VRIJE
UNIVERSITEIT
BRUSSEL

MVC USING SCALA PLAY

Assignment 2 - Software Architectures

Lennert Bontinck

Second session, 2020-2021

Student number: 568702

Computer Science: AI

Contents

General remarks

1.1 Notes on the assignment

None of the assignments for the Software Architectures course were submitted in the first examination period due to personal reasons. Because of this, all of the code written for this assignment is written specifically for the second examination period. All code was written by using the course material and the online Akka Play documentation[1]. The solution of exercise 7 was used as a starter template for this project. A previous MEAN stack school project of mine based on Bootstrap, CarMeets (Bontinck, 2019), was used as concept and styling (HTML + CSS) inspiration for this project.

1.2 Important files

All code written is available on the GitHub repository for this assignment (Bontinck, 2021). Rights to this private GitHub repository can be granted upon request. A copy of this GitHub repository is accompanied by this report. An overview of important files is given below:

- `README.md`
  - General information of the GitHub repository as well as some technical details about the used environment and how to run the project. The general flow of the application for the end-user is also demonstrated with some screenshots. The pre-loaded account information is also supplied here.
- `assignment.pdf`, `Lennert-Bontinck-SA2.pdf` and `report/`
  - The assignment PDF, this report and the source files of the report. A VUB themed LaTeX template by De Smet (2020) was used to create this report.
- `code/Lennert-Bontinck-SA2/`
  - The folder containing the code of the assignment solution.
  - Instructions on how to run the code are available in the `README.md` file.

[1] https://www.playframework.com/documentation/2.8.x/Home

Project defence

In what follows a high-level discussion is held on the created code. Multiple figures and screenshots were made to illustrate the different parts of the project. Some of these figures are provided in the extra figures list at the end of this report. For more technical details the reader is invited to read the documentation and comments in the code. The `README.md` file contains a screenshot and discussion of every important part for the end-user.

2.1 Ideology of CarLovers

CarLovers is a social media platform where registered users can share automobile-related pictures with all other users or a select group of users. Other users can interact with the shared post by liking and/or commenting on it. The ideology and styling (HTML + CSS) of this application are inspired by and sometimes taken from the earlier discussed CarMeets application (Bontinck, 2019) where users can share and plan car meetings.

2.2 Protecting pages from unwanted users

As with any project, security is an important aspect. With web-based applications, the most challenging part is often limiting what request which users can and can't execute. The general strategy of handling page requests is shown in figure 2.1. Due to the use of an MVC pattern, requests sent by the user are processed by a controller. With Akka Play the controller provides `Actions` of a certain type to process these requests. The custom made classes `AuthenticatedUserAction` and `AuthenticatedUserActionWithMessageRequest` are responsible for blocking any request that doesn't have a cookie containing a username. Such actions can thus be used to limit requests to signed-in users, as is done for most of the pages. The denial page that is shown by these `Actions` upon a blocked request is the login required page shown in figure 2.2. Sometimes a request is only allowed for a specific user, to process this, the code provided inside the Action should handle the blocking on a per Action basis.

From this login required page, it becomes apparent the user can indeed register an account and log in to his/her account. Since no database had to be used, the users, and all other objects, are just stored in memory and plain text. It is noted that this approach may not be used in the real world since such data must be hashed and stored according to various regulations. Simply having a username cookie is also a very weak account validation mechanism. The login and register forms are made with Akka Play's form methods and provide various types of validation and error handling. Figure 2.3 shows the login and register page with some of those validation and error handling features present. Once the user is logged in, a different menu is shown where the user can log out, add a post or view his/her profile.

