***Laboratory work #4***

**Topic: "Database design. Designing the user interface"**

**Purpose of the work:**

* *Introduction to basic database design techniques.*
* *Introduction to user interface design techniques*.

**Task:**

**Database Design**

1. Study additional material on the theory of database design (see the section "additional information") design the application database for storing processed data.
   1. The database should contain a table with information about processed files (minimum set of columns: file name, date of processing).
   2. The table of processed files should limit the main table with data by foreign key.
2. To write SQL- a script for creating the structure of the designed database (see the example SQL- script in the archive **pikpo4\_python.zip**).
3. Study the sample code from the archive **pikpo4\_python.zip**. Based on this example, implement the necessary **CRUD** (Create, Read, Update, Delete) operations for working with the database (see the link to the material on the basics SQL). **Upload the app code to GitHub.**
4. Check execution CRUD-operations on the test database (only SQLite).

**Designing the user interface**

During the laboratory work, it is necessary to describe the expected behavior of the developed system from the point of view of an external user, that is, to "design" external interactions of the future IC with the user without specifying its internal structure.

1. **Define the structure of the designed user interface** (visual design that is responsible for presenting information to the user; system functionality that includes a set of features for effective performance of professional activities; user interaction techniques with the system + additional system functionality), taking into account the task (lectures, recommendations Appendix 1, additional system functionality (example) see Appendix 3).
2. **Define user interface styles** (graphic (GUI, web- interface (WUI), object-oriented interface) based on the task (lectures, recommendations Appendix 1).
3. **Decide on the placement of user interface elements**(buttons, icons, drop-down lists, fields for writing text, etc.) (Lectures, recommendations Appendix 1)**.**
4. **Write Requirements for the user interface**(See Appendix 2 for an example).
5. **Develop the interaction of the developed program with the user**: a script (you can use a sequence diagram or interaction diagram), on-screen forms, a set of hints (enumeration), and so on.
6. **Get acquainted with the methodological material on the basic layout of web pages**.
7. Based on **item 1-5** develop (compare) basic html pages for your app, using CSS styles and HTML5 layout model (see the implementation example HTML5-pages in a folder **www**). **Upload the code html-pages on GitHub.**

**Write a report.** The report should contain:

* 1. ER- diagram of the designed database.
  2. Listing SQL- a script for creating a database structure.
  3. Listing of the main functions for working with the database (with explanations of what each function does).
  4. Screenshot of part of the content of the main data table and related tables.
  5. Screenshots of the main ones html-pages with explanations and marking of the basic layout. HTML5-tags (similar to Figure 2.4).

**Fix it separately:** organization of the program's dialog with the user; composition of the menu, submenu...; description of the function keys ' actions; all on-screen forms or protocol screen messages; messages issued to the user during the program session and answers given to them by the user; error messages; hints to the user, organization of "help"; description and preparation of input data; output printed forms for writing specifications in the laboratory work 7.

**Additional information:**

**pikpo4\_python.zip–**sample app code on **Python.**

**pikpo4\_java.zip–**sample app code on **Java (higher difficulty task).**

To launch the app, you need to install the following **Python-packages**:

pip install **PyMySQL**

pip install **pysqlite3** (optional)

**Language Basics SQL:**

[**https://htmlacademy.ru/blog/php/sql**](https://htmlacademy.ru/blog/php/sql)

**Additional information on database design:**

https://metanit.com/sql/tutorial/1.1.php – основы.

https://habr.com/ru/post/535588/ - detailed material from G. A. Makeev

**Tools for working with databases:**

[https://dbeaver.io/files/dbeaver-ce-latest-x86\_64-setup.exe**–DBeaverCommunity**](https://dbeaver.io/files/dbeaver-ce-latest-x86_64-setup.exe%E2%80%93DBeaverCommunity)universal client for working with various databases.

<https://metanit.com/sql/mysql/2.5.php> **–**information on working with foreign keys in SQL The database.

