SOURCE CODE

DATA BASE :

Schema :

```
    const entrySchema = new mongoose.Schema({
    title: String,
    content: String,
    category: String,
    createdBy: String,
});
const Entry = mongoose.model("Entry", entrySchema);
```

HTML :

```
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Personal Journal</title>
    <style>
        body {
            font-family: 'Roboto', Arial, sans-serif;
            background: linear-gradient(135deg, #83a4d4, #b6fbff);
            color: #333;
            display: flex;
            flex-direction: column;
            align-items: center;
    justify-content: center;
    min-height: 100vh;
    margin: 0;
    padding: 20px;
}

#journal-form {
    width: 100%;
    max-width: 500px;
    padding: 25px;
    background-color: #fff;
    box-shadow: 0 8px 15px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
    display: flex;
    flex-direction: column;
    gap: 15px;
}

h1 {
    font-size: 2.5em;
    color: #fff;
    margin-bottom: 20px;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.4);
}

input[type="text"], textarea, select {
    width: 100%;
    padding: 12px;
    border: 1px solid #ddd;
    border-radius: 8px;
    font-size: 1em;
    box-sizing: border-box;
    transition: border-color 0.3s, box-shadow 0.3s;
}

input[type="text"]:focus, textarea:focus, select:focus {
    border-color: #007bff;
    box-shadow: 0 0 5px rgba(0, 123, 255, 0.5);
    outline: none;
}

textarea {
    resize: vertical;
    min-height: 120px;
}

select {
    background-color: #f9f9f9;
    cursor: pointer;
}

button {
    background: linear-gradient(135deg, #007bff, #0056b3);
    color: #fff;
    padding: 12px;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 1em;
    font-weight: bold;
    text-transform: uppercase;
    transition: background 0.3s, transform 0.2s;
}

button:hover {
    background: linear-gradient(135deg, #0056b3, #003a80);
    transform: scale(1.05);
}

h2 {
    font-size: 2em;
    color: #333;
    margin: 20px 0 10px;
    text-align: center;
}

#entries {
    width: 100%;
    max-width: 500px;
    display: flex;
    flex-direction: column;
    gap: 15px;
}

.entry {
    background: #fff;
    border: 1px solid #ddd;
    padding: 15px;
    border-radius: 8px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
```

```
            transition: transform 0.2s, box-shadow 0.3s;
        }

        .entry:hover {
            transform: scale(1.02);
            box-shadow: 0 8px 20px rgba(0, 0, 0, 0.15);
        }

        .entry h3 {
            margin: 0 0 10px;
            font-size: 1.5em;
            color: #007bff;
        }

        .entry p {
            font-size: 1em;
            margin: 0 0 10px;
            color: #666;
            line-height: 1.5;
        }

        .entry span {
            font-size: 0.9em;
            color: #999;
        }
    </style>
</head>
<body>
    <h1>Personal Journal</h1>
```

```
<form id="journal-form">
  <input type="text" id="title" placeholder="Title" required>
  <textarea id="content" placeholder="Write your journal entry..." required></textarea>
  <select id="category">
    <option value="Work">Work</option>
    <option value="Personal">Personal</option>
    <option value="Travel">Travel</option>
  </select>
  <button type="submit">Add Entry</button>
</form>

<h2>Entries</h2>
<div id="entries"></div>

<script>
  document.getElementById('journal-form').addEventListener('submit', function (e) {
    e.preventDefault();
    const title = document.getElementById('title').value;
    const content = document.getElementById('content').value;
    const category = document.getElementById('category').value;

    const entry = document.createElement('div');
    entry.classList.add('entry');
    entry.innerHTML = `
      <h3>${title}</h3>
      <p>${content}</p>
      <span>Category: ${category}</span>
    `;
    document.getElementById('entries').prepend(entry);
    document.getElementById('journal-form').reset();
```

```
        });
    </script>
</body>
</html>
```

SERVER :

```
const express = require("express");
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");
const bodyParser = require("body-parser");
const session = require("express-session");

const app = express();
const PORT = 3000;

app.use(express.static("public"));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(
    session({
        secret: "your-secret-key",
        resave: false,
        saveUninitialized: true,
    })
);

mongoose.connect("mongodb://127.0.0.1:27017/journalApp", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});
```

```
const userSchema = new mongoose.Schema({
  username: { type: String, unique: true },
  password: String,
  entries: [
    {
      title: String,
      content: String,
      category: String,
    },
  ],
});
const User = mongoose.model("User", userSchema);

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/views/login.html");
});

app.post("/login", async (req, res) => {
  const { username, password } = req.body;
  const user = await User.findOne({ username });
  if (user && (await bcrypt.compare(password, user.password))) {
    req.session.userId = user._id;
    res.redirect("/index");
  } else {
    res.send("Invalid credentials");
  }
});

app.post("/signup", async (req, res) => {
  const { username, password } = req.body;
  const hashedPassword = await bcrypt.hash(password, 10);
  try {
    await User.create({ username, password: hashedPassword });
    res.redirect("/");
  } catch (err) {
    res.send("Error creating user");
  }
});

app.get("/index", async (req, res) => {
  if (!req.session.userId) {
    return res.redirect("/");
  }
  const user = await User.findById(req.session.userId);
  res.render("index.ejs", { username: user.username, entries: user.entries });
});

app.post("/add-entry", async (req, res) => {
  if (!req.session.userId) {
    return res.redirect("/");
  }
  const { title, content, category } = req.body;
  await User.findByIdAndUpdate(req.session.userId, {
    $push: { entries: { title, content, category } },
  });
  res.redirect("/index");
});

app.get("/logout", (req, res) => {
  req.session.destroy();
```

```
    res.redirect("/");
});

app.set("view engine", "ejs");

app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

CSS :

