Config file code:

```
//Connection file to mongo db
import mongoose from "mongoose";
import colors from "colors";

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {
      useUnifiedTopology: true,
      useNewUrlParser: true,
      useCreateIndex: true,
    });
    console.log(`MongoDB Connected: ${conn.connection.host}`.cyan.underline);
  } catch (error) {
    console.error(`Error: ${error.message}`.red.bold);
    process.exit();
  }
};

export default connectDB;
```

Post controllers code:

```
import Post from "../models/postModel.js";
import asyncHandler from "express-async-handler";

//@description     Create new Post
//@route           POST /api/post/create
//@access          Private
const createPost = asyncHandler(async (req, res) => {
  const { caption, pic, feeling } = req.body;

  if (!caption) {
    return res.status(422).json({ error: "Please add all the feilds" });
  }

  const post = await Post.create({
    caption,
    pic,
    feeling,
    postedBy: req.user,
  });

  if (post) res.status(201).json(post);
  else {
    res.status(400);
    throw new Error("Post not Found");
  }
});

export { createPost };
```

User Controller code:

```
import asyncHandler from "express-async-handler";
import User from "../models/userModel.js";
import generateToken from "../utils/generateToken.js";

//@description     Register new user
//@route           POST /api/user/register
//@access          Public
const registerUser = asyncHandler(async (req, res) => {
  const { name, email, password, pic } = req.body;

  const userExists = await User.findOne({ email });

  if (userExists) {
    res.status(404);
    throw new Error("User Already Exists");
  }

  const user = await User.create({ name, email, password, pic });

  if (user) {
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      pic: user.pic,
    });
  } else {
    res.status(400);
    throw new Error("User not Found");
```

```
  }
});

//@description     Login the user
//@route           POST /api/user/login
//@access          Public
const loginUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({ email });

  if (user && (await user.matchPassword(password))) {
    res.send({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
      pic: user.pic,
      token: generateToken(user._id),
    });
  } else {
    res.status(401);
    throw new Error("Invalid Email or Password");
  }
});

export { registerUser, loginUser };
```

Middle ware code:

```
import jwt from "jsonwebtoken";
import User from "../models/userModel.js";
import asyncHandler from "express-async-handler";

const protect = asyncHandler(async (req, res, next) => {
  let token;

  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith("Bearer")
  ) {
    try {
      token = req.headers.authorization.split(" ")[1];
       //decodes token id
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.id).select("-password");
      next();
    } catch (error) {
      res.status(401);
      throw new Error("Not authorized, token failed");
    }
  }
    if (!token) {
    res.status(401);
    throw new Error("Not authorized, no token");
  }
});
export { protect };
```

Model code:

```
import mongoose from "mongoose";
const { ObjectId } = mongoose.Schema.Types;
const postSchema = new mongoose.Schema(
 {
   caption: {
    type: String,
    required: true,
   },
   feeling: {
    type: String,
    default: null,
   },
   pic: {
    type: String,
    default: null,
   },
   likes: [{ type: ObjectId, ref: "User" }],
   comments: [
    {
      type: String,
      postedBy: { type: ObjectId, ref: "User" },
    },
   ],
   postedBy: {
    type: ObjectId,
    ref: "User",
   },
 },
 { timestamps: true }
);
```

```
const Post = mongoose.model("Post", postSchema);
export default Post;
```

Utils Code:

```
import jwt from "jsonwebtoken";

const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: "30d",
  });
};

export default generateToken;
```