Increment 3

Digitization of career events and job fairs

Team Members

Narra, Sailaja

Thungathurthy, Apurva

Gumireddy, Mahipal Reddy

Venkata Mahesh Mokkapati

Video link for this project:

https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6890d0ab-99c9-4217-a750-ade10060fe09

PPT:
https://docs.google.com/presentation/d/14qrFZVlgYe4QiFJYShJTj532Kdfba9JU/edit?usp=sharing&ouid=109043198401589299519&rtpof=true&sd=true

Report:
https://drive.google.com/file/d/1jfSE5VYSCYmhNnFOZLzvsPlrWlRXDclA/view?usp=sharing

Source code: https://github.com/Mahesh68/Web_Project2021_Fall

The Problem

Every day, it appears like more and more of the business world is moving online. Today's kids can expect to work in both the virtual and physical worlds in the future. Many of these youngsters, fortunately, grew up in an age of technology and have negotiated online activities and interactions. But, at a totally digital career fair, are college students prepared to give it their all?

It's difficult to make the shift from student to employee. In the best of conditions, presenting yourselves to recruiting recruiters is stressful.

Idea for the project

A physical career fair provides students with the structure they require to distinguish their student identities from their developing professional identities. It prepares them for success.

A digital career fair has various advantages. Employers and students will find it handier. Employers are not required to travel to attend additional events. You save money by not renting a location or hiring event staff, and there are no traffic or parking headaches. Students can check in between classes on their phones without worrying about their appearance. Career events is a valuable result of an activity that was designed to achieve the dream of students after graduation.

Project Design

For this project we used MEAN stack to develop the application for career events. As mentioned, full stack refers from the front end to the back end and the code that connects the two.

Front end: Angular

Backend: Node, Express JS

Database: Mongo DB

Front end: It is the part of a website or online application that is visible to the user and is responsible for their experience. The user interacts directly with the web application or website's front end.

Front End Framework:

Angular is an open source front-end framework written in JavaScript that is mostly used to create single-page web apps (SPAs). It is a framework that is always evolving and providing better ways to construct online apps. The static HTML is replaced by dynamic HTML. It is a free and open source project that anyone can use and modify. It adds Directives to HTML attributes and uses HTML to bind data.

Angular is widely regarded as one of the most popular technologies for developing flexible web applications among software professionals. According to the results of the latest Annual Developer Survey, Angular is used by 26.5 percent of professional developers around the world. So I preferred Angular for this application development.

Backend:

It refers to the development of a web application or website from the server side, with a primary focus on how the website functions. It's in charge of querying and interacting with APIs via client-side commands to manage the database. The front end, back end, and database are the three essential components of this type of website.

The back end is made up of several libraries, frameworks, and languages.

Project:

Making the events online and digitization of the events will become a preferred career network for students after graduation. It will be a platform for engaging with students efficiently and effectively.

Features we provide:

- One code multiple associations

- Authentication
- Authorisation
- Chatbot
- Register to an events
- Save and Fetch data from database

One code multiple associations

You'll need a PHP and JavaScript professional for your front end if you decide to build a website for your online business. In the back end, at the very least, you'll need a SQL expert. In contrast to these two requirements, the MEAN stack supports JavaScript in every aspect of its operation.

Because it includes MongoDB, a developer can use it to handle any database.

Similarly, the use of Node.js and ExpressJS allows a developer to create highly scalable and efficient web applications on the front end.

Authentication and Authorisation

Authentication is the act of confirming a person's identity, whereas authorisation is the process of confirming a user's access to specific apps, files, and data.

When the user initially runs the app, we developed an intriguing, animated splash screen.

When the application loads the index file, home component is displayed first. Next, we designed the look of each module inside its own component.

We added services to communication between the components and lazy components to the application. We added most of the features technically compared to developing an full pledged application.

Finally, during the previous iteration, we tweaked the code to improve the UI and UX.

Code Snippets

We have added bootstrap and jquery to the angular.json file to reflect these libraries over the application.

```
      "assets": [
        "src/favicon.ico",
        "src/assets"
      ],
      "styles": [
        "node_modules/bootstrap/dist/css/bootstrap.min.css",
        "node_modules/bootstrap-icons/font/bootstrap-icons.css",
        "src/styles.css"
      ],
      "scripts": [
        "node_modules/jquery/dist/jquery.min.js",
        "node_modules/popper.js/dist/umd/popper.min.js",
        "node_modules/bootstrap/dist/js/bootstrap.min.js"
      ]
    },
    "configurations": {
```

When the application is loaded “angular.json” file is loaded and from here “main.ts” file and “index.html” will be loaded into the browser from the platformbrowser module of angular. This is imported in the main.ts file.

Navigation

All pages have navigation; if you wish to add, change, or remove a navigation item, you can do it here. If you wish to add a navigation item, you'll need to replicate an existing navigation item and then update the value.

```
        [routerLinkActiveOptions]="{exact:true}">Home</a>
</li>
<li class="nav-item px-2">
    <a class="nav-link" routerLink="/about" routerLinkActive="active">About</a>
</li>
<li class="nav-item px-2">
    <a class="nav-link" routerLink="/authors" routerLinkActive="active">Authors</a>
</li>
<li class="nav-item px-2">
    <a class="nav-link" routerLink="/products" routerLinkActive="active">Events</a>
</li>
<li class="nav-item px-2">
    <a class="nav-link" routerLink="/lazy" routerLinkActive="active"
        [routerLinkActiveOptions]="{exact:true}">Lazy One</a>
</li>
<li class="nav-item px-2">
    <a class="nav-link" routerLink="/lazy/two" routerLinkActive="active">Lazy Two</a>
</li>
<li class="nav-item px-2">
    <a class="nav-link" routerLink="/admin" routerLinkActive="active">Admin</a>
</li>
```

Module System

Here in the angular app the mechanism to load the components is by using the modules. It defines an application. It is a container for the different parts of an application. Angular applications are modular and Angular has its own module system called NgModules.

NgModules are containers for a logical piece of code dedicated to a specific application domain, workflow, or set of capabilities. Components, service providers, and other code files whose scope is defined by the contained NgModule can be found in them. They can import functionality from other NgModules, as well as export functionality for usage by other NgModules.

Routing:

In this application every component loads based on the routing which is configured in the app routing module file

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'authors', component: AuthorRootComponent },
  { path: 'services', component: OfferingsComponent },
  {
    path: 'events',
    component: ProductsComponent,
    children: [
      { path: '', component: ProductNotSelectedComponent },
      { path: ':id', component: ProductDetailsComponent }
    ]
  },
  {
    path: 'lazy',
    loadChildren: () => import('./lazy/lazy.module').then(m => m.LazyModule),
    data: { preload: true, delay: 5000 }
  },
  { path: 'admin', component: AdminComponent, canActivate: [CanActivateAdminGuard] },
  { path: 'login', component: LoginComponent },
  {
    path: 'crud',
    component: CrudRootComponent,
    children: [
      { path: 'add', component: CrudAddEditComponent },
      { path: 'edit/:id', component: CrudAddEditComponent }
    ],
    canActivate: [CanActivateAdminGuard]
  },
```

Authorisation

Components can be accessed based on the user roles. If the criteria is satisfied these components will be routed or else will be navigated to the home page.

This criteria is handled as shown below

```
},
{ path: 'admin', component: AdminComponent, canActivate: [CanActivateAdminGuard] },
{ path: 'login', component: LoginComponent },
{
  path: 'crud',
  component: CrudRootComponent,
  children: [
    { path: 'add', component: CrudAddEditComponent },
    { path: 'edit/:id', component: CrudAddEditComponent }
  ],
  canActivate: [CanActivateAdminGuard]
},
{ path: '**', component: NotFoundComponent },
;
```

Connection with backend and database:

Here the connection from Angular to backend is made by using Http client service

```
@Injectable()
export class ProductsService {
  private url: string;

  constructor(private httpClient: HttpClient) {
    this.url = "http://localhost:8000/api/products";
  }

  getAllProducts() {
    return this.httpClient.get<Array<Product>>(this.url).pipe(
      retry(3),
      catchError(this._handleError<Array<Product>>('getAllProducts', []))
    )
  }

  getProduct(productId: number) {
    return this.httpClient.get<Product>(`${this.url}/${productId}`).pipe(
      retry(3),
      catchError(this._handleError<Product>('getProduct', undefined))
    )
  }
```

In the port 8000 the backend server is up and running.

Connection to the Mongo database is made using mongojs module in node.js

We have also jsonwebtoken module to generate a token based authentication to the user.

```
const mongojs = require("mongojs");
const jwt = require("jsonwebtoken");

const db = mongojs("career-job-fair", ["users"]);

const secretKey = "testing";

exports.requestToken = function (req, res, next) {
  var user = {
    email: req.body.username,
    password: req.body.password,
  };