```
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: 'Arial', sans-serif;
    /* Change the URL to point to a local image */
    background: url(https://img.freepik.com/free-photo/anime-moon-landscape_23-2151645903.jpg?t=st=1731222411~exp=1731226011~hmac=b84eb3ef7d1933999a466935a9942f8e4cf190424fdf32f8b1f9c9e3ffef4eba&w=1060) no-repeat center center fixed;
    background-size: cover;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    color: #fff;
}

.container {
    width: 100%;
    max-width: 400px;
```

```css
    background: rgba(0, 0, 0, 0.7); /* Black background with transparency */
    border-radius: 10px;
    padding: 20px;
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.5);
}

.form {
    display: flex;
    flex-direction: column;
}

h2 {
    margin-bottom: 20px;
    text-align: center;
}

input {
    margin-bottom: 15px;
    padding: 10px;
    border: none;
    border-radius: 5px;
    outline: none;
}

input[type="text"],
input[type="email"],
input[type="password"] {
    background: rgba(255, 255, 255, 0.1); /* White with transparency */
    color: #fff;
}
```

```css
button {
   padding: 10px;
   border: none;
   border-radius: 5px;
   background: #28a745;
   color: white;
   font-size: 16px;
   cursor: pointer;
   transition: background 0.3s ease;
}

button:hover {
   background: #218838;
}

p {
   text-align: center;
}

a {
   color: #28a745;
   text-decoration: none;
}

.hidden {
   display: none;
}

/* Responsive Styles */
```

```
@media (max-width: 600px) {
    .container {
        width: 90%;
    }
}
```

FRONT END :

```
const express = require("express");
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");
const bodyParser = require("body-parser");
const session = require("express-session");

const app = express();
const PORT = 3000;

app.use(express.static("public"));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(
    session({
        secret: "your-secret-key",
        resave: false,
        saveUninitialized: true,
    })
);

mongoose.connect("mongodb://127.0.0.1:27017/journalApp", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});
```

```
const userSchema = new mongoose.Schema({
   username: { type: String, unique: true },
   password: String,
});
const User = mongoose.model("User", userSchema);

const entrySchema = new mongoose.Schema({
   title: String,
   content: String,
   category: String,
   createdBy: String,
});
const Entry = mongoose.model("Entry", entrySchema);

app.get("/", (req, res) => {
   res.sendFile(__dirname + "/views/login.html");
});

app.post("/login", async (req, res) => {
   const { username, password } = req.body;
   const user = await User.findOne({ username });
   if (user && (await bcrypt.compare(password, user.password))) {
      req.session.userId = user._id;
      res.redirect("/index");
   } else {
      res.send("Invalid credentials");
   }
});

app.post("/signup", async (req, res) => {
    const { username, password } = req.body;
    const hashedPassword = await bcrypt.hash(password, 10);
    try {
        await User.create({ username, password: hashedPassword });
        res.redirect("/");
    } catch (err) {
        res.send("Error creating user");
    }
});

app.get("/index", async (req, res) => {
    if (!req.session.userId) {
        return res.redirect("/");
    }
    const user = await User.findById(req.session.userId);
    const entries = await Entry.find();
    res.render("index.ejs", { username: user.username, entries });
});

app.post("/add-entry", async (req, res) => {
    if (!req.session.userId) {
        return res.redirect("/");
    }
    const { title, content, category } = req.body;
    const user = await User.findById(req.session.userId);

    const newEntry = new Entry({
        title,
        content,
        category,
        createdBy: user.username,
    });
    await newEntry.save();

    res.redirect("/index");
});

app.get("/logout", (req, res) => {
    req.session.destroy();
    res.redirect("/");
});

app.set("view engine", "ejs");

app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

CSS STYLE 2:

```
document.getElementById('journal-form').addEventListener('submit', async function(event)
{
    event.preventDefault();
    const title = document.getElementById('title').value;
    const content = document.getElementById('content').value;
    const category = document.getElementById('category').value;

    const response = await fetch('http://localhost:3000/api/journal', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ title, content, category })
    });

    if (response.ok) {
        loadEntries();
    }
});

async function loadEntries() {
    const response = await fetch('http://localhost:3000/api/journal');
    const entries = await response.json();
    const entriesDiv = document.getElementById('entries');
    entriesDiv.innerHTML = entries.map(entry => `
        <div class="entry">
            <h3>${entry.title}</h3>
            <p>${entry.content}</p>
            <p><em>${entry.category}</em></p>
        </div>
    `).join('');
}

loadEntries();
```