Unplagged Developers Manual

Building the Plagiarism Detection Cockpit

Term paper for the master project I
Mentoring Teacher: Prof. Dr. Debora Weber-Wulff

Department Economics II
HTW Berlin – University of Applied Sciences

Elsa Mahari (s0534556) <Elsa.Mahari@gmx.de>
Tien Nguyen (s0512510) <s0512510@htw-berlin.de>
Dominik Horb (s0534217) <dominik.horb@googlemail.com>
Benjamin Oertel (s0522720) <contact@benjaminoertel.com>
Heiko Stammel (s0534218) <heiko.stammel@googlemail.com>

Contents

Introduction

After Minister Guttenberg had to resign, because of the plagiarisms found in his doctoral thesis, the big media coverage and interest in plagiarism in Germany has very much subsided (**?**). However, the initial idea for the creation of the "Unplagged" project, whose development approach will be described here, can be found in this very case of plagiarism. Related to it were the formation of the GuttenPlag and it's descendent VroniPlag. Both are Wiki-based communities that are collaboratively discovering and collecting plagiarism in their respective cases and are kind of the role models for the way the Unplagged system is developed.

The project idea and context were provided by our professor Dr. Debora Weber-Wulff and the two-term master project, every media informatics student at the HTW-Berlin has to take. Professor Weber-Wulff is a well known expert in Germany on the topic of plagiarism. As she has also done research in this field for over ten years and is actively involved in the VroniPlag community under her synonym "WiseWoman"(**?**), she came up with the idea to build a dedicated system — a "Plagiarism Detection Cockpit"(**?**) — that is modeled after the experiences that were made with the workflow used in VroniPlag and GuttenPlag.

So, to put it in a catchy marketing phrase, here is what Unplagged aims to become:

> **Unplagged is a simple, web-based, collaborative system to help discover, collect and document plagiarism in scientific papers.**

To make things a bit more conceivable, we also often refer to it as a mixture of a very specialised text editor, with a focus on comparing texts and marking passages and a modern project management tool like Redmine or JIRA, to manage the collaborative

aspects of the system. The big distinction we make to other plagiarism software on the market is, that the approach is not to autodetect plagiarism, but focused on aiding the workflow of the users while searching for plagiarized fragments inside a scientific paper, a homework assignment or any other kind of probable textual plagiarism.

This present document will be the handbook that gets you started if you are interested in helping us with the development of this open source project, which is licensed under the GNU GPLv3.

Chapter Overview

One of the biggest problems we faced at the start was, that none of the team members had written a longer scientific text than a bachelors thesis and therefore the experience we got with actual scientific writing was very limited and very specific to the field of computer science. We understand the ethical problems, that come with the betrayal of good scientific practice of plagiators, but we simply can not relate easily to the amount of work that has to be put into a PhD., or be as passionate about plagiarism as Prof. Weber-Wulff always is, because we never experienced it ourselves.

That is why we had a lot of catching up to do on the most important history behind VroniPlag, the different types of plagiarism, different citation styles and the research Prof. Weber-Wulff and others had already done on systems that try to help finding plagiarism. Chapter 1, A Plagiarism Primer, will give a brief overview of the most important topics to get you up to speed with the domain of the software, if you are not already familiar with it.

Chapter 2, Project Workflow and Requirements, will be the place, where the development process is described and a collection and description of the parts of the system, that already exist or that we identified as necessary parts of Unplagged will be given. As the system is developed with an agile project management style, this will be done primarily based on the user stories.

If you know all those things already and simply want to get started working and coding,

you should probably jump to Developing Unplagged. This chapter will give the technical insights into the system, the basic installation steps and all necessary tools for you as a developer.

Conventions

To markup important words in the text, the following typgraphical conventions are used:

Italic
: First used technical terms

`Constant Width`
: Programm code, file names, paths

`Bold Constant Width`
: Variables that have to be changed by the user

1. A Plagiarism Primer

1.1. Basic Classification of Plagiarisms

1.1.1. Copy&paste

1.1.2. Copy, shake&paste

1.1.3. Patchwriting (rewording)

1.1.4. Structural plagiarism

1.1.5. Translations

1.2. How to detect plagiarism

1.2.1. Software systems

1.2.2. Human approach

1.3. Vroni Plag

2. Project Workflow and Requirements

First of all, we've got a confession to make: Unplagged is like a big playground of new workflows and technologies for us, as we are aiming to incorporate "best-practices" wherever possible, or at least what we currently consider to be best-practices.

We believe this approach is necessary, because of the fact, that we are essentially trying to incubate Unplagged as a real open source project and this will only work if it is well crafted and if cutting-edge workflows and technologies are used. Nearly all of the team members are also working in some kind of web related side job, so we all got enough experiences with the problems that can occur during the maintenance of badly designed software.

Most of the times this works pretty well, but sometimes we are still trying to figure out how to get everyone up to speed with every technology and part of the system or how to divide the responsibilites carefully.

To start this project, we opted to use *Scrum* as our agile development approach. If you are familiar with this methodology, you may notice, that there could be a few problems when considering, that the team is working mostly distributed without a common office and with very different time tables for each of the members.

We struggled a bit to the tweak the workflow that is required by Scrum in

2.1. The Workflow

To make it possible to work efficiently together in this kind of environment, we chose to use Redmine as our project management tool, which you can access under:

- http://tickets.unplagged.com

If you register there, an administrator should grant you access to the tickets and the wiki, so that you can participate in solving the problems at hand.

2.1.1. Team Meetings

2.1.2. "Debbie Meetings"

Figure 2.1.: Scrum Meeting

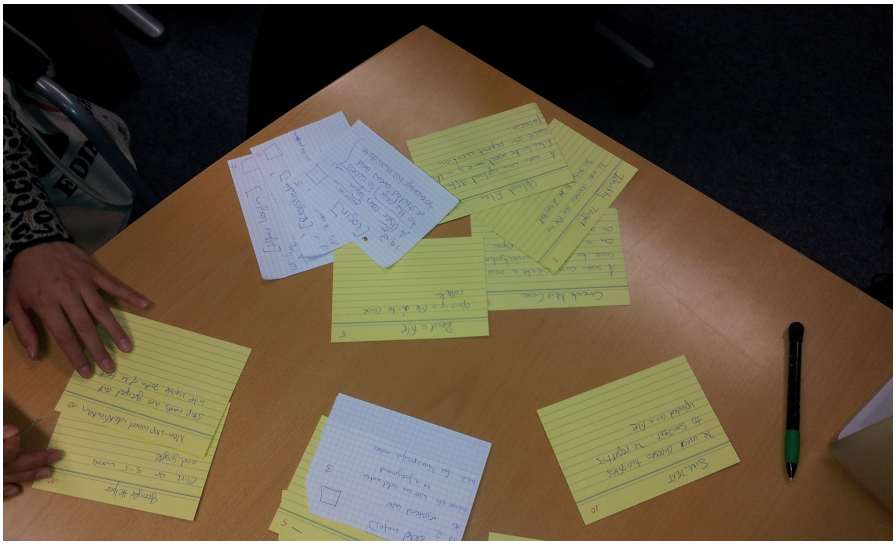

Figure 2.2.: User Stories

2.2. Target Group

2.3. User roles

As the Unplagged system will provide a permission based user system, our goal is to make it possible, to create custom user roles from an administration area and make it possible for users to have multiple user roles in one case and also different roles for different cases. The standard roles which will be provided by the system are:

Guest A user without a valid login can only see the parts of cases that are set to be public.

Registered Registered users can get "promoted" to higher roles and contribute to publicly editable cases.

Collaborator Collaborators are registered users who were granted access to a specific case. Collaborators can access and edit these projects.

Case-Manager Case-Managers can set up new cases and manage colloborators for their

cases and project versions. They may have the permission to add or remmove project members.

Admin An admin owns all permissions, such as user administration or project administration. They also hvae the ability to to block/unblock an existing case.

2.4. Basic functionalities

2.5. Document Parser

2.6. Detection Modes

2.7. Plugin Architecture

2.8. Use Cases

3. Developing Unplagged

Coming from the Project Workflow and Requirements here we have yet another set of requirements, before we can start with the actual description of the way you can help us develop the system. This time it's about what we believe will be helpful or sometimes even necessary prerequisites.

First of all, the programming languages mostly used in Unplagged are PHP and JavaScript, both of which in conjuction with a framework. Teaching programming languages is, as you probably can imagine, well beyond the scope of this document, but we will at least try to cover the most important concepts of the frameworks as they occur.

The used frameworks are jQuery for Javascript and ZEND for PHP respectively. jQuery is kind of the industry standard for unobtrusive scripting with about 50% of the Top 10.000 websites using it according to **?** and the Zend framework is also well established and brings a lot of features, that are useful to this project.

