Portfolio

2017-2018

Jordy Van der Haegen

GITHUB.COM/JORDYVANDERHAEGEN

Swappit

Swappit is an auction platform for festival tickets that connects both the buyer and seller in a secure environment. This platform guarantees that the end-user can exchange his festival-ticket in a safe and simple way.

This project is using Laravel as an API, connected to a MySQL database. The API is consumed in an Angular.

Backoffice

- Role-based authentication with Laravel Passport
- Control panel to perform CRUD actions
- Ticket checkout with Laravel Queue
- Email system with Laravel Queue

Frontoffice

- Pagination service
- Shopping cart / checkout service
- Route guards

Authentication

To authenticate users I've used Laravel Passport. Through Passport the API can hand out unique tokens which in turn gives you access to all account-based features on Swappit.

```php
public function signUp(RegisterRequest $request)
{
    if(User::where('email', request( key: 'email'))->whereNotNull('restore_token')->exists()) {
        dd( ...args: 'User has already signed up!');
    }
    else {

        $user = User::create([
            'name' => request( key: 'name'),
            'email' => request( key: 'email'),
            'password' => Hash::make(request( key: 'password')),
            'verify_token' => str_random( length: 40),
            'role' => 'guest',
        ]);

        $user->sendVerificationMail();
    }

    return response()->json( data: 'An email has been sent.');
}
```

Limiting user access

Some users should not have the same access as others, that's why there was the need of a role-based system.
I've opted to go for a simple solution where there are 3 fixed roles. The disadvantage to this solution is having no flexiblity to adjust the roles but provides a better performance.

```php
public function up()
{
    Schema::create( table: 'users', function (Blueprint $table) {
        $table->increments( column: 'id');
        $table->string( column: 'name');
        $table->enum( column: 'role',['superadmin','admin','guest']);
        $table->string( column: 'verify_token')->nullable()->unique();
        $table->string( column: 'restore_token')->nullable()->unique();
        $table->softDeletes();
        $table->string( column: 'email')->unique();
        $table->string( column: 'password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Checking out

All of the tickets on Swappit need to have a consistent state, we don't want users to buy tickets that have already been sold. That's why I have implemented a check-out-system divided in different phases. This secures that the user will always see what's still available and what's not.

```php
public function pay_store($id)
{
    $order = Order::findOrFail($id);
    $user = User::findOrFail($order->user_id);
    if(Auth::user()->id === $user->id && $order->status === 'placed'){
        $order->update(['status' => 'payed', 'payed_at' => Carbon::now()]);
        $user->sendOrderCompletedMail($order);
        $tickets = $order->tickets()->get();
        foreach ($tickets as $ticket) {
            $ticket->user->sendTicketSoldMail($ticket);
            $ticket->update(['sold' => 1]);
        }
        $order->update(['status' => 'completed', 'completed_at' => Carbon::now()]);
        return response()->json( data: 'Successfully payed order' . $order->id);

    }
    return response()->json( data: 'Access to this payment is denied.', status: 401);
}
```

```php
public function pay_create($id)
{
    $order = Order::with( relations: 'tickets.ticket_type.festival','tickets.user')->findOrFail($id);
    $user = User::findOrFail($order->user_id);
    if(Auth::user()->id === $user->id && $order->status === 'placed')
    {

        return response()->json(new OrderResource($order));
    } else {
        return response()->json( data: 'Access to this payment is denied.', status: 404);
    }
}
```

```php
public function pay_cancel($id) {
    $order = Order::findOrFail($id);
    $user = User::findOrFail($order->user_id);
    if(Auth::user()->id === $user->id && $order->status === 'placed') {
        $order->update(['status' => 'cancelled', 'cancelled_at' => Carbon::now()]);
        $tickets = $order->tickets()->get();
        foreach ($tickets as $ticket) {
            $ticket->update(['sold' => 0, 'published' => 1, 'order_id' => null]);
        }
    }

}
```

```php
public function pay($id) {
    $order = Auth::user()
        ->orders()
        ->findOrFail($id);