  db.users.findOne({ email: user.email }, (err, stu_user) => {
    if (err) {
      reject({
        success: false,
      });
      return;
```

Here comes the full stack MEAN application. All the CRUD operations are handled here in the product service and this service is used as a provider for the whole application.

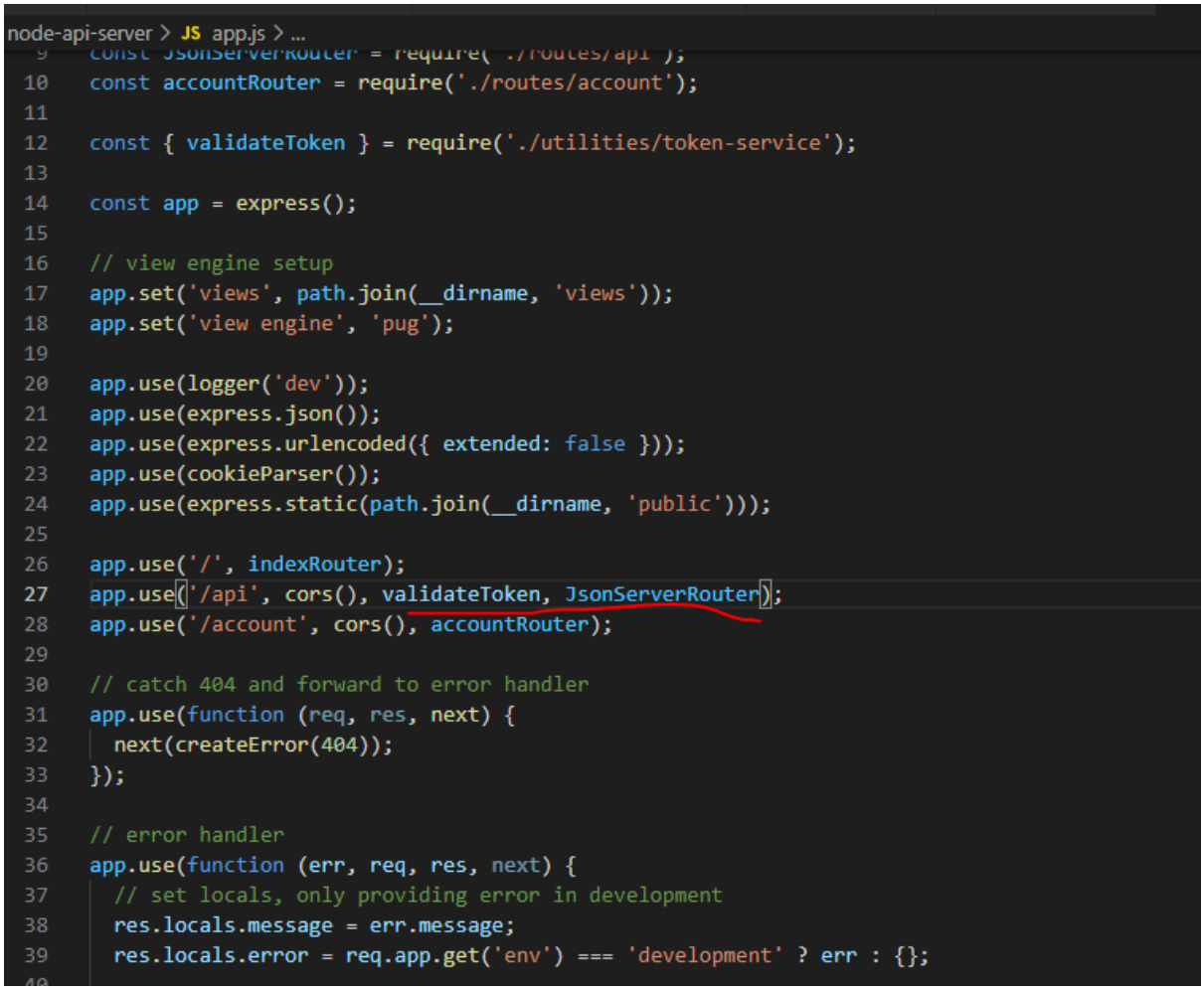
If the user is not authenticated there will be no valid json token for this session. Once the user wants to load any authenticated data he will be redirected to login component.

This feature is handled by the admin guard which returns a Boolean value if the user is authenticated or not. This admin guard is loaded by the can Activate module which is added for the component if required in the app routing module.ts file

When the request hits the node js server the initial file that's loaded is app.js file.

When two applications are communicating, if they are not from same domain the browser will block the incoming responses which is called as CORS(Cross Origin Resource Sharing).

So here in the app.js file we added a CORS module along with every request. So for every incoming request JSON web token is verified and then the data is fetched from the JSON file and DB.



Here the data that's coming for the events is handled in the local JSON file. This JSON can be also be made available as an API to the front end application.

For every request there will be request headers that's added in the interceptors while sending the response. HTTP Interceptors are those which modify and change the outgoing requests and incoming responses.

In our application every outgoing HTTP request is handled and 'x-access-token' is added to the headers. Similarly incoming responses can be handled and response can be modified.

```
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthenticatorService } from '../services/authenticator.service';

@Injectable()
export class TokenInterceptor implements HttpInterceptor {
  constructor(private authenticatorService: AuthenticatorService) { }

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    if (new RegExp('api').test(request.url)) {
      let token = <string>this.authenticatorService.getToken();

      const authReq = request.clone({
        setHeaders: {
          'x-access-token': token ? token : ""
        }
      });

      return next.handle(authReq);
    } else {
      return next.handle(request);
    }
  }
}
```

This token is verified and then response is given to the next handler.

```
exports.validateToken = function (req, res, next) {
  var token = req.headers["x-access-token"];

  if (token) {
    jwt.verify(token, secretKey, function (err, decoded) {
      if (err) {
        res.statusCode = 403;
        res.json({
          success: false,
          message: "Invalid Token Found",
        });
      } else {
        req.decoded = decoded;
        console.log(decoded);
        next();
      }
    });
  } else {
    res.statusCode = 403;
    res.json({
      success: false,
      message: "No Token Found",
    });
  }
};
```

Similarly for the user authentication, authentication service is used. Where we used the user data in the mongo DB and fetched the data as shown:

```
export const environment = {
  production: false,
  usersUrl: 'http://localhost:8000/api/users',
  accountUrl: 'http://localhost:8000/account/getToken'
};
```

In the environments file we stored the URL for accessing the server. This is useful for when making a builds for different environments.

Career events in action:

Mongo DB server :

DB Collections and the data

Angular application is up and running

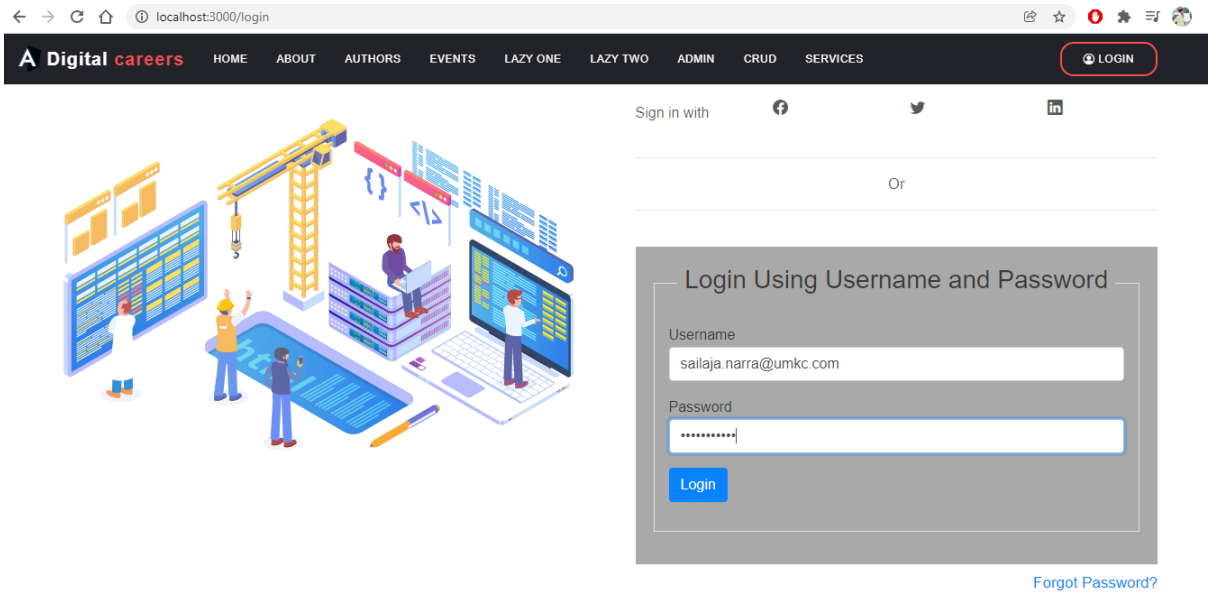
```
M:\UMKC\Fall2021\Web Programming\Repository\Web-fall-project\routing-app>ng serve
√ Browser application bundle generation complete.

Initial Chunk Files   | Names           |      Size
vendor.js             | vendor          |   3.33 MB
styles.css, styles.js | styles          | 558.30 kB
polyfills.js          | polyfills       | 475.08 kB
main.js               | main            | 240.51 kB
scripts.js            | scripts         | 170.16 kB
runtime.js            | runtime         |   9.00 kB

                      | Initial Total   |   4.75 MB

Lazy Chunk Files      | Names           |      Size
lazy-lazy-module.js   | lazy-lazy-module |   7.62 kB

Build at: 2021-11-14T23:55:43.599Z - Hash: f564c69f4a97b7cd9e45 - Time: 9000ms

** Angular Live Development Server is listening on localhost:3000, open your browser on http://localhost:3000/ **


√ Compiled successfully.
```

Node server is up and running

```
M:\UMKC\Fall2021\Web Programming\Repository\Web-fall-project\node-api-server>npm start

> ngo-api-service@0.0.0 start
> nodemon ./bin/www

[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./bin/www`
```

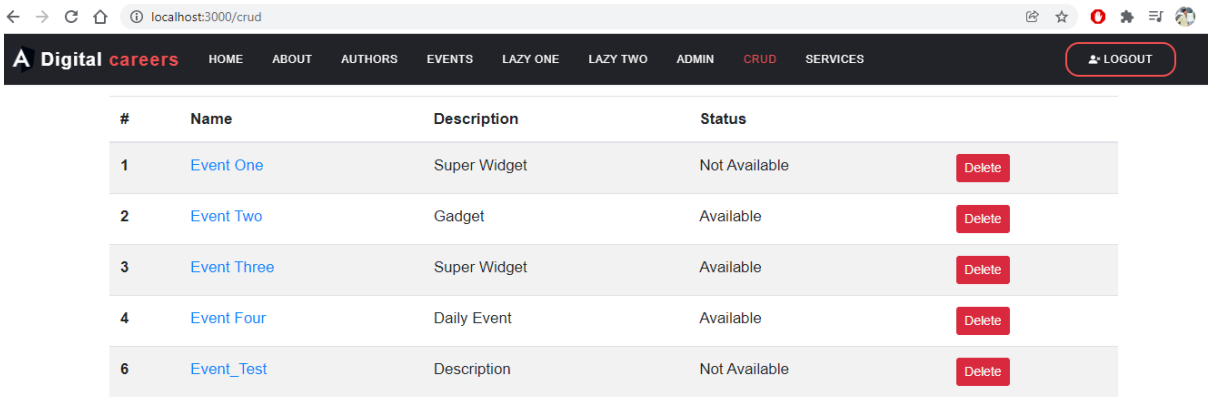
Login Component

Whenever the user clicks on login or some of the authentication required components, login component will be loaded initially if the token is not available as a cookie.



Once the authenticated user is logged in the authorization guard will be activated and the access will be permitted.

If the user is authorised below CRUD component will be loaded into the applicaition.



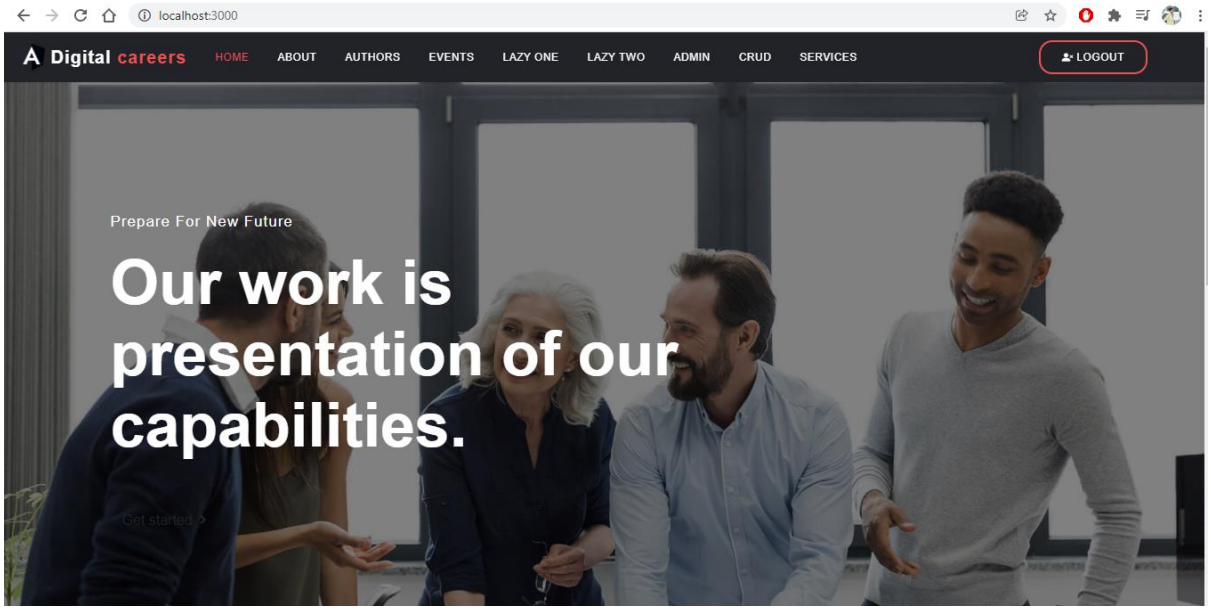
Here user will be accessed to the CRUD module if that user has role “Hr”