For most of the other topics, we will give you some (hopefully) helpful resources on the way, if it isn't covered thoroughly by us. But just to let you know, here is a list of the buzzwords, er technologies that will be mentioned:

- Scrum
- HTML5 and CSS3
- Continuous Integration
- Responsive Webdesign

- Progressive Enhancement
- Git, Netbeans, Redmine
- Tesseract, Imagemagick
- Simtext

As said in section **??**, the system is developed, so that it should work on multiple platforms. This makes it sometimes difficult to describe certain installation processes in a way that would work for everybody. As it's often most problematic, to get some Linux software running on Windows, we will mostly concentrate on the way those things are done on this platform and give the instructions for other operating systems as an aside if necessary.

3.1. Development Environment

The following parts will mostly focus on the way you can get a development version of unplagged up and running on your system.

3.1.1. Git

The source code and files of all parts of the Unplagged project are managed through Git, with the repository hosted at Github. Git is a distributed version control system, that exists since 2005 and gained more and more track in recent years. Many developers prefer it over other version control systems, because it is much easier to create different branches and merge them again or simply initalize local repositories. This made it also interesting for us to use it for Unplagged.

However, none of the team memebers had ever used Git before in a bigger context so it was a challenge to get it running on all the systems. But we took it, to explore all the features Git offers. .

Installing Git Bash

First of all let's find out, how to install the Git console application, called Git Bash. Unfortunately all the GUIs we were evaluating didn't work consistently, so we decided to use it from the command line only. A very good instruction on how to install the Git Bash can be found on the website of the github project:

Windows: http://help.github.com/win-set-up-git/

Linux: http://help.github.com/linux-set-up-git/

Mac OS X: http://help.github.com/mac-set-up-git/

Getting the source code of the unplagged project

Now it is time to get the project source code on your machine. As said before, the whole unplagged project is hosted on github, so if you want to be able to contribute source code later on, you first need to create an account there:

- https://github.com

This isn't necessary, if you simply want to look into the source code, which can be accessed via the repository URL:

- https://github.com/benoertel/unplagged

If you haven't been granted write access to the above mentioned repository by a project member (which is very likely when you are reading this document for the first time), you will need to do a fork of the Unplagged project right at github, like described in:

- http://help.github.com/fork-a-repo/

After this, the following steps are mostly the same for everybody, with the distinction of the project URIs, which should be the one of your newly created fork.

Open up the Git Bash and switch to the directory where you want the project to be located and clone the repository as you can see in listing 3.1.

Listing 3.1: Cloning a repository

```
1 cd Sites/unplagged.local
2 git clone
      https://<username>@github.com/benoertel/unplagged.git
```

After this you should have a local copy of all the repository data in the specified directory.

The most important git commands

You are now ready to use Git! Here are some more instructions on the most important commands and how to properly use it. However, if the given instructions in this manual are not enough, feel free to checkout the whole Git manual on:

- http://schacon.github.com/git/user-manual.html

The unplagged project consists of several branches, which are used to develop and store code independently of the other developers. Once a new feature is done, it is merged into the master branch. The master branch usually includes only fully tested and deployable source code.

As a new developer, it is important to create an own branch before doing anything else and switch to it.

Listing 3.2: Creating branches

```
1 git branch mynewfeature
2 git checkout mynewfeature
```

Now anything in the repository can be changed. At any point changes can be versioned in the repository by using the `git commit` command. If new files were created, `git add` has to be executed as well.

Listing 3.3: Commiting

```
1 git add .
2 git commit -m "A message that describes the changes."
```

When the feature is fully working and approved, it has to be merged back to the master branch, in order to get deployed to the staging environment. To do this, the master branch has to be checked out, updated with `git pull` and then all changes have to be merged from the new feature into the master branch. The feature branch can then be removed.

Listing 3.4: Merging branches

```
1 git checkout master
2 git pull
3 git merge mynewfeature
4 git branch -d mynewfeature
```

In comparison to Subversion for example, Git has one more step to really write back to the remote source repository. After a `git commit`, a `git push` has to be executed, each push can include multiple commits.

Listing 3.5: Pushing to the server

```
1 git push origin master
```

This is nearly it, the changes to the repository have been pushed to the master branch. The only thing, that probably has to be done now, is to open up a pull request on github, if you developed on your own fork of the project. This means, that you are asking the project members who have access to the "real" Unplagged github account, to integrate your changes into the actual project sources. A nice description of how this process is done can be found at github again:

- http://help.github.com/send-pull-requests/

Handling conflicts in merging process

It is possible, if two developers were working on the same part of file, that a conflict is found during the merge. Such a conflict could look like this:

Listing 3.6: Merge conflict

```
1 CONFLICT (content): Merge conflict in readme.txt
2
3 To https://github.com/benoertel/unplagged.git
4  ! [rejected]        master -> master (non-fast-forward)
5 error: failed to push some refs to 'https://github.com/
      benoertel/unplagged.git'
6 To prevent you from losing history, non-fast-forward updates
      were rejected
7 Merge the remote changes (e.g. 'git pull') before pushing
      again.  See the
8 'Note about fast-forwards' section of 'git push --help' for
      details.
9
10 # Unmerged paths:
11 #   (use "git add/rm <file>..." as appropriate to mark
      resolution)
12 #
13 #    both modified:      readme.txt
14 #
```

To resolve the issues, open the files listed in the error message, in this case *readme.txt* and decide how the correct version should look like, by removing all the "< < < < < < < HEAD" and "> > > > > > > b478801d68267ef479acc5ca54544634c52c545c" parts accordingly or using a dedicated merge tool, that is able to show you the changes that were made

Here is an example of how this process would work:

Listing 3.7: Conflicted file

```
1 <<<<<<< HEAD
2 The goal of this project is the creation of an easy-to-use,
      web-based
3 system to document and detect plagiarism in scientific papers.
4
5 hello world
6 =======
7
8 The goal of this project is the creation of an easy-to-use,
      web-based
9 system to document and detect plagiarism in scientific papers.
10
11 >>>>>>> b478801d68267ef479acc5ca54544634c52c545c
12 Just a change for educational purposes.
```

Could look like this after merging:

Listing 3.8: Fixed conflict after merging

```
1 The goal of this project is the creation of an easy-to-use,
      web-based
2 system to document and detect plagiarism in scientific papers.
3
4 hello world
5
6 Just a change for educational purposes.
```

3.1.2. Local Deployment

This subsection will describe how to configure a virtual host properly. A virtual host is a domain that is mapped to the local web server. It is assumed that Apache, MySQL and PHP are already running on the machine. If not, here are some tutorial to get them all running:

Windows:

http://www.apachefriends.org/de/xampp-windows.html#1098

Mac OS:

http://www.djangoapp.com/blog/2011/07/24/installation-of-mysql-server-on-mac-os-x-lion/
http://www.quarkstar.at/index.php/2009/05/18/webserver-aktivieren-und-konfigurieren-in-mac-os-x/

Most Linux distribution should already have this kind of server stack installed.

The main goal is to create a local domain and add the virtual host from listing 3.12 to the vhost config.

In Max OS X and Linux this can be done via the command line:

Listing 3.9: Mac OS X: Creating virtual host

```
1 sudo vi /private/etc/hosts
2 #add the following line:
3 "127.0.0.1 unplagged.local"
4
5 sudo vi /private/etc/apache2/extra/httpd-vhosts.conf
```

On Windows you need to open up your *hosts* file, which is mostly located in *C:\WINDOWS\system32\drivers\etc\hosts*, and add the following line on the bottom:

Listing 3.10: New host declaration

```
1 127.0.0.1 unplagged.local
```

Now you need to open your apache configuration file *C:\xampp\apache\conf\httpd.conf* and remove the hash symbol(uncomment) from the following line

Listing 3.11: httpd.conf

```
1 #Include conf/extra/httpd-vhosts.conf
```

Add the following configuration to the httpd-vhosts.conf file you just included:

Listing 3.12: Apache configuration

```
1 <VirtualHost *:80>
2   ServerName unplagged.local
3   DocumentRoot "/Users/me/Sites/unplagged.local/public"
4   SetEnv APPLICATION_ENV "development"
5
6   <Directory "/Users/benjamin/Sites/unplagged.local/public">
7     Options +Indexes +FollowSymLinks +ExecCGI
8     DirectoryIndex /index.php
9     AllowOverride All
10    Order allow,deny
11    Allow from all
12  </Directory>
13 </VirtualHost>
```

You can tryout your new configuration by entering *unplagged.local* in your browser.

3.1.3. Netbeans

Configuring Tests

Documentation

Unplagged uses ? for the generation of a HTML page of all the source code documentation comments, because of it's superior and much more beautiful user interface in comparison to the older ?.

Sadly the automatic generation is not yet supported by Netbeans, but as it will be soon(?), we are currently only generating this server-side as described in section ??.

This section will be enhanced, when the Netbeans user interface becomes available. If you are interested, you could install the software for yourself and use it over the command line.

3.1.4. Additional Software

As we currently have no installer or script that checks for installed software, you still have to install some additional dependencies to make some parts of the system work. Those are mainly command line tools that we use for the optical character recognition or text comparison.

Most of the times the software wouldn't break completely if those dependencies were not installed, but some parts would silently fail, which is of course one of the more annoying problems to debug.

