**Backend:**

├─ middleware/

├── admin.js

├── async.js

├── auth.js

├── error.js

├── validate.js

├── validateObjectId.js

├───── models/

├── admin.js

├── user.js

├───── routes/

├── login.js

├── forgot.js

├── dashboard.js

├──── startup/

├── config.js

├── cors.js

├── db.js

├── logging.js

├── routes.js

├── validation.js

├──── index.js

**Module:index.js**

Packages used: express

Modules used: All modules of startup folder.

**Scope:** Server creation logic should be implemented.

**Details:** In this module we should create the server and here we should Initialize all startup files and pass app argument to required files.

**Module:config.js (startup/)**

**Scope:** checks for the correct configuration.

**Details:** This module exports a function which checks whether the private key is defined in environment variables or not.

**Module:cors.js (startup/)**

Packages used: cors

**Scope:** Enables cross origin requests.

**Details:** This module exports a function with app parameter and initializes cors (app.use(cors()).

**Module:db.js (startup/)**

Packages used: mongoose

**Scope:** This module deals with the connecting implementations of database.

**Details:** This module exports a function with app parameter and initializes cors (app.use(cors()).

**Module:logging.js (startup/)**

Packages used: express-async-errors

**Scope:** Initializes the logging service.

**Module:routes.js (startup/)**

Packages used: express

Modules used: login.js, forgot.js, dashboard.js, error.js

**Scope:** Initializes all routes.

**Details:** This module initializes all the routes during the startup including the error middleware.

**Module:validation.js (startup/)**

Packages used: joi, joi-objectid

**Scope:** creating new property called object Id in Joi package.

**Details:** This module exports a function which adds new property (Joi.objectId=require(“joi-objectid”)(Joi)) to Joi class

**Module:admin.js (middleware/)**

**Scope:** checks if the user is admin or not.

**Details:** If req.user.admin does not exist then return this status code 403 and message unauthorized. If req.user.admin exists then call next().

**Module:async.js (middleware/)**

**Scope:** This handles the async error caused by all routes while making connection to database.

**Details:** we should export an anonymous async function with handler parameter and inside that function we should return another function with parameter of req, res, next and inside that function we should await handler in try catch block. It if causes exception then call next with exception argument passed from catch block.

**Module:auth.js (middleware/)**

Packages used: jsonwebtoken

**Scope:** This handles the authentication and checks the user is logged in or not.

**Details:** This module exports a middleware function, Inside that function we need to get token from user defined header (req.header(“user defined header”)) If there is no token, then return access denied with status code 401. If token has a value, then inside try catch block verify jwt token with secret key from server. If token is valid then assign decoded token into req.user and call next(), If token is invalid it will automatically throw an error and catch block will catch that error and in that catch block we should send invalid token message with status code 400 .

**Module:error.js (middleware/)**

**Scope:** Here you should log all your errors.

**Details:** Here we should export a function with err,req,res,next paramenters. Inside this function we should log all errors and after logging we should send “something failed” message with status code 500.

**Module:admin.js**

Packages used: joi, mongoose

**Scope:** This module deals with the admin collection.

**Details:** Admin collection should have document of every admin. Each Admin collection document should contain name, username, password, resettoken, isAdmin.

{

name:String,

username:String,

password:String,

resettoken:String,

isAdmin:Boolen

}

By default resettoken should be null and isAdmin should be false.

**Module:user.js**

Packages used: joi, mongoose

**Scope:** This module deals with the user collection.

**Details:** Admin collection should have document of every admin. Each Admin collection document should contain name, username, password, resettoken, scrapeddata.

{

name:String,

username:String,

password:String,

resettoken:String,

Scrapeddata:{

scrapeId:{data}

…

}

}

By default resettoken should be null and scrapeddata should be empty object.

**Routes list:**

api/login

api/forgot

api/dashboard

**Module:login.js** (/login)

Modules used: User.js, Admin.js

Packages used: express(express router), jsonwebtoken

**POST**

**Scope:** Post request in this route receives the login credentials (email Id and password) from the user or administrator.

