user.model.js

```
import mongoose from "mongoose";
import jwt from "jsonwebtoken";
import bcrypt from "bcrypt";

const userSchema = new mongoose.Schema(
  {
    watchHistory: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Video",
      },
    ],
    username: {
      type: String,
      required: true,
      unique: true,
      lowercase: true,
      trim: true,
      index: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
      lowercase: true,
      trim: true,
    },
    fullName: {
      type: String,
      required: true,
    },
    avatar: {
      type: String, //cloudinary url
      required: true,
    },
    coverImage: {
      type: String, //cloudinary url
    },
    password: {
      type: String,
      required: [true, "Password is required"],
    },
    refreshToken: {
      type: String,
    },
  },
  { timestamps: true }
);

userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});

userSchema.methods.isPasswordCorrect = async function (password) {
```

```
  return await bcrypt.compare(password, this.password);
};

userSchema.methods.generateAccessToken = function () {
  return jwt.sign(
    {
      _id: this._id,
      email: this.email,
      username: this.username,
      fullName: this.fullName,
    },
    process.env.ACCESS_TOKEN_SECRET,
    { expiresIn: process.env.ACCESS_TOKEN_EXPIRY }
  );
};
userSchema.methods.generateRefreshToken = function () {
  return jwt.sign(
    {
      _id: this._id,
    },
    process.env.REFRESH_TOKEN_SECRET,
    { expiresIn: process.env.REFRESH_TOKEN_EXPIRY }
  );
};
export const User = mongoose.model("User", userSchema);
```

user.routes.js

```
import { Router } from "express";
import { verifyJWT } from "../middlewares/auth.middleware.js";
import {
  loginUser,
  logoutUser,
  refreshAccessToken,
  registerUser,
} from "../controllers/user.controller.js";
import { upload } from "../middlewares/multer.middleware.js";

const router = Router();

router.route("/register").post(
  upload.fields([
    {
      name: "avatar", //frontend field ka bhi naam avatar hona chahiye
      maxCount: 1,
    },
    {
      name: "coverImage",
      maxCount: 1,
    },
  ]),
  registerUser
);

router.route("/login").post(loginUser);
```

```
//secured routes
router.route("/logout").post(verifyJWT, logoutUser);
export default router;
router.route("/refresh-token").post(refreshAccessToken);
```