**Libraries for working with data files:**

https://pandas.pydata.org/docs/reference/frame.html (**Python Pandas**)

**Libraries for working with databases:**

https://pythonru.com/biblioteki/vvedenie-v-sqlite-python (**Python sqlite3**)

https://docs.python.org/3/library/sqlite3.html#sqlite3.Cursor

**The basics HTML5 layout options:**

https://developer.mozilla.org/ru/docs/Learn/HTML/Introduction\_to\_HTML/Document\_and\_website\_structure

https://html5book.ru/vyorstka-stranicy-sayta

**The basics CSS:**https://html5book.ru/css-css3

**Editing tools HTML and CSS:**

1. NotePad++
2. Atom
3. VSCode + the plugin HTML CSS Support

**Tools for working with style templates:**

1. Paint.Net: https://paintnet.ru/download/
2. Gimp: https://www.gimp.org/downloads/
3. Photoshop

***Report design requirements:***

The text execution method should be the same for all work. **Font**– **TimesNewRoman**, size 14, **line spacing**– 1,5, **field sizes**: left – 30 mm; right-10 mm, top-20 mm; bottom-20 mm. Abbreviations of words in the text are allowed only in general terms. **Paragraph indent (1,25)**it should be the same in all work. **Page numbering**the main text should be end-to-end. The page number is not indicated on the title page. The number itself is located at the bottom center of the page or on the right.

**Methodological recommendations**

**1. Working with the database SQLite**

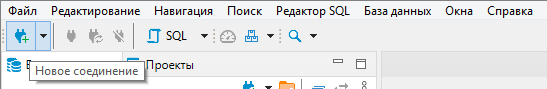
SQLite – compact embedded relational database. The main feature of this database is – this works without using the client-server interaction scheme. SQLite provides a library for working with a file as a relational database, which allows it to be used in embedded systems, as well as for local testing and debugging of the application.

However, despite a number of advantages, SQLite it also has some limitations that should be taken into account during the development process:

* Multiple processes or threads can simultaneously read data from the same database, but writing to the database is only possible if no other requests are currently being served.
* Dynamic data typing is used, i.e. if the field is declared as "**INTEGER**", SQLite allows you to enter any type of value in this column (999, "abc", "123", 678.525). If the inserted value is not an integer, SQLite will try to convert it to an integer (the string " 123 "will turn into an integer 123), and the remaining values will be written"as is".
* Unavailable**RIGHT and FULL OUTER JOIN**. Implemented only **LEFT OUTER JOIN**.
* Partially implemented features**ALTER TABLE** (available only **RENAME TABLE** and **ADD COLUMN)**.
* Partial support for triggers. Available only **FOR EACH ROW** triggers.
* Unable to record in **VIEWS**. **VIEWS** they are read-only.
* Due to the implementation of the database as a single file and a departure from the concept of "client-server", access rights management capabilities are not supported (**GRANT** and **REVOKE)**.
* By default, foreign keys are disabled.

To get started with the SQLite database, create an empty one txt-file and change the extension to **\*.db** or **\*.sqlite**.

Open the manager for working with the database (DBeaver). In the toolbar, select "New connection", then "Connection type".SQLite"(Figure 1.1):



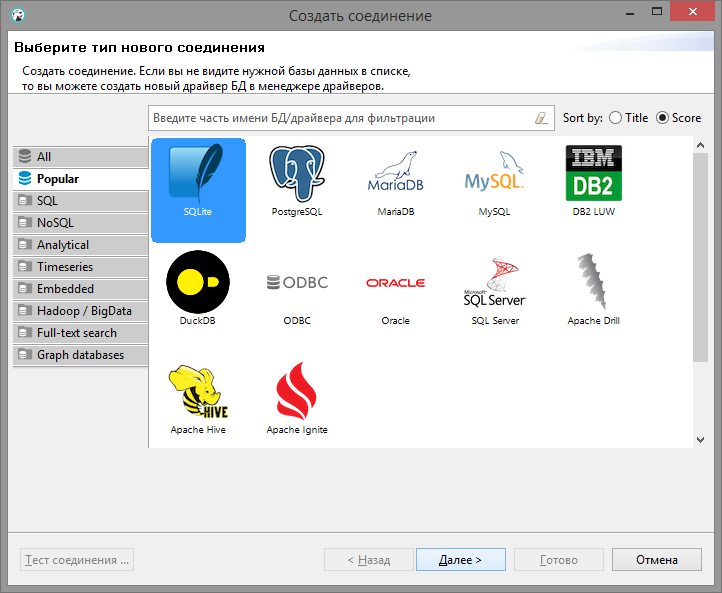


Figure 1.1-Creating a new DB connection

In the window that opens, click "Find" and select the created file (when connecting to a new type of database for the first time DBeaver automatically downloads the required driver):