    $order->update(['status' => 'payed', 'payed_at' => Carbon::now()]);
    $order->user->sendOrderCompletedMail($order);
    $tickets = $order->tickets()->get();
    foreach ($tickets as $ticket) {
        $ticket->user->sendTicketSoldMail($ticket);
        $ticket->update(['sold' => 1]);
    }
    $order->update(['status' => 'completed', 'completed_at' => Carbon::now()]);
    return response()->json( data: 'Successfully payed order.');
}
```

Sending mail

When the API has to send multiple emails at once, the web request can take some time. To cut down this time and let the user have a faster experience I decided to include Laravel Queue into this project.

When the API has to send out an email it gets sent to a seperate table in the MySQL database. This table serves as a queue for certain tasks and will send out the email when it's ready.

Next to being significantly faster, a sytem crash will not cause your emails to not arrive since they are all stored in a database.

	id	queue	payload	attempts	reserved_at	available_at	created_at
1	263	default	{"displayName":"App\\Notifications\\OrderCompletedMail","job":"Illu…	255	<null>	1528443530	1528443530
2	264	default	{"displayName":"App\\Jobs\\CancelOrderAutomatically","job":"Illumin…	0	<null>	1528443670	1528443610
3	265	default	{"displayName":"App\\Jobs\\CancelOrderAutomatically","job":"Illumin…	0	<null>	1528444783	1528444723
4	266	default	{"displayName":"App\\Jobs\\CancelOrderAutomatically","job":"Illumin…	0	<null>	1528444975	1528444915
5	267	default	{"displayName":"App\\Jobs\\CancelOrderAutomatically","job":"Illumin…	0	<null>	1528445525	1528445465
6	268	default	{"displayName":"App\\Notifications\\OrderCompletedMail","job":"Illu…	0	<null>	1528445470	1528445470
7	269	default	{"displayName":"App\\Notifications\\TicketSoldMail","job":"Illumina…	0	<null>	1528445470	1528445470
8	270	default	{"displayName":"App\\Jobs\\CancelOrderAutomatically","job":"Illumin…	0	<null>	1528445547	1528445487
9	271	default	{"displayName":"App\\Jobs\\CancelOrderAutomatically","job":"Illumin…	0	<null>	1528448432	1528448372
10	272	default	{"displayName":"App\\Notifications\\OrderCompletedMail","job":"Illu…	0	<null>	1528448395	1528448395
11	273	default	{"displayName":"App\\Notifications\\TicketSoldMail","job":"Illumina…	0	<null>	1528448395	1528448395
12	274	default	{"displayName":"App\\Notifications\\VerifyMail","job":"Illuminate\\…	0	<null>	1528448687	1528448687

Shopping Cart

Saving the shoppingcart in the local storage gives the user the ability to perform actions in real-time without having to wait for a web-request.

The disadvantage of saving the cart locally is having a less consistent state of the tickets. A user might see a ticket available in his cart, but at check-out that ticket could be flagged as sold. This has been partially been solved due to checking ticket availablity more often.

```
clearCart(): void {
  localStorage.removeItem( key: 'cart');
  this._sidenavService.closeCartSideNav();
}

calculateCart(cart: ShoppingCart): void{
  if(cart.items.length > 0) {
    cart.price_total  = cart.items.map( callbackfn: (item) => parseFloat(item.price) ).reduce((previous, current) => previous + current);
    console.log(cart.price_total);
  }
  else {
    cart.price_total = 0;
  }
}

