Web, Mobile and Security

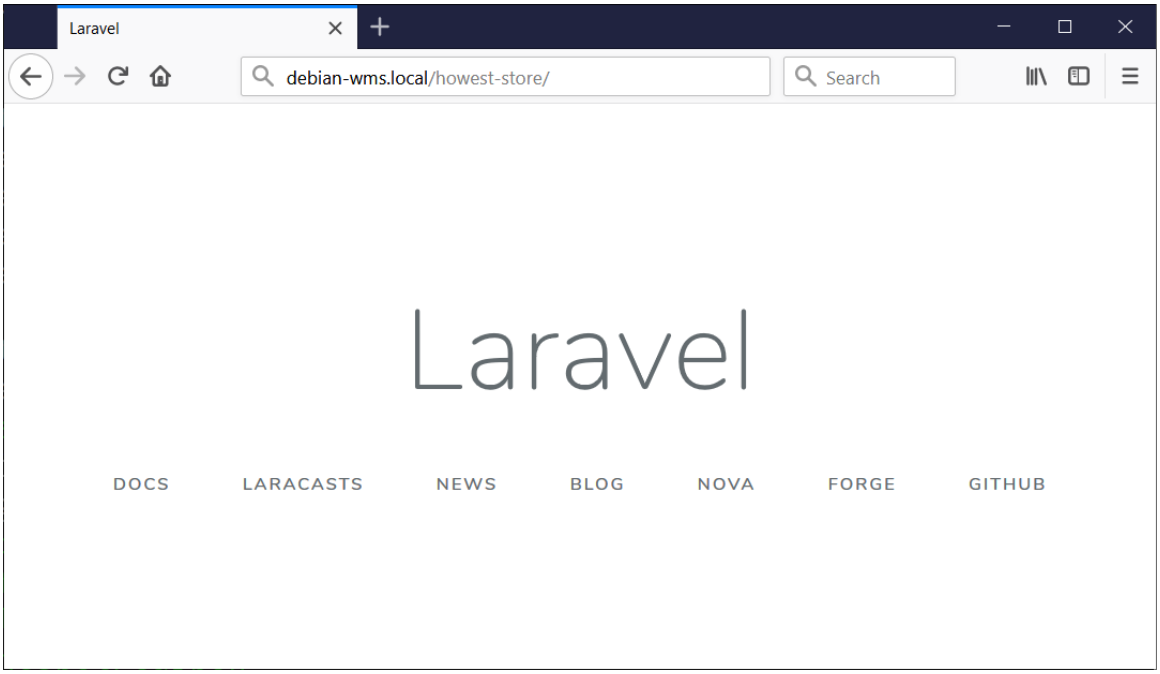
Lab: Howest Air

1. Creating the project

- Create a new Laravel project called `howest-air` by executing the following command from within your PuTTY:

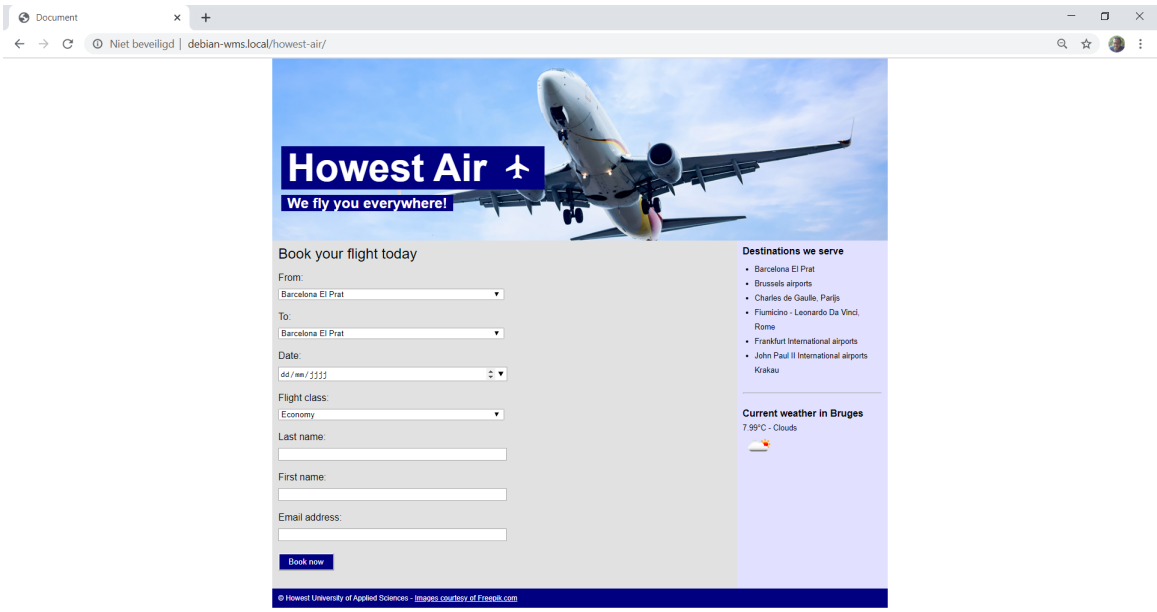
```
./build-laravel-project howest-air
```