```
ngOnInit(): void {

  const role = sessionStorage.getItem("role");
  if (role !== "Hr") {
    this.router.navigate(['login']);
  } else{
    this.subscription = this.pubSubService.on('products-updated').subs
    this.loadProducts();
  }
}
```

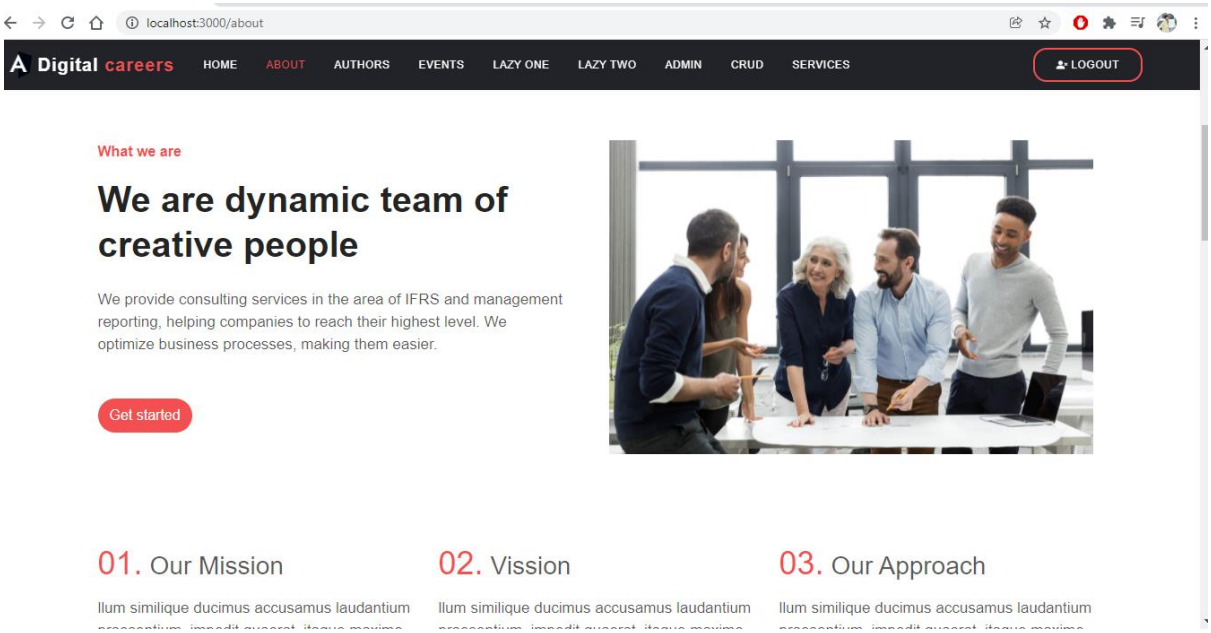
For all other users page will be redirected to the login component. Once the Login component is loaded into the application user will be logged out.

```
constructor(private formBuilder: FormBuilder, private route: ActivatedRoute,
  private router: Router, private authenticatorService: AuthenticatorService) {
  this.loginForm = this.formBuilder.group({
    username: ['', Validators.required],
    password: ['', Validators.required]
  });
}