removeItem(cartitem: CartItem): void {
  let cart = this.getCart();
  let item = cart.items.findIndex( predicate: (x) => x.ticketId === cartitem.ticketId);
  cart.items.splice(item, deleteCount: 1);
  this.calculateCart(cart);
  this.saveCart(cart);
}
```

Design

When it comes down to the design of Swappit, I have only used 3 colours throughout the website. Thanks to the simple selection of colours I can indicate to the user what is more important by using colours instead of typography.

SWAPPIT

Veilingsplatform.

Typography.

Headings : Raleway Bold.

Normal text: PT Serif.

Colors.

Accent color.
#333333

Primary color.
#FDDD00

Background color.
#FFF

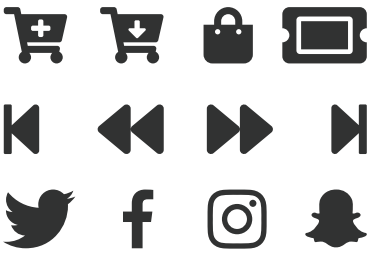
Buttons.

PRIMARY

SECONDARY

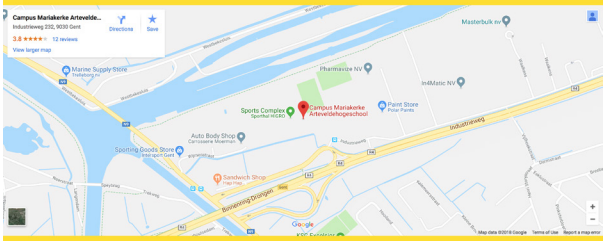
Icons.

Powered by Font awesome.



Swappit
TICKET SWAPPING

HOME TICKETS ACCOUNT CONTACT



Get in touch.

Industrieweg 232
9030 Gent-Mariakerke
Belgium
09 362 34 19

Email us.

Your name

Email address

Your phonenumber

Your message

SUBMIT

Swappit
TICKET SWAPPING

Email us at info@swappit.com
Call us: 09 362 81 20

Industrieweg 232
9030 Gent, Mariakerke

© Swappit 2018

Swappit
TICKET SWAPPING

HOME TICKETS ACCOUNT CONTACT

PRICE ↓ TICKET TYPE ↓

Tickets

Festival	Ticket	Prijs	Naam
Graspop	1x Dagticket Confort	€62.00	Estella McGlynn
Suikerrock	1x Dagticket	€256.00	Mr. Fermin Stiedemann
Dour	1x Combiticket Confort	€47.00	Selina Blick
Laundry Day	1x Food / drinks	€172.00	Felicita Murray
Suikerrock	1x Combiticket Confort	€121.00	Jazmin Schaden
Graspop	1x Combiticket	€79.00	Mr. Fermin Stiedemann
Suikerrock	1x Combiticket Confort	€111.00	Sydni Rosenbaum
Dour	1x Dagticket Confort	€244.00	Libbie Abbott
Dour	1x Dagticket Confort	€71.00	Madeline Russel II

1 2

Swappit
TICKET SWAPPING

Email us at info@swappit.com
Call us: 09 362 81 20

Industrieweg 232
9030 Gent, Mariakerke

© Swappit 2018

Swappit
TICKET SWAPPING

HOME TICKETS ACCOUNT CONTACT

All of your

Ticket #2

€46.00

Graspop - Dagticket Confort

Camping + Parking + Pendelbus Inbegrepen, voor een dag naar keuze van het festival.

Sold: ✖

Published: ✖

PUBLISH BUMP

Id	Date			Published	
#2	Graspop - D				
#30	Suikerrock -				
#46	Laundry Day - Combiticket Confort	€165.00	✓	✓	
#51	Laundry Day - Combiticket	€139.00		✓	

Swappit
TICKET SWAPPING

HOME TICKETS ACCOUNT CONTACT

DOUR - DAGTICKET CONFORT

Jul 11, 2018 - Jul 15, 2018

Camping + Parking + Pendelbus Inbegrepen, voor een dag naar keuze van het festival.

3 Available
8 Sold
0 Wanted

Festival	Ticket	Prijs	Naam
Dour	1x Dagticket Confort	€244.00	Libbie Abbott
Dour	1x Dagticket Confort	€71.00	Madeline Russel II
Dour	1x Dagticket Confort	€55.00	Sydni Rosenbaum

Swappit
TICKET SWAPPING

Email us at info@swappit.com
Call us: 09 362 81 20

Industrieweg 232
9030 Gent, Mariakerke

© Swappit 2018

Swappit
TICKET SWAPPING

HOME TICKETS ACCOUNT CONTACT

PUKKELPOP

Aug 15, 2018 - Aug 18, 2018

Pukkelpop is een jaarlijks terugkerend Belgisch muziekfestival in Kiewit, Hasselt. Het festival vindt traditioneel plaats in de tweede helft van augustus en duurde tot 2014 drie dagen. In 2015 werd het een vierdaags festival. Het festival wordt georganiseerd door The Factory vzw van onder anderen Chokri Mahassine.

TICKETS →

DOUR

Jul 11, 2018 - Jul 15, 2018

Het Dour Festival of kortweg Dour is een vijfdaags muziekfestival dat zichzelf de naam van 'European Alternative Music Event' geeft. Het vindt sinds 1989 ieder jaar plaats bij het Waalse dorp Dour.

TICKETS →

LAUNDRY DAY

Sep 1, 2018

Laundry Day is een Belgisch dancefestival in Antwerpen. Het festival ontstond in 1998 in de Antwerpse Kammenstraat. Het wordt elk jaar op de eerste zaterdag van september gehouden en is één van de laatste festivals van de Belgische festivalzomer.

TICKETS →

GRASPOP

Jun 21, 2018 - Jun 24, 2018

Graspop Metal Meeting is een jaarlijks meerdaags metalfestival in Dessel, in de Belgische provincie Antwerpen. Sinds 2008 trekt het festival elk jaar rond de 135.000 bezoekers.

TICKETS →

SUIKERROCK

Jul 27, 2018 - Jul 29, 2018

Suikerrock is een Belgisch muziekfestival, in de binnenstad van Tienen. De naam is ontleend aan de "suikerstad" Tienen: hier ligt namelijk de belangrijkste suikerraffinaderij van België. Suikerrock is ontstaan in 1986.

TICKETS →

Swappit
TICKET SWAPPING

Email us at info@swappit.com
Call us: 09 362 81 20

Industrieweg 232
9030 Gent, Mariakerke

© Swappit 2018

Greeward

Greeward started of as a hackaton-project to make Ghent a more environmentally friendly place. Students get rewarded for using their bike throughout the city.

It is build on NodeJS / Express and uses MongoDB as a database. The front-end is developped in React and Redux.

Backoffice

- Local and Facebook authentication
- Control panel to perform CRUD actions
- Google maps Distance Matrix API
- Liking and commenting on posts
- Following other users

Frontoffice

- Global state with Redux
- Displaying route on Google maps API

Authentication

Greeward is using PassportJS to authenticate users. Through Passport, the API hands out JSON web tokens which in turn gives you access to all account-based features.