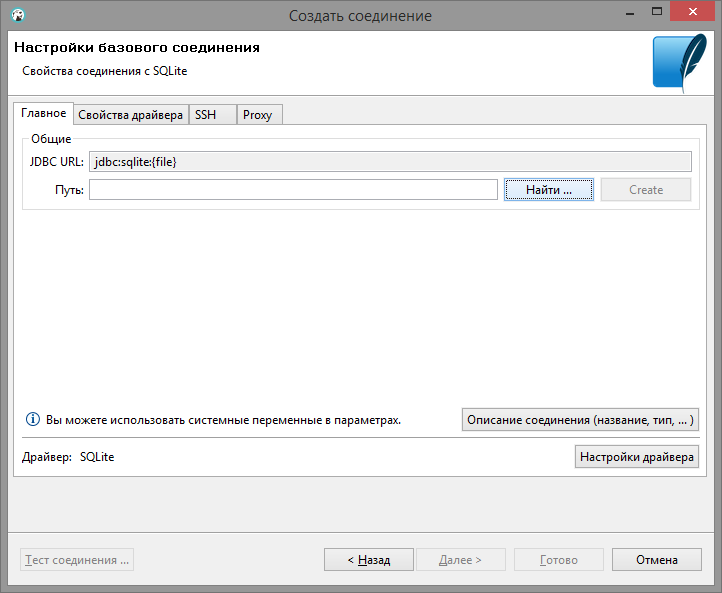


Figure 1.2-Selecting a database file

In the left window, select your database from the list and click on the toolbar "SQL". This opens the panel for creating an account SQL- the script (Figure 1.3):

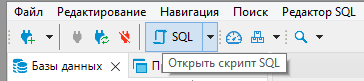


Figure 1.3-Creating an account SQL-scripts

This panel allows you to execute the entire script as a whole(Figure 1.4).:

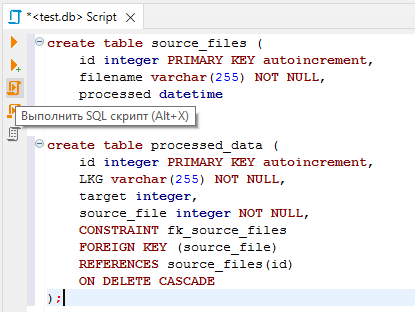


Figure 1.4-Launch SQL-scripts

So run the expression separately, previously selected with the mouse (Figure 1.5):

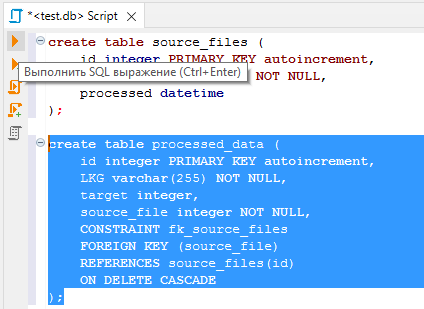


Figure 1.5-Launch SQL- expressions

After execution SQL- requests need to update the connection in the list so that the changes are correctly displayed in the list. DBeaver:

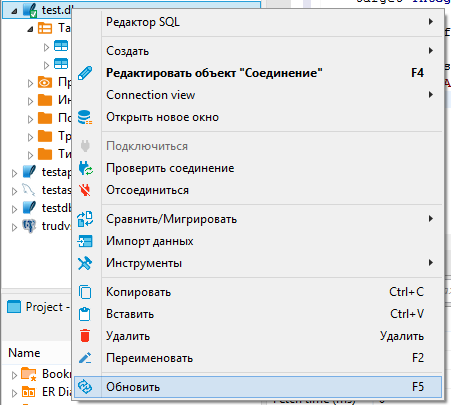


Figure 1.6-Connection update in the database selection panel

You can also use the left panel to view the structure of database tables and their contents (via **ViewData**, figure 1.7):

