Experiment No - 09

Roll No.	52
Name	Kshitij Nangare
Class	D15B
Subject	Full Stack Development
Lab Outcome	L6
Date of Performance / Submission	13/10/2025 27/10/2025
Signature & Grades	

Experiment 9

Aim : CI/CD Deployment with GitHub Actions + Render/Vercel

Code :

backend > app.js > ...

```javascript
import dotenv from 'dotenv';  7.2k (gzipped: 3.1k)
import express from 'express';
import cookieParser from 'cookie-parser';  5k (gzipped: 2k)
import cors from 'cors';  5k (gzipped: 2.1k)
import {connectDB} from './config/db.js';

import authRoutes from './routes/authRoutes.js';
import sessionRoute from './routes/sessionRoutes.js';

dotenv.config();

const app = express();
connectDB();

// Middleware
app.use(cors({ origin: process.env.FRONTEND_URL, credentials: true }));
app.use(cookieParser());
app.use(express.json());

app.use('/api/auth', authRoutes);
app.use('/api/session', sessionRoute);

// Server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

```

Figure 1

backend > config > db.js > ...

```javascript
import mongoose from 'mongoose';  581.7k (gzipped: 145.3k)
import dotenv from 'dotenv';  7.2k (gzipped: 3.1k)

dotenv.config();

export const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log('MongoDB Connected...');
  } catch (err) {
    console.error('Database connection error:', err.message);
    process.exit(1);
  }
};
```

Figure 2

```
import mongoose from "mongoose";  581.7k (gzipped: 145.3k)

const SessionSchema = new mongoose.Schema({
  user_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true
  },
  title: {
    type: String,
    trim: true
  },
  description: {
    type: String,
    default: ""
  },
  youtube_url: {
    type: String,
    trim: true
  },
  tags: {
    type: [String],
    default: []
  },
  status: {
    type: String,
    enum: ["draft", "published"],
    default: "draft"
  },
  imageUrl: {
    type: String,
    default: ""
  },
  likes: {
    type: Number,
    min: 0,
    default: 0
  },
  // NEW: Array to store IDs of users who liked this session
  likedBy: {
    type: [mongoose.Schema.Types.ObjectId],
    ref: "User", // Assuming you have a 'User' model
    default: []
  },
},{
  timestamps: true // This will automatically add createdAt and updatedAt
});


const Session = mongoose.model("Session", SessionSchema);

export default Session;
```

Figure 3

```
export const getAllPublishedSessions = async (req, res) => {
  try {
    const { search } = req.query; // Extract search term from query parameters

    let query = { status: 'published' };
    if (search) {
      // Create a case-insensitive regex for title or tags
      const searchRegex = new RegExp(search, 'i');
      query.$or = [
        { title: { $regex: searchRegex } },
        { tags: { $in: [searchRegex] } } // Search within the tags array
      ];
    }

    const sessions = await Session.find(query)
      .populate('user_id', 'name email') // Populate creator details
      .sort({ createdAt: -1 }); // Sort by creation date, newest first

    res.json(sessions);
  } catch (err) {
    console.error("Error in getAllPublishedSessions:", err); // Log the error
    res.status(500).json({ message: err.message || "Server error" });
  }
};

// Get logged-in user's sessions
export const getMySessions = async (req, res) => {
  try {
    const sessions = await Session.find({ user_id: req.user._id })
      .populate('user_id', 'name email')
      .sort({ createdAt: -1 });
    res.json(sessions);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

Figure 4

```
export const getSessionById = async (req, res) => {
  try {
    const { id } = req.params; // Get the session ID from the URL parameters

    // Find the session and populate the user_id field to get creator's name and email
    const session = await Session.findById(id).populate('user_id', 'name email');

    if (!session) {
      return res.status(404).json({ message: 'Session not found.' });
    }

    res.status(200).json(session);
  } catch (error) {
    console.error('Error fetching session by ID:', error);
    // Handle specific CastError if ID format is invalid
    if (error.name === 'CastError') {
        return res.status(400).json({ message: 'Invalid session ID format.' });
    }
    res.status(500).json({ message: 'Server error.', error: error.message });
  }
};

// Create new session
export const createSession = async (req, res) => {

  const session = new Session({
    ...req.body,
    user_id: req.user._id
  });

  try {
    const newSession = await session.save();
    res.status(201).json(newSession);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};
```

Figure 5

backend > controllers > sessionsController.js > ...

