IWGS-II – Informatische Werkzeuge in den Geistes- und Sozialwissenschaften – SS 2019

Michael Kohlhase
Informatik, FAU Erlangen-Nürnberg
For Course Purposes Only

August 1, 2019

Contents

Assignment 1 (First Steps with GIT) – Given 3. 5. 2019, Due 12. 5. 2019

Problem 1.1 (Make a GitLab Account)

We will use the GitLab instance at `http://gitlab.cs.fau.de` to manage your GIT repositories. Make an account there with your FAU Single Sign On, and set a password on the account. 20pt

Problem 1.2 (Make an IWGS Homework Repository)

Make a private repository in your personal group on your FAU GitLab from Problem 1.1, clone it on your machine, move all your IWGS homework submissions there, adds them, commits, and pushs. 20pt

It is a good idea to keep all the IWGS homework submissions there. We are going to re-use them in the IWGS project.

Problem 1.3 (IWGS-II Project)

During the remainder of the semester, you will be developing an information system for the "Farmer-Fair Pictures" data set. 20pt

1. find two other people you want to work with, form a working group (WG), and give the group a name.
2. Make a GitLab repository in the personal group of one of the WG members, and give the other WG members developer or admin permission.
3. add a file README.md that shortly explains the purpose of the repository – two sentences are enough.

Solution: *No master solution for this problem; there is an example in the problem text above.*

Problem 1.4 (Using GitLab)

The aim of this problem is to get your hands dirty in using a revision control system, and to show you that this is actually quite simple. Here are your tasks: 20pt

1. Using your account from Problem 1.1, clone the repository `https://gitlab.cs.fau.de/IWGS-SS19/collaboration` to get a local working copy.
2. We want to collaboratively build a membership file for the IWGS course, you can find it as `users.txt`. Add your personal information (account name, real name, and e-mail) and commit it. If there are conflicts, you need to resolve them (make sure that you do not delete the information of your peers). Do not forget to give a meaningful commit message and to push your changes.
3. Add a file ⟪account⟫.txt with a friendly note about your experience with IWGS, where ⟪account⟫ is your account name to the users directory and commit/push it.

Assignment 2 (Issues in Git) – Given 10. 5. 2019, Due 19. 5. 2019

Problem 2.1 (Issues in GitLab)

The aim of this problem is to get your hands dirty in using a bugtracker, and to show you that this is actually quite simple. Here are your tasks: 30pt

1. On the collaboration project from Problem 1.4 you can find the IWGS course notes as the file notes.pdf for reference. Report an issue about the slides, this can be a bug report or a feature request. Label it appropriately and assign it to one of your peers.
2. Comment on one of the existing issues. If you have been assigned an issue, be sure to comment on that as well.
3. Then add the number of the issue you created and the issues you comment upon, in the line with your data in the file `users.txt`, commit it and push.

Hint: For the issues just use the GitLab web interface at `https://gitlab.cs.fau.de/IWGS-SS19/collaboration/issues`

Problem 2.2 (Bug Report in GitLab)

In his free time Dr. Leonardo is developing a system for automatic detection of famous buildings in paintings. To aid with the image processing, he is using an open source framework available on Gitlab. Unfortunately for certain images the framework sometimes causes his program to crash when loading images from the hard drive. Therefore Dr. Leonardo decides to file a bug report on the Gitlab page, hoping that the developers fix the problem. By doing some research, he gathers the following information: 40pt

1. The function responsible for image loading crashes for images which are higher than wide (i.e. upright images).
2. The bug only seems to appear on Windows 10. On Linux, Mac and older versions of Windows the function works as expected.
3. The problem only occurs for certain image types, `jpg` and `gif`, not for `png`.
4. The bug disappears, if he converts the image to grayscale in Photoshop beforehand [1].
5. Other image processing frameworks handle the problematic images just fine.

Help Dr. Leonardo formalize all these facts in a bug report.

Problem 2.3 (Markdown)

Being happy with his newly acquired bug report writing skills, Dr. Leonardo now wants to make it nicer in order to better catch the developers attention. He found out that he can use a markdown language to achieve his goal! 30pt

Format your bug report from Problem 1.2 in **Markdown syntax**! Make sure to use at least one *heading*, two *sub-headings*, one *list*, and one element of *text formatting* (e.g., bold, italic, underline). Of course, you are free to use as many as you want!