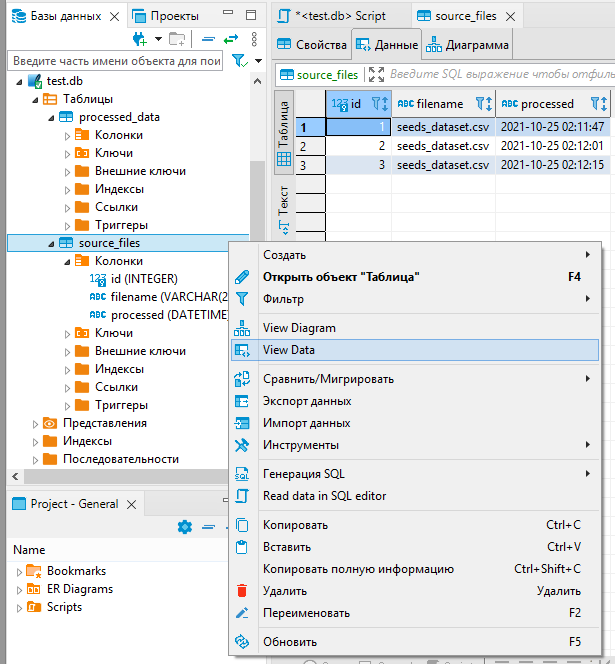


Figure 1.7-Viewing Table data

**2. Working with content resources html-in pages.**

**2.1. Preparing the working directory.** Before starting work, we create the site's file structure – an index page with separate folders for images and images. css- styles (Figure 2.1):

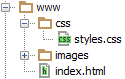


Figure 2.1-File structure in the working folder

**2.2. Design development.** After creating the file structure, select the style of content pages and work out the layout of elements based on **recommendations for developing the user interface (see Appendix 1)**.

Graphic elements for the page design can be taken from the template in the format PSD For example, here: **http://www.pcklab.com/templates**

File PSD – this is a saved bitmap graphic project of the program Photoshop saved without compression and containing additional information, such as: layer hierarchy, element transparency, etc.

One of the most popular freely distributed programs for working with this format is Gimp. Download this software and open it PSD-file. The structure of the project layers will be shown in the lower-right menu (Figure 2):

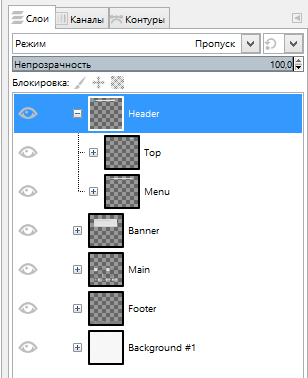


Рис. 2.2 – Дерево слоёв в Gimp

Figure 2.2-Layer tree in Gimp

Using this structure, we "cut" the necessary ones for implementation html- use graphic elements. The main ones are:

* Logos.
* Background images for the header block, main section, and basement (**Footer**If the background images have a repeating pattern (grid, lines, repeating pattern, etc.), then you need to "slice" the background so that it can be correctly combined when copying horizontally or vertically.
* Markers, menu icons, and social networks.

To save graphic elements, select the appropriate layer in the menu and press the keyboard shortcut Ctrl+C, then you need to open the program Paint.Net and copy the content via Ctrl+V or Shift+Insert. In Paint.Net click "Crop by selection" in the toolbar and save the data in the format PNG using the"File" menu🡪 "Save as..." (Figure 2.3).

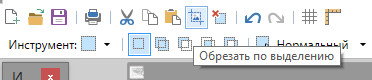


Figure 2.3-Cropping the canvas to fit the copied image

We save all the graphic elements in the folder **images**.

