

Exercise 5: Create basic Express website using Nodejs

Aap.js

```
require("dotenv").config();
const createError = require("http-errors");
const express = require("express");
const path = require("path");
const cookieParser = require("cookie-parser");
const logger = require("morgan");
const mongoose = require("mongoose");
const session = require("express-session");
const passport = require("passport");
const flash = require("connect-flash");
const Category = require("../models/category");
var MongoStore = require("connect-mongo")(session);
const connectDB = require("../config/db");

const app = express();
require("../config/passport");

// mongodb configuration
connectDB();
// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");

// admin route
const adminRouter = require("../routes/admin");
app.use("/admin", adminRouter);

app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, "public")));
app.use(
  session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    store: new MongoStore({
      mongooseConnection: mongoose.connection,
    }),
    //session expires after 3 hours
    cookie: { maxAge: 60 * 1000 * 60 * 3 },
  })
)
```

```

);
app.use(flash());
app.use(passport.initialize());
app.use(passport.session());

// global variables across routes
app.use(async (req, res, next) => {
  try {
    res.locals.login = req.isAuthenticated();
    res.locals.session = req.session;
    res.locals.currentUser = req.user;
    const categories = await Category.find({}).sort({ title: 1 }).exec();
    res.locals.categories = categories;
    next();
  } catch (error) {
    console.log(error);
    res.redirect("/");
  }
});

// add breadcrumbs
get_breadcrumbs = function (url) {
  var rtn = [{ name: "Home", url: "/" }],
    acc = "", // accumulative url
    arr = url.substring(1).split("/");

  for (i = 0; i < arr.length; i++) {
    acc = i !== arr.length - 1 ? acc + "/" + arr[i] : null;
    rtn[i + 1] = {
      name: arr[i].charAt(0).toUpperCase() + arr[i].slice(1),
      url: acc,
    };
  }
  return rtn;
};

app.use(function (req, res, next) {
  req.breadcrumbs = get_breadcrumbs(req.originalUrl);
  next();
});

//routes config
const indexRouter = require("./routes/index");
const productsRouter = require("./routes/products");
const usersRouter = require("./routes/user");
const pagesRouter = require("./routes/pages");
app.use("/products", productsRouter);
app.use("/user", usersRouter);
app.use("/pages", pagesRouter);

```

```

app.use("/", indexRouter);

// catch 404 and forward to error handler
app.use(function (req, res, next) {
  next(createError(404));
});

// error handler
app.use(function (err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get("env") === "development" ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render("error");
});

var port = process.env.PORT || 3000;
app.set("port", port);
app.listen(port, () => {
  console.log("Server running at port " + port);
});

module.exports = app;

```

cart.js

```

const mongoose = require("mongoose");

const cartSchema = new mongoose.Schema({
  items: [
    {
      productId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Product",
      },
      qty: {
        type: Number,
        default: 0,
      },
      price: {
        type: Number,
        default: 0,
      },
      title: {
        type: String,

```

```

    },
    productCode: {
      type: String,
    },
  },
],
totalQty: {
  type: Number,
  default: 0,
  required: true,
},
totalCost: {
  type: Number,
  default: 0,
  required: true,
},
user: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "User",
  required: false,
},
createdAt: {
  type: Date,
  default: Date.now,
},
});

module.exports = mongoose.model("Cart", cartSchema);

```

Category.js

```

const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const slug = require("mongoose-slug-updater");

mongoose.plugin(slug);

const categorySchema = Schema({
  title: {
    type: String,
    required: true,
  },
  slug: {
    type: String,
    unique: true,
    slug: "title",
  },
});

```

```
    },  
  });
```

```
module.exports = mongoose.model("Category", categorySchema);
```

order.js

```
const mongoose = require("mongoose");  
const Schema = mongoose.Schema;  
  
const orderSchema = Schema({  
  user: {  
    type: Schema.Types.ObjectId,  
    ref: "User",  
  },  
  cart: {  
    totalQty: {  
      type: Number,  
      default: 0,  
      required: true,  
    },  
    totalCost: {  
      type: Number,  
      default: 0,  
      required: true,  
    },  
    items: [  
      {  
        productId: {  
          type: mongoose.Schema.Types.ObjectId,  
          ref: "Product",  
        },  
        qty: {  
          type: Number,  
          default: 0,  
        },  
        price: {  
          type: Number,  
          default: 0,  
        },  
        title: {  
          type: String,  
        },  
        productCode: {  
          type: String,  
        },  
      },  
    ],  
  },  
});
```

```

    ],
  },
  address: {
    type: String,
    required: true,
  },
  paymentId: {
    type: String,
    required: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
  Delivered: {
    type: Boolean,
    default: false,
  },
});

module.exports = mongoose.model("Order", orderSchema);