ngOnInit(): void {
  this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/admin';
  this.authenticatorService.logout();
}
```

Landing page is about the application home page and as it is the main page for the application all the features are shown in this page

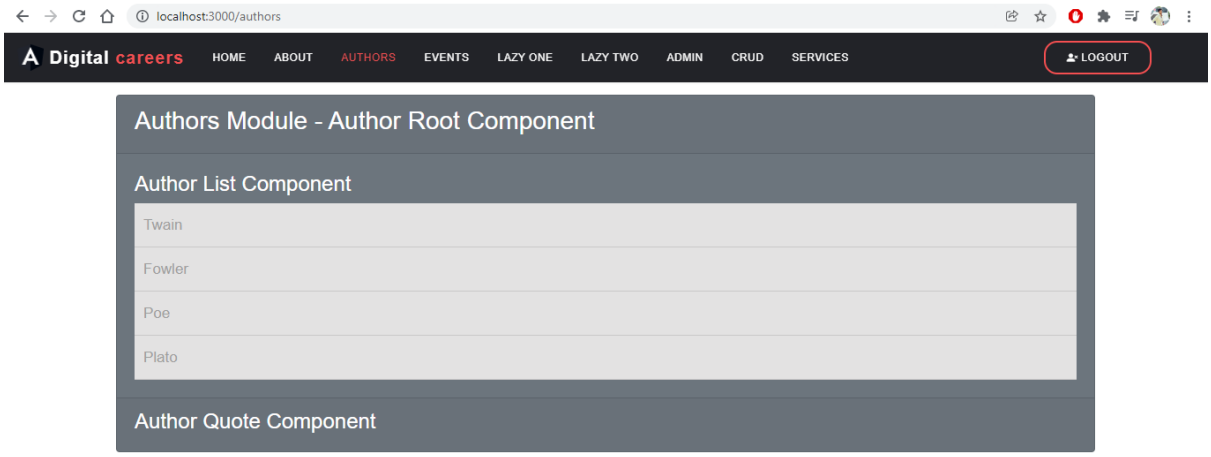


About my application is as shown below: Page URL will be navigated to ‘about’

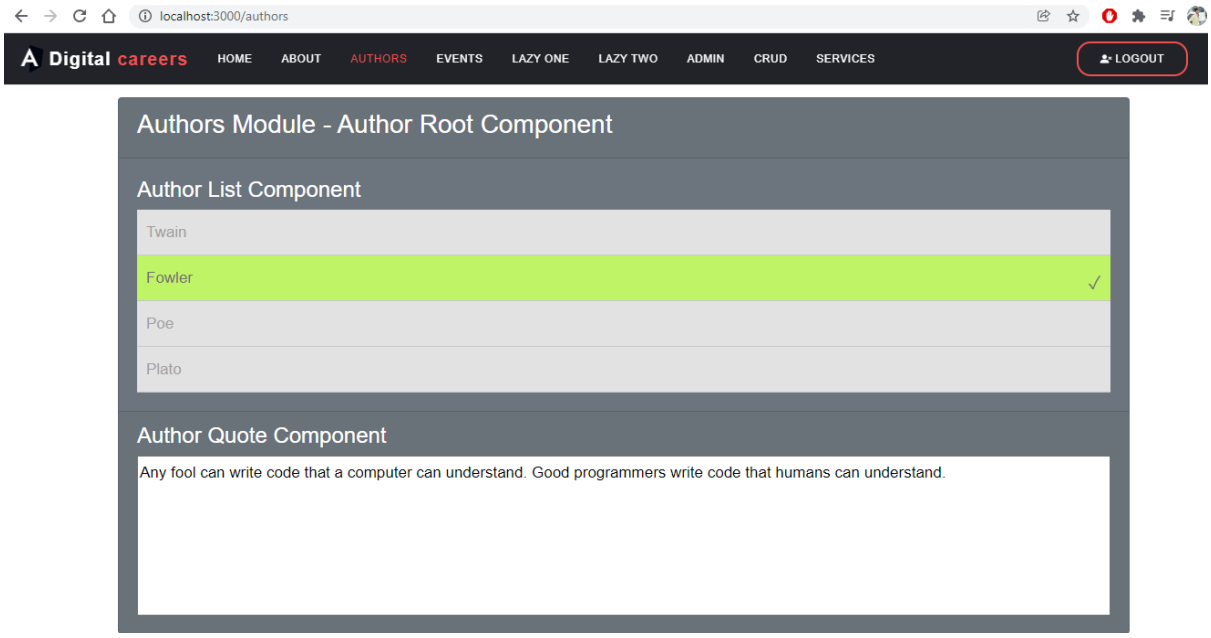


This is a SPA application so even if the user bookmarks the URL page will be redirected back to the same page without missing any data.

Authors component is loaded into the application when the user hits the authors menu. We developed this page for this application mainly to create an interesting topics to the students

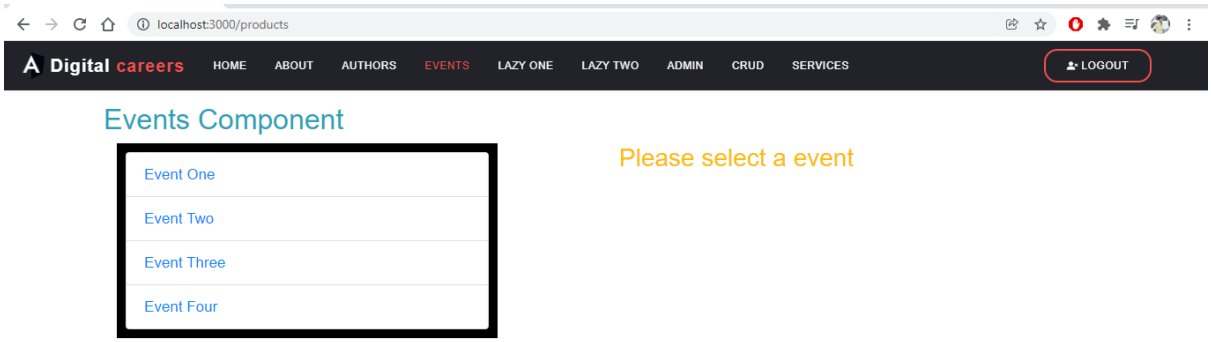


Once the user clicks on any author respective data will be displayed below

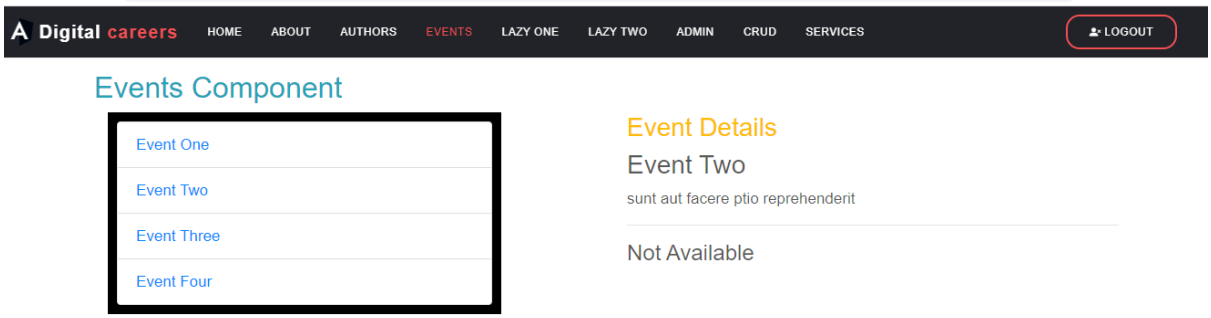


Events data

For every user who uses this application can have access to the events data. Not only to the events data, any component which not requires any authentication can be accessed.



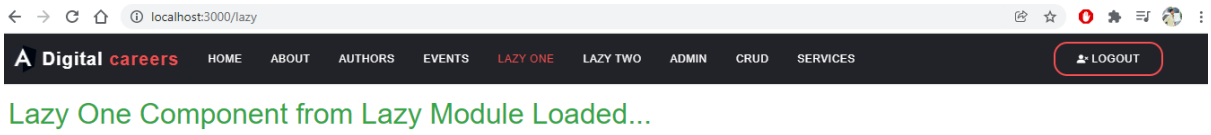
Once the user clicks on any event the data will be displayed to the right of the page.



Data is presented nicely as above. For this application UI can be modified and adjusted as required by the Product owner.

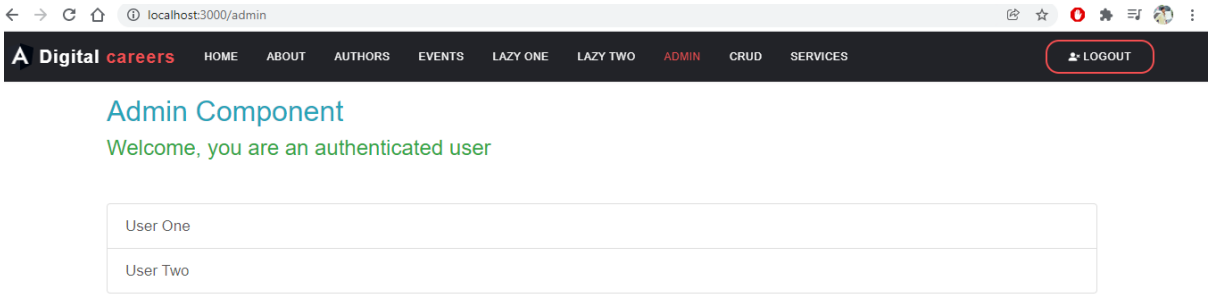
We have also covered the lazy module mechanism in our application. Everytime some of the menus are not opened by the user, if that's the case loading that component into the application is not necessary. This will impact on the performance of the application.