```javascript
export const likeSession = async (req, res) => {
  const userId = req.user._id; //  auth middleware sets req.user.id
  const { id } = req.params; // Session ID from the URL parameter

  try {
    const session = await Session.findById(id);

    if (!session) {
      return res.status(404).json({ message: 'Session not found.' });
    }

    let updatedSession;

    // Check if the user has already liked this session
    if (session.likedBy.includes(userId)) {
      // If already liked, UNLIKE IT: Decrement likes and remove user ID
      updatedSession = await Session.findByIdAndUpdate(
        id,
        {
          $inc: { likes: -1 },         // Atomically decrements the 'likes' count by 1
          $pull: { likedBy: userId }  // Removes the user's ID from the 'likedBy' array
        },
        {
          new: true,
          runValidators: true
        }
      ).populate('user_id', 'name email');;
      return res.status(200).json({ message: 'Session unliked successfully.', session: updatedSession });
    } else {
      // If not liked, LIKE IT: Increment likes and add user ID
      updatedSession = await Session.findByIdAndUpdate(
        id,
        {
          $inc: { likes: 1 },         // Atomically increments the 'likes' count by 1
          $push: { likedBy: userId }  // Adds the user's ID to the 'likedBy' array
        },
        {
          new: true,
          runValidators: true
        }
      ).populate('user_id', 'name email');;
      return res.status(200).json({ message: 'Session liked successfully.', session: updatedSession });
    }
  } catch (error) {
    console.error('Error processing like/unlike for session:', error);
```

Figure 6

```
// Update a session
export const updateSession = async (req, res) => {
  try {
    const session = await Session.findOneAndUpdate(
      { _id: req.params.id, user_id: req.user._id },
      req.body,
      { new: true }
    ).populate('user_id', 'name email');
    if (!session) return res.status(404).json({ message: 'Session not found' });
    res.json(session);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};


// Delete a session
export const deleteSession = async (req, res) => {
  try {
    const session = await Session.findOneAndDelete({
      _id: req.params.id,
      user_id: req.user._id
    });
    if (!session) return res.status(404).json({ message: 'Session not found' });
    res.json({ message: 'Session deleted' });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

Figure 6

```
import User from '../models/User.js';
import { generateToken } from '../utils/jwtToken.js';
import bcrypt from 'bcryptjs';  20.1k (gzipped: 9k)

export const register = async (req, res) => {
  try {
    const { email,name, password } = req.body;

    // Validate input
    if (!email || !password || !name) {
      return res.status(400).json({ message: 'Email and password are required' });
    }

    // Check if user exists
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(409).json({ message: 'Email already in use' });
    }

    // Hash password
    const salt = await bcrypt.genSalt(10);
    const password_hash = await bcrypt.hash(password, salt);


    // Create new user
    const user = new User({ email,name, password: password_hash });
    await user.save();

    // Generate JWT
    const token = generateToken(user);

    res.status(201).json({
      token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (error) {
    console.error('Registration error:', error);
    res.status(500).json({ message: 'Registration failed' });
  }
};
```

Figure 7

```
export const login = async (req, res) => {
  try {
    const { email, password } = req.body;

    // Validate input
    if (!email || !password) {
      return res.status(400).json({ message: 'Email and password are required' });
    }

    // Find user
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ message: 'Invalid Email or Password' });
    }

    // Check password
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).json({ message: 'Invalid Email or Password' });
    }

    // Generate JWT
    const token = generateToken(user);

    res.json({
      token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({ message: 'Login failed' });
  }
};
```