[1] `https://en.wikipedia.org/wiki/Grayscale`

Assignment 3 (Image Database Part 1 - Adding entries) – Given 17. 5. 2019, Due 26. 5. 2019

Note: In the following weeks, we will gradually build an image database for the Kirmes data set provided by Prof. Peter Bell. The images are available under `https://gitlab.cs.fau.de/iwgs-ss19/KirmesDH`. Note, that you are only allowed to clone the repository, not push to it.

In this first assignment you will set up a simple SQLite database and add images to it, along with some metadata. As a reference for how to create a database, set up the tables and insert data, you can refer to `http://www.sqlitetutorial.net/sqlite-python/`, where all relevant parts are demonstrated. We recommend you download and install the *DB Browser for SQLite* which allows you to browse your database.

Since this is the first assignment working towards our project, we recommend you work in groups!

We have prepared a code skeleton (BuildDB.pyy) for you to complete, so that you can concentrate on the important parts. It is heavily commented. Familiarize yourself with the code before starting with the assignment. The parts for you to fill out are marked by `TODO`.

Problem 3.1 (Setting up the Database) 30pt

In this exercise we will set up our database tables. Start by cloning the KirmesDH repository[2]. The dataset consists of a directory img/, which contains images and a folder metadata/ containing CSV files. The other directories are not important for this assignment.

Familiarize yourself with the metadata format. As you can see most files employ the same columns, however some data may be missing. We will mirror the given column structure in our database.

1. In the given code skeleton, change the values of the variables metadataFolder and imageFolder at the top of the file according to your folder structure.
2. Establish a connection to the database. Use the databaseName variable.
3. Create a table with name Images in the database with the following column structure:
   - FileName, type TEXT
   - Title, type TEXT
   - Subtitle, type TEXT
   - Archive, type TEXT
   - Artist, type TEXT
   - Location, type TEXT
   - Date, type TEXT

[2] `https://gitlab.cs.fau.de/iwgs-ss19/KirmesDH`

- Genre, type TEXT
- Material, type TEXT
- Url, type TEXT
- Content, type BLOB

4. At the end of the file, commit all changes you made to the database and close it.

Run your script and open the resulting database file in the DB Browser for SQLite. Make sure that you see the Images table and that its layout is correct.

Hint: CREATE TABLE fails to create a table if one with this name already exists. Before creating a table you should therefore issue the DROP TABLE IF EXISTS <tablename> command.

Problem 3.2 (Parsing the Input Data)

In this exercise we will parse the metadata files and extract all relevant data. Since the input data is not curated very carefully and some entries may be missing, we need to design our program as robustly as possible. 30pt

Amend the `parseMetadata` function in the given python script for this assignment. The prepared code opens the `CSV` file and uses the module `csv` to parse it. Detailed information on the `csv.DictReader` can be found here: `https://docs.python.org/3/library/csv.html#csv.DictReader`.

In the loop do the following for each row of the file:

1. Use the `getValue` function to extract the relevant data.
2. Call the `addImage` function with the data.

Make sure that the data is parsed correctly by running your program and printing the extracted values. Assure that the program does not crash if certain data fields are not available.

Problem 3.3 (Inserting Data into the Database)

40pt

In this last exercise we fill our database with the parsed data. Before starting with this task, assure that the previous two assignments work correctly.

Complete the `addImage` function.

1. Check whether in the img/ folder a file with the specified file name exists. If yes, open and read it and store the content in the `imageData` variable.
2. Insert all data fields into the database by issuing the correct SQL command.

Run your script. Make sure it does not crash and check your database in the *DB Browser*. All values should be in the correct column. Some rows should have values in the Content column. In the *DB Browser* you can see the image when you click on the table cell.

Assignment 4 (Image Database Part 2 - Setting up the Web Server) – Given 24. 5. 2019, Due 2. 6. 2019

Note: Last week we set up a simple database for the Kirmes Image data. In this exercise we will start establishing a web server, using the `bottle` framework we introduced last semester. We are building on top of the code from last week, so you may either continue with your own code or use the sample solution from last week as a starting point for this exercise.

For the web server we again prepared a code skeleton for you (Server_Skeleton.py and Index_Skeleton.tpl).

Problem 4.1 (Adding a Primary Key to our Table) 20pt

Our table Images from last week supports nearly all functionality we need. However currently it lacks the ability to uniquely identify a single entry, since all properties could be featured in multiple entries.

We therefore introduce primary keys. To this end, amend your Images table by adding a field Id of type INTEGER. Mark it as a primary key. When inserting into your database, you don't actually have to provide a value for the Id, since SQLite will simply use the next free number.