- Try surfing to http://debian-wms.local/howest-air and make sure you get the default Laravel start page:

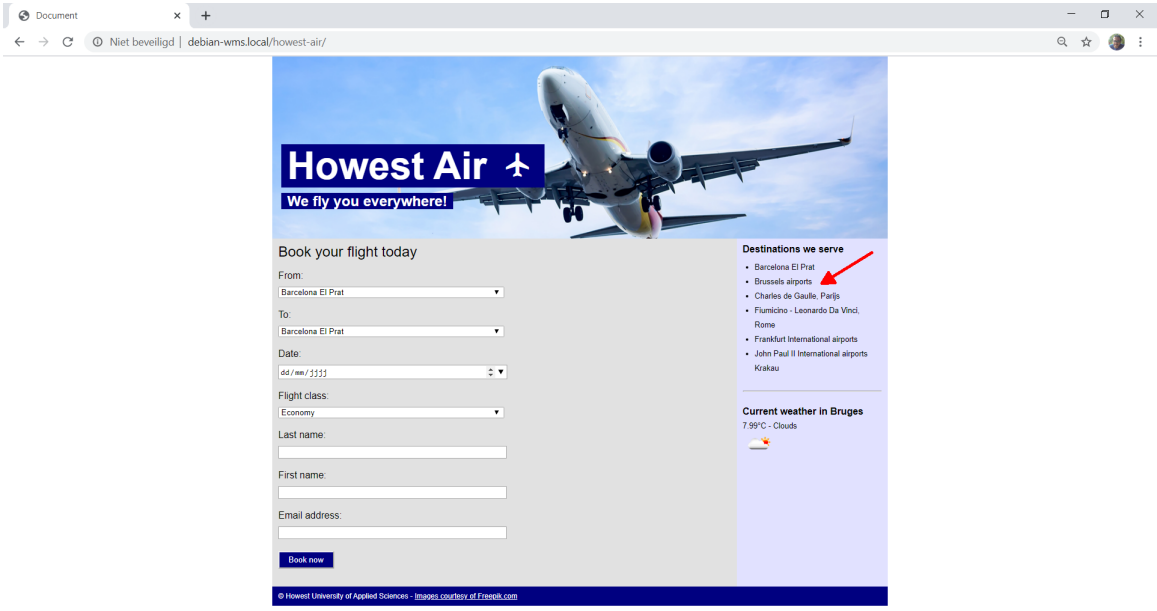


2. General overview of the application

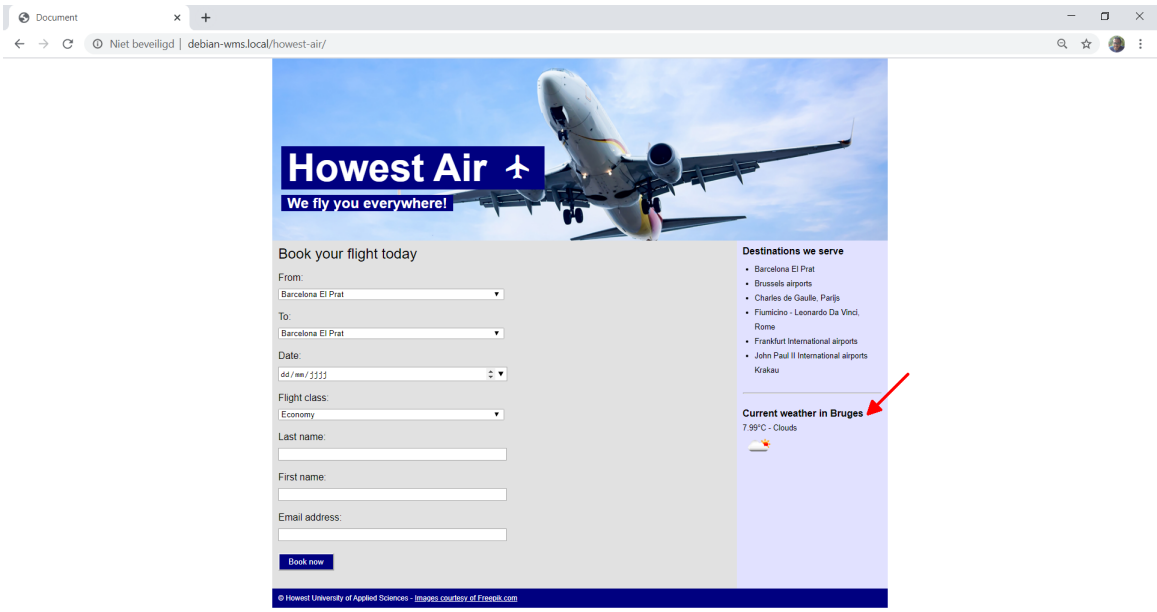
- We are creating the brand new e-commerce site for the *Howest Air* airline, allowing visitors to book flight tickets.
- When opening the website, the user is presented with a booking form:



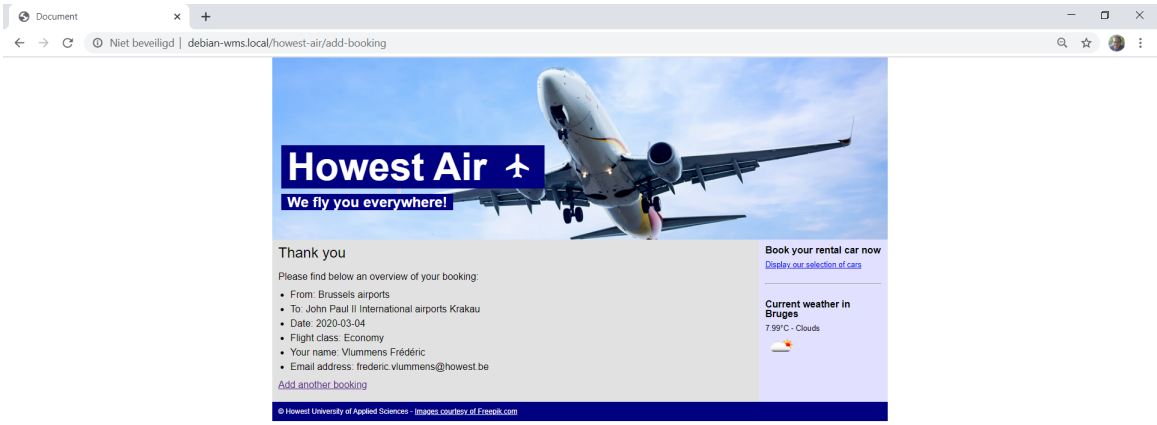
- The form has the following fields:
    - From: a dropdown list of airports
    - To: a dropdown list of airports
    - Date: a date field
    - Flight class: a dropdown list containing the different classes
    - Last name: a text field
    - First name: a text field
    - Email address: an email field
- All information regarding the airports and flight classes is retrieved dynamically from the database (see later).
- As you can see, the destinations (=different airports) are also rendered as an unordered list at the right side of the page:



- Last but not least, the OpenWeatherMap API is consulted to display the weather situation in Bruges (condition + temperature + icon):



- After submission, a thank you message is displayed. Also note that although the weather situation remains visible, the list of airports is replaced by a link to a (non-existing) page containing an overview of rental cars:



- The booking itself is stored in the database

3. Creating the actual application

3.1. Importing the database

- We provide the database for you. However, you will need to import it in your MySQL database server.
- To do so, launch SQLyog, connect to `debian-wms.local` using user `user` and password `user`.
- Next, retrieve the script `howest-air.sql` from Leho and execute its contents.
- This will create the database, as well as the various airport definitions and flight classes.