```
exports.user_auth_local_post = function (req, res, next) {
  passport.authenticate('local', config.jwtSession, function (err, user, info) {
    if (err) { return next(err); }
    if (!user) {
      return res.status(401).json({
        'message': 'User Not Authenticated'
      });
    }
    req.auth = {
      id: user.id
    };
    const token = tokenUtils.createToken(req.auth);
    res.status(200).json({
      user: {
        id: user.id,
        email: user.email,
        first_name: user.first_name,
        last_name: user.last_name,
        avatar: user.avatar,
        stats: user.stats
      },
      token: `${token}`,
      strategy: 'local'
    });
  })(req, res, next);
}
```

Authenticating with Facebook

I had to implement Facebook authentication into Greeward which brought up an important issue, the local and facebook account not being linked. Pulling the e-mail from the Facebook-account and comparing it against all other e-mail adresses in the database has solved this issue.

```
exports.user_auth_facebook_post = function (req, res, next) {
  passport.authenticate('facebook-token', config.jwtSession, function (err, user, info) {
    if (err) { return next(err); }
    if (!user) {
      return res.status(401).json({
        'message': 'User Not Authenticated'
      });
    }
    req.auth = {
      id: user.id
    };
    const token = tokenUtils.createToken(req.auth);
    res.status(200).json({
      user: {
        id: user.id,
        email: user.email,
        first_name: user.first_name,
        last_name: user.last_name,
        avatar: user.avatar,
        stats: user.stats,
      },
      token: `${token}`,
      strategy: 'facebook-token'
    });
  })(req, res, next);
}
```

Likes and comments

Greeward also aims for interaction between users, that's why responding to someone else's activities is vital.

```
exports.like_create_post = function (req, res, next) {
  const id = req.params.activityId;
  Activity.findById(id, (err, activity) => {
    if (!activity) {
      res.status(404).json('User heeft geen volgers');
    }
    if (activity) {
      const isInArray = activity.likes.some((like) => {
        return like.equals(req.userId);
      });
      if (isInArray) {
        return errorHandler.handleAPIError(500, `Already liked a post with id: ${id}`, next);
      }
      if (!isInArray) {
        Activity.findByIdAndUpdate(id, {
          $push: { likes: req.userId }
        }, { new: true }).then(activity => {
          if (!activity) {
            return errorHandler.handleAPIError(404, `Activity not found with id: ${id}`, next);
          }
          res.status(201).json(req.userId);
        }).catch(err => {
          if (err.kind === 'ObjectId') {
            return errorHandler.handleAPIError(404, `Activity not found with id: ${id}`, next);
          }
          return errorHandler.handleAPIError(500, `Could not update activity with id: ${id}`, next);
        });
      }
    }
  });
}
```

