4МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных технологий

Кафедра информатики

**ТЕМА**

(курсовой проект по дисциплине «Основы программирования на языке Python»)

Выполнили:

студенты группы 4.205-1,

Ряховский Михаил

Исупова Сабрина

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*(подпись)*

Научный руководитель:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(степень, звание)*

Рябов Иван Юрьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)*

Работа защищена:

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Барнаул 2024

**СОДЕРЖАНИЕ**

[1. АКТУАЛЬНОСТЬ ПРОЕКТА 3](#_Toc7137)

[1.1. Цель проекта 3](#_Toc7262)

[1.2. Актуальность приложения 4](#_Toc23757)

[1.3. Технологический стек 5](#_Toc13432)

[2. ТРЕБОВАНИЯ К ПРИЛОЖЕНИЮ И ПОСТАНОВКА ЗАДАЧИ 6](#_Toc1662)

[2.1. Характеристики пользователя 6](#_Toc22919)

[2.2. Функциональные требования 6](#_Toc18575)

[2.3. Нефункциональные требования 6](#_Toc14529)

[2.4. Анализ проблемы и постановка задачи на разработку 6](#_Toc3899)

[3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЯ 7](#_Toc14798)

[3.1. Описание источников и структуры данных 7](#_Toc1660)

[3.2. Разработка приложения 7](#_Toc30701)

[3.3. Обзор полученного пользовательского интерфейса 18](#_Toc10014)

[4. БИБЛИОГРАФИЧЕСКИЙ СПИСОК 20](#_Toc12594)

[5. ПРИЛОЖЕНИЕ 22](#_Toc9583)

## АКТУАЛЬНОСТЬ ПРОЕКТА

### Цель проекта

Традиционный метод ввода китайского иероглифа – pinyin. Он представляет собой ввод с помощью латинской транслитерации с последующим выбором нужного китайского иероглифа, соответствующего запросу. Так же традиционным методом ввода является ввод китайского иероглифа с помощью его отрисовки на экране смартфона. Проблемой ввода китайского иероглифа с помощью pinyin является то, что при незнании латинской транслитерации ввод иероглифа невозможен, даже если ты знаешь его значение и написание. Проблемой ввода китайского иероглифа с помощью отрисовки на экране смартфона заключается с том, что это практически невозможно для людей, не являющихся носителями, так как метод очень сложный. По этим причинам был создан новый метод ввода китайских иероглифов – графический, с использованием отдельно выделенных частиц иероглифов – графем. Было выделено около 350 графем, различных по внешнему признаку, которые, в свою очередь, позволяют, как пазл, собрать китайский иероглиф. Так же был введен определенный способ фильтрации, позволяющий ускорить ввод китайского иероглифа: при выборе одной графемы, остальные графемы, которые с ней не используются, отсекаются, и остаются лишь те, которые могут быть введены с данной графемой. С помощью такой фильтрации мы сокращаем время написания китайского иероглифа и делаем сам процесс более комфортным для использования.

Целью проекта является создание локального веб портала для эксперимента с новым методом ввода иероглифов, который позволит увидеть статистику по времени ввода отдельного иероглифа.

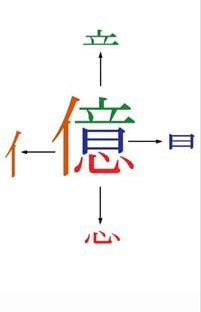


Рисунок 1 - Иероглиф и его графемы

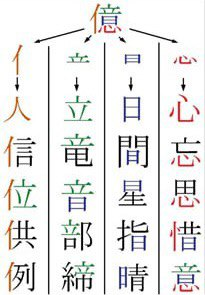


Рисунок 2 - Пример применения одинаковых графем в разных иероглифах

### Актуальность локального веб портала

Эта платформа позволит увидеть численную статистику по времени ввода отдельного иероглифа для каждой группы испытуемых и понять является ли метод эффективным.