Tesseract

Tesseract is an open source OCR[1] software, that is used to

Simtext

Imagemagick

3.1.5. Continuous Integration

To always have a running version of the latest code, we use an automated workflow, that always deploys everything that has been pushed to the Github repository on the Unplagged staging server. The machine this is done with, is a simple Ubuntu web server, that is also used for hosting our collaboration tools and the webpage.

As you can see in figure 3.1 the mechanism used for this is a callback, the *post-receive hook* of git, which github employs to let it's users enter a *post-receive URL* to call a URL after someone has pushed to the repository. The URL that gets called is located on our staging server and gets answered by a Redmine plugin called "redmine_github_hook",

[1] Optical character recognition

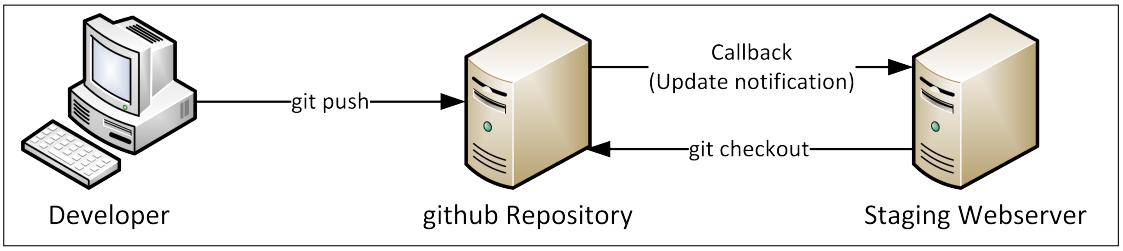

Figure 3.1.: Deployment workflow

that would normally only call a checkout on the server-side repository, so that the newest sources can be seen via the Redmine web frontend.

Listing 3.13: Changes to redmine_github_hook.rb

```
1  # Fetches updates from the remote repository
2  def update_repository(repository)
3    command = git_command('fetch origin', repository)
4    if exec(command)
5      command = git_command("fetch origin '+refs/heads/*:refs/
           heads/*'", repository)
6      exec(command)
7
8      #custom checkout to preview area
9      system('sh /usr/local/etc/scripts/buildUnplaggedPreview.sh
           ')
10   end
11 end
```

However, we tweaked the source code of this plugin slightly, as you can see on line 9 in listing 3.13 so that it also calls the below bash script(listing 3.14) to initiate the deployment process.

Listing 3.14: Deployment script

```
1 #!/bin/bash
2
3 cd /var/git/unplagged.git/
4 GIT_WORK_TREE=/var/www/preview.unplagged.com git checkout -f
5
6 cd /var/www/preview.unplagged.com
```

```
7 #generate phpdoc
8 apigen -s application/ -s library/Unplagged/ -d docs/phpdoc --
      title "Unplagged Documentation" --todo yes
9
10 #run database build scripts
11 cd scripts/build
12 php initdirectories.php
13 php doctrine_staging.php
14
15 cd /var/www
16 chown www-data:www-data preview.unplagged.com
```

The bash script is then used to do a "clean checkout"(without hidden `.git` folders) of the repository and to run "Apigen", an engine to process the PHPDoc comments inside the project. Those two things can be accessed by the already prepared vHosts on the server:

- http://preview.unplagged.com/
- http://phpdoc.unplagged.com/

If you would like to get access to the preview areas, you need to obtain the password and username from a team member.

Possible Improvements

The above described workflow is, as we believe, already on a good way, but it still has a lot of room for improvement. First of all, it would be nice to only let the deployment go through, if the unit tests ran successfully on the server and to have some sort of email notification mechanism if this wasn't the case.

Another improvement would also be to have a separation into a staging environment with the newest commits and an actual preview environment, that can be deployed to a known stable state/commit of the system in a simple manner.

3.2. User Interface / User Experience

This chapter will explain the progress and development of the user interface of our project. As we tried to follow the typical project workflow, we first drew a lot of mockups in the beginning, which represented the main features of Unplagged. At first the wireframes, or also called mockups were drawn by hand, before we digitalized them. As an example the mockups of the 'new case' page are shown below. All the other mockups can be found in the appendix.

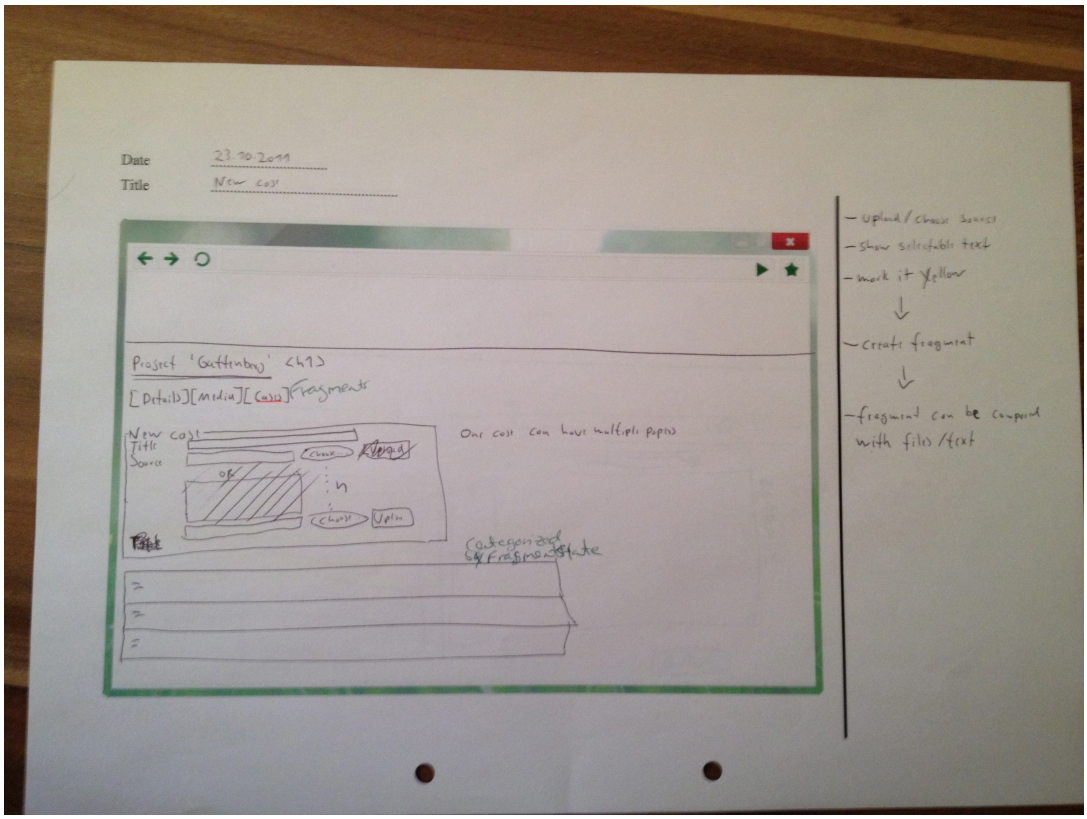

Figure 3.2.: Mockup – New case – hand-drawn

After we had a basic idea how the main interface should be structured, we created a first screen in Photoshop. Therefore we got several helpful suggestions from the website PremiumPixels: http://www.premiumpixels.com/.

The next step, before the HTML template got created, we defined the key features, our user interface should take care of:

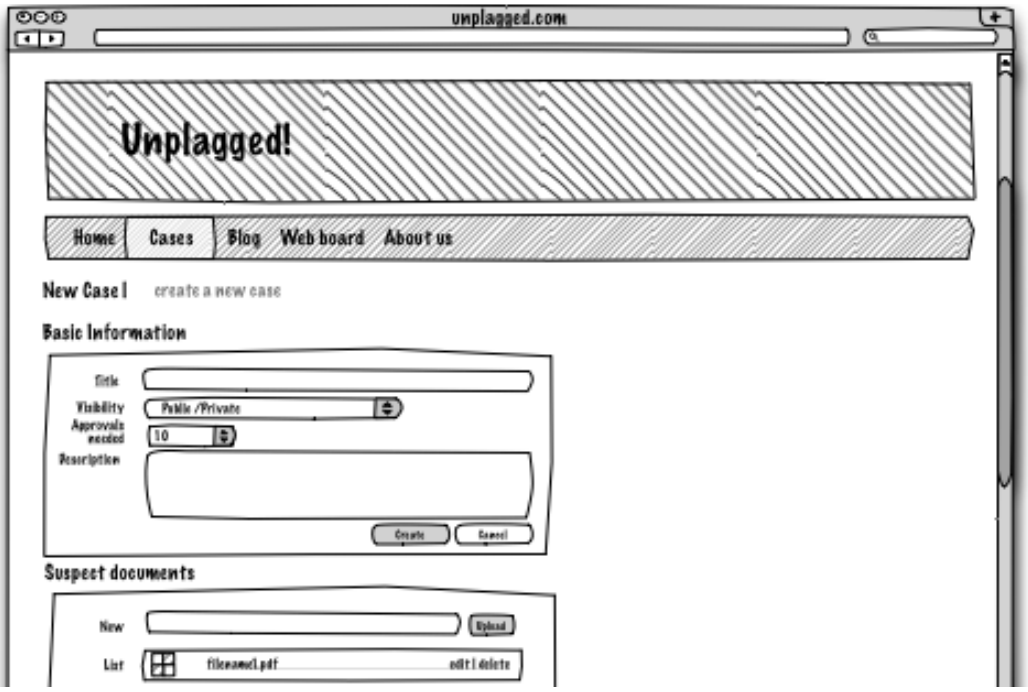

Figure 3.3.: Mockup – New case – digitalized

- Reponsive Layout – optimized layouts for different devices
- Cross-Browser-Compatibility
- Light-weight and w3c-conform XML Markup
- Progressive enhcancement with CSS3 – CSS instead of images where possible