```
exports.comment_create_post = function (req, res, next) {
  if (!req.body || !req.body.body) {
    return errorHandler.handleAPIError(400, `Comment must have a body`, next);
  }
  // Get the activity
  const id = req.params.activityId;
  // Create new comment
  const comment = new Comment({
    _user: req.userId,
    _activity: id,
    body: req.body.body,
  });
  comment.save((err, comment) => {
    if (err) return errorHandler.handleAPIError(500, `Could not save the new comment`, next);
    // Find activity and update comments
    Activity.findByIdAndUpdate(id, {
      $push: { comments: comment }
    }, { new: true })
      .then(activity => {
        if (!activity) {
          return errorHandler.handleAPIError(404, `Activity not found with id: ${id}`, next);
        }
        //res.send(activity);
      }).catch(err => {
        if (err.kind === 'ObjectId') {
          return errorHandler.handleAPIError(404, `Activity not found with id: ${id}`, next);
        }
        return errorHandler.handleAPIError(500, `Could not update activity with id: ${id}`, next);
      });
      const query = Comment.findById(comment._id).populate({path: '_user'});
      query.exec((err, com) => {
        res.status(201).json(com);
      });
  });
}
```

Generating routes

For testing purposes I didn't have any trajectories of bicycle-rides available so I had to come up with a solution. By generating two random latlong points in Ghent and feeding them to the Google Distance Matrix API I was able to make the right measurements.

After getting the data back from the API, I can create an Activity and store the needed information in the database.

```
exports.activity_create_post = function (req, res, next) {
  let start = getRandomLatLng();
  let stop = getRandomLatLng();
  googleMapsClient.distanceMatrix({
    origins: { lat: start.lat, lng: start.lng },
    destinations: { lat: stop.lat, lng: stop.lng },
    mode: 'bicycling',
  })
    .asPromise()
    .then((response) => {
      let distance = response.json.rows[0].elements[0].distance.value;
      let moving_time = response.json.rows[0].elements[0].duration.value;
      let points = distance / 200;
      let co = distance / 1000;
      const activityDetail = {
        start_lat: start.lat, start_lng: stop.lng, stop_lat: stop.lat, stop_lng: stop.lng,
          distance: distance, points: points, moving_time: moving_time, _user: req.userId
      };
      const activity = new Activity(activityDetail);
      activity.save((err, activity) => {
        updateUserStats(activity, distance, points, req.userId);
        if (err) return errorHandler.handleAPIError(500, `Could not save the new activity`, next);
        res.status(201).json(activity);
      })
    })
    .catch((err) => {
      console.log(err);
    });