**Details:** Receives JSON data(email Id and password) from user and checks whether the user or admin exists, if user Id and password matches then valid web token is sent to client and the page is redirected to **/dashboard** route, else wrong user Id or password message is returned with status code 400.

**User.js:** This module contains the User model and checks the user with given username and password.

**Admin.js:** This module contains the Admin model. If the given user Id and password didn’t find a match in User model then we have to search in Admin model.

**Module:forgot.js** (/forgot)

Modules used: User.js, Admin.js

Packages used: express(express router), nodemailer, jsonwebtoken, crypto

**POST**

**Scope:** Post request in this route receives the email Id from the user or administrator and sends reset link to their email Address using nodemailer package.

**Details:** Receives email id and checks whether the email id exist in database. If not exist then send message “given email id not registered” with status code 404. If exist then send reset link to the given email id. Reset link should be https://site.com/api/forgot/{provide token}

Token should be jwt token generated by using mongoDB id and email Id. Token should be encrypted and stored in database also.

**User.js:** This module contains the User model and checks the user with given email id exist.

**Admin.js:** This module contains the Admin model. If the given email Id didn’t find a match in User model then we have to search in Admin model.

**PUT**

**Scope:** Put request in this route receives the password and confirm password from the user or administrator and updates password.

**Details:** First find the user by email id(email can be found by decoding the jwt token sent by user) then decrypt the token stored in database and check whether the token sent by user and token in database are equal. If tokens are equal then check for matching of password and confirm password. If password and confirm password matches then update password with new password user email can be get by decoding the jwt token which is passed as a token params and after updating password delete token stored in database. If passwords didn’t match then send “passwords don’t match” message with status code 400.

**Module:dashboard.js** (/dashboard)

Modules used: User.js

Packages used: express(express router), childprocess

**POST**

**Scope:** Post request in this route receives the HTML file uploaded or HTML link sent by the user and then the HTML table is scraped and stored it in the the database in JSON format.

**Details:** After receiving the link sent by the user first we need to check whether the link is protected or not. If it is protected send message “Protected link cannot be used” with status code 400. If it is not protected then scrape the HTML table using beautifulsoup4 package in python and return the data from python process in JSON format and let node.js store it in database.

For scraping the HTML link or file we should spawn the childprocess and use python beautifulsoup4 package.

**User.js:** This module contains the User model and saves JSON data in database.

**POST/:id**

**Scope:** Post request in this route receives the scrape Id sent by the user via params and question is received through body of the request and the answer should be returned to client.

**Details:** After receiving the scrape Id and question we should understand what the question is and query the answer from the database using scrape Id and send clearly framed sentence to the client. We should understand the question using NLP and NLU in python.

**User.js:** This module contains the User model and searches the data in the database.

**Frontend:**

├─ src/components/

├── loginForm.jsx

├── forgotPassword.jsx

├── resetPassword.jsx

├── dashboard.jsx

├── askQuestion.jsx

├── common/

├── form.jsx

├── input.jsx

├── protectedRoute.jsx

├───── services/

├── authService.js

├── httpService.js

├── logService.js

├── scarpeService.js

├──── index.js

├── App.js

**Module:index.js**

Packages used: react-router-dom(BrowserRouter), bootstrap, react-toastify

Modules used: App.js.

**Scope:** Main file of this program.

**Details:** Here we should encapsulate App component inside BrowserRouter.

**Module:App.js**

Packages used: react-router-dom(Route, Redirect, Switch), bootstrap, react-toastify(ToastContainer)

Modules used: protectedRoute.jsx, loginForm.jsx, forgotPassword.jsx, resetPassword.jsx, dashboard.jsx, askQuestions.jsx, auth.js.

**Scope:** This module renders all components.

**Details:** This module encapsulate Route Component inside Switch component and wrap these in ToastContainer component. ToastContainer is user to show info, error etc to the user in a clean way. Encapsulate all these in ReactFragment and return inside render method. In this App module use ComponentDidMount method to check the authenticity of the user and set the user state.

