ message: 'Server error' });  }}
export async function updateTask(req, res) {
  try {
    const { id } = req.params;
    const { title, description, completed } = req.body;
    const task = await Task.findOne({ _id: id, user: req.userId });
    if (!task) return res.status(404).json({ message: 'Task not found' });
    if (title !== undefined) task.title = title;
    if (description !== undefined) task.description = description;
    if (completed !== undefined) task.completed = completed;
    await task.save();
    return res.json(task);
  } catch (err) {
    console.error('Update task error:', err.message);
    return res.status(500).json({ message: 'Server error' });
  }}
export async function deleteTask(req, res) {
  try {
    const { id } = req.params;
    const task = await Task.findOneAndDelete({ _id: id, user: req.userId });
    if (!task) return res.status(404).json({ message: 'Task not found' });
    return res.json({ message: 'Task deleted' });
  } catch (err) {
    console.error('Delete task error:', err.message);
    return res.status(500).json({ message: 'Server error' });
  }}
```

Src/middleware/auth.js :

```
import jwt from 'jsonwebtoken';
export default function auth(req, res, next) {
  try {
    const token = req.cookies?.token || (req.headers.authorization?.startsWith('Bearer ') ?
req.headers.authorization.split(' ')[1] : null);
    if (!token) return res.status(401).json({ message: 'Not authenticated' });
    const payload = jwt.verify(token, process.env.JWT_SECRET || 'dev_secret');
    req.userId = payload.id;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid or expired token' }); }}
```

src/models/Task.js :

```
import mongoose from 'mongoose';
const taskSchema = new mongoose.Schema(
  {
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    title: { type: String, required: true, trim: true },
    description: { type: String, default: '' },
    completed: { type: Boolean, default: false }
  },
  { timestamps: true }
);
export default mongoose.model('Task', taskSchema);
```

src/models/User.js :

```
import mongoose from 'mongoose';
const userSchema = new mongoose.Schema(
  {
    username: { type: String, required: true, unique: true, trim: true },
    email: { type: String, required: true, unique: true, lowercase: true, trim: true },
    password: { type: String, required: true, select: false }
  },
  { timestamps: true }
);
export default mongoose.model('User', userSchema);
```

src/routes/auth.js :

```
import { Router } from 'express';
import { register, login, me, logout } from '../controllers/authController.js';
import auth from '../middleware/auth.js';
const router = Router();
router.post('/register', register);
router.post('/login', login);
router.get('/me', auth, me);
router.post('/logout', auth, logout);
export default router;
```

src/routes/tasks.js :

```
import { Router } from 'express';
import auth from '../middleware/auth.js';
import { createTask, getTasks, updateTask, deleteTask } from '../controllers/taskController.js';
const router = Router();
router.use(auth);
router.get('/', getTasks);
router.post('/', createTask);
router.put('/:id', updateTask);
router.delete('/:id', deleteTask);
export default router;
```

FRONTEND /

Src/api/client.js :
```
import axios from 'axios';
const api = axios.create({
  baseURL: import.meta.env.VITE_API_BASE_URL || 'http://localhost:5000/api'
});
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});
export default api;
```
src/components/Navbar.jsx:
```
import { Link, useNavigate } from 'react-router-dom';
export default function Navbar() {
  const navigate = useNavigate();
  const user = JSON.parse(localStorage.getItem('user') || 'null');
  function logout() {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    navigate('/login');
  }
  return (
    <div className="navbar">
      <div className="left">
        <Link className="brand" to="/feed">Microblog</Link>
      </div>
      <div className="right">
        {user ? (
          <>
            <Link to="/feed">@{user.username}</Link>
            <Link to={`/u/${user.username}`}>Profile</Link>
            <Link to="/explore">Explore</Link>
            <Link to="/settings">Settings</Link>
            <button className="button secondary" onClick={logout}>Logout</button>  </>
        ) : (
          <>
            <Link to="/login">Login</Link>
            <Link to="/register">Register</Link>
          </>  )}
      </div></div>  );}
```
Src/components/ProtectedRoute.jsx :
```
import { Navigate, useLocation } from 'react-router-dom';
export default function ProtectedRoute({ children }) {
  const token = localStorage.getItem('token');
  const location = useLocation();
  if (!token) return <Navigate to="/login" state={{ from: location }} replace />;
  return children;
}
```
Src/pages/Explore.jsx :
```
import { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import api from '../api/client.js';
```