Figure 8

backend > controllers > authController.js > register

```javascript
export const changePassword = async (req, res) => {
    const { currentPassword, newPassword } = req.body;

    // 1. Basic validation
    if (!currentPassword || !newPassword) {
        return res.status(400).json({ message: 'Please provide current password and new password.' });
    }

    if (newPassword.length < 4) {
        return res.status(400).json({ message: 'New password must be at least 4 characters long.' });
    }

    try {
        // req.user will come from your authentication middleware (e.g., JWT verification)
        // It should contain the ID of the authenticated user
        const user = await User.findById(req.user._id).select('+password'); // Select password field explicitly

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

        // 2. Compare current password
        const isMatch = await bcrypt.compare(currentPassword, user.password);

        if (!isMatch) {
            return res.status(401).json({ message: 'Current password is incorrect.' });
        }

        // 3. Check if new password is the same as current password
        if (newPassword === currentPassword) {
            return res.status(400).json({ message: 'New password cannot be the same as the current password.' });
        }

        // 4. Hash the new password and save
        const salt = await bcrypt.genSalt(10);
        user.password = await bcrypt.hash(newPassword, salt); // Hash with a salt round of 10
        await user.save();

        res.status(200).json({ message: 'Password changed successfully!' });

    } catch (error) {
        console.error('Error changing password:', error);
        res.status(500).json({ message: 'Server error. Could not change password.' });
    }
};
```

Figure 9

frontend > src > pages > Dashboard.jsx > ...

```jsx
const DashboardPage = () => {
    const { user, logout } = useAuthStore();
    const [searchTerm, setSearchTerm] = useState('');
    const [isLoading, setIsLoading] = useState(true);
    const [activeTab, setActiveTab] = useState('all'); // 'all' or 'my'
    const [allSessions, setAllSessions] = useState([]);
    const [processingCardId, setProcessingCardId] = useState(null); // For like/unlike
    const navigate = useNavigate();

    // Function to fetch all published sessions
    const fetchAllPublishedSessions = useCallback(async () => {
        try {
            setIsLoading(true); // Keep loading true for all sessions tab
            // Append searchTerm as a query parameter for backend filtering
            const response = await axiosInstance.get(`/api/session/get-all-sessions?search=${searchTerm}`);
            setAllSessions(response.data);
        } catch (error) {
            console.error('Failed to fetch all published sessions:', error);
            toast.error('Failed to load public sessions.');
        } finally {
            setIsLoading(false);
        }
    }, [searchTerm]); // Re-run when searchTerm changes

    // Fetch dashboard data on component mount and when searchTerm changes (for 'all' tab)
    useEffect(() => {
        if (activeTab === 'all') {
            fetchAllPublishedSessions();
        }
    }, [activeTab, fetchAllPublishedSessions]); // Also re-fetch if activeTab changes to 'all'

    // Handle like/unlike for sessions displayed in 'All Sessions' tab
    const handleLikeUnlike = async (sessionId) => {
        setProcessingCardId(sessionId);
        try {
            // Send the request to your dedicated like/unlike endpoint
            const response = await axiosInstance.post(`/api/session/like/${sessionId}`);

            // The backend is now sending { message: '...', session: updatedSession }
            // We need to extract the session object from the response.data
            const updatedSession = response.data.session;
            const message = response.data.message; // Get the message from backend

            // Update the allSessions state with the modified session
```

Figure 10

```
frontend > src > pages > Dashboard.jsx > ...
17  const DashboardPage = () => {
77      return (
78          <div className="min-h-screen bg-gray-50">
79              {/* Header */}
80              <Navbar />
81
82              {/* Main Content */}
83              <main className="max-w-7xl mx-auto px-6 py-8">
84
85                  {/* Tab Navigation */}
86                  <div className="border-b border-gray-200 mb-6">
87                      <nav className="-mb-px flex space-x-8">
88                          <button
89                              onClick={() => setActiveTab('all')}
90                              className={`whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm ${
91                                  activeTab === 'all'
92                                      ? 'border-indigo-500 text-indigo-600'
93                                      : 'border-transparent text-gray-500 hover:text-gray-700 hover:border-gray-300'
94                              }`}
95                          >
96                              All Sessions
97                          </button>
98                          <button
99                              onClick={() => setActiveTab('my')}
100                             className={`whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm ${
101                                 activeTab === 'my'
102                                     ? 'border-indigo-500 text-indigo-600'
103                                     : 'border-transparent text-gray-500 hover:text-gray-700 hover:border-gray-300'
104                             }`}
105                         >
106                             My Sessions
107                         </button>
108                     </nav>
109                 </div>
110
111                 {/* Search Bar (Only for All Sessions tab) */}
112                 {activeTab === 'all' && (
113                     <div className="relative mb-6">
114                         <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
115                             <Search className="h-5 w-5 text-gray-400" aria-hidden="true" />
116                         </div>
117                         <input
118                             type="text"
119                             name="search"
120                             id="search"
```