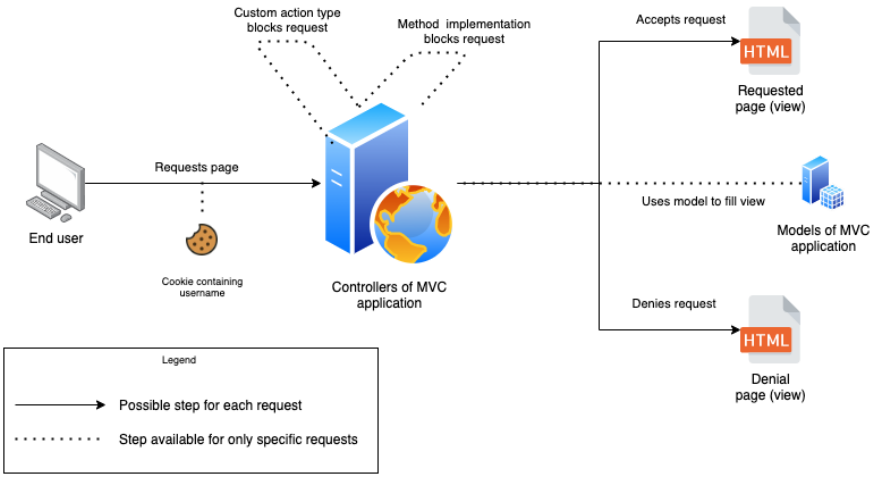


Figure 2.1: Simplified representation of page request procedure.

2.3 Post overview, single post and profile pages

The main pages of the project revolve around viewing posts from users. Since many of these pages share the same components, these shared components are made available as sub-views. An obvious such sub-view is `main`, which provides the header and navigation bar as well as the footer. As discussed, a different navigation bar (menu), is shown for logged in users. Less obvious sub-views include the like button (`postLikeButtonAndCount.scala`), a *post card* which shows a singular post for an overview page (`postListItem.scala`), comments (`commentListItem.scala`) and more. This reuse of sub-views is very visible with the overview and the profile page shown in figure 2.4. The same sub-view for showing a *post card* can be used but the view is just supplied with a different list of `PostWithInfo` objects, e.g. from a specific user or ordered differently. Such a `PostWithInfo` object contains a `Post` and `Visibility` object as well as a list of `Likes` and `Comments`. The MVC pattern is present: after a user request, the controller sets up the model so that the view can display information to the end-user. The controller is thus responsible for reasoning about which `PostWithInfo` objects should be visible to the user, based on the information from the `Visibility` object. Figure 2.7 shows that the homepage for two users can be different based on the `Visibility` information.

Figure 2.4a shows that a user can like a post from the overview page and sees at most three comments per post on the overview page. If the post in question is placed by the logged-in user, the delete and edit buttons are also present. The user is forwarded to the single page of a post when clicking read more, the image, or the like button. As shown in figure 2.5, all comments can be read on this single post page and a comment can be placed by the user.

2.4 Managing and visibility of posts

The last important part of the puzzle for this project is managing posts. Compared to previous forms where Akka Play neatly created instances of objects upon form bind, uploading a file and working with lists provides some manual work. Luckily the Akka Play documentation pointed in the right direction[1]. The add post form is shown in figure 2.6a and a correct `PostWithInfo` object is made in two different parts. Firstly the standard form approach is used to create a `Post` object and validate the description field by using hidden fields in a pre-filled form for the system-generated values. For the image and visibility settings, the form body is manually checked for correctness and error messages are shown manually through view settings. The code in the `PostController` is well documented and explains this procedure in greater detail.

After submitting the form successfully, the user is forwarded to the just created post. The edit visibility page is shown in figure 2.6b and demonstrates that also this form is pre-filled with the settings chosen on post creation. Deleting the post simply takes the user to his profile page after a successful deletion with a flash message notifying him of the successful deletion.

2.5 Other caveats

This assignment has proven to be rather lengthy and explaining each portion in great detail is simply not possible in a document that is already over the page limit. The `README` file provides screenshots for each important step and discusses briefly what is important in each step. As always, the code is very well documented and the reader is invited to take a look at specific methods to have a better understanding of how certain parts of the code work.

[1] https://www.playframework.com/documentation/2.8.x/ScalaFileUpload

Extra figures

To make the report more readable some figures are not provided directly in the text. These figures are provided here.

