ABSCHLUSS-ARBEIT M133

"Webapplikationen mit Session-Handling realisieren"

INA3a, Viscardi Simone

Documentation

Analysis

The toy rental company *SpielGut* has given us the Task to implement a web application for giving their customers the ability to autonomously register themselves to rent toys and manage their account. The web application **will not** handle the management of the toys themselves, as this is already implemented in a separate application.

Requirements

Functional Requirements

- Users must be able to:
  - …register themselves.
  - …log into their accounts.
  - …view and edit their personal details.
  - …make a rental for a toy.
  - …view all their past and active rentals.
  - …extend their active rentals.
  - …reset their passwords.
- Additionally when logging in for the first time, the user must provide personal details about himself.

Nonfunctional Requirements

- The toys and rentals shall be saved server side on the file system.
- Every toy shall only be available once, therefore it can only be rented once at any given time.
- Every rental shall only consist of one toy each.
- Rental of multiple toys by the same user at the same time is possible, but the user has to split them up over multiple rentals of single items.

Non-Defined Requirements

These are some requirements which have not been defined by the received documentation of the requirements. So these are decisions made by us.

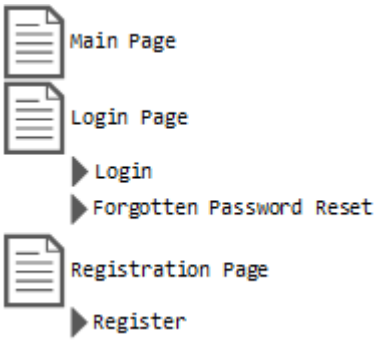
- Rentals can be extended an unlimited number of times, as long as the extension takes place during the active period of the rental.
- When renting a toy, the rental starts instantly and its start cannot be postponed.
- The initial rental does not have a limit of time for which it can be rented.

Planning

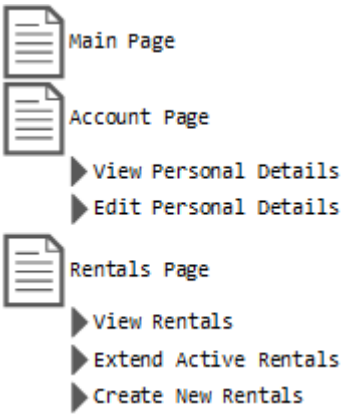
Page Layout

This is the initial planning of the applications page layout.

If the visitor is not yet logged in, he shall be presented with following page layout:



Once logged in, this is what the user can see:



Page Design & UX

In order to present the user with a solid UX experience and a consistently appearing UI, Google's *Material Design Lite* library shall be user as the library allows for an easy, quick and seamless integration and is very lightweight.

To ease the process of providing a consistent design and flexible interface, the design will be based on *material* stylized cards. This means the page's content is presented on cards of adaptive sizes, which in turn makes it easy to append content later without breaking page flow or layout.

Test Criteria

Login & Registration

- As long as the page visitor is not authenticated, the *Login* and *Register* menu entries shall be displayed and the menu entries for logged in users (like *Account*) shall not.
- When the visitor tries to registrate or login and does not provide correct information (like a malformed e-mail address or an incorrect combination of username and password) or an error occurred in the processing of given information he shall be notified with an explanation to what went wrong and/or how to fix the issue, if possible.

Rentals

- Once a toy is in rental it shall no longer be available for rental by other users.
- A toy which is not actively in rental shall be selectable in the corresponding menu.
- At any point during the rental time, the toy can be extended, as long as the rental is still active.
- Once a rental is no longer active the option to extend the rental ceases to be available.

User Accounts & User Data

- Users can, at any time, view and edit their given personal details.
- If a user does no longer remember his password and is not able to log in, he can reset his password using his e-mail address.

Concept

Functionality

When registering as a new customer, the user must fill in his personal details, including, but not limited to, his full name, his address and his phone number.

After a successful registration and/or login, the user will be able to navigate to his personal profile page, where the user has the ability to view or edit his personal details.

The user may also choose to navigate to his *Rentals* page where he is able to view his past rentals and view or extend the currently active ones as well as choose a toy and duration for a new rental.

Every toy is only found once in the library and can therefore only be rented once at the same time.

Realization Concept

Time Table

This is the initial rough time table for the planning as well as the realization.

Date	Plans
06.12.2016	Start of the project, Received the requirements
09.12.2016	Finished analysis parts of documentation and rough estimate of time table
13.12.2016	Mostly finished concepts
18.12.2016	Done with implementation and realization chapters of documentation
20.12.2016	Project deadline (at 1700)
08.01.2017	Finished preparing the presentation
10.01.2017	Held the presentation

Data Structure

All data except for the user data will be saved in the *XML* format as files on the file system of the server running the application. All user data will be saved in an *SQL* database.

There will be the need to save three types of data: user data, rentals and toys. The user data, as mentioned before, shall be saved in an *SQL* database instead of the file system, this is due to the ability to use the built in types in the *ASP.NET* framework, which provides this functionality without much effort by default. The other two types of data which have to be stored, shall be stored in *XML* formatted files in directories relative to the application's root directory.