Hence this concept is called as lazy loading. So in our application we have added two lazy components so that this concept is not missing into our learning. Browser will load this module only if the user clicks on that component. This will increase the initial app load performance and no delay will be seen to the end user.

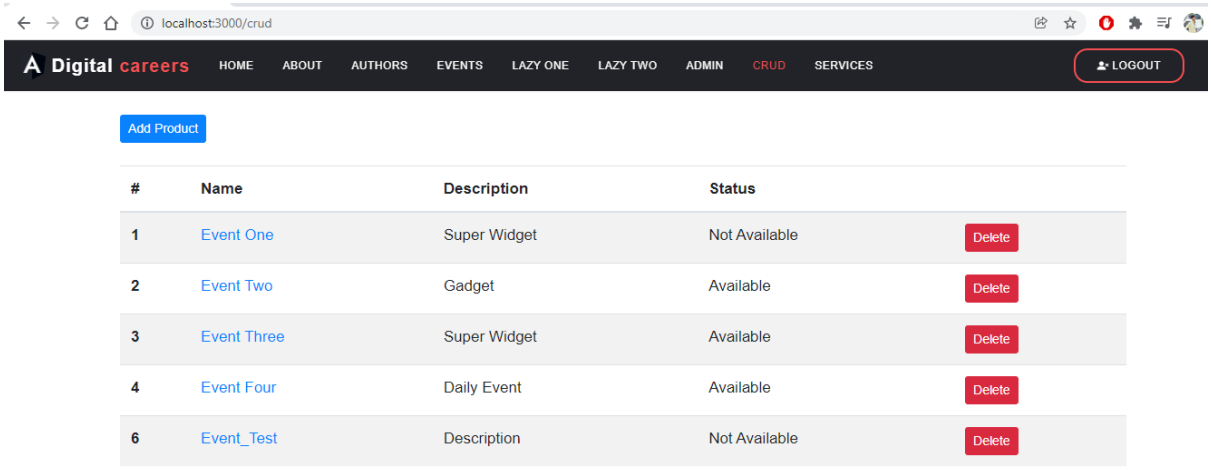


Admin Component:

Admin component will have all the users list. This requires authentication and authorisation.

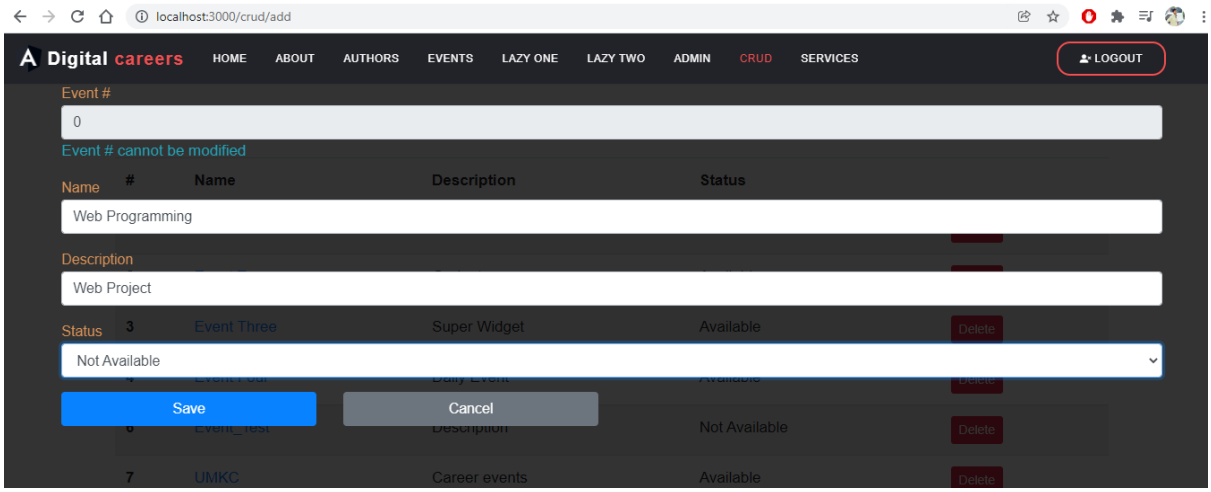


Register to an event can takes place from the CRUD module.



When user clicks on ADD product app routing module will load crud-add component into the UI. For this modal a nice animation has been added.

With this UI will be displayed as below



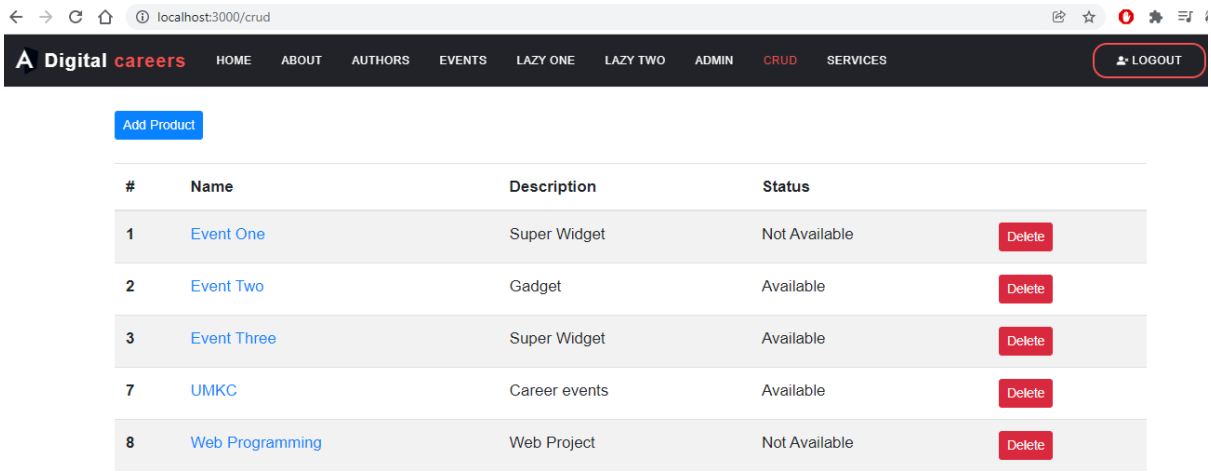
After user saves the data, API call will happen with the user entered data to the node server and data will be stored to the JSON file in the node server.

Add Product

#	Name	Description	Status	
1	Event One	Super Widget	Not Available	Delete
2	Event Two	Gadget	Available	Delete
3	Event Three	Super Widget	Available	Delete
4	Event Four	Daily Event	Available	Delete
6	Event_Test	Description	Not Available	Delete
7	UMKC	Career events	Available	Delete
8	Web Programming	Web Project	Not Available	Delete

Once the data is saved products will be loaded again and the actual response from the node server will be displayed. This happened because on click of save button page will be navigated back to the CRUD page. So here on page initialisation, getAllEvents will be called which gets the data in background.

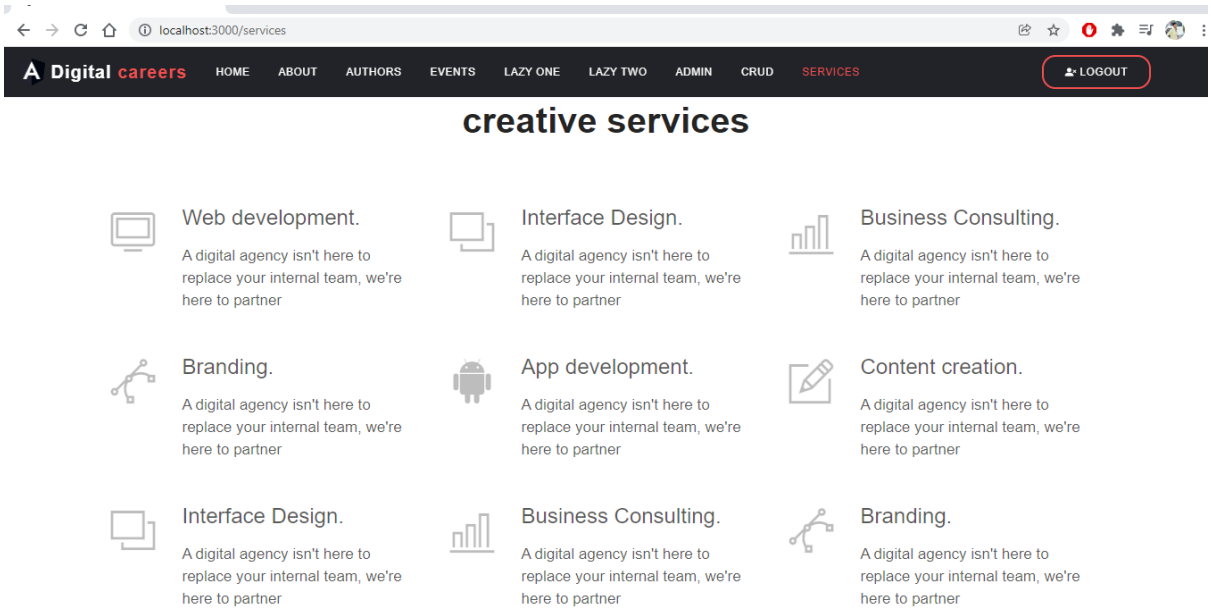
On click of delete button data will be removed from the DB and response is given back to the UI.



An delete HTTP call is made and on successful operation we will get the response with the products list.

As the product is an observable and there is change in data, UI will be updated here. Thanks to the rxjs observables for this nice and rich enhanced features.