}
```

Managing state

Sometimes it's not easy to get the state of other components in React, that's why I needed a global state manager like Redux.

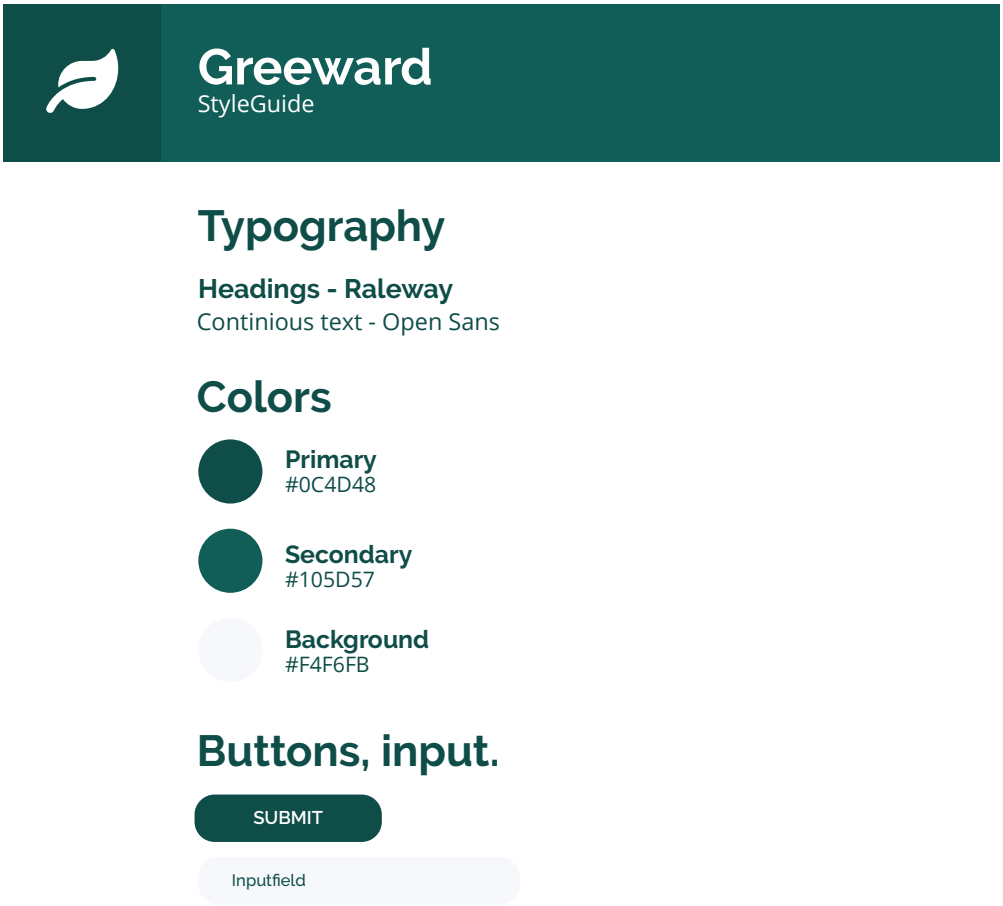
Redux has been implemented into Greeward to store the authenticated user's information like name, jwt-token so I can easily access them from anywhere.

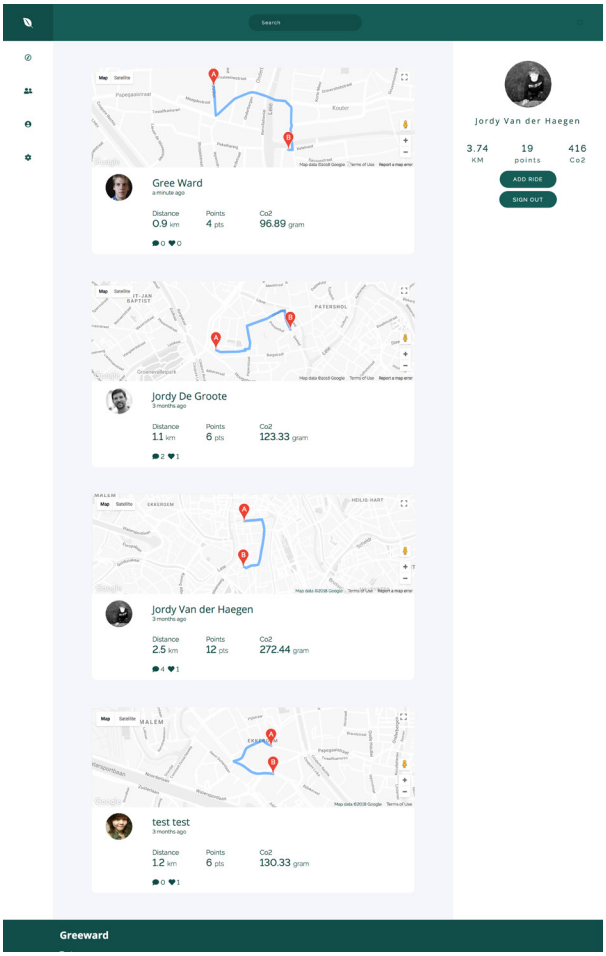
```
export function signInActionLocalStrategy({ email, password }, history) {
  return async (dispatch) => {
    try {
      const postData = new Blob([JSON.stringify({ email: email, password: password }, null, 2)], { type: 'application/json' });
      const options = {
        method: 'POST',
        body: postData,
        mode: 'cors',
        cache: 'default'
      };
      const response = await fetch('https://greeward.herokuapp.com/api/v1/auth/local', options);
      if(response.status !== 200) {
        throw new Error("Not 200 response");
      }
      const responseJson = await response.json();

      dispatch({
        type: AUTHENTICATED,
        payload: responseJson
      });
      localStorage.setItem('mobdev2_auth', JSON.stringify(responseJson));

