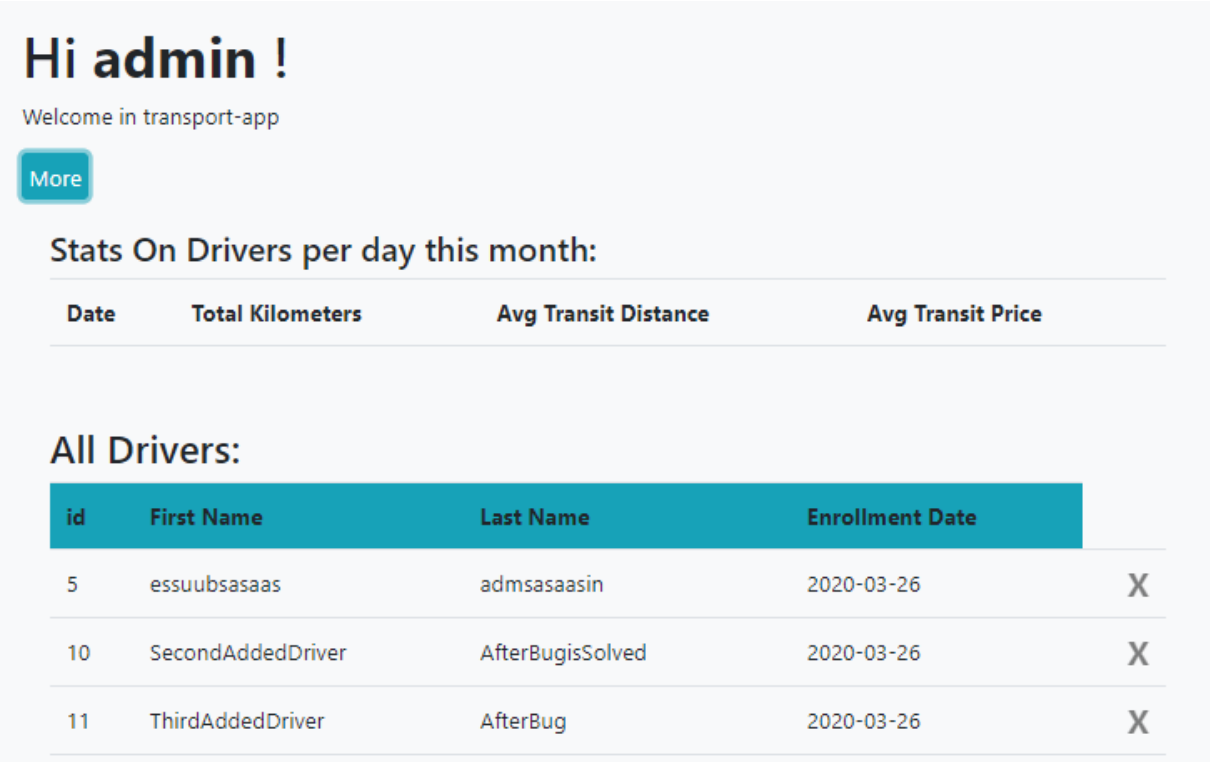
Inetrnet Engineering Project – Report

- Application main functionality is to communicate with REST API server. Application has several form that collects data from user and sends it as JSON and then shows responses to user.



Above we see home page of application, it has list of drivers that are stored in database of server.
Above the list there is small table that contains additional information about drivers' transits.
When we click on the 'X' in the above list driver is deleted in database.

Driver: **essuubsasaas admsasaasin**, id: **5**

Enrollment Date: **2020-03-26**

Extend

Totals:

Total Distance	Longest Transit	Most Expensive Transit
1427.17	596.6	2112

Per Month

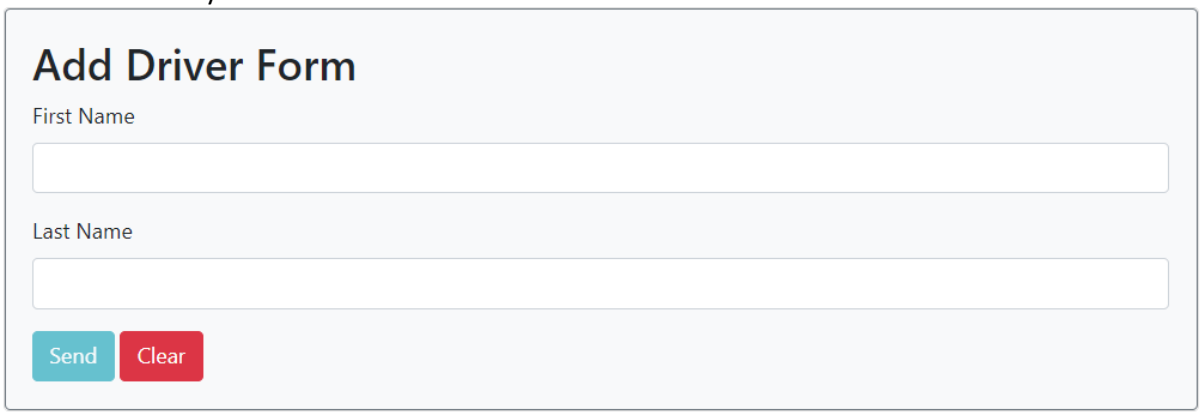
Date	Distance Sum	Longest Transit	Most Expensive Transit
2010-12	91.77	91.77	2112
2020-02	94.28	94.28	212
2020-04	1060.02	596.6	950
2020-05	181.1	95.75	1111

Transits:

id	Source	Destination	Price	Date	Distance	Delete
20	Tarnów, miłorzębowa 3	Kraków, pigonia 23	212	2020-02-20	94.28	X

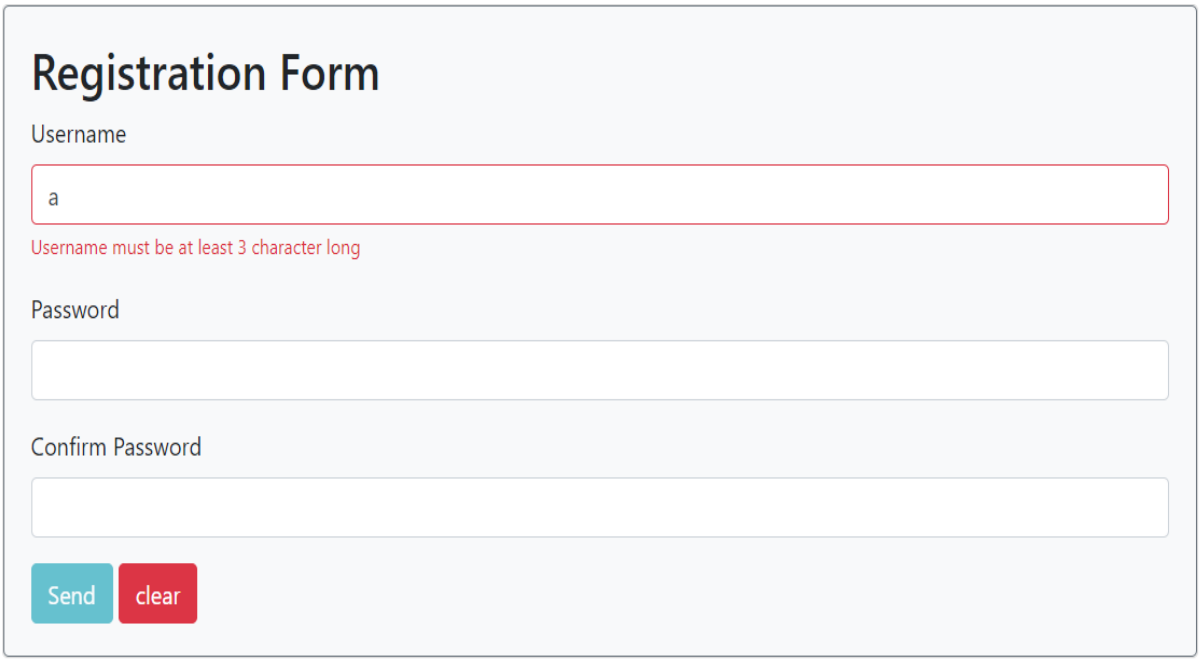
Screen shot above presents the driver page that is shown when the driver is clicked in the home page. This page contain list of driver's transits and also short report about that transits. When we click on the 'X' in the above list transit is deleted in database.