### Технологический стек

1. Python
2. HTML
3. JavaScript
4. CSS

Библиотеки:

1. Flask
2. Random
3. Zipfile
4. Os
5. json

## ТРЕБОВАНИЯ К ПРИЛОЖЕНИЮ И ПОСТАНОВКА ЗАДАЧИ

### Характеристики пользователя

Пользователями будут являться группы испытуемых, способные вводить иероглифы графическим и традиционными(pinyin и ввод на экране смартфона) методами. Эксперимент будет проводиться с разными группами испытуемых с разным уровнем владения китайским языком:

Незнающие язык: не сталкивались с языком ранее, ничего о нем не знают

Начинающие изучение китайского языка: знают азы, небольшое количество иероглифов и основные правила и понятия китайского языка

Находящиеся на более продвинутым уровне: хорошо владеют языком, знают много различных иероглифов, способны ввести некоторое количество иероглиф традиционными методами

Носители китайского языка: вводят иероглифы быстрее остальных групп и способны полноценно использовать традиционные методы ввода

### Функциональные требования

Локальный веб портал (написанный на Python), позволяющий проводить эксперимент в ограниченных по времени условиях, направленный на выявление эффективности ввода китайского иероглифа с помощью графического ввода для разных групп испытуемых.

### Нефункциональные требования

Требования для проведения эксперемента:

1. Windows 10 и выше

### Анализ проблемы и постановка задачи на разработку

Оценка эффективности ввода китайского иероглифа графическим способом (через графемы) с количественными показателями на разных группах испытуемых с помощью данного локально веб портала.

## ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЯ

### Описание источников и структуры данных

Источники:

Flask Документация, Python Документация, CHATGPT – помощь в поиске информации, JS Документация, HTML Документация

Структура данных:

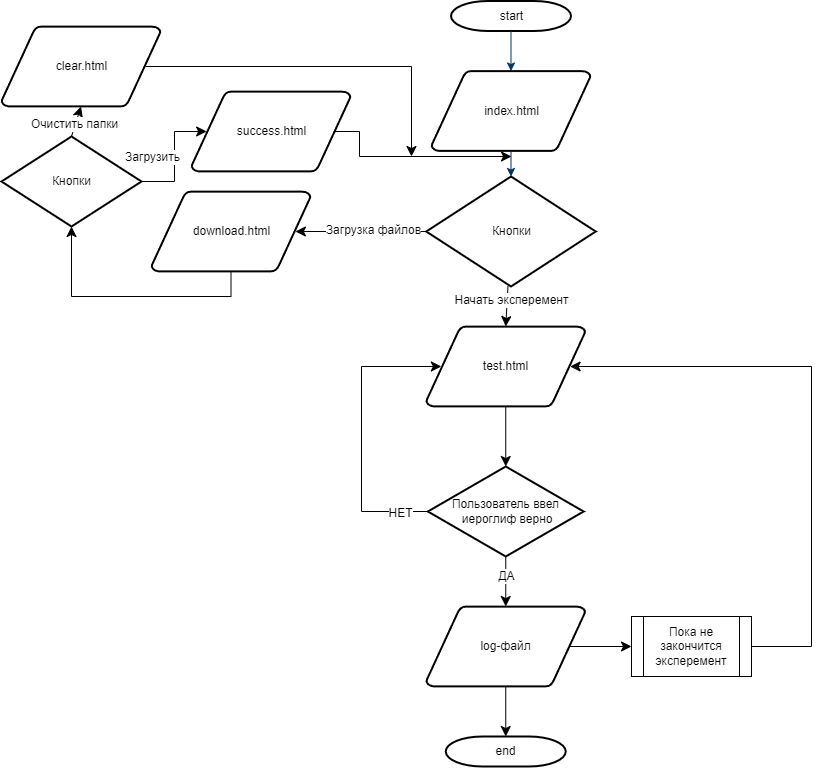


Рисунок 3 – Структура данных локального веб-портала

### 

### Разработка локального веб портала

Этапы разработки веб портала:

1. Разработка страницы для загрузки файлов
2. Разработка алгоритма для распаковки архивов в нужные директории
3. Алгоритм для считывания базы данных из текстового файла
4. Создание контейнера картинок графем на странице
5. Создание кнопок из картинок графем
6. Создание модуля отображения иероглифа на странице
7. Реализация алгоритма выбора и вывода графем
8. Алгоритм для поиска графем иероглифа и постановление их как задачи
9. Алгоритм сортировки графем в графемном поле
10. Создание секундомера для измерения времени эксперимента
11. Создание обработчика шаблона эксперимента
12. Алгоритм подтверждение иероглифа
13. Создание алгоритма для записи времени и иероглифов в лог файл
14. Упаковка в exe файл
15. Написание курсовой работы

Разработка:

Для того чтобы начать работу и тестирование нам нужно было сначала разработать страницу загрузки файлов. На ней нужно было разместить 4 кнопки выбора файла (две для txt формата, две для zip формата) и еще две обычные кнопки (подтверждение загрузки и удаление всех загруженных файлов). Кнопки для загрузки файлов реализованы так:

<form method="POST" enctype="multipart/form-data">  
 <p>Графемы ( Графемы.zip )</p>  
 <input type="file" name="graf" accept=".zip"> /// или .txt

…  
</form>

Остальные кнопки было решено сделать с переходом на другие страницы и отправкой запроса в основной код python:

<form action="http://127.0.0.1:5000/clear">  
 <button>Очистить папки</button>  
</form>

Далее для того чтоб работать с этими файлами нам нужно было создать алгоритм распаковывания zip-файлов. Для его создания мы использовали библиотеку zipfile. Распаковка файлов выглядит вот так:

with zipfile.ZipFile('uploads/Графемы.zip', 'r') as zip\_ref:  
 zip\_ref.extractall('static/graf/')

Но перед тем как распаковывать архив проверяется пустая ли папка, для того чтобы в папку не сохранялись дубликаты.  
После того как все файлы были загружены и распакованы, нам нужно было считать базу данных в код. Сначала мы думали сделать через sql, но после нескольких попыток поняли, что нам не нужна такая сложная система, ведь эксперимент будет локальный и создавать отдельную базу нет необходимости, поэтому было принято решение просто считывать базу в массив. Алгоритм считывания выглядит вот так:

db = [x.split(':') for x in database]  
for i in range(len(db)):  
 db[i][1] = db[i][1][:-1]  
 db[i][1] = db[i][1].split(',')  
 bd.append(db[i])

После обработки им файла, массив выглядит вот так:  
[…['4fed', ['001', '002', '003', '004', '003']], ['4fee', ['001', '005', '006', '007']],…].

На данном этапе нам нужно было уже реализовывать место, где будет отображаться вся информация, нам нужно было создать саму страницу эксперемента (test.html). По параметрам, которые дал заказчик, там должны были находиться: графемное поле, картинка иероглифа, кнопка подтверждения собранного иероглифа, секундомер и поле выбранных иероглифов. Мы решили начать с графемного поля. Для его реализации был выделен контейнер, который занимает 80% экрана:

.container1 {  
 …  
 padding: 1%;  
 flex: 1 1 80%;  
 height: 80vh;  
}

Так же был выделен блок под сами картинки графем (нужен для того что графемы располагались строго в контейнере):

.col-4 {  
 display: flex;  
 flex-wrap: wrap;  
 height: auto;  
 width: 100%;  
 align-items: flex-start;  
}

Далее был создан вывод графем из папки:

{% for image in folder1\_images %}  
 <button type="button" class="btn1" id="image-{{ image }}" onclick="selectImage('{{ image }}')">  
 <img src="{{ url\_for('static', filename='graf/' + image) }}" alt="{{ image }}" class="img-fluid">  
 </button>  
{% endfor %}

Здесь сразу было решено сделать из картинок кнопки, чтоб не создавать дополнительных задач. Теперь нам нужно было создать второй контейнер под всю остальную информацию.

.container2 {  
 …  
 padding: 1%;  
 flex: 1 1 20%;  
 height: 80vh;  
}

И так же сделать стиль для того чтобы вся информация находилась строго внутри container2:

.col-12 {  
 display: flex;  
 flex-wrap: nowrap;  
 gap: 2px;  
 width: 25px;  
 height: 25px;  
 align-items: center;  
}

В контейнере 2 были реализованы: секундомер, вывод иероглифа(который нужно собирать), вывод выбранных графем и кнопка подтвердить иероглиф.