3.2.1. Responsive Layout using CSS3 Media Queries

Since the worldwide amount of smartphones and templates is growing very fast, website developers should optimize for these devices as well. And so do we. Some functionality as uploading a file, doesn't work on iOS at all, but at least all functions that are working on mobile devices, should work. So the goal is creating a user interface that uses the same markup, but displays is differently on different devices. Therefore CSS media queries can be used, these are basically conditions that execute a part of CSS only if the condition is

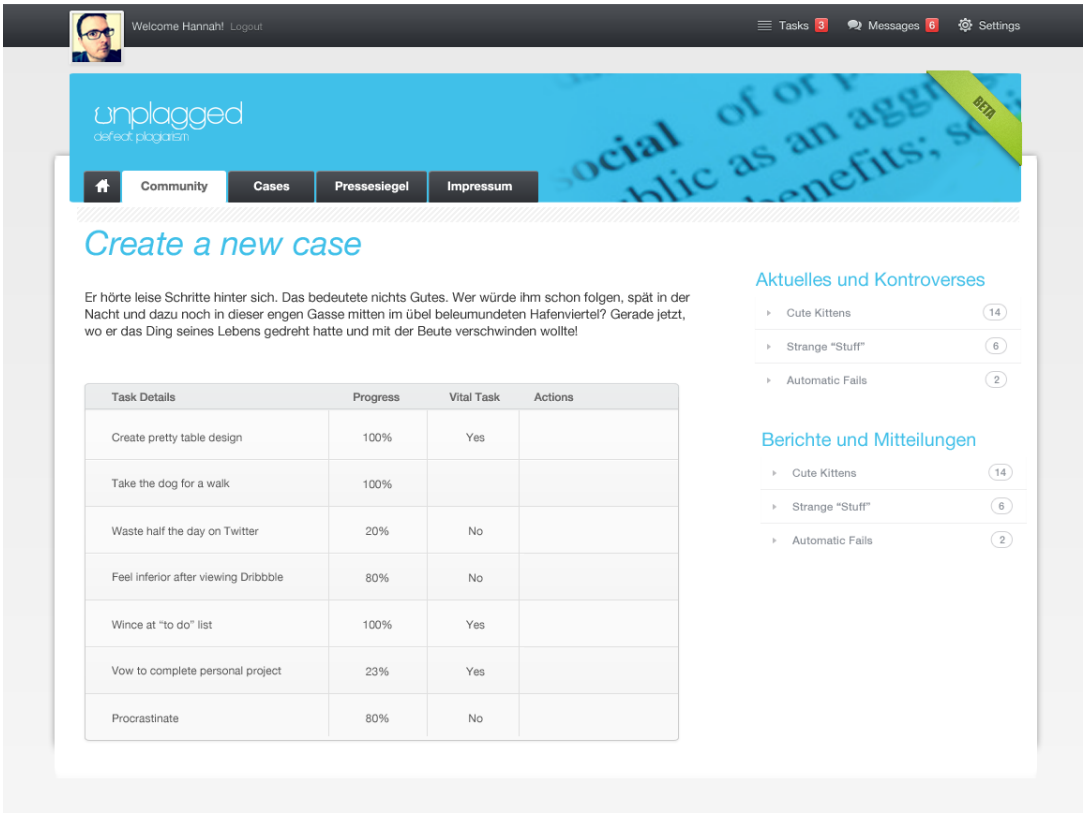

Figure 3.4.: Initial Screen PSD

true. The most prominent conditions are the following:

Listing 3.15: CSS Media Queries

```
1 max-width: 600px  # max browser width 600px
2 min-width: 300px  # min browser width 300px
3 orientation:landscape # current orientation landscape mode
4 orientation:portrait # current orientation portrait mode
5 -webkit-min-device-pixel-ratio: 2 # min pixel ration (iPhone
      4, Retina)
```

These media queries can be combined in any order to display an optimized page for each device. Since only the CSS changes, the HTML does not have to be touched. An example query looks like this:

Das entsprechende CSS Media Query sieht wie folgt aus:

Listing 3.16: CSS Media Query

```
1 @media only screen and (max-width: 600px) { }
```

Media queries are a new feature of modern browsers, except IE9. Though, this isn't a problem, because browsers that don't support them, just display the default css for desktop browsers.

3.2.2. Javascript and fallbacks

Even though many websites require Javascript as mandatory for using the whole functionality range of the page, it is important to provide as much functionality as possible, when Javascript is turned of. A short example will show how to implement a pagination with and without Javascript.

Usally when Javascript is disabled, the whole page will be reloaded, when the user changes to another page of the paginated content. The URL in this case will look something like this: http://unplagged.local/document/list/page/2. When Javascript is enabled, it

is much faster to only refresh the area of the page, that really needs to reload, in this case the table with the content of the next page. It is only possible to change the hash of an URL, the part after the hash key (#), using Javascript, the changed URL will be: http://unplagged.local/document/list/#page/2. An event called 'hashchange' can trigger a change of this part of the url and then reload the content through an AJAX request.

The pagination has the same HTML markup with and without Javascript, but with Javascript enabled it is much more convenient.

Listing 3.17: Javascript Pagination

```
1  $(".pagination a").live("click", function() {
2          var href = $(this).attr("href");
3          if(href) {
4            var substr = href.split('/');
5            var hash = substr[substr.length-2] + "/" + substr[
                 substr.length-1];
6
7            window.location.hash = hash;
8          }
9          return false;
10     });
11
12     $(window).bind('hashchange', function(){
13         var newHash = window.location.hash.substring(1);
14
15         if (newHash) {
16             var substr = newHash.split('/');
17             var hash = substr[substr.length-2] + "/" + substr[
                   substr.length-1];
18
19             var url = window.location.pathname;
20             if(url.charAt(url.length-1) != '/') {
21               url += '/';
22             }
23             url += hash;
24             $("#main-wrapper").load(url + " #main");
25         };
26
```

```
27        });
```

3.3. Frameworks

When we first discussed which programming language and frameworks, the Unplagged project should be built on, we figured out, that everyone was familiar with PHP. Since programming in a group requires a much better structure, than programming on your own, we needed a framework that requires a comfortable Model-View-Controller structure, we decided to use the Zend Framework. And to get rid of all the database issues as well as getting a flexibility in the used database system, we decided to use an Object-Relational-Mapping framework, called Doctrine. What an ORM is, will be discussed later on.

3.3.1. Zend Framework

The Zend Framework is a typical PHP Framework using the Model-View-Controller pattern. Due to it's pre-defined directory structure, it is easy to get well seperated code.

The directories and their meaning:

- application — includes controller, models and view
- data — currently only includes i18n stuff (language stuff)
- docs — PHP Documentation and Developers Manual
- library — extrnal and internal frameworks and extension to the Zend framework
- public — files that are accessible directly through the browser

- scripts — build scripts, deploying scripts
- temp — data that is overridden at any deployment
- tests — PHP Unit tests

The Zend framework offers REST-ful URLs that follow a fixed pattern: unplagged.local/controller/action/key/value/key2/value2. Each controller is defined as ControllerNameController.php in the application/controllers directory and includes all possible actions. For example a file controller will look like this:

Listing 3.18: Persisting an object to the database in Doctrine

```
1 class FileController extends Zend_Controller_Action{
2   public function init(){
3   }
4
5   public function indexAction(){
6   }
7
8   public function uploadAction(){
9   }
10
11    public function listAction(){
12  }
13
14  public function downloadAction(){
15  }
16 }
```

By default, if no action is defined, the indexAction is called. For each action the appropriate view is by default rendered in the application/views/scripts/conntrollerName/actionName.phtml file.

The models directory includes all the objects, this will be discussed in more detail in the following chapter about Doctrine.

3.3.2. Doctrine

The whole database connection management of Unplagged is implemented using the Doctrine Framework in version 2. It consists of two layers, a database abstraction layer (DBAL) and an object relational mapping framework (ORM). The DBAL uses PDO, a PHP framework for encapsulating database statements. The DBAL manages the communication with any kind of SQL database and offers an own query syntax. This has the advantage, that the database behind the framework can be changed from MySQL, to OracleSQL, PostgreSQL or SQLite at any time. The DBAL is the agent between PDO and the ORM. The ORM is the connection between PHP objects and the DBAL.