Above we see fully validated form which is used to transit to database.



Above is fully validated form which is used to add driver to database.

- All forms in application has validation for input data.



In the above example we see, that user entered too short username and form notified him about it.

- Every url in application except for registration and login urls I protected with authentication guard, it means user cannot use application without logging in.

```
export class AuthGuard implements CanActivate {

  constructor(private router: Router,
              private authenticationService: UserService) {
  }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    const currUser = this.authenticationService.currentUserValue;
    if (currUser) {
      //means user is authorized
      return true;
    } else {
      this.router.navigate( commands: ['/login'],  extras: {queryParams: {returnUrl: state.url}});
      return false;
    }
  }
}
```

Picture above contains code of authentication guard. We can see, if user is not logged in, and wants to enter protected url, then user is redirected to login page.

- Application communicates with server using JSON and http.

```
public login(username, password) {
  return this.http.post<any>(this.loginUrl,  body: {username, password})
    .pipe(map( project: user => {
      //save logged user to local storage
      localStorage.setItem('currentUser', JSON.stringify(user));
      //set current user
      this.currentUserSubject.next(user);
      return user;
    }));
}
```

In the picture above, we see that application uses http client to send request to server, and also sends body as JSON object. In this case user wants to log in, app takes given username and password, puts them to JSON body and finally sends http post request.

- Application has navigation menu which makes every functionality accessible to user

Home Logout Add Driver Add Transit Delete Driver Delete Transit Get Driver Scope Report

- Application properly handles communication errors with server.

```
this.userService.login(
  this.login.value,
  this.password.value
)
  .subscribe( next: data => {
      this.router.navigate( commands: [this.returnUrl]);
    },
     error: error => {
      let msg = 'Login failed - ' + (error.error ? error.error : 'unknown error');
      this.alertService.error(msg);
    });
this.clearForm();
```

In above example we see that if the logging in is successful then user is redirected to home page,
Otherwise when there is error user is informed about it.

Login failed - Bad credentials ×

- Application is connected to JWT protected Server therefore, this application utilizes JWT also.

```
public login(username, password) {
  return this.http.post<any>(this.loginUrl,  body: {username, password})
    .pipe(map( project: user => {
      //save logged user to local storage
      localStorage.setItem('currentUser', JSON.stringify(user));
      //set current user
      this.currentUserSubject.next(user);
      return user;
    }));
}
```

In above code we see that, after successful registration, application stores user credentials in its memory. User credentials contains JWT token obtained from server.

```
export class JwtInterceptor implements HttpInterceptor {
  constructor(private authenticationService: UserService) {
  }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    let currUser = this.authenticationService.currentUserValue;

    if (currUser && currUser.token) {  //if user logged in and has token then add header to request
      const tokenVal = 'Bearer ' + currUser.token;
      req = req.clone( update: {
        setHeaders: {
          'Authorization': tokenVal,
        }
      });
    }
    return next.handle(req);
  }
}
```

Above we see, that this class intercepts every request send to server from application. Then previously received JWT token is added to request so that user has access to protected resources in the server.

- App listens to data using “JSRX” library.

```
this.subscription = this.alertService.getAlert()
  .subscribe( next: message => { // if there is new message do what is typed below
    switch (message && message.type) {
      case 'success':
        message.cssClass = 'alert alert-success mt-2';
        this.hideAlertAfter( mills: 8000);
        break;
      case 'error':
        message.cssClass = 'alert alert-danger mt-2';
        this.hideAlertAfter( mills: 8000);
        break;
      case 'wait':
        message.cssClass = 'alert alert-primary mt-2';
        this.hideAlertAfter( mills: 8000);
    }
    this.message = message;
  });
```

When there is error, during communication, app automatically shows notification to user. We can see above the Observer pattern. When there is new event, above lambda expression is used to handle it. Example:

Registration failed: username is not available ×