Login required page

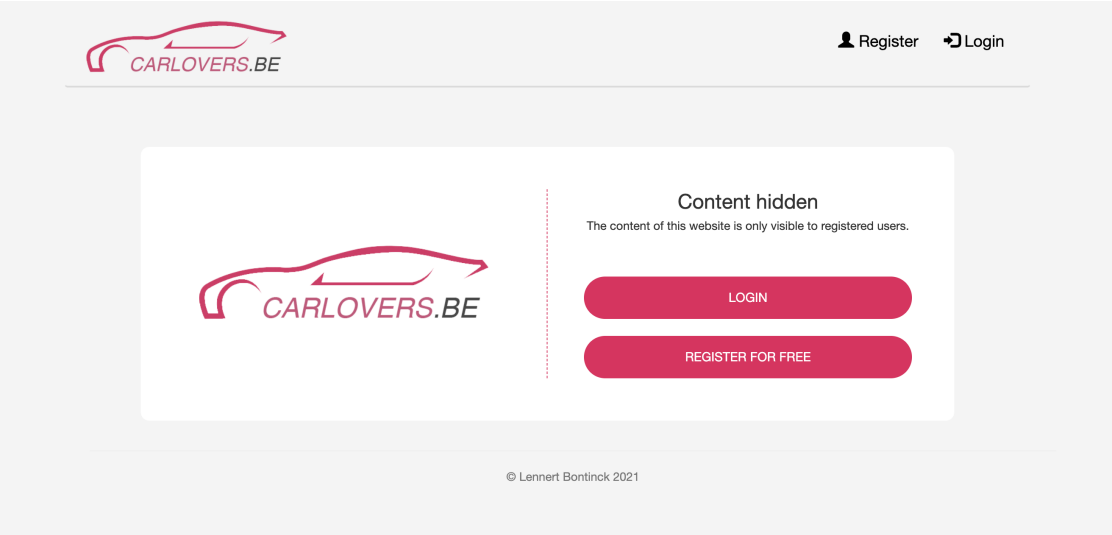
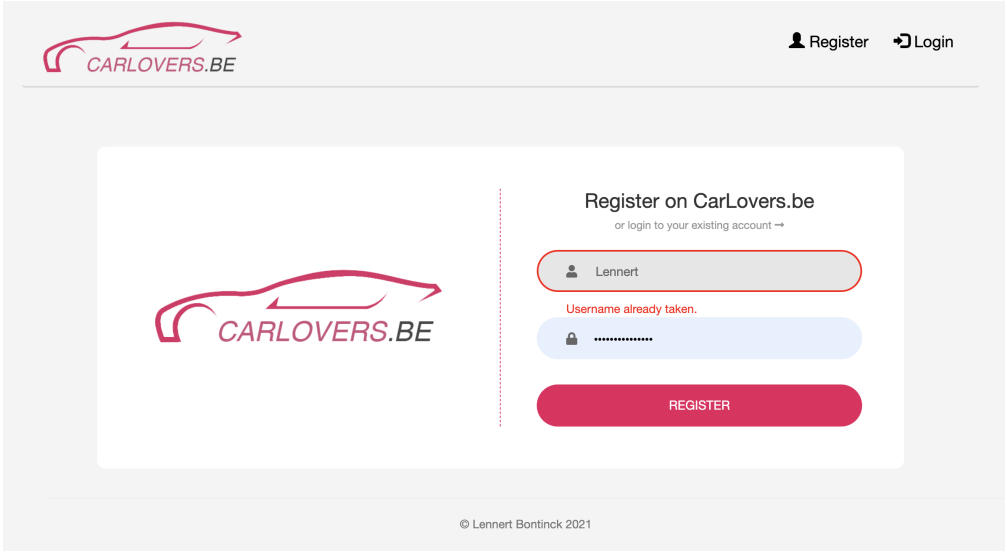
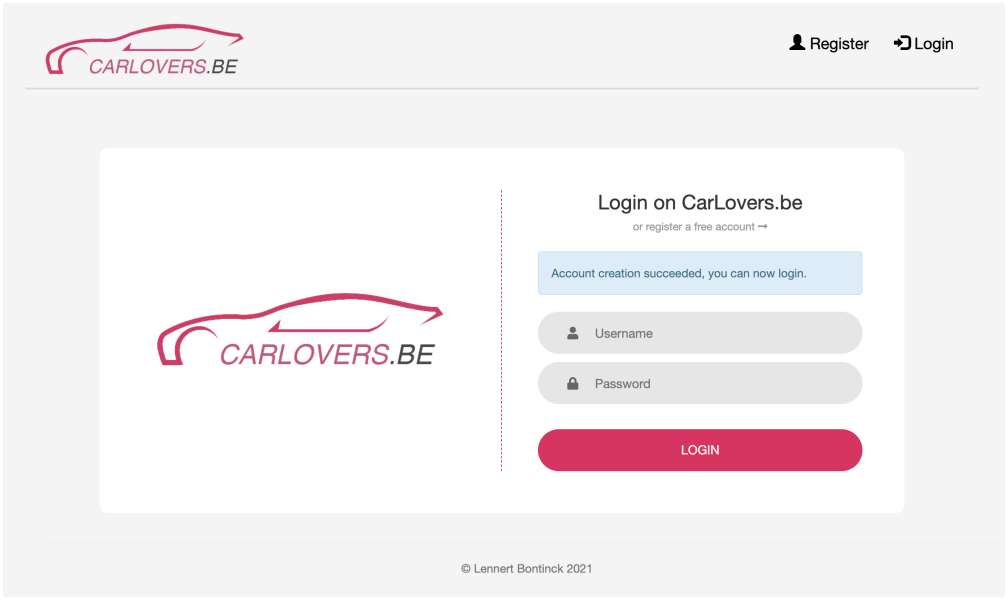