Before the use of Doctrine is described, it will be explained, how the database on a new machine can be created and how the database structure can be updated, whenever something changed in the structure. Actually this is very easy, it is required to have a local my sql database at this point having 'root' as a username, no password and a database called 'unplagged'. It is also possible to create a new configuration in the application/configs/application.ini file, although this step will not be described in this chapter. If the database is created, the build script can be executed:

Listing 3.19: Updating database structure

```
1 php unplagged.local/scripts/build/doctrine.php
```

Now, the database is created or updated. If the database already existed, the data in it will not be removed! As described below, the big advantage of ORM is, that the programmer can stay in the PHP object context at any time. The only thing that has to be done additionally, is adding comments to the member variables of a class, that define the fields in the database:

Listing 3.20: Defining a class in Doctrine

```
1 /** @Entity */
2 class UserClass
3 {
4 /** @Column(type="integer") */
5 private $id;
6 /** @Column(length=30) */
```

```
7 private $username;
8 }
```

The whole syntax documentation of doctrine can be found here: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/index.html

To persist a new object to the database, the persist method on this object has to be called, this writes the object into the doctrine cache. It stays in the cache, until the flush method is called, which actually executes all the previous operation since the last flush. These cann be deleting, updating, or editing an object.

Listing 3.21: Persisting an object to the database in Doctrine

```
1 $user = new User();
2 $user->setUsername('Max');
3 $em->persist($user);
4 $em->flush();
```

As the previous examples show, no database programming is necesarry to create or persist a new object to the database, everyhting can be done in the PHP object context.