user.controller.js

```
import { asyncHandler } from "../utils/asyncHandler.js";
import { ApiError } from "../utils/apiError.js";
import { ApiResponse } from "../utils/apiResponse.js";
import { User } from "../models/user.model.js";
import { uploadOnCloudinary } from "../utils/cloudinary.js";
import jwt from "jsonwebtoken";

const generateAccessAndRefreshTokens = async (userId) => {
  try {
    const user = await User.findById(userId);
    const accessToken = user.generateAccessToken();
    const refreshToken = user.generateRefreshToken();

    user.refreshToken = refreshToken;
    await user.save({ validateBeforeSave: false });

    return { accessToken, refreshToken };
  } catch (error) {
    throw new ApiError(
      500,
      "Something went wrong while generating access and refresh tokens"
    );
  }
};

const registerUser = asyncHandler(async (req, res) => {
  // get user details from frontend/postman
  const { username, email, fullName, password } = req.body;

  // validation - not empty or undefined
  if (
    [fullName, email, username, password].some(
      (field) => field === undefined || field?.trim() === ""
    )
  ) {
    throw new ApiError(400, "All fields are required");
  }

  // check if user already exists? username, email
  const existedUser = await User.findOne({
    $or: [{ username }, { email }],
  });

  if (existedUser) {
    throw new ApiError(409, "User with email or username already exists");
  }

  // check for images, check for avatar
  const avatarLocalPath = req.files?.avatar[0]?.path;
  if (!avatarLocalPath) {
    throw new ApiError(400, "Avatar file is required");
```

```
  }
  // upload avatar to cloudinary
  const avatar = await uploadOnCloudinary(avatarLocalPath);
  console.log(avatar);
  if (!avatar) {
    throw new ApiError(400, "Avatar file is required");
  }

  // check for images, check for coverImage
  // const coverImageLocalPath = req.files?.coverImage[0]?.path;
  let coverImageLocalPath;
  if (
    req.files &&
    Array.isArray(req.files.coverImage) &&
    req.files.coverImage.length > 0
  ) {
    coverImageLocalPath = req.files.coverImage[0].path;
  }
  // upload coverImage to cloudinary
  const coverImage = await uploadOnCloudinary(coverImageLocalPath);

  // create user object - create entry in db
  const user = await User.create({
    fullName,
    avatar: avatar.url,
    coverImage: coverImage?.url || "",
    email,
    password,
    username: username.toLowerCase(),
  });

  // check for user creation and remove password and refresh token
  const createdUser = await User.findById(user._id).select(
    "-password -refreshToken"
  );

  if (!createdUser) {
    throw new ApiError(500, "Something went wrong while registering");
  }

  // return user response
  return res
    .status(201)
    .json(new ApiResponse(200, createdUser, "User registered successfully"));
});

const loginUser = asyncHandler(async (req, res) => {
  // req body -> data
  const { email, username, password } = req.body;
  // username or email
  if (!username && !email) {
    throw new ApiError(400, "username or email is required");
  }


  // Here is an alternative of above code based on logic discussed in video:
  // if (!(username || email)) {
  //     throw new ApiError(400, "username or email is required")

  // }
  // find the user
```

```
  const user = await User.findOne({ $or: [{ username }, { email }] });
  if (!user) {
    throw new ApiError(404, "User does not exist");
  }
  // check password
  const isPasswordValid = await user.isPasswordCorrect(password);

  if (!isPasswordValid) {
    throw new ApiError(401, "Invalid user credentials");
  }
  // access and refresh token
  const { accessToken, refreshToken } = await generateAccessAndRefreshTokens(
    user._id
  );
  const loggedInUser = await User.findById(user._id).select(
    "-password -refreshToken"
  );
  // send Cookie
  const options = {
    httpOnly: true,
    secure: true,
  };

  return res
    .status(200)
    .cookie("accessToken", accessToken, options)
    .cookie("refreshToken", refreshToken, options)
    .json(
      new ApiResponse(
        200,
        { user: loggedInUser, accessToken, refreshToken },
        "User Logged In Successfully"
      )
    );
});

const logoutUser = asyncHandler(async (req, res) => {
  User.findByIdAndUpdate(
    req.user._id,
    { $set: { refreshToken: undefined } },
    { new: true }
  );

  const options = {
    httpOnly: true,
    secure: true,
  };
  return res
    .status(200)
    .clearCookie("accessToken", options)
    .clearCookie("refreshToken", options)
    .json(new ApiResponse(200, {}, "User logged out successfully"));
});


const refreshAccessToken = asyncHandler(async (req, res) => {
  const incomingRefreshToken =
    req.cookies.refreshToken || req.body.refreshToken;
  if (!incomingRefreshToken) {
    throw new ApiError(401, "unauthorized request");
  }
```

```
    try {
      const decodedToken = jwt.verify(
        incomingRefreshToken,
        process.env.REFRESH_TOKEN_SECRET
      );
      const user = await User.findById(decodedToken?._id);
      if (!user) {
        throw new ApiError(401, "Invalid Refresh Token");
      }
      if (incomingRefreshToken !== user?.refreshToken) {
        throw new ApiError(401, "Refresh token is expired or used");
      }
      const options = {
        httpOnly: true,
        secure: true,
      };

      const { accessToken, newRefreshToken } =
        await generateAccessAndRefreshTokens(user._id);

      return res
        .status(200)
        .cookie("accessToken", accessToken, options)
        .cookie("refreshToken", newRefreshToken, options)
        .json(
          new ApiResponse(
            200,
            { accessToken, refreshToken: newRefreshToken },
            "Access toke refreshed"
          )
        );
    } catch (error) {
      throw new ApiError(401, error?.message || "Invalid Refresh Token");
    }
  });
  export { registerUser, loginUser, logoutUser, refreshAccessToken };
```

auth.middleware.js

```
  import jwt from "jsonwebtoken";
  import { ApiError } from "../utils/apiError.js";
  import { User } from "../models/user.model.js";

  export const verifyJWT = async (req, _, next) => {
    try {
      const token =
        req.cookies?.accessToken ||
        req.header("Authorization")?.replace("Bearer ", "");

      if (!token) {
        throw new ApiError(401, "Unauthorized Request");
      }

      const decodedToken = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET);
      const user = await User.findById(decodedToken?._id).select(
        "-password -refreshToken"
      );
```

```
    if (!user) {
      throw new ApiError(401, "Invalid Access Token");
    }
    req.user = user;
    next();
  } catch (error) {
    throw new ApiError(401, error?.message || "Invalid Access Token");
  }
};
```

mutler.middleware.js

```
import multer from "multer";

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "./public/temp");
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + file.originalname);
  },
});

export const upload = multer({ storage: storage });
```