Figure 2.2: Page notifying the user an account is required.

Register and login page



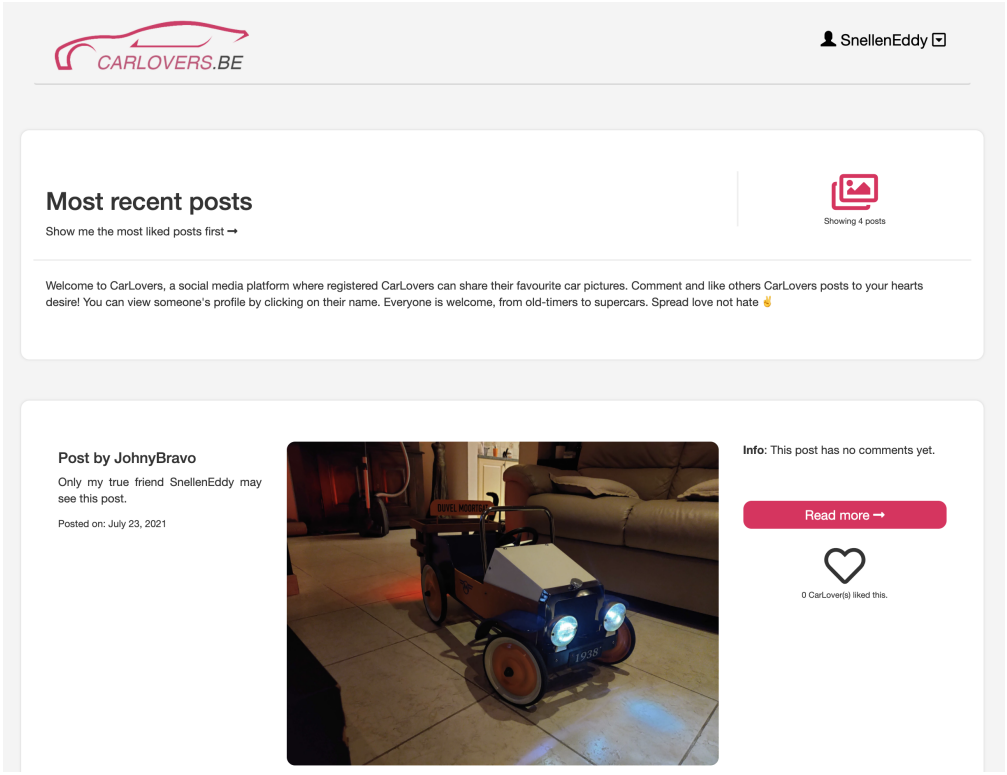
(a) Register page with validation that username is already in use.