3.4. Architectural Goals

3.4.1. Progressive Enhancement

3.4.2. Test Driven Development

A. Sprints

B. Minutes

C. Time Logging

D. Mockups

D.1. Hand-Drawn

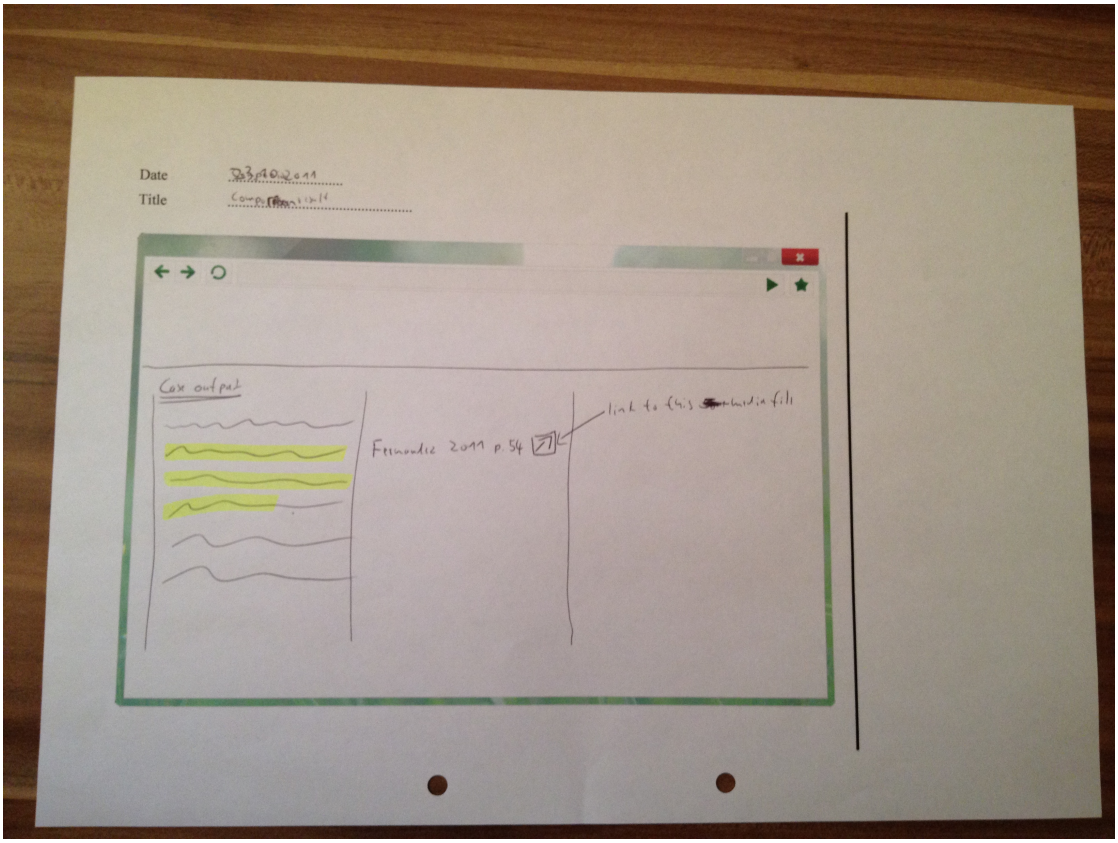

Figure D.1.: Mockup – Compare results – digitalized

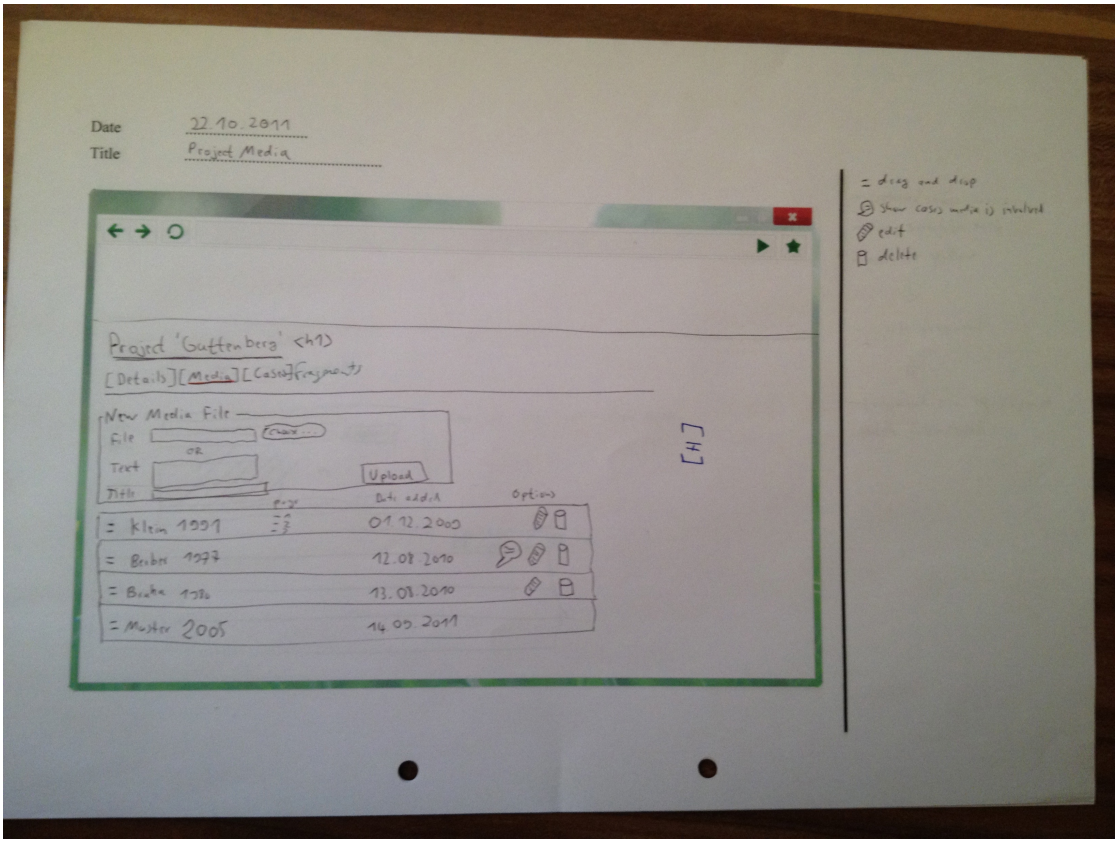

Figure D.2.: Mockup – Media list – digitalized

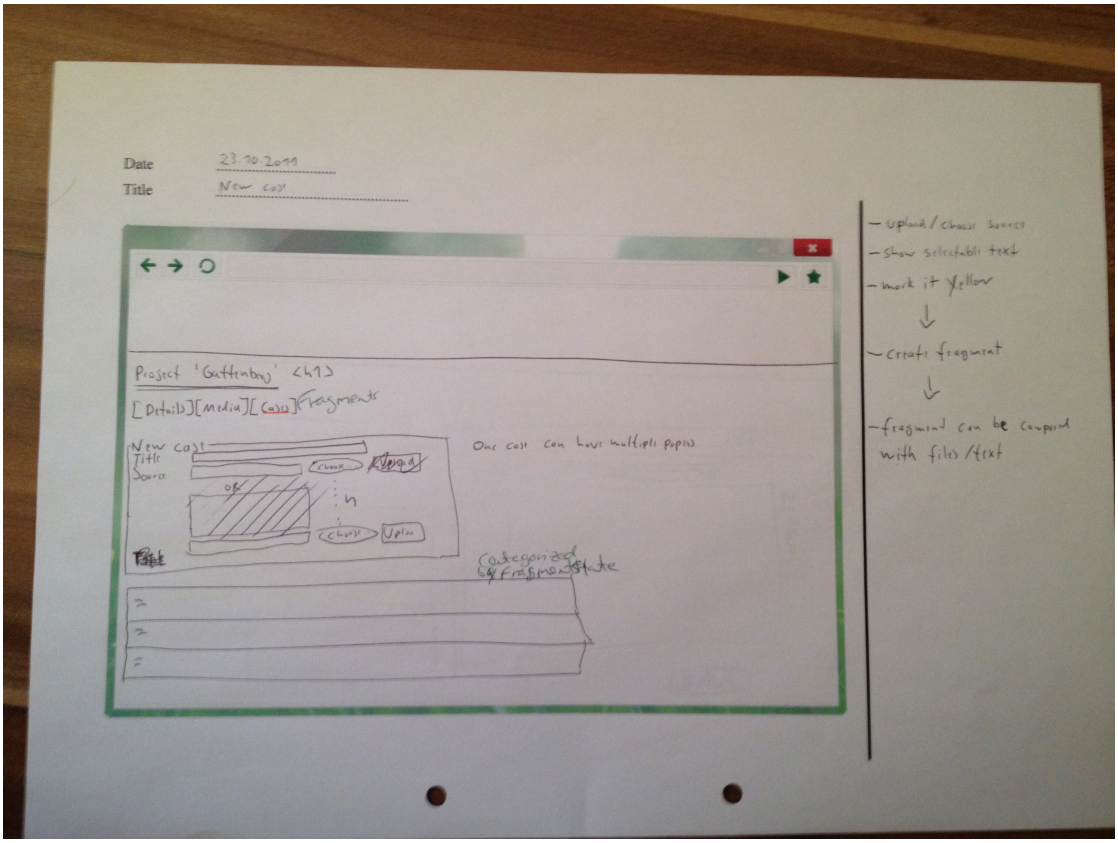