**2.3. Implementation HTML-code.**Base code for html-pages:

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>**This is the page title**</title>

<link rel="stylesheet" href="css/styles.css">

</head>

**<body>**

**Here you will see the content of the page**

**</body>**

</html>

Basic "framework" **HTML5** the document (fig. 2.4):

**<body>**

<header></header>

<nav></nav>

<main>

**Здесь будет представлен уникальный контент страницы**

**<aside>Опционально</aside>**

**<article>**

**</article>**

</main>

<footer></footer>

**</body>**

* **<main>**the element is intended for content***unique to this page***. The <main> block is only used ***one*** once on the page and placed inside **<body>**.
* **<header>** it represents a group of introductory content. If it is a child element of <body>, it defines the global title of the web page. It can include a logo and/or the name of a website or company. If <header If it is located as a child element of <article> or <section>, then it defines a specific title for that section (try not to confuse it with titles and headings).
* **<nav>** contains the main navigation functions for the page. Secondary links, etc. are not included in the navigation.
* **<article>** contains a block of linked content that contains the main semantic load of the page (for example, a table with data).
* **<section>**an auxiliary element that allows you to divide content in the <article> block into separate subsections. It usually contains a separate header as well. In these blocks, for example, you can separate a table with data and an additional description of the table, callouts, etc.
* **<aside>**this auxiliary element includes content that is not directly related to the main content, but may contain additional information that is indirectly related to it: dictionary, author's biography, related links, and so on.
* **<footer>** it represents a group of final content for a page, the so-called "basement".