- Take a moment to study the structure and contents of the various tables (primary keys and other columns, ...).

3.2. Configuring the database connection

- In the `.env` file (found in the root of your Laravel project), add the correct credentials:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=howest-air
DB_USERNAME=user
DB_PASSWORD=user
```

- This will tell Eloquent that:
    - the database is of type mysql;
    - the database server is running on the same machine as the webserver;
    - the server is listening on port 3306;
    - the database name is `howest-air`;
    - the database user is `user` and password idem.

3.3. Building the model

- Generate an Eloquent model called `Airport` by using the command

```
php artisan make:model Airport
```

- We will also be needing a model for `Flightclass`.
- Finally, each booking is to be stored in the database. Add a model `Booking`.

3.4. Building the master layout page

Create a blade "*master page*" called `master.blade.php`, which will function as template for the various views.

- Base yourself on the screenshots above. Which part(s) will be different per view? Isolate them using `@yield` instructions.
- Your CSS files (`reset.css` and `screen.css`) should be placed in the subdirectory `/public/css` of your application.
- To reference your CSS file(s), do not forget to use the Blade function `asset`.

3.5. Creating the Booking Form

- The first view is activated when navigating to the route `/`.

- It displays the form, filling in the available airports in From and To based on the corresponding records in the database table `airports`.
- The flight classes are also filled in, based on the corresponding records in the database table `flightclasses`.
- **Important:**
    - The airports and flight classes are rendered *dynamically* based on what the models return. Should we add a new airport in the DB table `airports`, the page should automatically also display said airport.
    - Therefore, you will not include static HTML in the view with airport names and flight classes, **everything** comes from the DB.

3.6. Processing the booking

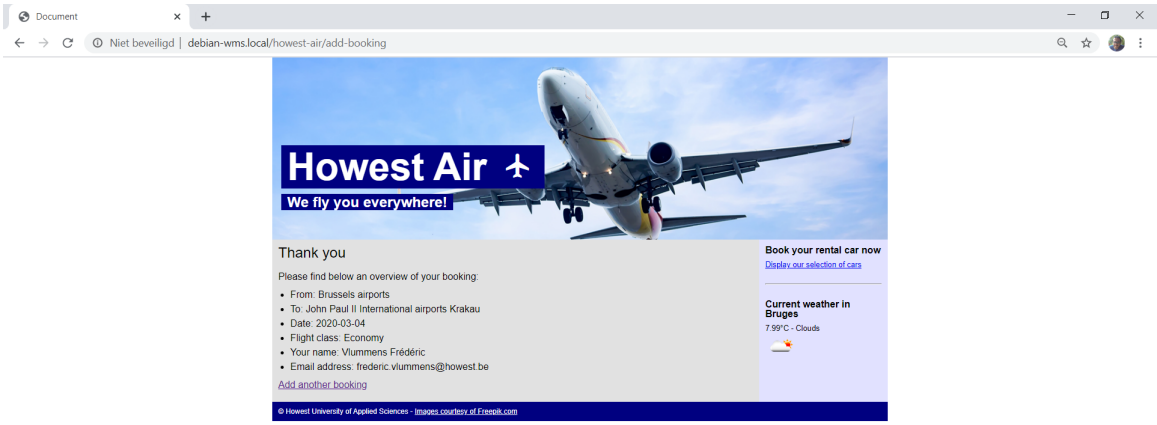
- Upon successful validation of user input, two things occur:
    1. a new booking is added to the database;
    2. a thank you message is displayed, summarizing the booking.

3.6.1. Adding the booking to the database

- Create a new instance of the model class `Booking`, passing in the submitted information.
- Next, call the model's `save()` method, which should effectively execute the SQL INSERT.
- Verify in the database table `bookings` that the record has effectively been added.

3.6.2. Showing the thank you page

- Finally, make sure a thank you page is also displayed, summarizing the booking:



- At the bottom of the screen, a link is provided to start the process over again.

3.7. Current weather in Bruges

- Make sure that on all pages of the website, the current weather of Bruges is displayed.
- Use a regular fetch to do so, **not** JSON-P.
- Place your JavaScript file(s) in the `public` folder and as with the CSS, use the `asset` function to reference them.