**Module:loginForm.jsx(/src/components/)**

Packages used: react-router-dom(Redirect), joi-browser

Modules used: Form.jsx, auth.js

**Scope:** This module deals with the logic and rendering of login page.

**Details:** This LoginForm class should extend from Form class. This should contain two input box one for username and another for the password. We need to validate username and password using joi-browser package. If user is already logged in then redirect to home page.

**Module:forgotPassword.jsx(/src/components/)**

Packages used: react-router-dom(Redirect)

Modules used: Form.jsx, auth.js

**Scope:** This module deals with the logic and rendering of forgot password form.

**Details:** This ForgotPassword class should extend from Form class. This should contain one input box for email Id. We need to validate email using joi-browser package. If user is already logged in then redirect to home page. After clicking submit http post request should be sent to /api/forgot route with email Id.

**Module:resetPassword.jsx(/src/components/)**

Packages used: react-router-dom(Redirect)

Modules used: Form.jsx, auth.js

**Scope:** This module deals with the logic and rendering of reset password form.

**Details:** This ResetPassword class should extend from Form class. This should contain two input box one for password and another for confirm password. We need to validate passwords using joi-browser package. If user is already logged in then redirect to home page. After clicking submit http put request should be sent to /api/forgot/{token} route with password and confirm password.

**Module:dashboard.jsx(/src/components/)**

Packages used: react-router-dom(Redirect)

Modules used: Form.jsx, auth.js

**Scope:** This module deals with the submission of HTML file or a link.

**Details:** This Dashboard class should extend from Form class. If a user is not logged in then redirect to home page. This should contain one inputbox for entering the link, there should be a option to upload HTML file and a submit button. Either one of this should be selected at a time. And on the left side there should be all previous scraped items which is clickable, if user clicks that this should be redirected to askQuestion route where we can ask question related to that scraped data. This should send a http post request to /api/dashboard route.

**Module:askQuestion.jsx(/src/components/)**

Packages used: react-router-dom(Redirect)

Modules used: Form.jsx, auth.js

**Scope:** This module deals with asking questions.

**Details:** This askQuestion class should extend from Form class. If a user is not logged in then redirect to home page. This page should get a scrape Id from the user. If user didn’t get a scrape Id then redirect to dashboard. This page should display a message that “Your data is ready” and below that there should be a inputbox to ask question and there should be a submit button. After clicking submit button a Post request with id as parameter should be sent to the backend, then backend will send the answer to frontend and it should be displayed in this page.

**Module:form.jsx(/src/components/common)**

Packages used: joi-browser

Modules used: input.jsx

**Scope:** This module deals with all form logic.

**Details:** This module has all form logic and validation takes place in this module. This reflects the changes in the value of text box and updates the state accordingly.

**Module:input.jsx(/src/components/common)**

**Scope:** This module deals with inputbox.

**Details:** he skeleton of inputbox defined here and whenever any module need textbox they can use this.

**Module:protectedRoute.jsx(/src/components/common)**

**Scope:** This module protects the route from unauthorized user.

**Details:** This is user defined component to protect access to this route from unauthorized users. If any user try to access this route without logging in then the user should be redirected to login page.

**Module:authService.js(/src/services)**

Packages used: jwtDecode

Modules used: httpService.js

**Scope:** This module deals with authentication.

**Details:** Authentication services like login, logout, getting the current user and other user related things are dealt here.

**Module:httpService.js(/src/services)**

Packages used: axios, react-toastify(toast)

Modules used: logService.js

**Scope:** This module deals with http services.

**Details:** This deals with setting http headers and encapsulating all http related code in this module in case if we change the service. If any error occurs we need to call logging service and log the error.

**Module: logService.js(/src/services)**

Packages used: jwtDecode

Modules used: httpService.js

**Scope:** This module deals with logging error.

**Details:** All logging related code is encapsulated in this module. In case if we change the logging service we need to only here.

**Module:scrapeService.js(/src/services)**

Modules used: httpService.js

**Scope:** This module deals with scraping and asking questions.

**Details:** This module deals with calling backend service for uploading the HTML link or a file and asking questions and other all non user related endpoint.