Problem 4.2 (Setting up our Web Server) 80pt

We will now set up a simple web server using the bottle framework. As a starting point you can use the Server_Skeleton.py and Index_Skeleton.tpl we provide you.

You might need to install the bottle package first. In your command prompt (terminal) issue the following command:

```
pip install bottle
```

You should now be able to run the provided code. Make sure you adapt the value of the variable databasename to match your database file.

After starting you can access your website by visiting the URL `http://localhost:8080/` in your browser. The content of this page is for you to implement.

We provide a route /imageraw in the getImage function. Follow the instructions in the code to try out the function and see how it works. For all operations which need to display images from the database on your website you should use this route.

Your job is to implement the index function, which is called when the home page is visited. In the end this page should display a large table where all entries of your database are listed.

1. Start by querying your database for the data you want to display. Select at least the Id, Title, Subtitle, Artist, Material and Archive of each entry. Issuing the appropriate SQL command should provide you a large list of entries. Make sure that this works before continuing.

2. Last semester we created websites in bottle by creating HTML code from python. This does not scale well to larger projects. We will therefore use bottle's own template

engine, which allows you to write normal HTML documents, which you can augment with snippets of python code. You can read about the templating in the bottle documentation: `https://bottlepy.org/docs/dev/tutorial.html#templates`.

From the index function, pass the data you queried from the database to the template function. In the Index_Skeleton.tpl file, create a HTML table. This should employ columns for each data field you queried (Title, Subtitle, etc).

Inject python code with the appropriate syntax, which loops over the queried data and fills the table. The Archive field should be a link, which leads you the archive's website. Run your server, visit its URL and check if everything works.

3. Augment your HTML table by adding one more column called Thumbnail. This should display a small version of the image stored in each data entry. For this refer to the following tutorial: `https://www.w3schools.com/tags/tag_img.asp`.

   Set the thumbnail to an appropriate size (e.g. 200 pixels). As source use the /imageraw route described above. Make sure you specify the correct id for each entry.

   Test your website and enjoy it!

Assignment 5 (Image Database Part 3 - Details Page) – Given 31. 5. 2019, Due 9. 6. 2019

Note: In this exercise we will augment our web server by another route, which displays detailed information for a single image entry. As always, you can continue with your own code or start with the sample solution from last week.

As a reminder: The code skeleton is available on StudOn together with this assignment sheet or in the Kirmes repository. Just pull the latest version of the repo!

Please read the entire assignment sheet before starting to program!

Problem 5.1 (Details Page) 100pt

Our overview table is nice, but we would like the user to be able to inspect a certain entry more closely. We will therefore create a new route, which displays information for a single image on its own page.

1. In your Server.py file, create a new route /details/<id:int>. Given an Id as parameter, the function should query the database for this entry. If no entry with the Id can be found, use bottle's abort function to display an error with the code 404: `https://bottlepy.org/docs/dev/tutorial.html#http-errors-and-redirects`.

2. Create a new template file Details.tpl. From your python code, call the template with the information you queried from the database. In the template, write HTML code which displays the given information in a nice and easy-to-read way.

   Some information might not be available (NULL/None). Handle this case!

   Test your page by navigating to the details URL for some example image, e.g. `http://localhost:8080/details/27`. Make sure, that all data is displayed correctly.

3. On the details page, also display the image in full size. You may again use the /imageraw/id route from last week as source.

4. Amend your Index.tpl from last week in the following way: Each image thumbnail in the table should be a link (<a href=...>), which leads to the details page of this respective entry, i.e. by clicking on the thumbnail of image 27 your website should navigate to the URL `http://localhost:8080/details/27`.

Test all your code and enjoy!

Assignment 6 (Image Database Part 4 - Image Manipulation) – Given 7. 6. 2019, Due 16. 6. 2019

Note: In this exercise we use Pillow to manipulate images. Install Pillow by typing one of the following lines in your command line:

```
pip install Pillow
py -m pip install Pillow
```

The exercise this week is independent of the current web server code base. In the next exercise we will incorporate this week's functionality into the server.

Please read the entire assignment sheet before starting to program!

Problem 6.1 (Basic Image Manipulation)

40pt

In this exercise we will explore Pillow's image manipulation capabilities. Create a new Python file ImageManip.py and import the Image and ImageOps modules like this:

```
from PIL import Image, ImageOps
```

Write a Python function transformImage, which takes as arguments an image and a string. The string describes, which transformation should be applied to the image. For example, if the value of the passed string is "gray", your function should convert the image to grayscale and return the resulting image.