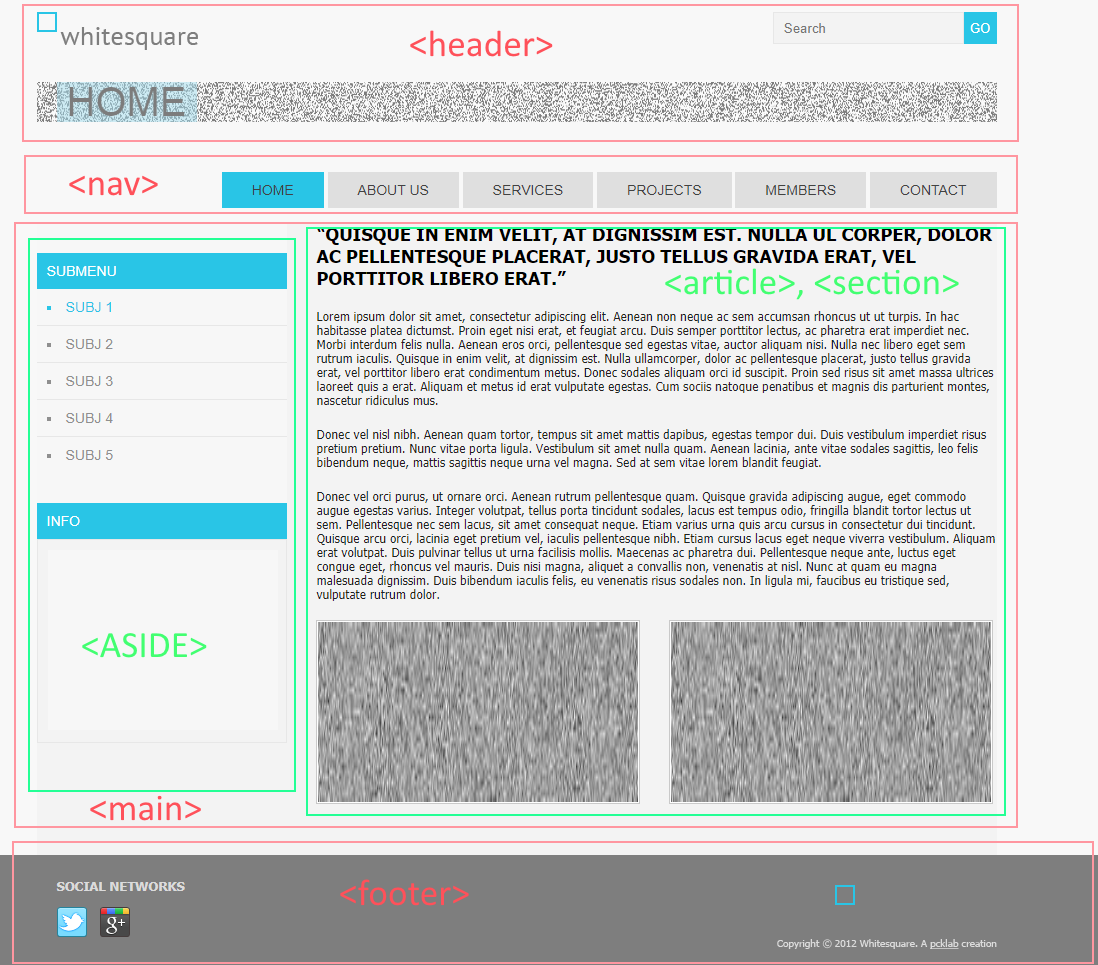


Figure 2.4-Marking up basic blocks on the page

The main condition of a well-laid-out page is the ability to move blocks to any location. As well as the ability to add blocks of the same type without breaking the general order of elements. For example, if you have a news block, then the news block can be supplemented with a block with a summary of the weather forecast, while the remaining blocks will keep their current location.

Appendix 1

**Recommendations for developing the user interface**

Decide which ones to use

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. general principles | 2. create interface elements (EI) | 3. what should be the EI design? | 4. where to place EI correctly | 5. how EIs should behave |

1. **General principles**

|  |  |  |
| --- | --- | --- |
| *Interface* must be *intuitive*(so that the user doesn't have to explain how to use it. | https://habrastorage.org/r/w1560/getpro/habr/post_images/982/62f/a7b/98262fa7b1ad088f48e7aae810da1742.png | |
| *Return the user* in *the place where he finished the job last time*. | | |
| *Often, users in the interface first search for an entity (noun), and then for an action (verb) to it*. Follow the rule "noun - > verb". | https://habrastorage.org/r/w1560/getpro/habr/post_images/9ae/d9d/3c4/9aed9d3c414f98b3c2164e6532c61198.png | |
| *Help is needed to simplify the learning process*. Literally - a graphical hint that explains the meaning of a particular EI. The complete guide should be part of the interface, accessible at any time. | | |
| *The sooner a person sees the result , the better*. An example is a "live" search, when options are available, while typing a search query. | | |
| Basic principle: *the program should interact with the user based on the smallest significant input unit*. | | |
| *Using quasi-modes*. For example, entering uppercase letters while holding down the key *shift* — this is a quasi-mode. With enabled *capslock* — mode. | https://habrastorage.org/r/w1560/getpro/habr/post_images/9b9/2ee/6b6/9b92ee6b67c2e6c873cd0a653f81240c.png  The main difference is that a person can forget what mode they are in, but in a quasi-mode (with the additional button pressed). using the button) this is much more difficult to do. | |
| *You should be careful to provide the user with the ability to set personal settings*. | | |
| *The more the user works with a particular task, the more they focus on it and the less they stop noticing the hints and messages displayed by the program*.  The more critical the task, the less likely it is that the user will notice warnings about certain potentially dangerous actions. | | https://habrastorage.org/r/w1560/getpro/habr/post_images/1d4/5c1/b66/1d45c1b66df6789c0dafe30c75c9d326.png |

1. **1. What EIs should I create?**

|  |  |
| --- | --- |
| Defining the task or set of tasks that the product is intended for. | |
| The simple must remain simple. No need to complicate the interfaces. Constantly think about how to make the interface easier and clearer. | |
| Users don't think twice about how the program works. All they see is the interface. Therefore, from the consumer's point of view, it is the interface that is the final product. | |
| The interface should be human-oriented, i.e. it should meet the needs of a person and take into account their weaknesses. You need to constantly think about what difficulties the user may face. | |
| Think about the behavior and habits of your users. Don't change well-known EIs to unexpected ones, but make new ones intuitive. | https://habrastorage.org/r/w1560/getpro/habr/post_images/222/0e5/946/2220e594653fd61397f816ad87f4abab.png |
| Design the interface based on the principle of the least possible number of actions on the part of the user. | |