Figure D.3.: Mockup – New case – digitalized

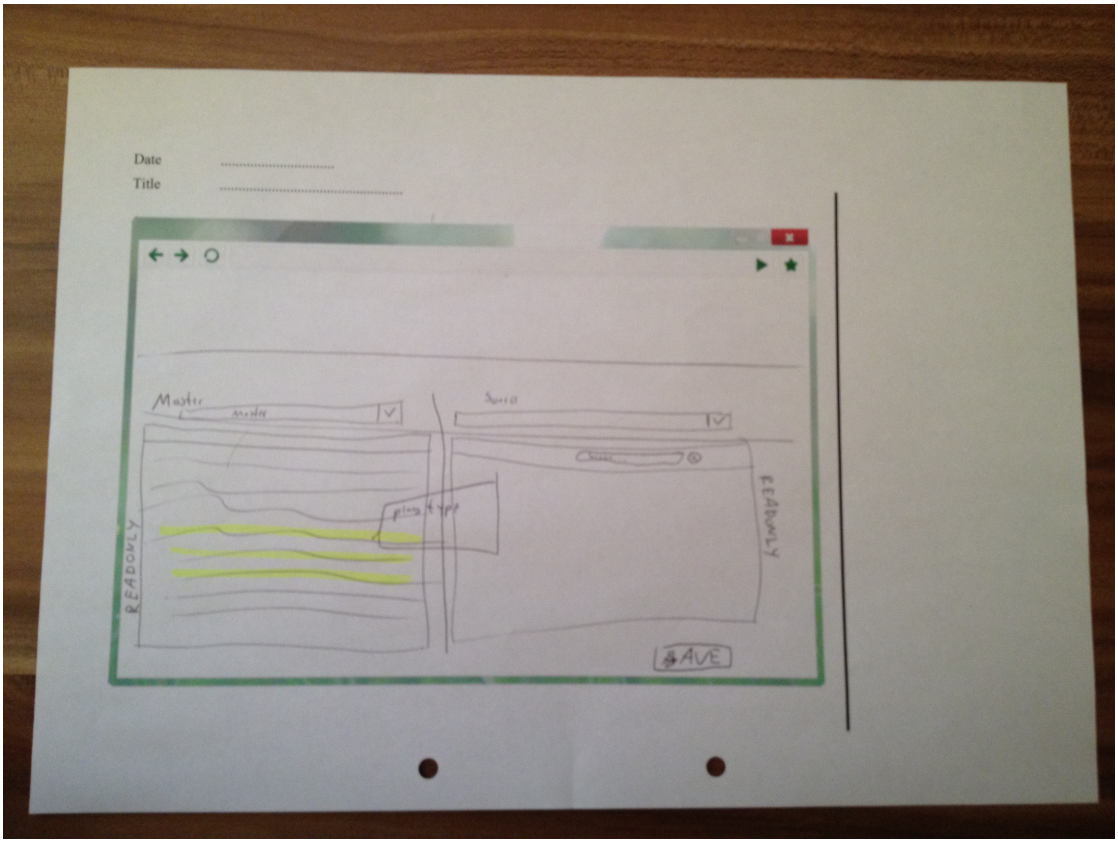

Figure D.4.: Mockup – New fragment – digitalized

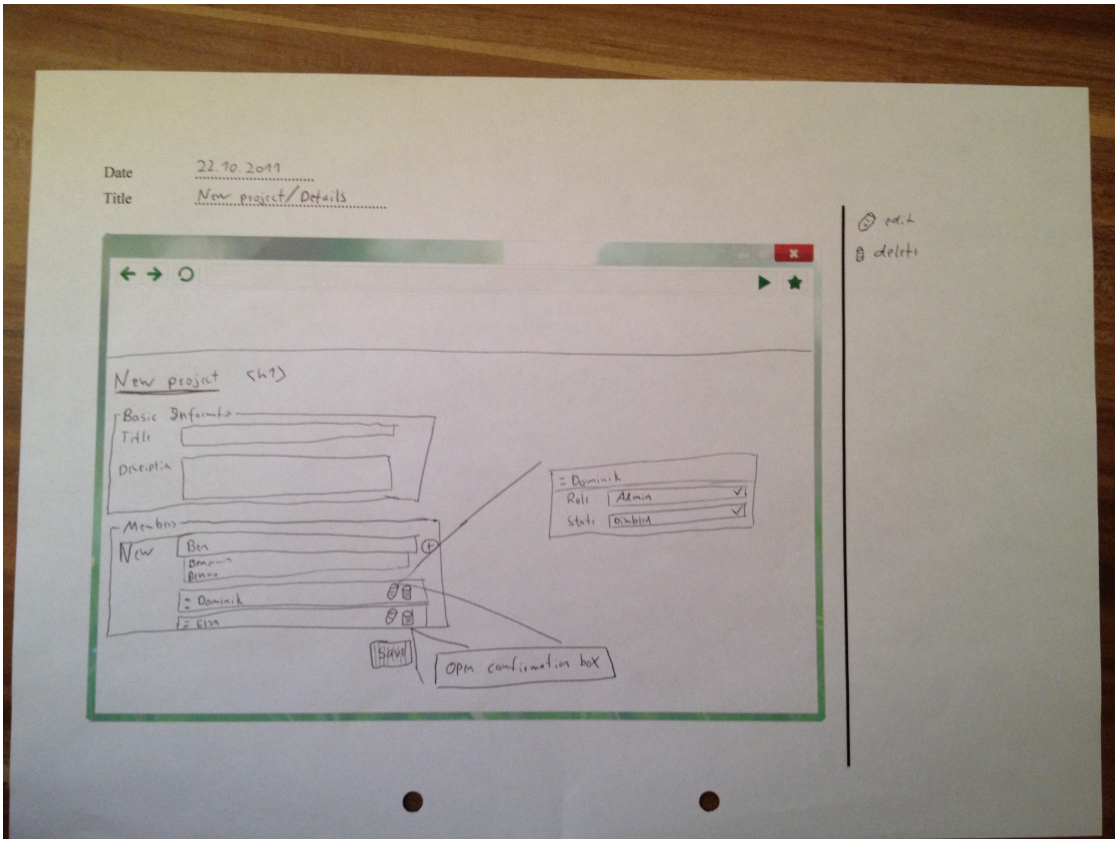

Figure D.5.: Mockup – New project – digitalized

D.2. Digitalized

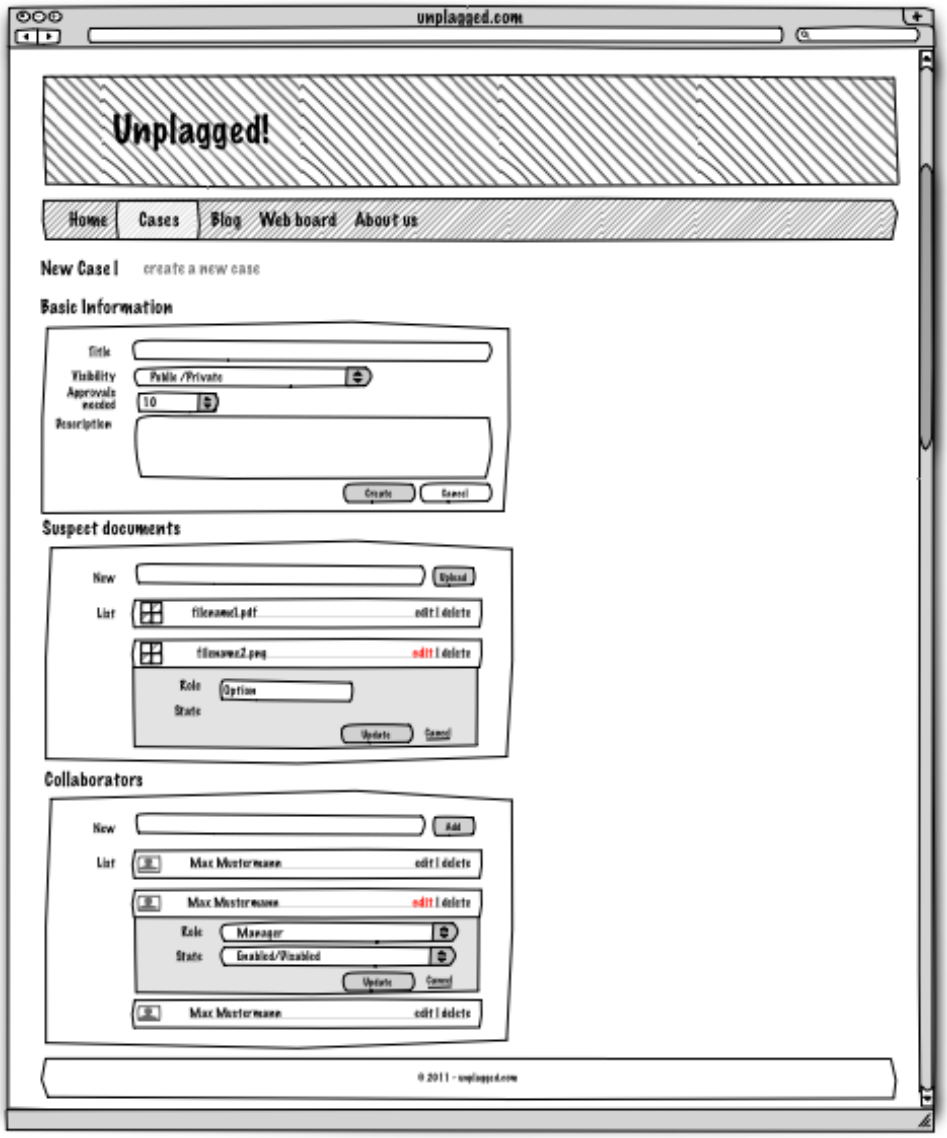

Figure D.6.: Mockup – New case – digitalized