You find a complete list of Pillow's image manipulation functions here: `https://pillow.readthedocs.io/en/stable/reference/ImageOps.html`. Your function should at least support five of them.

You can freely choose the string value you want to assign each operation. For example, if you want to support the grayscale operation, you can choose whether the expected string is supposed to be "gray" or "grayscale" or something else, as long as it is sensible.

If the passed string does not match any operation, just return the original image.

Outside the function, load an image from your hard drive using Pillow's Image.open function. You may use one of the images in the Kirmes repository or use one of your own images.

Test your transformImage function by passing the image, along with some strings specifying the image operation. Display the transformed image using Pillow's show functionality.

Refer to the course notes for examples of the open and show methods.

Problem 6.2 (Watermarking Images)

60pt

In this exercise we will add functionality to apply a watermark to an image. We provide a watermark image (Watermark.png) together with this assignment (StudOn and Kirmes repository), but feel free to create one yourself.

Create a new Python function applyWatermarkToImage, which takes an image as argument. In the function, load the watermark image from your hard drive. Then use Pillow's alpha_composite function to overlay the watermark on top of the input image: `https://`

pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.alpha_composite

Note that there are two versions of alpha_composite in Pillow. The one we are using here directly modifies the original image and does not return a new one.

Hint: alpha_composite requires that both images have an alpha channel. The watermark image already has one, but you have to make sure that the input image also does.

Therefore, at the start of your function, convert the input image to RGBA. For this use the convert function[3] and pass it the string "RGBA". Then apply the alpha compositing to this converted image.

At the end of your function, convert the watermarked image back to RGB (analogous to above) and return the result.

Test your function and show the watermarked image! You can also use the save function to write the image to your hard drive:

```
im.save(''filename.jpg'', ''JPEG'')
```

Optional for the highly motivated: Check out the following tutorial, if you want to write arbitrary text as watermark: https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html#example-draw-partial-opacity-text Note: When they load a font (fnt = ImageFont.truetype(...)), just pass "arial.ttf" as argument (or another font which is installed on your PC).

Test all your code and enjoy!

[3] https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.convert

Assignment 7 (Image Database Part 5 - Better Thumbnails) – Given 14. 6. 2019, Due 23. 6. 2019

Note: Please read the entire assignment sheet before starting to program!

Problem 7.1 (Putting Thumbnails in Database) 100pt

Our image database and front-end are taking shape. On the home page we currently show an overview of all entries including thumbnails.

These thumbnails are small (200 pixels wide), yet we always load the full size image from the database. This is not particularly efficient, since all these (potentially very large) images need to be transferred to the client. We will try to fix this in this exercise.

We provide two new Python files with this exercise (ImageManip.py and ImageHelper.py). The first provides some basic image manipulation techniques (from last week). The latter provides functionality to create Pillow images from binary data (and vice versa) or to load Pillow images from a URL.

Familiarize yourself with the two files. You do not need to understand everything in the Python code, but make sure that you read the comments and that you understand what kind of functionality is given.

Now perform the following tasks:

1. In the BuildDB.py script, import the two provided files and Pillow:

   ```
   import ImageHelper
   import ImageManip
   from PIL import Image
   ```

2. In the BuildDB.py script add one more column to the database called Thumbnail of type BLOB. This will store our thumbnail.

3. Adapt the addImage function, such that it creates a Pillow image from the imageData variable (look in the ImageHelper file for a function you can use for this task). Create the thumbnail image (see file ImageManip). Then convert the image back to a binary blob and store it in the Thumbnail field of our database.

   See the comments in the BuildDB.py file for more details.

4. In the Server.py script add a new route /thumbnail/<id:int>. This should be exactly the same as the /imageraw/<id:int> route (which already exists), with one exception: It should return the Thumbnail instead of the Content field.

5. Lastly, in the Index.tpl make sure, that your new /thumbnail route is used instead of the /imageraw. On the details page the original sized image should stay of course.

Test all your code and enjoy!

Assignment 8 (Image Database Part 6 - Displaying Annotations) – Given 28. 6. 2019, Due 7. 7. 2019

Note: Please read the entire assignment sheet before starting to program!

Problem 8.1 (Displaying Annotations) 100pt

In this exercise we will finally give our database frontend the ability sto display annotations on top of our images. For now, these annotations come from files already provided in the Kirmes repository in the xml/ subfolder. Each of the files in this directory describes areas (rectangles) in a given image, along with a description text.