Figure 11

```
frontend > src > pages > Dashboard.jsx > ...
17  const DashboardPage = () => {
123                                 value={searchTerm}
124                                 onChange={(e) => setSearchTerm(e.target.value)}
125                                 />
126                             </div>
127                         )}
128
129                         {/* Sessions Display */}
130                         {activeTab === 'all' ? (
131                             <section>
132                                 <h2 className="text-xl font-semibold text-gray-900 mb-4">All Wellness Sessions</h2>
133                                 {isLoading ? (
134                                     <LoadingSpinner />
135                                 ) : allSessions.length === 0 ? (
136                                     <div className="bg-white p-8 rounded-lg shadow-sm text-center">
137                                         <p className="text-gray-500">No sessions found matching your search.</p>
138                                         <p className="text-gray-400 text-sm mt-2">Try a different keyword or check back later!</p>
139                                     </div>
140                                 ) : (
141                                     <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
142                                         {allSessions.map(session => ( // Use `allSessions` here, as filtering is done on backend
143                                             <SessionCard
144                                                 key={session._id}
145                                                 session={session}
146                                                 isEditable={false} // Public sessions are not editable from here
147                                                 onLike={handleLikeUnlike}
148                                                 hasLiked={user && session.likedBy && session.likedBy.includes(user.id)} // Check if
149                                                 isProcessing={processingCardId === session._id}
150                                             />
151                                         ))}
152                                     </div>
153                                 )}
154                             </section>
155                         ) : (
156                             <section>
157                                 <MySessionsPage />
158                             </section>
159                         )}
160                     </main>
161                 </div>
162             );
163 };
164
165 export default DashboardPage;
```

Figure 12

```
sessionsController.js   vercel.json ×   authRoutes.js
frontend > vercel.json > ...
1  {
2    "rewrites": [
3      { "source": "/(.*)", "destination": "/" }
4    ]
5  }
6
```

Figure 13

Output :