```

product.js

```

const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const productSchema = Schema({
  productCode: {
    type: String,
    required: true,
    unique: true,
  },
  title: {
    type: String,
    required: true,
  },
  imagePath: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  price: {
    type: Number,
    required: true,
  },
});

```

```

    },
    category: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Category",
    },
    manufacturer: {
      type: String,
    },
    available: {
      type: Boolean,
      required: true,
    },
    createdAt: {
      type: Date,
      default: Date.now,
    },
  });

module.exports = mongoose.model("Product", productSchema);

```

user.js

```

const mongoose = require("mongoose");
const bcrypt = require("bcrypt-nodejs");
const Schema = mongoose.Schema;

const userSchema = Schema({
  username: {
    type: String,
    require: true,
  },
  email: {
    type: String,
    require: true,
  },
  password: {
    type: String,
    require: true,
  },
});

// encrypt the password before storing
userSchema.methods.encryptPassword = (password) => {
  return bcrypt.hashSync(password, bcrypt.genSaltSync(5), null);
};

userSchema.methods.validPassword = function (candidatePassword) {

```

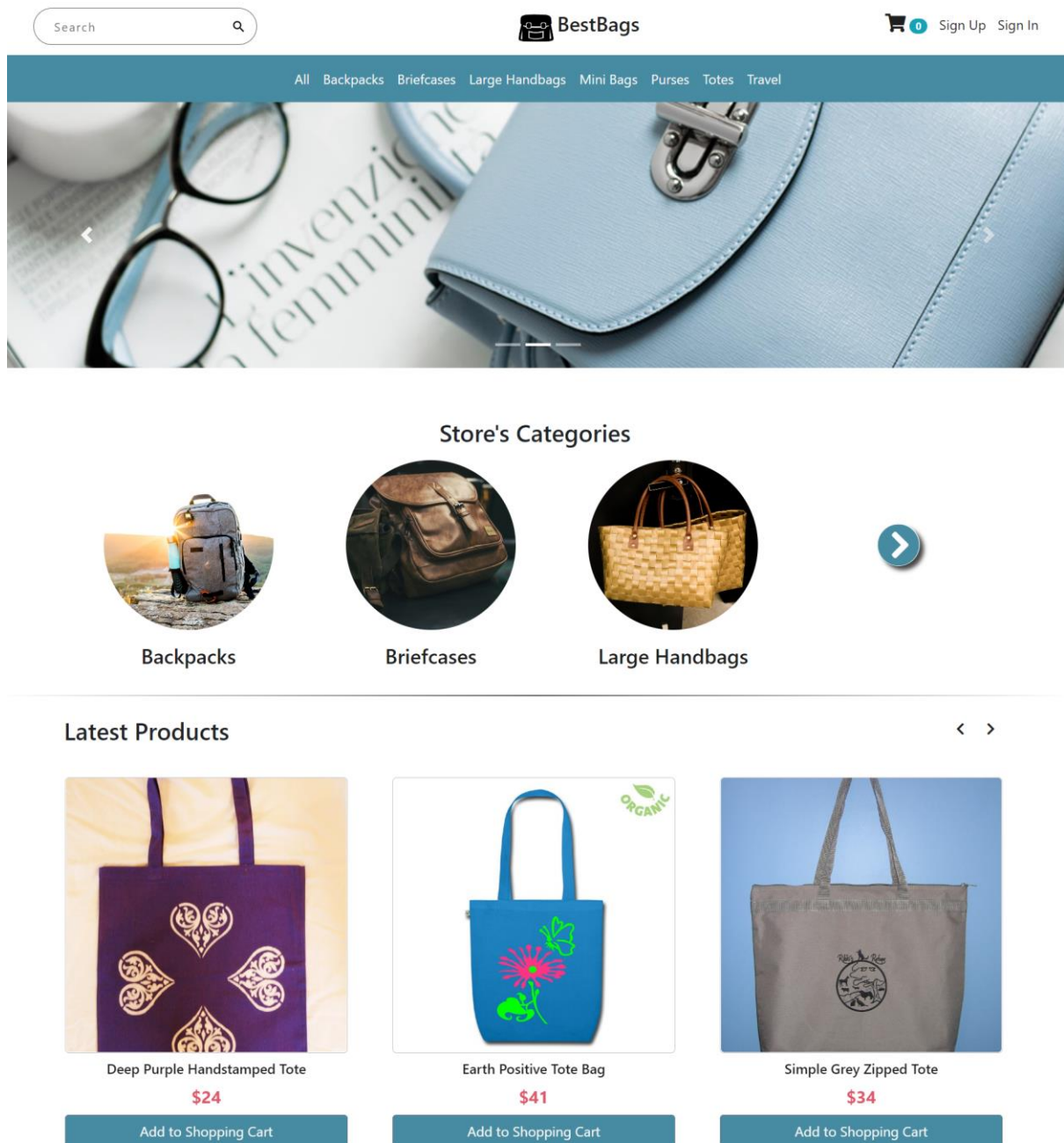
```

if (this.password !== null) {
  return bcrypt.compareSync(candidatePassword, this.password);
} else {
  return false;
}
};

module.exports = mongoose.model("User", userSchema);

