* Create a repository
* Initialize the repository
* node\_modules, package.json, package-lock.json
* Install express
* Create a server
* Listen to port 7777
* Write request handlers for /test , /hello
* Install nodemon and update scripts inside package.json
* What are dependencies
* What is the use of "-g" while npm install
* Difference between caret and tilde ( ^ vs ~ )
* initialize git
* .gitignore
* Create a remote repo on github
* Push all code to remote origin
* Play with routes and route extensions ex. /hello, / , hello/2, /xyz
* Order of the routes matter a lot
* Install Postman app and make a workspace/collectio > test API call
* Write logic to handle GET, POST, PATCH, DELETE API Calls and test them on Postman
* Explore routing and use of ?, + , (), \* in the routes
* Use of regex in routes /a/ , /.\*fly$/
* Reading the query params in the routes
* Reading the dynamic routes
* Multiple Route Handlers - Play with the code
* next()
* next function and errors along with res.send()
* app.use("/route", rH, [rH2, rH3], rH4, rh5);
* What is a Middleware? Why do we need it?
* How express JS basically handles requests behind the scenes
* Difference app.use and app.all
* Write a dummy auth middleware for admin
* Write a dummy auth middleware for all user routes, except /user/login
* Error Handling using app.use("/", (err, req, res, next) = {});
* Create a free cluster on MongoDB official website (Mongo Atlas)
* Install mongoose library
* Connect your application to the Database "Connection-url"/devTinder
* Call the connectDB function and connect to database before starting application on 7777
* Create a userSchema & user Model
* Create POST /sigup API to add data to database
* Push some documents using API calls from postman
* Error Handling using try , catch
* JS object vs JSON (difference)
* Add the express.json middleware to your app
* Make your signup API dynamic to recive data from the end user
* User.findOne with duplucate email ids, which object returned
* API- Get user by email
* API - Feed API - GET /feed - get all the users from the database
* API - Get user by ID
* Create a delete user API
* Difference between PATCH and PUT
* API - Update a user
* Explore the Mongoose Documention for Model methods
* What are options in a Model.findOneAndUpdate method, explore more about it
* API - Update the user with email ID
* Explore schematype options from the documention
* add required, unique, lowercase, min, minLength, trim
* Add default
* Create a custom validate function for gender
* Improve the DB schema - PUT all appropiate validations on each field in Schema
* Add timestamps to the userSchema
* Add API level validation on Patch request & Signup post api
* DATA Sanitizing - Add API validation for each field
* Install validator
* Explore validator library funcation and Use vlidator funcs for password, email, photoURL
* NEVER TRUST req.body
* Validate data in Signup API
* Install bcrypt package
* Create PasswordHash using bcrypt.hash & save the user is excrupted password
* Create login API
* Compare passwords and throw errors if email or password is invalid
* install cookie-parser
* just send a dummy cookie to user
* create GET /profile APi and check if you get the cookie back
* install jsonwebtoken
* IN login API, after email and password validation, create e JWT token and send it to user in cookies
* read the cookies inside your profile API and find the logged in user
* userAuth Middleware
* Add the userAuth middle ware in profile API and a new sendConnectionRequest API
* Set the expiry of JWT token and cookies to 7 days
* Create userSchema method to getJWT()
* Create UserSchema method to comparepassword(passwordInputByUser)
* Explore tinder APIs
* Create a list all API you can think of in Dev Tinder
* Group multiple routes under repective routers
* Read documentation for express.Router
* Create routes folder for managing auth,profile, request routers
* create authRouter, profileRouter, requestRouter
* Import these routers in app.js
* Create POST /logout API
* Create PATCH /profile/edit
* Create PATCH /profile/password API => forgot password API
* Make you validate all data in every POST, PATCH apis
* Create Connnection Request Schema
* Send Connection Request API
* Proper validation of Data
* Think about ALL corner cases
* $or query $and query in mongoose - <https://www.mongodb.com/docs/manual/reference/operator/query-logical/>
* schema.pre("save") function
* Read more about indexes in MongoDB
* Why do we need index in DB?
* What is the advantages and disadvantage of creating?
* Read this arcticle about compond indexes - <https://www.mongodb.com/docs/manual/core/indexes/index-types/index-compound/>
* ALWAYS THINK ABOUT CORNER CASES
* Write code with proper validations for POST /request/review/:status/:requestId
* Thought process - POST vs GET
* Read about ref and populate <https://mongoosejs.com/docs/populate.html>
* Create GET /user/requests/received with all the checks
* Create GET GET /user/connections
* Logic for GET /feed API
* Explore the $nin , $and, $ne and other query operatorators
* Pagination

NOTES:

/feed?page=1&limit=10 => 1-10 => .skip(0) & .limit(10)

/feed?page=2&limit=10 => 11-20 => .skip(10) & .limit(10)

/feed?page=3&limit=10 => 21-30 => .skip(20) & .limit(10)

/feed?page=4&limit=10 => 21-30 => .skip(20) & .limit(10)

skip = (page-1)\*limit;