File: index.js

The order/flow of acquiring and using libraries, setting template engine and static files, using middlewares, acquiring routes and triggering controllers, etc, is essential in the main index.js file.

For example, you can face an error/warning/bugs if you

- try to render views before setting the path for views (because the server will not know where to look for the view files)
- use routes before initialising the session and passport.js (because it needs to be checked if the user is authenticated or not before rendering the requested page – for eg profile page can't be rendered if the user isn't authenticated i.e. not logged in.)
- etc.

Right Order (Generic)

1. Acquire library(libraries) before using.

2. After acquiring libraries, set the bodyParser middleware to parse user input data before it passes onto the views. (Usually, all websites will require this as at least 2 forms are always present for login/sign up.)
   ```
   app.use(bodyParser.urlencoded({ extended: false }));
   ```

3. Set the folder path for static files (i.e. the 'assets' folder) before it passes onto the views (i.e. CSS and JS files have to be read before they can be used by the views).

   Avoid using direct folder paths as (since this may not work sometimes to load styling) :
   ```
   app.use(express.static(__dirname + './assets'));
   ```

   Instead, use:
   ```
   app.use(express.static(path.join(__dirname, 'assets')));
   ```

4. Set express layouts before setting the view engine and rendering views via routes so that the layouts are accessible.
   ```
   app.set('view engine', 'ejs');
   app.set('views', './views');
   ```

NOTE for writing middlewares:

The next step is to set routes. But before setting routes call all the middlewares that your app requires Since middlewares have to be triggered before the controller is called. For example, over here passport.js middleware is required and used.

There can be multiple middlewares that can be chained together. Sometimes a middleware needs to be executed before another because the other depends on the first one. You can refer to the documentation to know the right flow whenever required.

5. Now before setting routes (for further rendering pages), first a middleware has to be called that takes and encrypts the session cookie. This step is required to know if the user was previously logged in or not, and whether a session cookie associated with it exists. Before pages can be rendered, it's important to know if a session already exists or not so that pages can be rendered accordingly. For example, the profile won't be rendered if the user isn't logged in.
```
app.use(session({
    name: 'codeial',
    // TODO Change the secret before deployment in production
mode
    secret: 'blah something',
    saveUninitialized: false,
    resave: false,
    cookie: {
        maxAge: (1000*60*100)
    },
    store: mongoStore.create(
        {
            mongoUrl: 'mongodb://localhost/test-app',
            autoRemove: 'disabled',
        }
    )
}));
```

6. After checking for the session -
   initialize passport on every route call.
   `app.use(passport.initialize())`

   allow passport to use "express-session" for maintaining session
   `app.use(passport.session())`

   to check if a session cookie is present or not *(This middleware has to be called after initializing passport and allowing it to use express-session!*

Otherwise, passport won’t be able to use the isAuthenticated() method from the express-session library.)

```
app.use(passport.setAuthenticatedUser);
```

7. Use the express router (to further render the views via controllers)

```
app.use('/', require('./routes'));
```

8. Make the server listen to the required port.

```
app.listen(port, function(err){
    if(err){
        console.log(`Error in running the server: ${err}`);
        return;
    }
    console.log(`Server is running on port ${port}`);
});
```