Services offered:

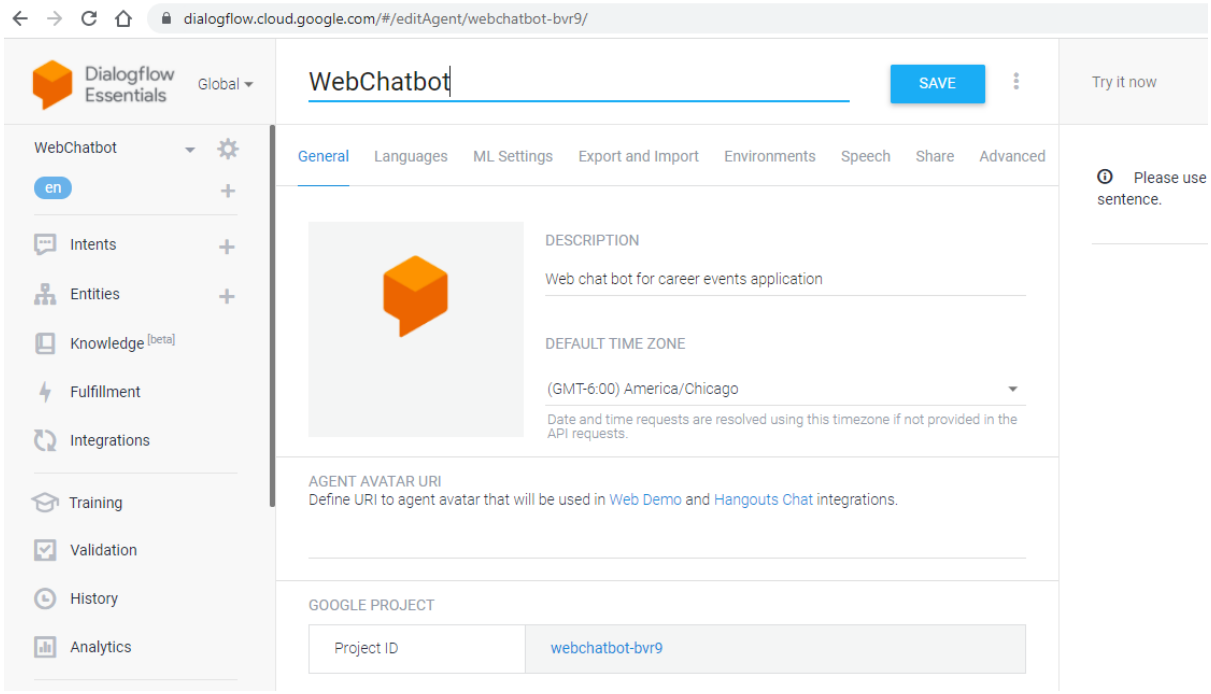


Chatbot

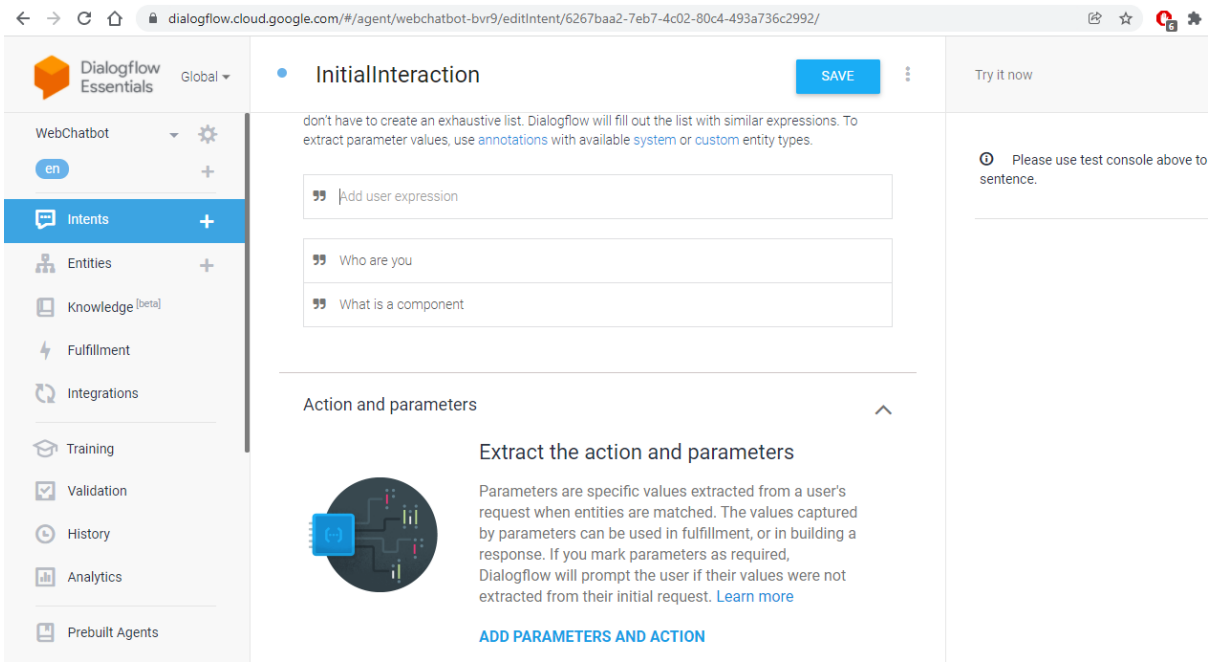
We introduced the chatbot into this nice and featured application. For this we want to use the google dialog flow application.

Created an account in the Google dialogflow app console with the school credentials.

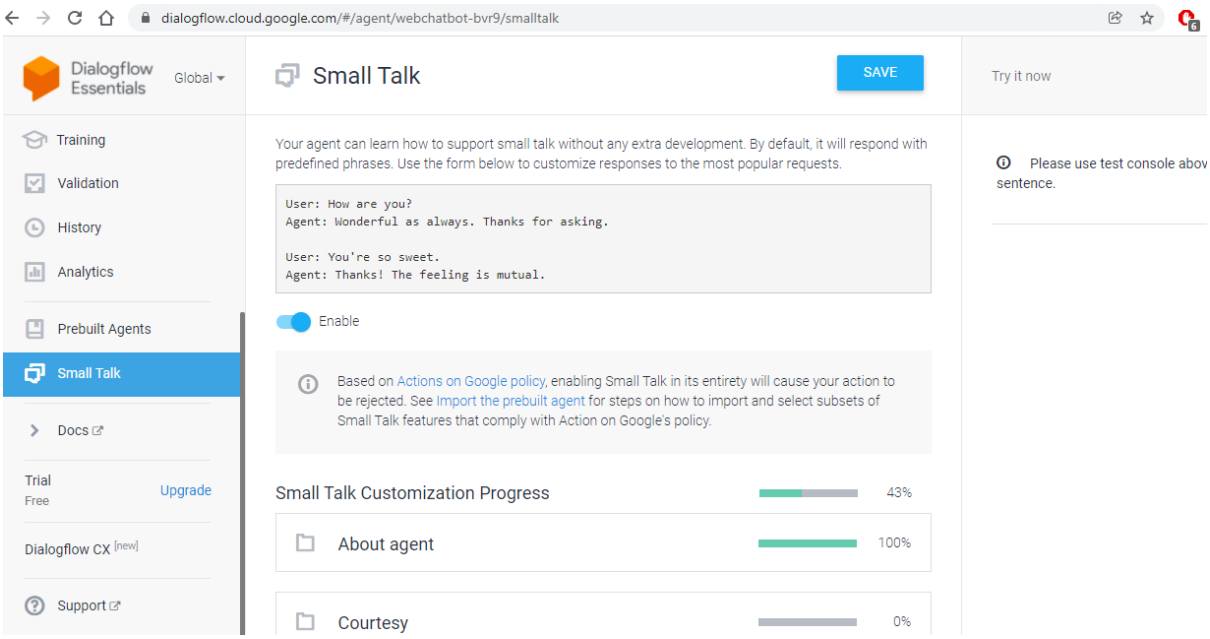
Created a new Web agent with the title and description



Next Intent is created with the title “InitialInteraction” and a training phrases are added to this intent. It represents the current context of a user’s request. For every new data new intent has to be created.



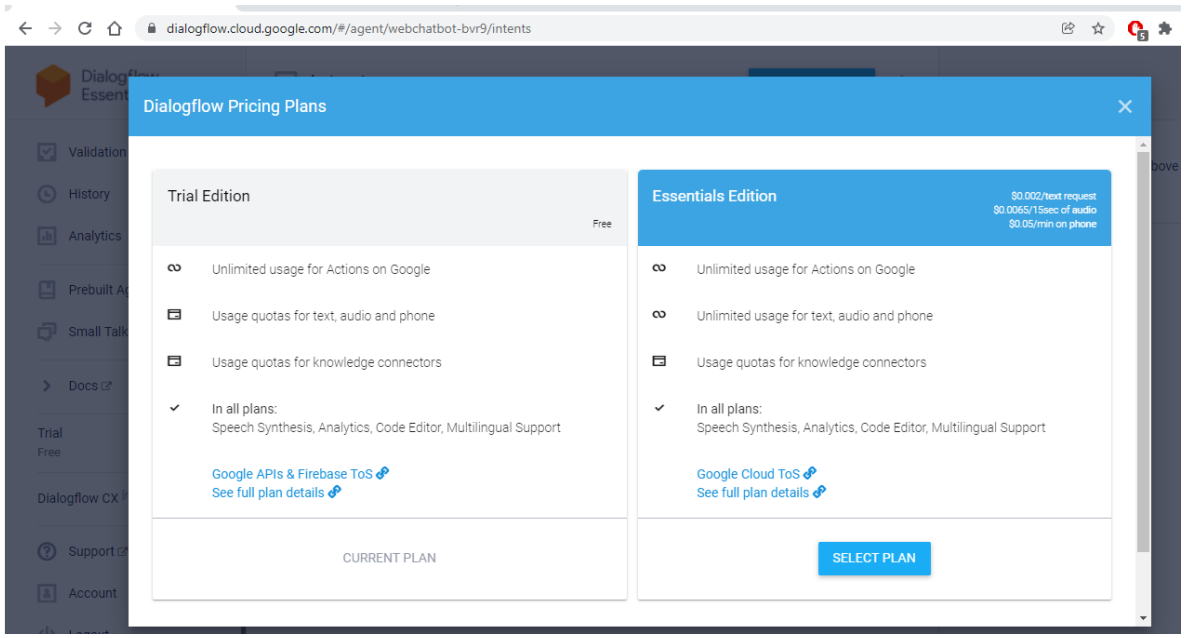
As creating an intent for every small conversation is a bit of lengthy procedure instead we can use the small talk feature given in this console.



Here basic questions and answers can be given to the chatbot to interact and make the conversation interactive and funny.

But the chatbot to use it requires an client key to use it in the application. If the client key is not available response form the chatbot will not happen.

Below are the plans for every request to the dialog flow API.



Hence we are not able to use the chatbot in the application development. But we have an idea how to use and develop the chatbot in our application. All the

interactions to the chatbot are ready for this feature. References for the development are given below in the references section.

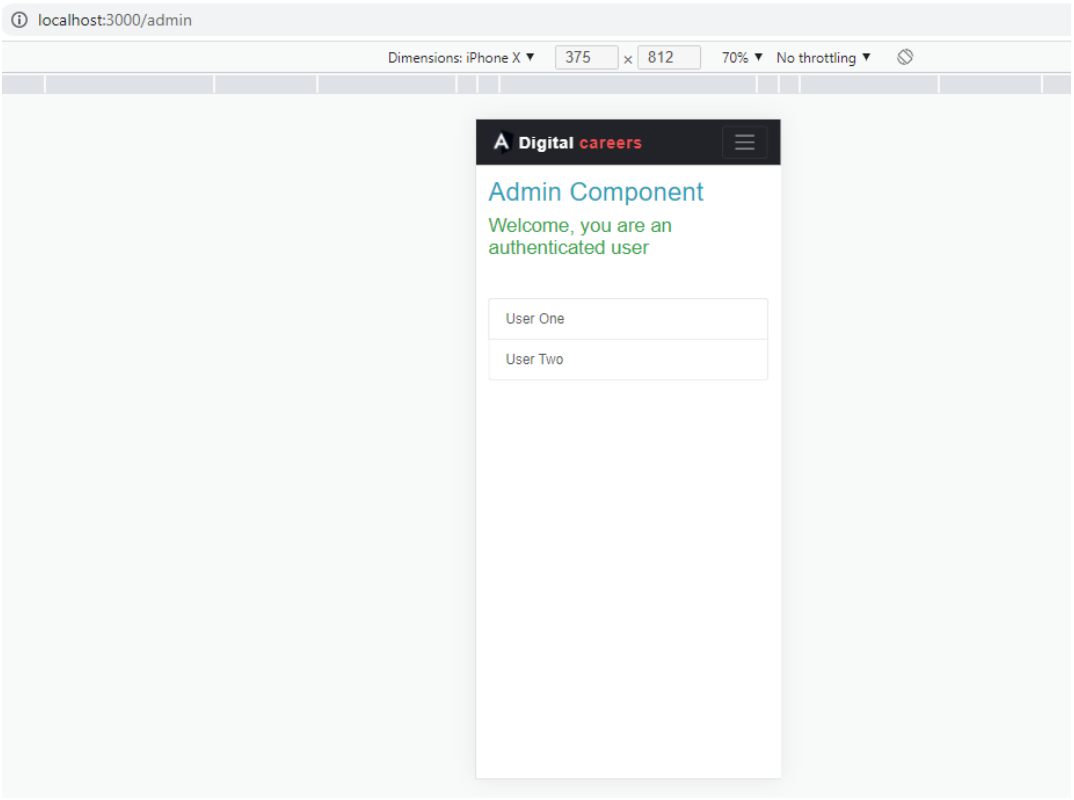
Filtering events

While the list of events are shown to the user he can also filter the events using filter button in the events menu.

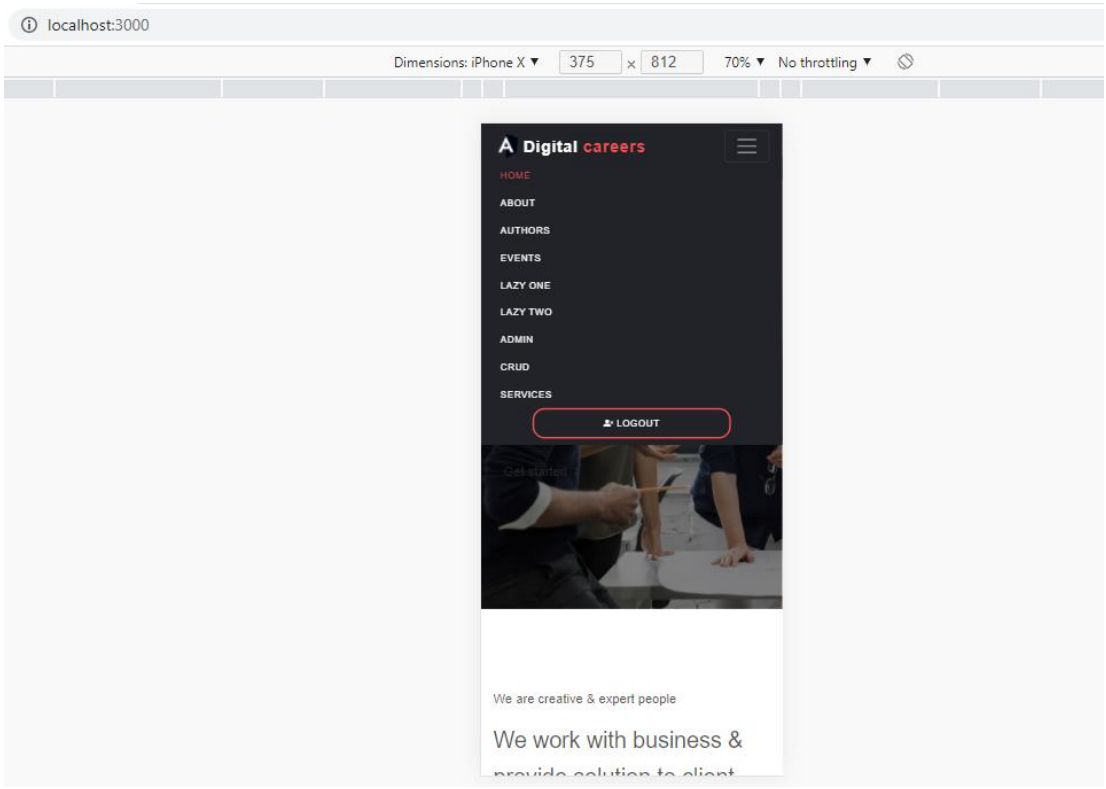