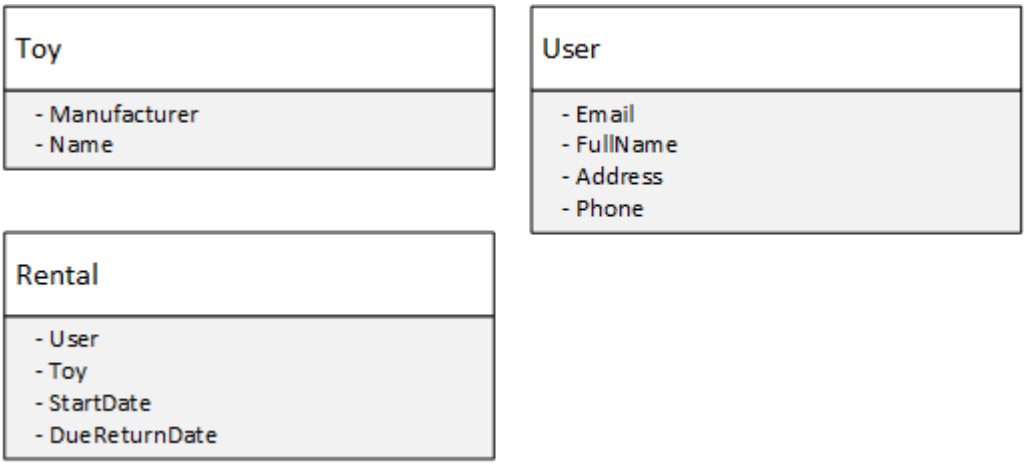
User Data

In order to create a new account, a valid e-mail address is required, which has not already been used to sign up. This address functions as a way of identifying the user as well as sending him important messages outside of our systems, like password reset and confirmation emails.

The password rules are set to allow any combination of characters as long as the minimum amount of 16 characters is met. No requirements for a minimum set of numbers, special characters or the like are there, since for user safety policies like these often make no sense. This is because a longer password is more secure than a shorter, more sophisticated one and enforcing policies for complex passwords usually encourages the user to just add required characters to the beginning or the end of their passwords and reuse the same passwords on multiple web sites.

Objects

The application implementation will use an *Object Oriented* approach to handle data, therefore there will be the need for multiple objects. The following are the objects that will probably be necessary for the implementation. There might be a need to add additional ones later down the developments process.


Toy
- Manufacturer
- Name
User
- Email
- FullName
- Address
- Phone
Rental
- User
- Toy
- StartDate
- DueReturnDate

Realization

Beginning

I decided to begin with the frontend of the application because in the past I've always begun with the backend which led to me spending a lot of time on creating a solid backend with an excessive amount of flexibility, expandability and features of which the frontend ended up only using very few.

Implementing the page layout

So I started with getting rid of the Bootstrap styling libraries, which are included in the *ASP.NET* framework by default and adding the *Material Design Lite* libraries.

The next step was to customize the menu bar and implement the rough base layout of the pages. This was quickly done and I could continue to add my own pages.

In order to make it easier to access relevant data like toys, rentals, etc. I chose to implement a class *ManagablePage* which handled the initialization and retrieval of data and makes sure that only authenticated users were allowed to access the page. All new pages could inherit this class to get easy access to this functionality.

I then added the *Details*, *Rentals* and *Account* pages each inheriting the before mentioned *ManagablePage* class. If a user tries to access any of those pages without having saved his personal details, he will be redirected to the *Details* page where he has to do so, before he can continue. Note that this does not apply to the *Default* page, as this page serves as the home page and is always accessible, whether or not the visitor is logged in or has provided his details yet.

The Backend

Although the pages did not have any content in them yet, I decided to start working on the backend. The business logic is located in the *Domain* directory where one can find classes to abstract real world objects like a toy under *Entity*, data managing classes under *Manager* and classes helping streamline the development under *Interface*.

The *Rental* and *Toy* entities are rather straight forward, they are simple classes implementing *IManagable* and contain a few properties. The users weren't as simple though, the user data properties had to be added to the *ApplicationUser* class under the *IdentityModels*. After changing the *ApplicationUser* class I ran into trouble since the database was still laid out for the old data structure, but thanks to this *StackOverflow* answer* by *Lin* I was able to resolve the issue.

* https://stackoverflow.com/a/20304312/2961178

The *Manager* and *Database* classes are separate so the database everything runs on can be swapped easily and without breaking a lot of code.

The *XmlDatabase<TManagable>* class is a generic class so it can handle any type of data but each instance of the class is specialized to handle one class and handle it

well. It is responsible for serializing and deserializing data and to write changes to the file system or read data from the file system. I initially had some trouble with serializing my generic classes using the *.NET XmlSerializer* class, but found some good advice in *Yan Cuis* blog post*.

* http://theburningmonk.com/2010/05/net-tips-xml-serialize-or-deserialize-dictionary-in-csharp/

The *Managers* are very simple classes which handle a set of their respective data entities. They handle things like initializing data, searching the database and manipulating data.

The Frontend

After finishing the backend to a point, where minimal required functionality was introduced, I continued to implement details in the frontend. Whenever I added functionality to the frontend which the backend could not yet accomplish, I either went to the backend to implement the functionality on the spot or made a fake implementation which did not yet work as intended but was enough to let me continue to work on the frontend and implement the functionality later.

Design

The Frontend design was mostly handled by the *Material Design Lite* libraries so I did not have to do a lot of work there. The bit of tweaking necessary in the styling was done in the *Site.css* file.

Sources

Documentation

Page 1, first image: "Oxfordwebapps"(www.oxfordwebapps.co.uk/)

Application

Google Material Design Lite Library: "Material Design Lite" (www.getmdl.io)

Some text on the main page was taken from third party sites:

About Us Section: "Marks and Spencer" (corporate.marksandspencer.com/aboutus)

Opening Times: "Kunsthaus Zürich" (www.kunsthaus.ch/en/information/)