    } catch (error) {
      dispatch({
        type: AUTHENTICATION_ERROR,
        payload: 'Invalid email or password'
      });
    }
  };
}
```

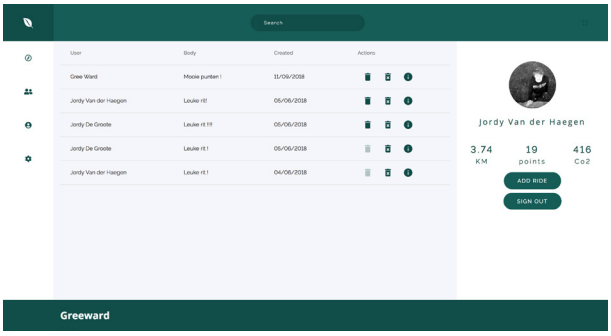
```
function authReducer(state = initialState, action) {
  switch (action.type) {
    case AUTHENTICATED:
      return Object.assign({}, state, {
        authenticated: true,
        auth: action.payload
      });
    case UNAUTHENTICATED:
      return Object.assign({}, state, {
        authenticated: false,
        auth: null
      });
    case AUTHENTICATION_ERROR:
      return Object.assign({}, state, {
        authenticated: false,
        error: action.payload
      });
    case SIGNEDUP:
      return Object.assign({}, state, {
        signedup: true,
      });
    case SIGNUP_ERROR:
      return Object.assign({}, state, {
        signedup: false,
        error: 'Not able to sign up new user.'
      });
    default:
      return state;
  }
}
```

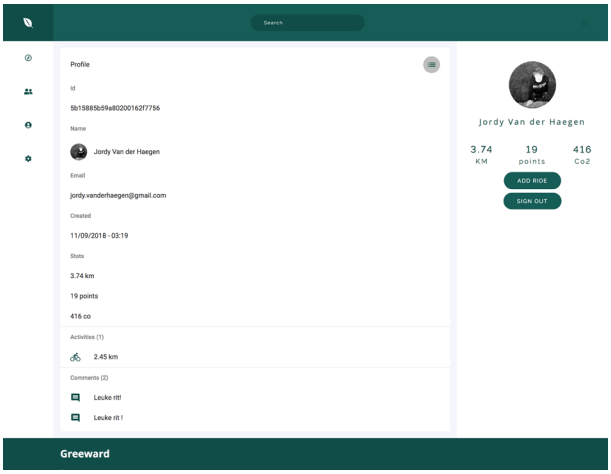
Design

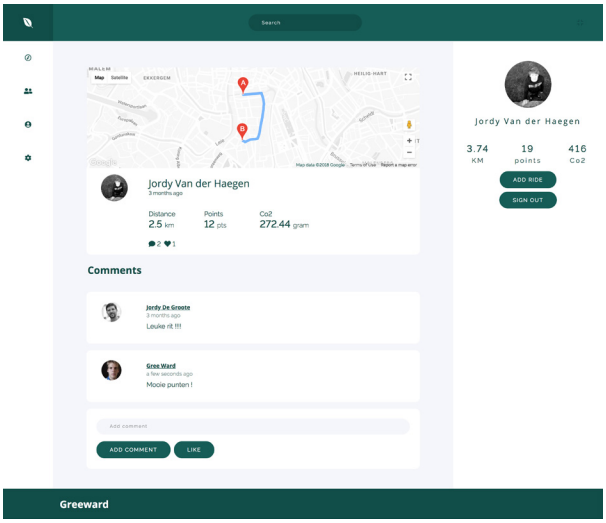
The design of Greeward is all based around activities, that's why the layout of the website primarily focuses on the section with the activity-feed. The green colours are there to remind the user about being an environment-friendly app.

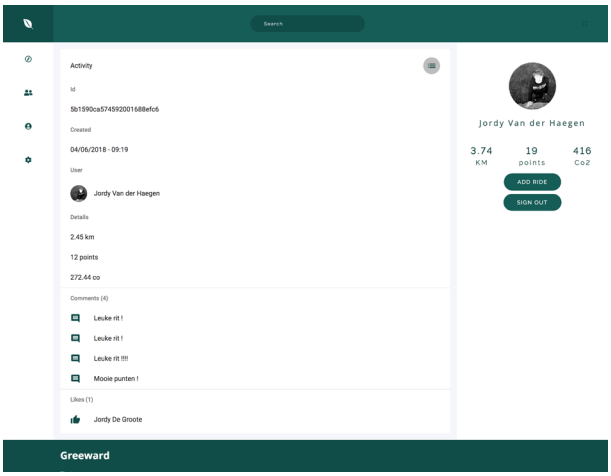


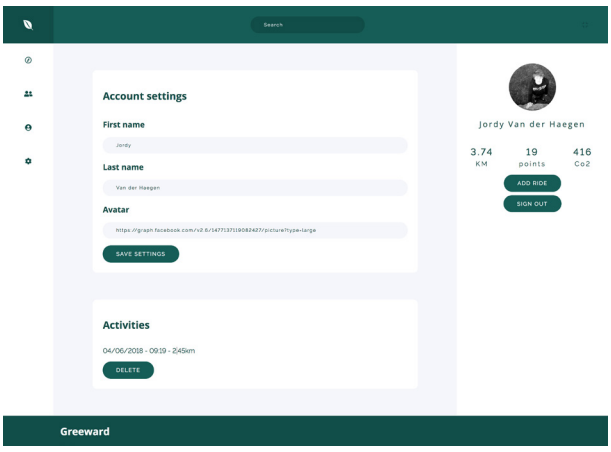

Search
Gree Ward
a minute ago
Distance 0.9 km
Points 4 pts
Co2 96.89 gram
Jordy De Groote
3 months ago
Distance 1.1 km
Points 6 pts
Co2 123.33 gram
Jordy Van der Haegen
3 months ago
Distance 2.5 km
Points 12 pts
Co2 272.44 gram
test test