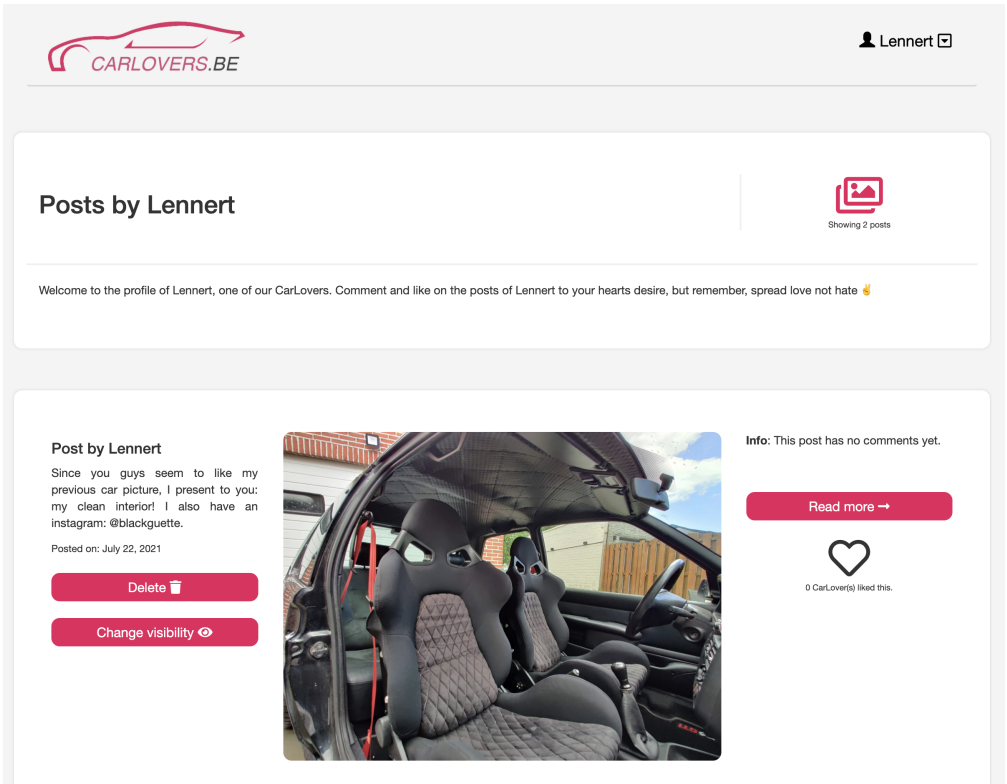
(b) Login page with flash message.

Figure 2.3: Login and register page of CarLovers website.

Overview and profile page



(a) The main overview page which may be sorted on date or likes.



(b) The profile page of a specific user.

Figure 2.4: Overview and profile page of Carlovers that share sub-views.