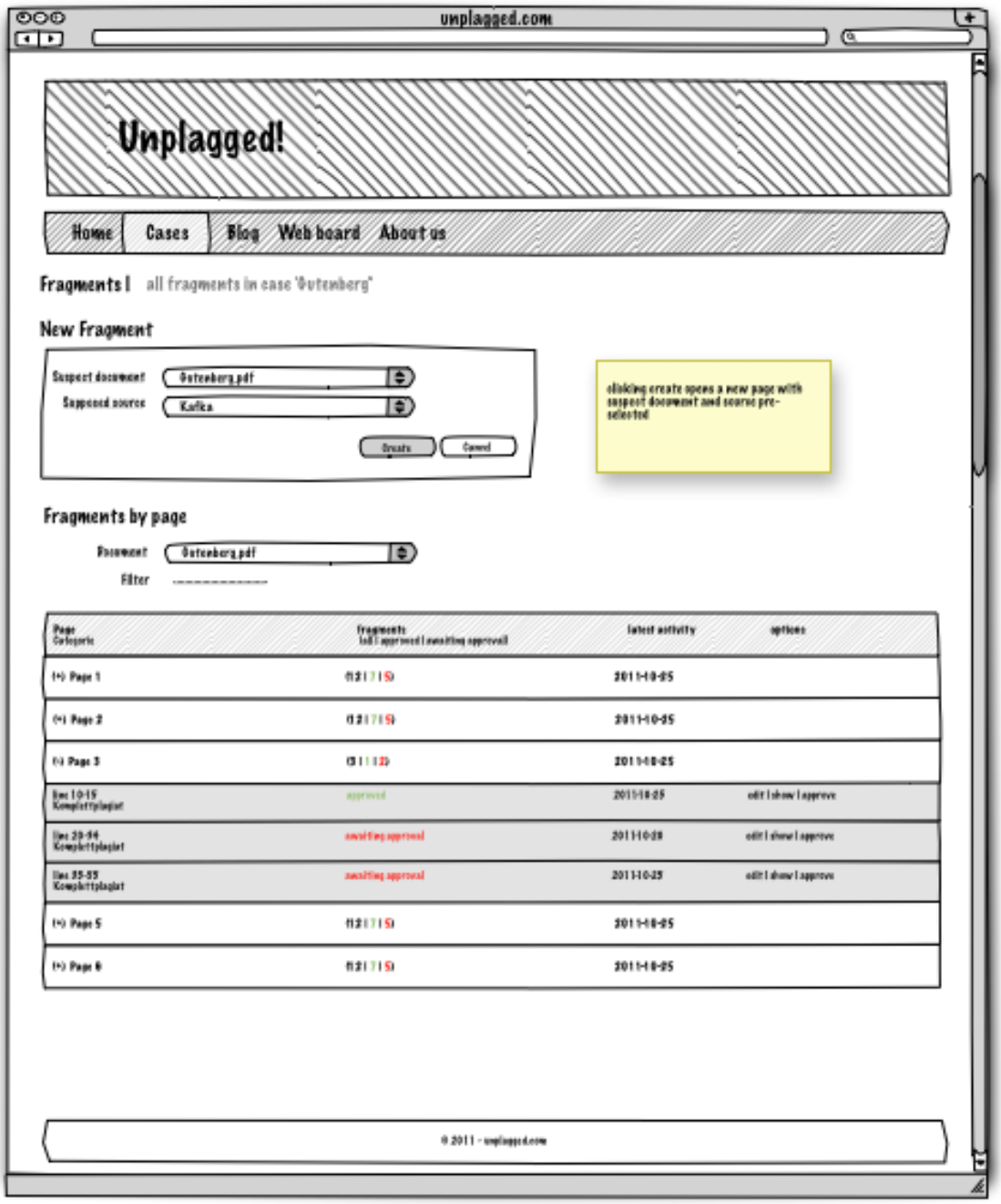

Figure D.7.: Mockup – List fragments – digitalized

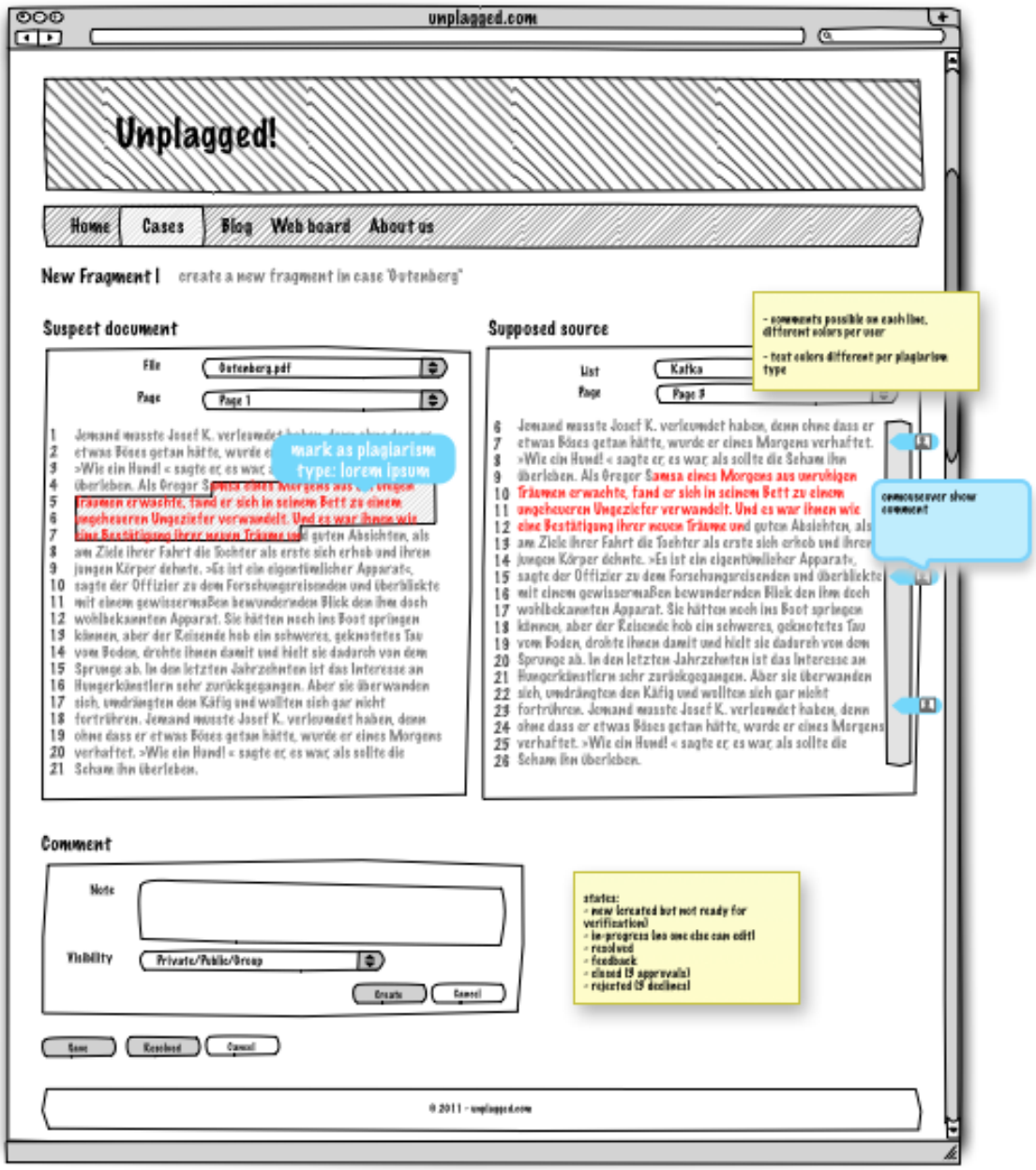

Figure D.8.: Mockup – New fragment – digitalized

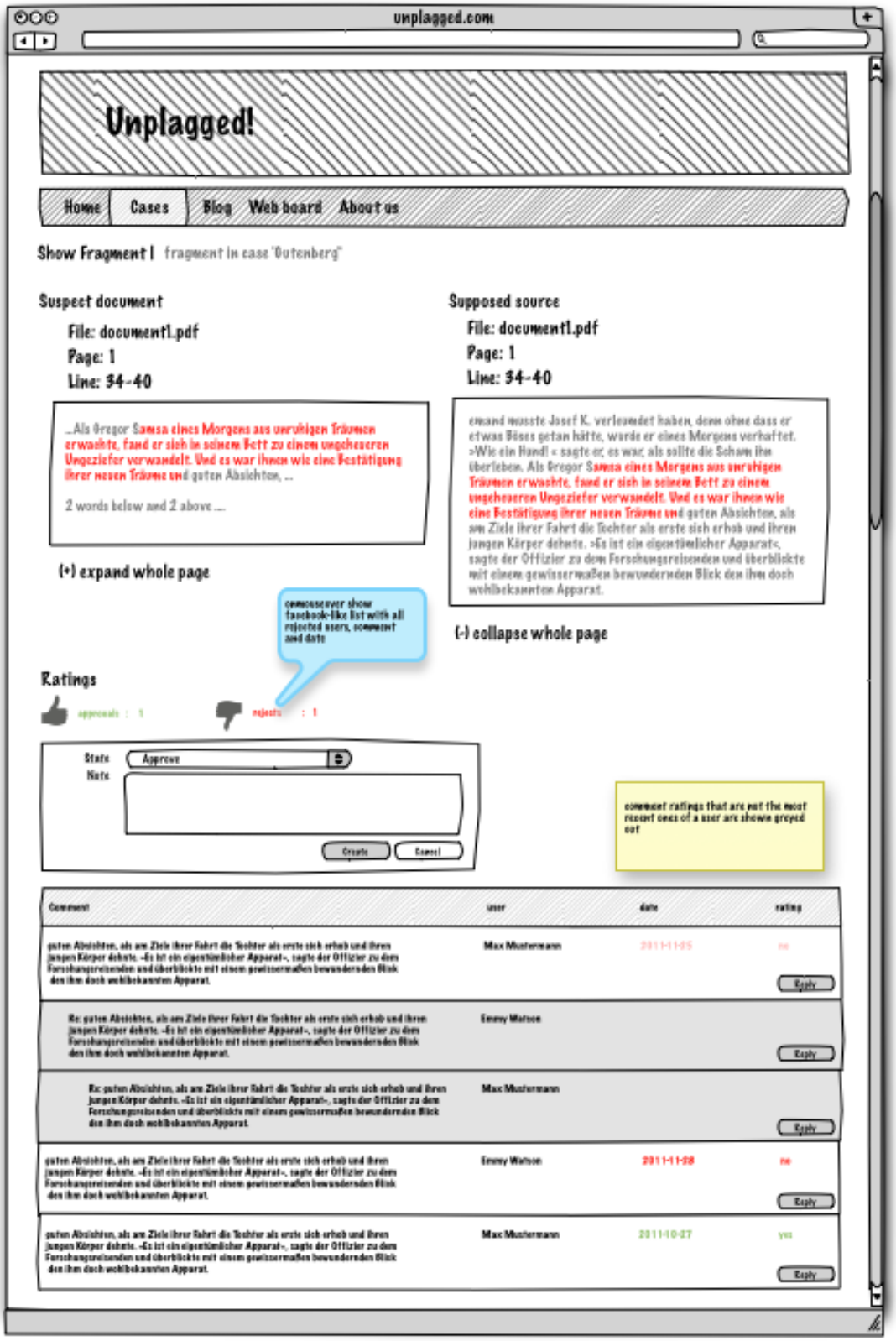

Figure D.9.: Mockup – Show fragment for approval – digitalized

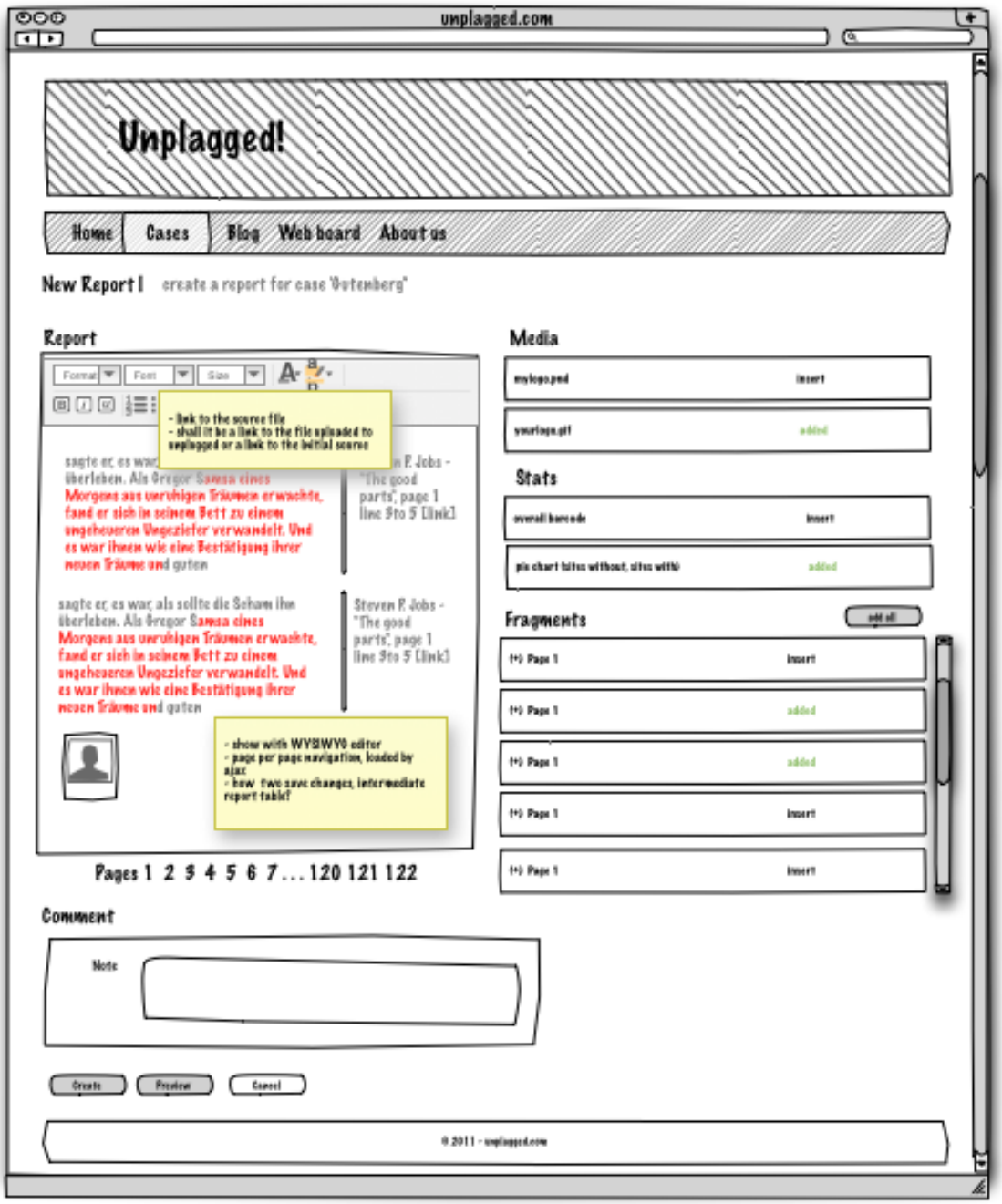

Figure D.10.: Mockup – New report – digitalized