```

Output



Exercise 6: Design a simple user Login system using Node js

Index.js

```
import React from "react";
import ReactDOM from "react-dom";
import Login from "./login.js";
import SignUp from "./signup.js";

import "./styles.css";

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: ""
    };
  }

  render() {
    return (
      <div className="App">
        <Login />
        <p className="signup-label">
          Don't have an account?{" "}
          <a href="#" className="link">
            Sign up
          </a>
        </p>
      </div>
    );
  }
}

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

Login.js

```
import React from "react";

class Login extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      username: "",
      password: ""
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    const target = event.target;
    const value = target.value;
    const name = target.name;
    this.setState({
      [name]: value
    });
  }

  handleSubmit(event) {
    event.preventDefault();
    console.log(this.state);
  }

  render() {
    const { username, password } = this.state;
    return (
      <form className="form" onSubmit={this.handleSubmit}>
        <div className="input-container">
          <label className="label">Username: </label>
          <input
            type="text"
            name="username"
            className="input"
            placeholder="Username"
            value={username}
            onChange={this.handleChange}
          />
        </div>
        <div className="input-container">
          <label className="label">Password: </label>
          <input
            type="password"
            name="password"

```

```

        className="input"
        placeholder="Password"
        value={password}
        onChange={this.handleChange}
      />
      <a href="#" className="link forgotten-password">
        Forgot password?
      </a>
    </div>
    <button type="submit" id="login-btn">
      Login
    </button>
  </form>
);
}
}

export default Login;

```

signup.js

```

import React from "react";

class SignUp extends React.Component {
  render() {
    return (
      <form className="form">
        <label for="fname">First Name</label>
        <input type="text" placeholder="First name" />
        <label for="sname">Last Name</label>
        <input type="text" placeholder="Last name" />
        <label for="email">Email Address</label>
        <input type="text" placeholder="Email Address" />
        <label for="password">Password</label>
        <input type="text" placeholder="Password" />
      </form>
    );
  }
}

export default SignUp;

```

styles.css

```
body {
  margin: 0;
  padding: 0;
}

.App {
  height: 500px;
  min-height: 100vh;
  background-image: url("background.png");
  background-size: cover;
  background-repeat: no-repeat;
  background-position: top;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  text-align: center;
  font-family: sans-serif;
}

.form {
  padding: 40px;
  background-color: rgba(255, 255, 255, 0.5);
  border-radius: 5px;
  display: flex;
  flex-direction: column;
  justify-content: space-evenly;
  align-items: flex-start;
}

.input-container {
  margin: 10px auto;
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
  align-items: flex-start;
  font-size: 14px;
}