Single post page

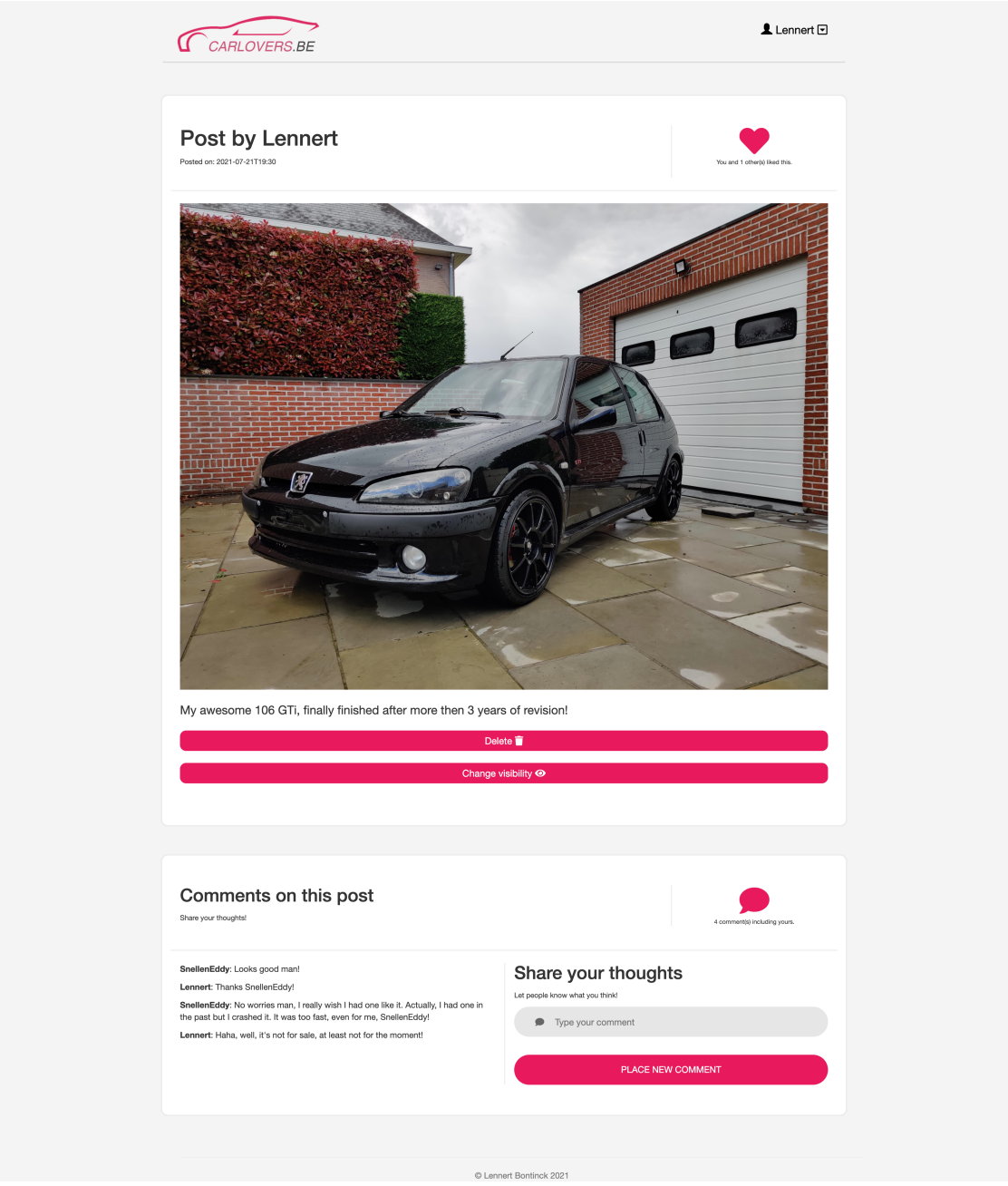
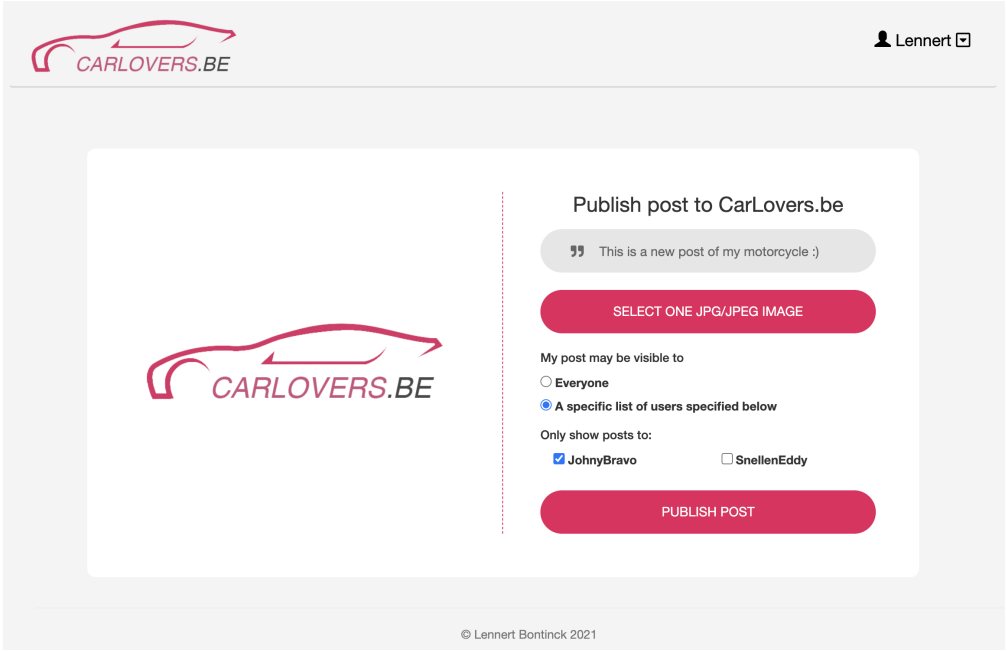
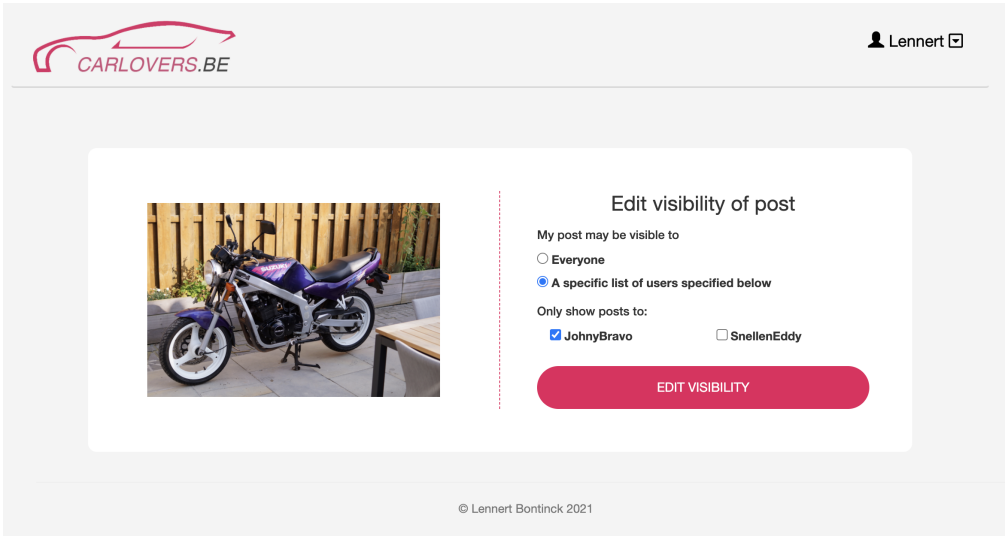