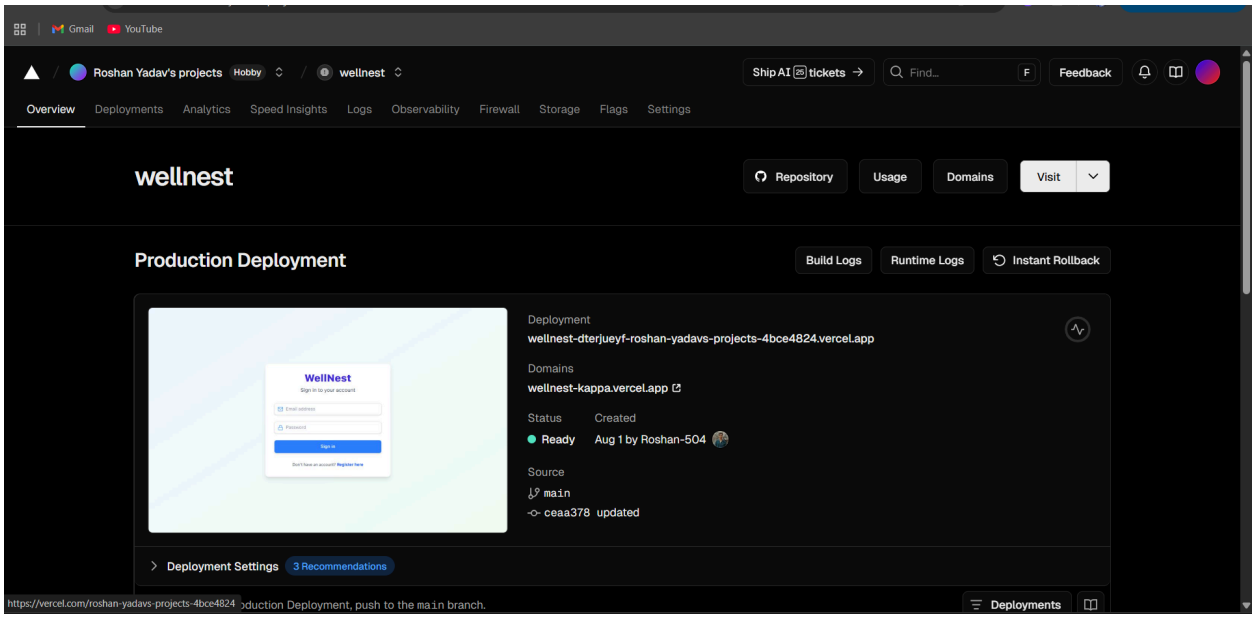
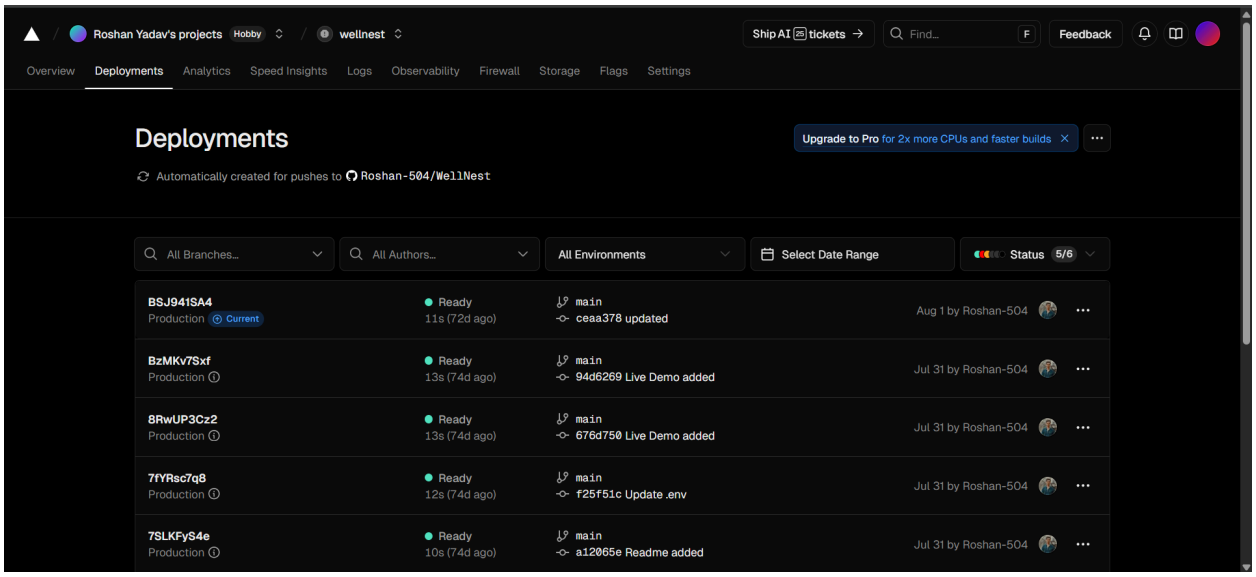