We have prepared the parsing of these files for you, so you don't need to change anything in the BuildDB.py script. Nevertheless, check the table creation near the end of the file (from line 246). In addition to the Images table we worked with for the last couple of weeks, we now have a second table in our database, called Annotations. This table stores the following information:

1. Id: The id of the annotation (analogous to the Id field in the Images table).
2. ImageId: The id of the annotated image.
3. Description: A text describing the annotations.
4. X, Y, Width, Height: The position and dimensions of the rectangle in the image.

The ImageId is a foreign key, which references the primary key Id attribute of the Images table. For example, an annotation entry with ImageId=27 defines an annotation for the image entry with Id=27. Note, that multiple annotations might reference the same image.

You don't need to do anything in this file, but make sure that you run it, so that your database is filled with the annotation data. Double check in the DB Browser, that the Annotation table is properly created and filled.

Now our frontend just needs to display the annotation information. To this end, amend the /details/ route in the Server.py script, such that for the given image id, it queries the database for annotations.

In the Details.tpl file, iterate over the annotations (if any exist), and create a <rect> and a <**text**> for each. Fill in the information from the annotation (position and size of the rectangle, description for the text). See the course notes for details, if you are unsure how this works.

Check if everything works as expected by visiting the /details/ page for an image, which has annotations. Not too many images actually have annotations, but some do. For example the image with id 146 should have a couple.

Make sure that by hovering the mouse over an annotation region, the rectangle highlights (gets brighter) and the description text is shown.

Test all your code and enjoy!

Assignment 9 (Image Database Part 7 - Adding Deleting and Editing Annotations) – Given 5. 7. 2019, Due 14. 7. 2019

Note: Please read the entire assignment sheet before starting to program!

In this exercise we will give the user the ability to edit annotations directly in the browser. The idea is that changing the values of an annotation (position, size, text) is always easier in a graphical user interface than by typing in the values in an XML file.

The process requires two parts. First the user must be able to interactively change the values in the browser. Second, the changes they made must be saved back to the database.

In order to ensure a pleasant user experience the first part should be performed directly in the browser, so that not every mouse click must be sent to the server and back. Since this requires JavaScript, we have provided this part for you.

Run your server and visit a details page of any image, which has annotations, e.g. `http://localhost:8080/details/146`. At the bottom you should see a checkbox Edit Annotations. If this is checked, you should see a list of all annotations.

The currently selected element in this list is editable. You can change the annotation description in the text box. You can change the position and size of the annotation rectangle by dragging the marked (red) rectangle in the image. Note that you can both move and resize the rectangle.

New annotations can be added with the New Annotation button at the bottom and deleted by clicking the bin icon.

The changes you made are sent to the server, when the Save Changes button is clicked. Saving the changes in the database is for you to implement.

Right now clicking Save Changes should do nothing (even though the website displays a notification saying that the changes have been saved).

You can verify that saving is not working by making some changes. Then click Save Changes and refresh the page. All changes should be gone (because they are not stored in the database).

Problem 9.1 (Editing Annotations)

In the Server.py script you can find a new route /edit_annotations. Since this receives data (i.e. the changes you made to the annotations), it is marked as POST. 30pt

The function loops over a list of changes and gets the necessary data.

Implement the following: For each entry in the list of changes, issue the correct SQL command to update the values (hint: UPDATE ...). At the end of the function, commit your changes to the database (db.commit()).

Test your function! In the browser, edit one or multiple annotations and click Save Changes. Refresh the page. Your changes should still be there!

Problem 9.2 (Deleting Annotations)

Complete the /edit_annotations route by issuing a DELETE command for each entry passed to this function. Again, don't forget to commit your changes. 30pt

Test your code by deleting entries in the browser and refreshing the page!

Problem 9.3 (Adding Annotations)

Adding new annotations (/new_annotations) is slightly more complicated (but not much). Note that this function takes in the imageID as an argument. 40pt

In the loop, extract the individual fields from the annotation variable (similar to the way it's done in /edit_annotations). Since this is a **new** annotation, there is no annotationID this time.

Issue an INSERT command for each new annotation. Then get the id of the newly stored entry (cursor.lastrowid) and append this id to the newIds list. These new ids will be sent back to the client (browser) at the end of the function. This is already implemented.

Lastly, test your functionality! You should now be able to add new annotations in the browser, which will persist even if you refresh the page.

Test all your code and enjoy!