1. **What should be the EI design?**

|  |  |
| --- | --- |
| **Color**. Colors are divided into warm (yellow, orange, red), cold (blue, green), and neutral (gray).  Usually, warm colors are used for EI. | https://habrastorage.org/r/w1560/getpro/habr/post_images/efc/e84/c71/efce84c71ec325914d31f0ccee8656b5.png  This is precisely related to the psychology of perception. It is worth noting that the opinion about the color is very subjective and can change even depending on the user's mood. |
| **Form.** In most cases, it is a rectangle with rounded corners. Or a circle. Again, the shape as well as the color is quite subjective. | |
| **The main EI (frequently used) should be highlighted**. For example, by size or color. | |
| **Icons in the program should be obvious**. If not, sign it. After all, in fact, instead of explaining, pictograms often require explanations for themselves. | |
| Try not to make the elements too small - they are very difficult to hit. | |

1. **How do I place the EI correctly on the screen?**

|  |  |
| --- | --- |
| There is a claim that visual appeal is based on proportions.  **The Golden Ratio principle**:  The bottom line is that  - the entire segment is related to most of it, just as a large part is related to a smaller one.  For example, the total width of the site(screen) is 900px, divide 900 by 1.62, get ~555px, this is the width of the content block. Now we subtract 555 from 900 and get 345px. This is the width of the smaller part. | https://habrastorage.org/r/w1560/getpro/habr/post_images/c4c/09d/e90/c4c09de901736423fadaaba945c9a692.png |
| EI follows **group by importance**. In other words, determine which ones are most important and which ones are less important. | |
| Usually, elements are placed in the following gradation: **left to right, top to bottom**. The most significant elements are located at the top left, while the least significant elements are located at the bottom right. This is due to the order in which the text is read. In the case of touch screens, the most important elements are located in the area of action of the thumbs. | |
| Must be considered **user's habits**. For example, if in Windows the close button is located in the upper-right corner, then the program should place a similar button in the same place. **the interface should have as many analogies as possible, with things known to the user**. | |
| The EI should be placed closer to the place where it is located. **most of the time the cursor is located** the user. So that he doesn't have to move the cursor, for example, from one end of the screen to the other. | |
| **Observe the proportions.**An interface element can be considered visible if it is either currently accessible to the human perception organs, or it was so recently perceived that it has not yet managed to get out of short-term memory. For the interface to work properly, only the necessary things should be visible - those that identify parts of the operating systems, and those that display the way the user can interact with the device. | |
| Make the margins between the EI equal or multiples of each other. | |

1. **How should EIs behave?**

|  |  |  |
| --- | --- | --- |
| *Users get used to it*. For example, when deleting a file, a confirmation window appears: "Yes" or"No". Over time, the user stops reading the warning and clicks "Yes"out of habit. | https://habrastorage.org/r/w1560/getpro/habr/post_images/df2/e62/189/df2e621890cddd6d7721990c5a08c9a8.png  Therefore, the dialog box, which was intended to provide security, absolutely does not fulfill its role. Therefore, it is necessary to give the user the ability to cancel the actions they have taken. | |
| If you give the user information that they need to enter somewhere or process in some way, then the information should remain on the screen until the person processes it. Otherwise, he might just forget. | | |
| **Avoid ambiguity.** For example, a flashlight has one button. By pressing the flashlight turns on, pressed again — turned off. If the light bulb in the flashlight has burned out, then when you click on the button, it is not clear whether we turn it on or not. | | https://habrastorage.org/r/w1560/getpro/habr/post_images/30f/103/169/30f10316904acddcf8850a523ef3bf4c.png  Therefore, instead of a single switch button, it is better to use a switch(for example, a checkbox with two positions: "on" and "off"). Except when the state of the task is obvious.  This switch directly reflects the state of the EI. |
| **Make monotonous interfaces** where the action can only be performed in one way. This approach will allow you to quickly get used to the program and automate your actions. | | |
| **Don't make adaptive interfaces that change over time**. Since it is better to learn only one interface rather than several to complete a task. An example is the home page of the Chrome browser. | | |
| **If there are delays** during program execution **unavoidable** or the action performed by the user is very significant. It is important that the interface provides the following information: **feedback that informs you about them**. For example, you can use the task progress bar (status bar). | | |
| **Active EIS must respond to**. If the user made a click, the EI must respond in some way so that the person understands that the click occurred. | | |