```
export default function Explore() {
  const [users, setUsers] = useState([]);
  const [search, setSearch] = useState('');
  const [page, setPage] = useState(1);
  const [hasMore, setHasMore] = useState(true);
  const [loading, setLoading] = useState(true);
  const [followingSet, setFollowingSet] = useState(new Set());
  const [requestedSet, setRequestedSet] = useState(new Set());
  const me = JSON.parse(localStorage.getItem('user') || 'null');
  useEffect(() => {
    load(1, true);
    (async () => {
      try {
        if (me?.username) {
          const { data } = await api.get(`/users/${me.username}/following`);
          const set = new Set((data || []).map((u) => u.username));
          setFollowingSet(set);
        } } catch (err) {
        console.error(err);
      }
    })();  }, []);
  function onSearch(e) {
    e.preventDefault();
    load(1, true);
  }
  return (
    <div>
      <h2>Explore</h2>
      <form onSubmit={onSearch} style={{ display: 'flex', gap: 8, marginBottom: 12 }}>
        <input className="input" placeholder="Search users by name or username" value={search} onChange={(e) =>
setSearch(e.target.value)} />
        <button className="button" type="submit">Search</button>
      </form>
      <div className="card">
        {users.length === 0 && !loading && <div>No users found</div>}
        {users.map((u) => (
          <div key={u.username} style={{ display: 'flex', justifyContent: 'space-between', padding: '10px 0', borderBottom:
'1px solid var(--border)' }}>
            <div>
              <div style={{ fontWeight: 600 }}>
                <Link to={`/u/${u.username}`}>@{u.username}</Link>
                {u.isPrivate && <span className="meta" style={{ marginLeft: 8 }}>🔒 Private</span>}
              </div>
              <div className="meta">{u.name}</div>
              {u.bio && <div style={{ marginTop: 6 }}>{u.bio}</div>}
            </div>
            <div style={{ display: 'flex', alignItems: 'center', gap: 8 }}>
              <div className="meta">Joined {new Date(u.createdAt).toLocaleDateString()}</div>
              {me?.username !== u.username && (
                u.isPrivate ? (
                  followingSet.has(u.username) ? (
                    <button className="button" onClick={() => toggleFollow(u)}>Unfollow</button>
                  ) : requestedSet.has(u.username) ? (
                    <button className="button secondary" disabled>Requested</button>
                  ) : (
                    <button className="button" onClick={() => toggleFollow(u)}>Request</button>
                  )
                ) : (
                  <button className="button" onClick={() => toggleFollow(u)}>
                    {followingSet.has(u.username) ? 'Unfollow' : 'Follow'}
                  </button>  ))}  </div></div>  ))}
        {hasMore && (
          <div style={{ display: 'flex', justifyContent: 'center', marginTop: 12 }}>
            <button className="button" onClick={() => load(page + 1)}>Load more</button>
          </div>  )}</div></div>);}
```

Index.html :

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Microblog</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

BACKEND /

Src/middleware/auth.js :

```
import jwt from 'jsonwebtoken';
export function authRequired(req, res, next) {
  const authHeader = req.headers.authorization || '';
  const token = authHeader.startsWith('Bearer ') ? authHeader.slice(7) : null;
  if (!token) return res.status(401).json({ message: 'Missing token' });
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET || 'dev_secret');
    req.userId = payload.sub;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid or expired token' });
  }}
```

Src/models/Post.js:

```
import mongoose from 'mongoose';
const { Schema } = mongoose;
const PostSchema = new Schema(
  {
    author: { type: Schema.Types.ObjectId, ref: 'User', required: true },
    content: { type: String, required: false, maxlength: 280, trim: true },
    imageUrl: { type: String },
    likes: [{ type: Schema.Types.ObjectId, ref: 'User' }],
    comments: [
      new Schema(
        {
          author: { type: Schema.Types.ObjectId, ref: 'User', required: true },
        content: { type: String, required: true, maxlength: 280, trim: true }
      },
      { timestamps: { createdAt: true, updatedAt: false } }
    ) ] },
  { timestamps: true }
);
PostSchema.index({ author: 1, createdAt: -1 });
export default mongoose.model('Post', PostSchema);
```

src/models/User.js :

```
import mongoose from 'mongoose';
const { Schema } = mongoose;
const UserSchema = new Schema(
  {
    username: { type: String, required: true, unique: true, trim: true, lowercase: true },
    email: { type: String, required: true, unique: true, trim: true, lowercase: true },
    name: { type: String, required: true, trim: true },
    bio: { type: String, default: '' },
    avatarUrl: { type: String, default: '' },
    dob: { type: Date },
    contact: { type: String, default: '' },
    passwordHash: { type: String, required: true },
    followers: [{ type: Schema.Types.ObjectId, ref: 'User' }],
    following: [{ type: Schema.Types.ObjectId, ref: 'User' }],
    isPrivate: { type: Boolean, default: false },
    followRequests: [{ type: Schema.Types.ObjectId, ref: 'User' }]
  },
  { timestamps: true }
);
UserSchema.index({ username: 1 });
UserSchema.index({ email: 1 });
export default mongoose.model('User', UserSchema);
```

src/routes/server.js :

```
import express from 'express';
import fs from 'fs';
import path from 'path';
import mongoose from 'mongoose';
import cors from 'cors';
import morgan from 'morgan';
import dotenv from 'dotenv';
import authRoutes from './routes/auth.js';
import userRoutes from './routes/users.js';
import postRoutes from './routes/posts.js';
import helmet from 'helmet';
import rateLimit from 'express-rate-limit';
dotenv.config();
const app = express();
const allowedOrigin = process.env.CORS_ORIGIN || '*';
app.use(
  cors({
    origin: allowedOrigin === '*' ? true : allowedOrigin,
    credentials: true }));
app.use(express.json());
app.use(morgan('dev'));
app.use(helmet());
const limiter = rateLimit({ windowMs: 15 * 60 * 1000, max: 300 });
app.use(limiter);
app.use('/api/auth', authRoutes);
app.use('/api/users', userRoutes);
app.use('/api/posts', postRoutes);
app.get('/health', (req, res) => res.json({ status: 'ok' }));
const uploadsDir = path.resolve(process.cwd(), 'uploads');
try {
  if (!fs.existsSync(uploadsDir)) fs.mkdirSync(uploadsDir, { recursive: true });
} catch {}
app.use('/uploads', express.static(uploadsDir));
const PORT = parseInt(process.env.PORT || '5000', 10);
const MONGODB_URI = process.env.MONGODB_URI || 'mongodb://127.0.0.1:27017/microblog';
async function startHttpServer(basePort) {
  const maxAttempts = 10;
  for (let attempt = 0; attempt <= maxAttempts; attempt++) {
    const portToTry = basePort + attempt;
    try {
      await new Promise((resolve, reject) => {
        const server = app.listen(portToTry, () => {
          console.log(`Server running on port ${portToTry}`);
          resolve();
        });
        server.on('error', (err) => {
          if (err && err.code === 'EADDRINUSE') {
            console.warn(`Port ${portToTry} is in use, trying ${portToTry + 1}...`);
            try { server.close(); } catch {}
            reject(err);
          } else {
            reject(err);
          }});});
      break;
    } catch (err) {
      if (!(err && err.code === 'EADDRINUSE')) {
        throw err;
      }
      if (attempt === maxAttempts) {
        throw new Error(`Unable to bind to any port from ${basePort} to ${basePort + maxAttempts}`);
      }   }  }}
if (process.env.NODE_ENV !== 'test') {
  mongoose
    .connect(MONGODB_URI)
    .then(async () => {
      console.log('MongoDB connected');
      await startHttpServer(PORT);
    })
    .catch((err) => {
      console.error('MongoDB connection error:', err);
      process.exit(1);
    });}
export default app;
```