Figure 14



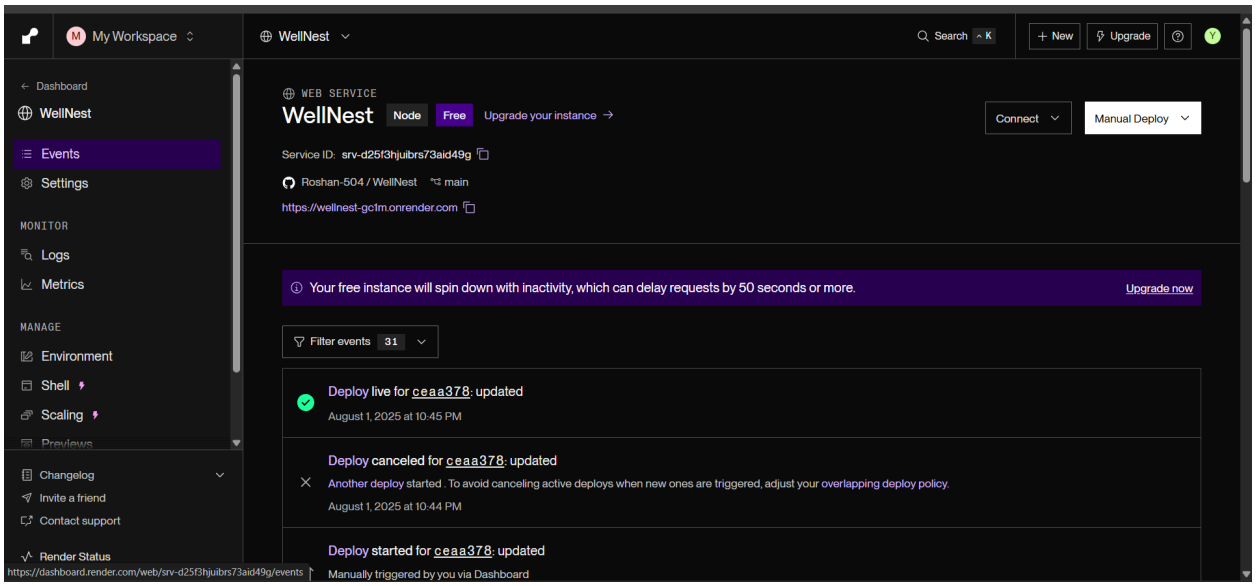


Figure 15

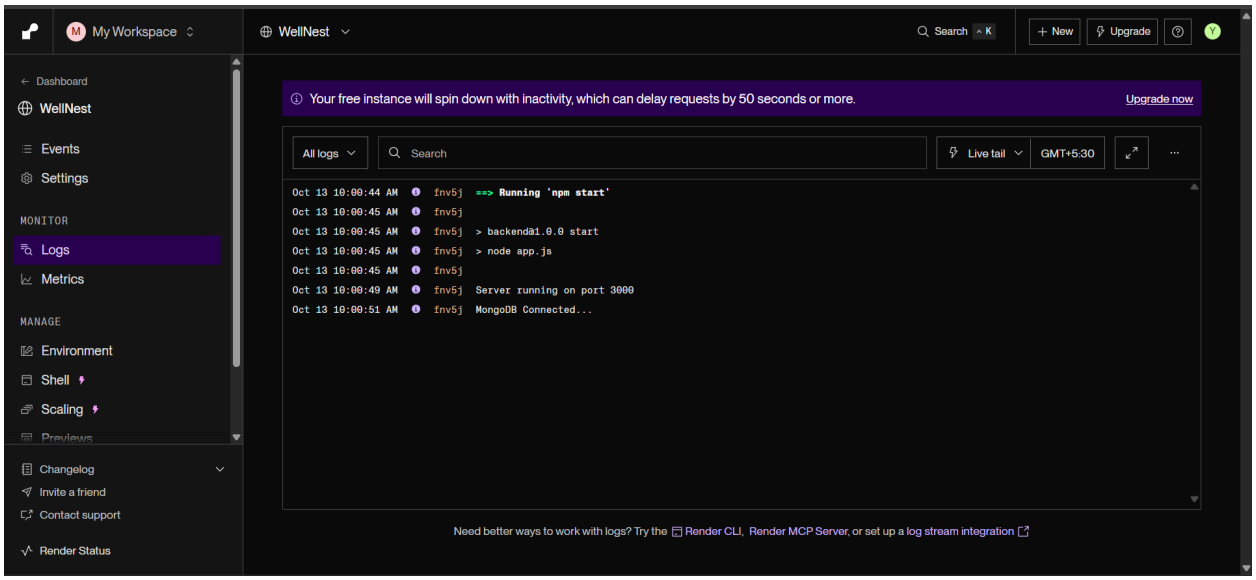


Figure 16