So based on the user role we can show only those events which are useful to the user. One more thing we can do is using filter button fetch only those events that are available in that area. But do to time constraints we are not adding this feature into the application.

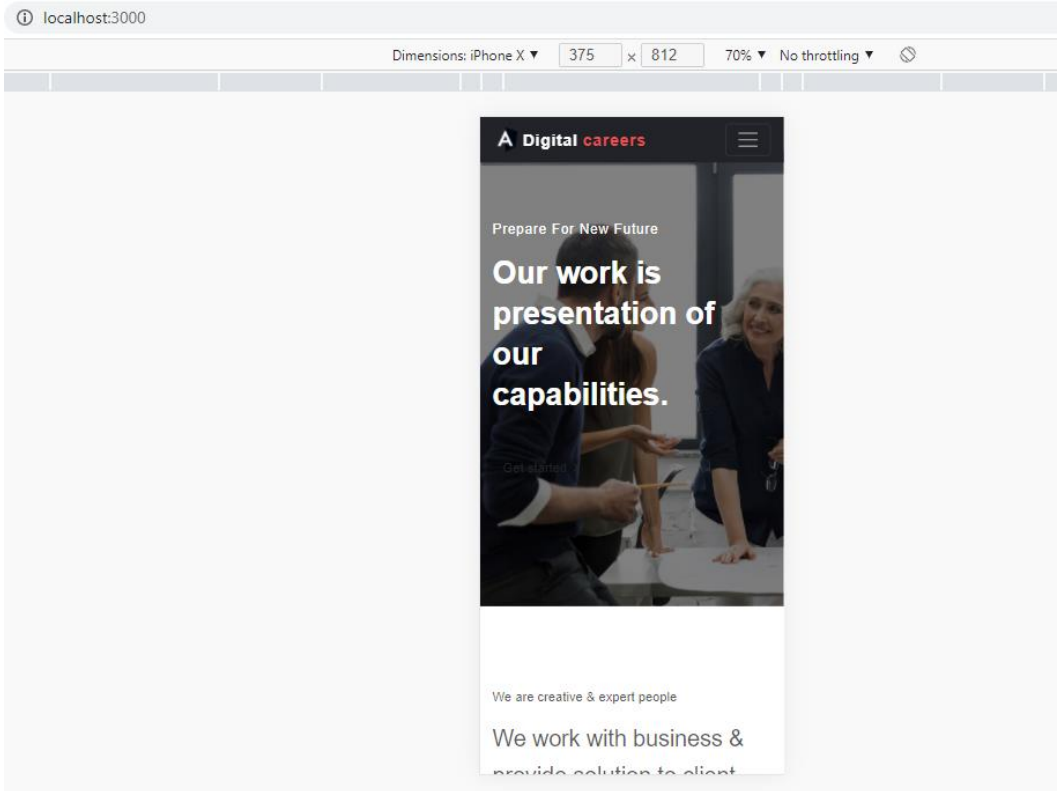
Responsiveness web design

Page is made responsive using bootstrap css and media queries

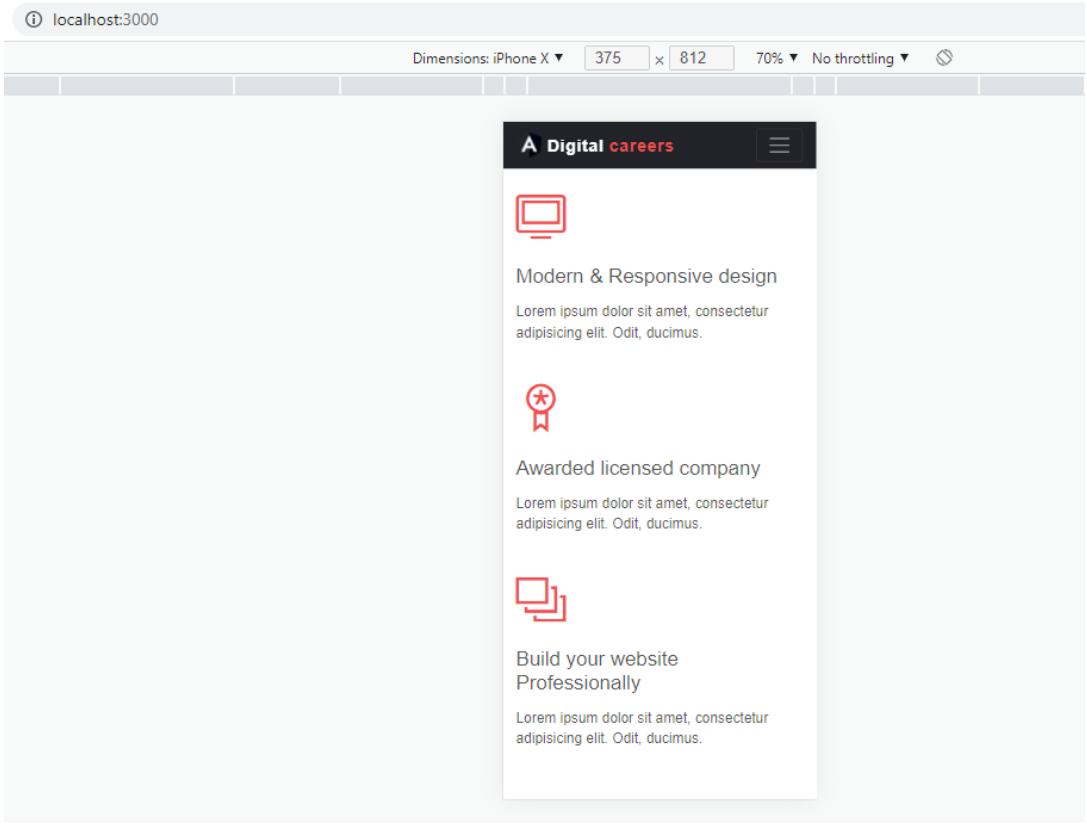


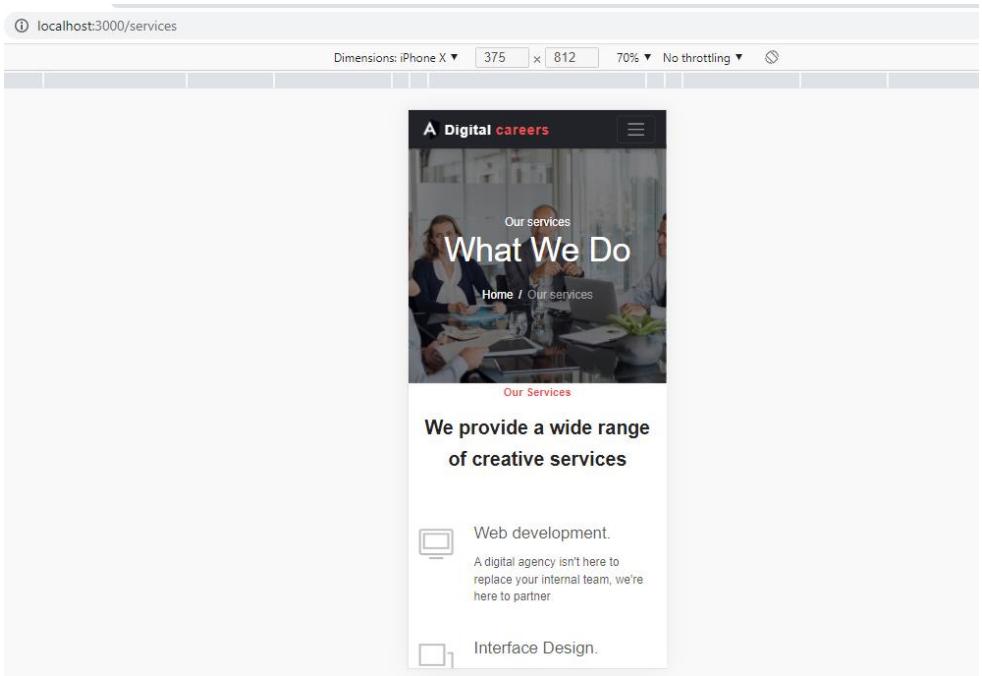
Menu with the hamburger icon

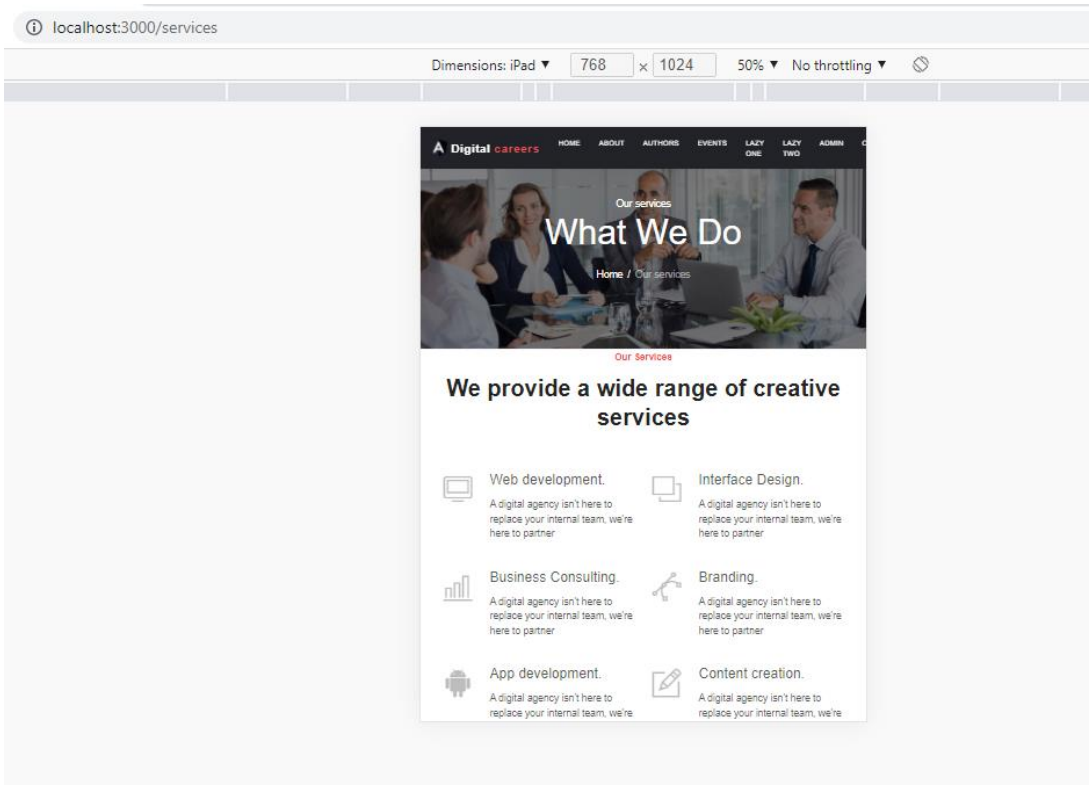


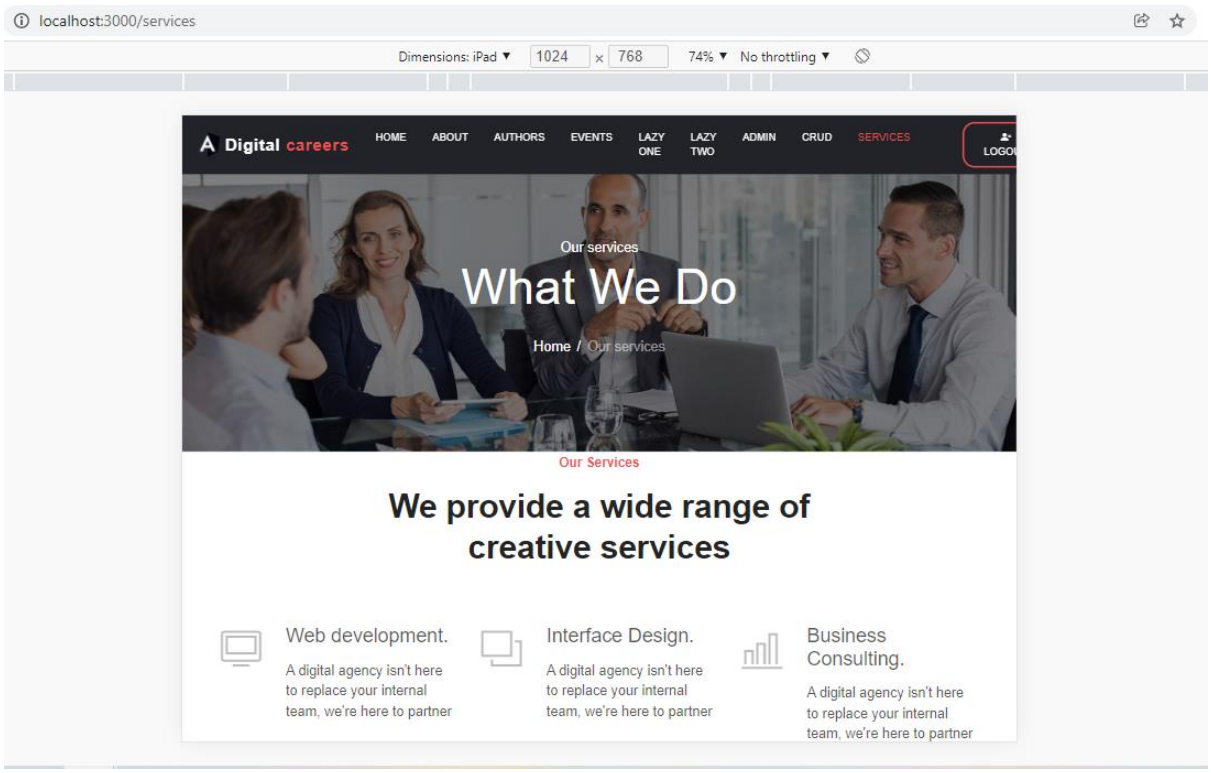


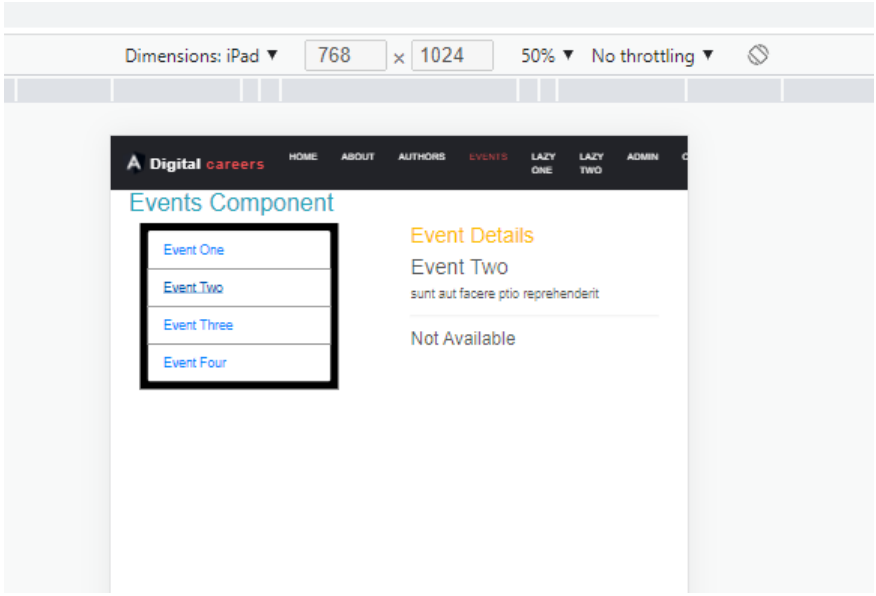
Services in action



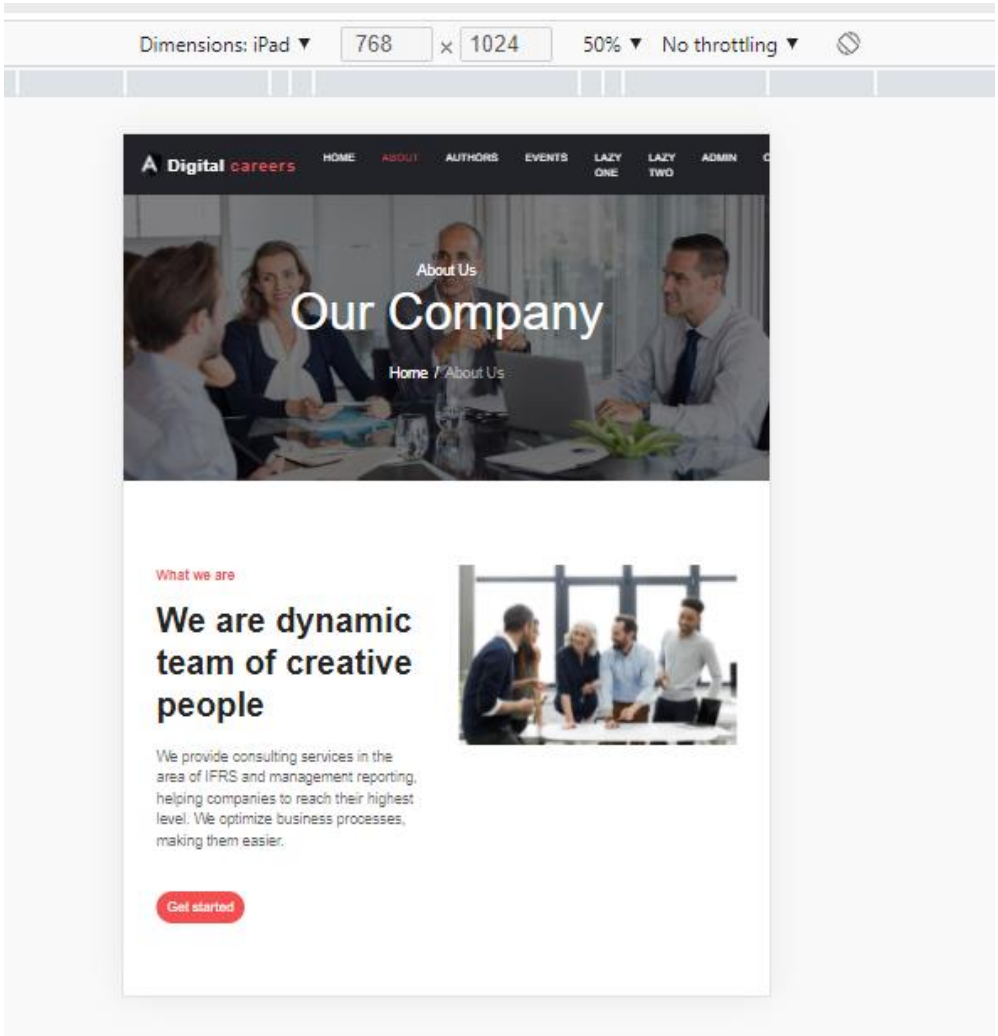



localhost:3000/services
Dimensions: iPad 768 x 1024 50% No throttling
Digital careers
HOME ABOUT AUTHORS EVENTS LAZY ONE LAZY TWO ADMIN
Our services
What We Do
Home / Our services
Our Services
We provide a wide range of creative services
Web development.
A digital agency isn't here to replace your internal team, we're here to partner
Interface Design.
A digital agency isn't here to replace your internal team, we're here to partner
Business Consulting.
A digital agency isn't here to replace your internal team, we're here to partner
Branding.
A digital agency isn't here to replace your internal team, we're here to partner
App development.
A digital agency isn't here to replace your internal team, we're
Content creation.
A digital agency isn't here to replace your internal team, we're


localhost:3000/services
Dimensions: iPad 1024 x 768 74% No throttling
Digital careers
HOME ABOUT AUTHORS EVENTS LAZY ONE LAZY TWO ADMIN CRUD SERVICES
Our services
What We Do
Home / Our services
Our Services
We provide a wide range of creative services
Web development.
A digital agency isn't here to replace your internal team, we're here to partner
Interface Design.