Figure 2.5: The detail page of single post.

Add post end edit visibility



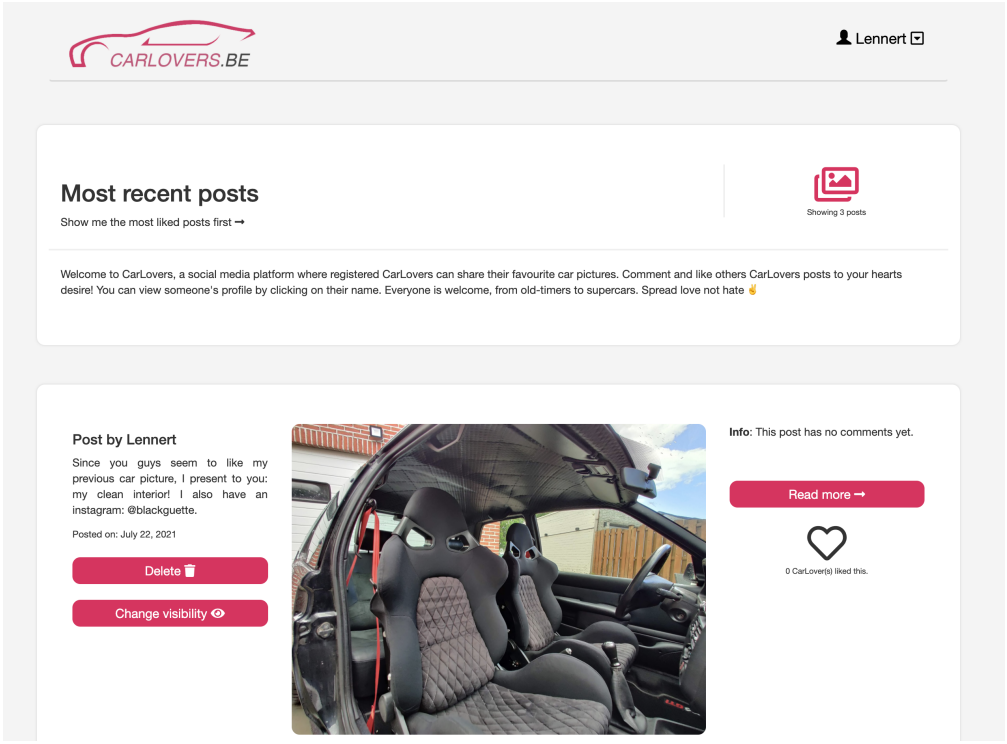
(a) The add post page.