3 months ago
Distance 1.2 km
Points 6 pts
Co2 130.33 gram
Jordy Van der Haegen
3.74 KM
19 points
416 Co2
ADD RIDE
SIGN OUT
Greeward


Search
User
Body
Created
Actions
Gree Ward
Mooie punten !
11/09/2018
Jordy Van der Haegen
Leuke rit!
05/06/2018
Jordy De Groote
Leuke rit !!!
05/06/2018
Jordy De Groote
Leuke rit !
05/06/2018
Jordy Van der Haegen
Leuke rit !
04/06/2018
Jordy Van der Haegen
3.74 KM
19 points
416 Co2
ADD RIDE
SIGN OUT
Greeward


Search
Profile
Id
5b15885b59a80200162f7756
Name
Jordy Van der Haegen
Email
jordy.vanderhaegen@gmail.com
Created
11/09/2018 - 03:19
Stats
3.74 km
19 points
416 co
Activities (1)
2.45 km
Comments (2)
Leuke rit!
Leuke rit !
Jordy Van der Haegen
3.74 KM
19 points
416 Co2
ADD RIDE
SIGN OUT
Greeward


Search
Jordy Van der Haegen
3 months ago
Distance 2.5 km
Points 12 pts
Co2 272.44 gram
Comments
Jordy De Groote
3 months ago
Leuke rit !!!
Gree Ward
a few seconds ago
Mooie punten !
Add comment
ADD COMMENT
LIKE
Jordy Van der Haegen
3.74 KM
19 points
416 Co2
ADD RIDE
SIGN OUT
Greeward


Search
Activity
Id
5b1590ca57459200168efc6
Created
04/06/2018 - 09:19
User
Jordy Van der Haegen
Details
2.45 km
12 points
272.44 co
Comments (4)
Leuke rit !
Leuke rit !
Leuke rit !!!!
Mooie punten !
Likes (1)
Jordy De Groote
Jordy Van der Haegen
3.74 KM
19 points
416 Co2
ADD RIDE
SIGN OUT
Greeward


Search
Account settings
First name
Jordy
Last name
Van der Haegen
Avatar
https://graph.facebook.com/v2.6/1477137119082427/picture?type=large
SAVE SETTINGS
Activities
04/06/2018 - 09:19 - 2.45km
DELETE
Jordy Van der Haegen
3.74 KM
19 points
416 Co2
ADD RIDE
SIGN OUT
Greeward