A digital agency isn't here to replace your internal team, we're here to partner
Business Consulting.
A digital agency isn't here to replace your internal team, we're here to partner


Dimensions: iPad
768
1024
50%
No throttling
Digital careers
HOME
ABOUT
AUTHORS
EVENTS
LAZY ONE
LAZY TWO
ADMIN
Events Component
Event One
Event Two
Event Three
Event Four
Event Details
Event Two
sunt aut facere ptio reprehenderit
Not Available

IPAD View


Dimensions: iPad
768
1024
50%
No throttling
Digital careers
HOME
ABOUT
AUTHORS
EVENTS
LAZY ONE
LAZY TWO
ADMIN
About Us
Our Company
Home / About Us
What we are
We are dynamic team of creative people
We provide consulting services in the area of IFRS and management reporting, helping companies to reach their highest level. We optimize business processes, making them easier.
Get started

Conclusion

We sought to make an application that might be used in a real-world situation for this project. We combined what we learnt in class and in our ICPs to produce a completely working and usable software with a fantastic UI and UX that students may use in their professions.

Contributions:

UI and FE handling: Sailaja Narra

Backend and DB: Venkata Mahesh Mokkapati

References:

https://artificialintelligence.oodles.io/blogs/building-chatbots-with-artificial-intelligence-and-angular/

https://www.youtube.com/watch?v=CKhV7-NF2OI&ab_channel=Fireship

https://dialogflow.cloud.google.com/

Video link for this project:

https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6890d0ab-99c9-4217-a750-ade10060fe09

PPT:
https://docs.google.com/presentation/d/14qrFZVlgYe4QiFJYShJTj532Kdfba9JU/edit?usp=sharing&ouid=109043198401589299519&rtpof=true&sd=true

Report:
https://drive.google.com/file/d/1jfSE5VYSCYmhNnFOZLzvsPlrWlRXDclA/view?usp=sharing

Source code: https://github.com/Mahesh68/Web_Project2021_Fall