.label {
  margin-bottom: 5px;
}

.input {
```

```
padding: 5px 15px 5px 5px;
}

.forgotten-password {
  align-self: flex-end;
  margin-top: 7px;
}

#login-btn {
  align-self: center;
  padding: 7px 30px;
  margin-top: 30px;
  border: none;
  color: #fff;
  background-color: rgb(255, 39, 147);
  box-shadow: 2px 2px 2px rgb(204, 4, 121);
  cursor: pointer;
  text-transform: uppercase;
  transition: all 0.2s ease-in;
}

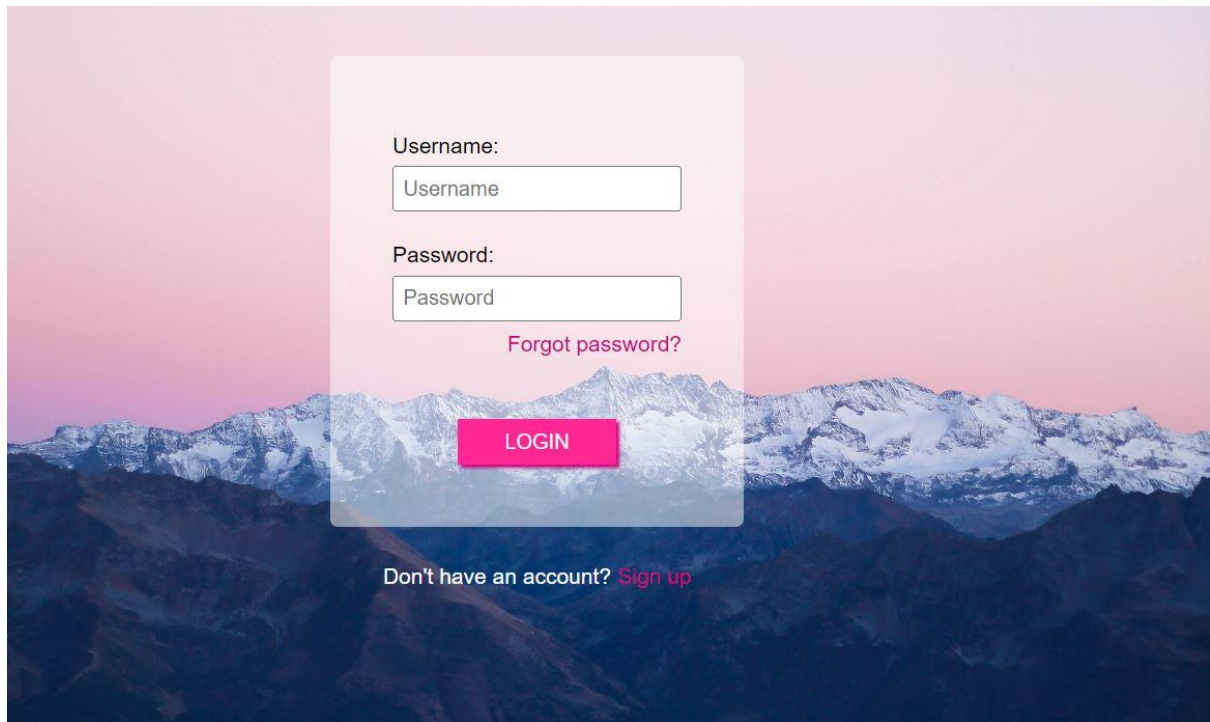
#login-btn:hover {
  background-color: rgb(204, 4, 121);
  box-shadow: none;
  transform: translateX(1px) translateY(1px);
}

.signup-label {
  position: relative;
  bottom: -10px;
  color: #fff;
  font-size: 14px;
}

.link {
  text-decoration: none;
  color: rgb(204, 4, 121);
}

.link:hover {
  text-decoration: underline;
}
```

Output

A login form is centered on a background image of a mountain range at dusk or dawn. The sky is a gradient of pink and purple, while the mountains are dark and silhouetted. A semi-transparent white rectangular box contains the login fields. Inside this box, there are labels for 'Username:' and 'Password:', each followed by a text input field. The 'Username' field contains the text 'Username' and the 'Password' field contains 'Password'. Below the password field, there is a link 'Forgot password?' in pink. At the bottom of the white box is a pink button with the text 'LOGIN' in white. Below the white box, centered on the background, is the text 'Don't have an account?' followed by a pink link 'Sign up'.