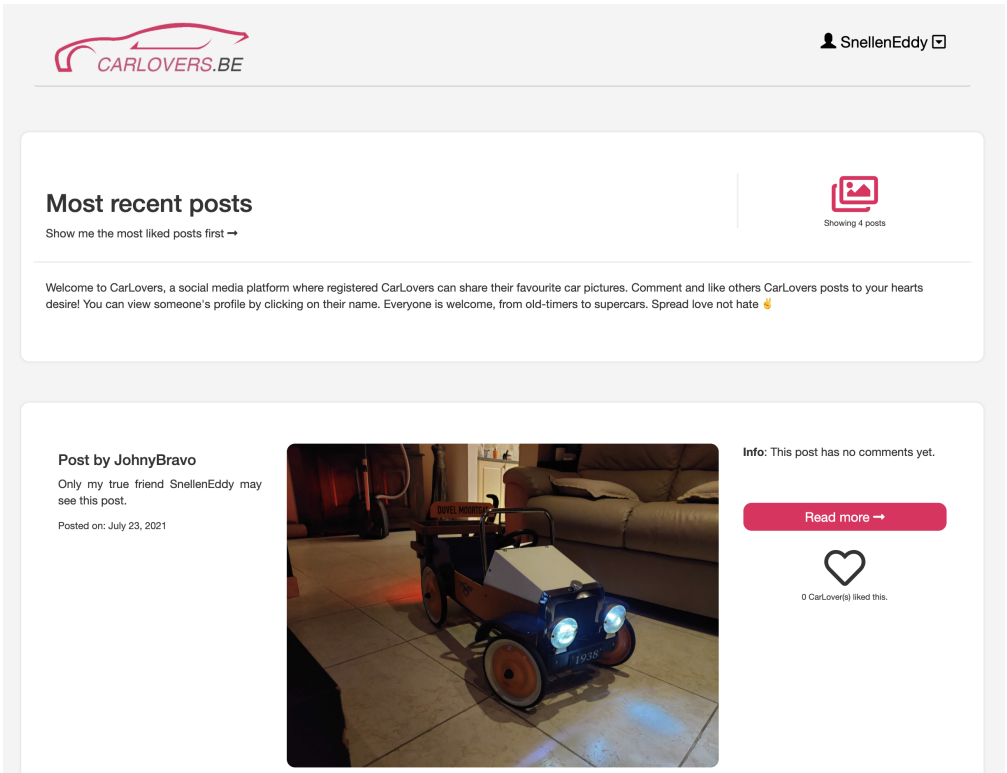
(b) The edit visibility page.

Figure 2.6: The add post page end edit visibility page for managing posts.

Limited visibility



(a) Homepage of user who can't see JohnyBravo's post.



(b) Homepage of user who can see JohnyBravo's post.

Figure 2.7: Different posts are shown on the home page for different users according to the visibility settings.

References

Bontinck, L. (2019). *Carmeets: Project web iv 2017-2018 bontinck lennert 2tina* [GitHub commit: a6f960b...]. Retrieved July 14, 2021, from https://github.com/pikawika/carmeets

Bontinck, L. (2021). *Assignment 2 of software architecture course* [GitHub commit: 4afd31e]. Retrieved August 5, 2021, from https://github.com/pikawika/VUB-SA-assignment-2

De Smet, R. (2020). *Vub latex huisstijl* [GitHub commit: d91f55...]. Retrieved November 2, 2020, from https://gitlab.com/rubdos/texlive-vub