Секундомер пока не являлся главной задачей, поэтому было решено реализовать его потом.

Вывод иероглифа был сделан по такой же логике, как и вывод графем (просто берется случайный из папки):

<img src="{{ url\_for('static', filename='ierog/' + folder2\_image) }}" alt="{{ folder2\_image }}" class="ier" id="ier-image">

Далее о реализации выбора графем и их вывода:  
Сначала мы было принято решение хранить информацию о выборе исключительно на сервере, для упрощения работы программы, но несколько неудачных попыток дали нам понять, что хранить информацию только на сервере плохая идея (т.к. графемы то не отображались на сайте, то не записывались в python), поэтому было принято решение сделать так, было создано два массива(один в python, один в javascript).

selected\_grafs = [] #python

let selected\_grafs = {{ selected\_grafs|tojson|safe }}; #js

Далее они были напрямую связаны через AJAX запрос. То есть, при выборе графемы, она сначала добавлялась в массив python, а после он просто копируется в массив в js.

@app.route('/update\_selected\_grafs', methods=['POST'])  
def update\_selected\_grafs():  
 selected\_grafs = request.form['selected\_grafs']  
 selected\_grafs = json.loads(selected\_grafs)  
 return jsonify({'selected\_grafs': selected\_grafs})

$.ajax({  
 url: '/update\_selected\_grafs',  
 type: 'POST',  
 data: {selected\_grafs: JSON.stringify(selected\_grafs)},  
 success: function(response) {  
 var list = $('#selected-grafems');  
 list.empty();  
 for (var i = 0; i < selected\_grafs.length; i++) {  
 list.append('<img src="/static/graf/' + selected\_grafs[i] + '" alt="' + selected\_grafs[i] + '" class="img-fluid" onclick="selectImage(\'' + selected\_grafs[i] + '\', true)"style="width: 100%;height: auto;display: flex;">');  
 }  
 }  
});

Далее мы приступили к самой реализации выбора графем. Был добавлен ajax привязанный к python через адрес “/select\_image”. Он передает название нажатой кнопки в python и записывает ее в массив. Далее страница обновляется, и нажатая графема отображается в блоке выбранных графем. Так же все сохраняется в локальное хранилище браузера, чтобы при обновлении страницы вся информация не стиралась.   
Теперь нам нужно сделать так, чтобы графемы можно было удалить.  
Была реализована функция selectImage в js и так же к каждой грамефе присвоено значение remove. То есть у каждой графемы в графемном поле это значение равно false, а у каждой графемы в поле выбранных графем оно равно true. Это означает, что при нажатии на графему в графемном поле она может только добавить, а в поле выбранных графем она может только удалиться.

function selectImage(image\_name, remove) {  
 if (remove) {  
 selected\_grafs = selected\_grafs.filter(function(value) {  
 return value !== image\_name;  
 });  
 } else {  
 // Add the image to the array  
 selected\_grafs.push(image\_name);  
 }

Далее была создана передача этого параметра в python:

$.ajax({ #js  
 url: '/set\_remove',  
 type: 'POST',  
 data: {  
 remove: remove  
 },  
 success: function(response) {  
  
 }  
});

@app.route('/set\_remove', methods=['POST']) #python  
def set\_remove():  
 global isremove  
 remove = request.form.get('remove')  
 isremove = remove  
 return jsonify({'status': 'ok'})

Далее нам нужно придумать, как сравнивать выбранные графемы с графемами иероглифа, который нужно собрать.

Мы создали алгоритм, который берет название выведенного иероглифа, пробегает по массиву базы данных и ищет внутри него массив, где первый элемент совпадает с этим названием, далее берет вторую часть массива с графемами и сохраняет в отдельный массив zadach.

def foundindex(a):  
 for i in range(len(bd)):  
 if bd[i][0] == a[:-4]:  
 zadach[0] = bd[i][0]  
 zadach[1] = bd[i][1]  
 for j in range(0,len(zadach[1])):  
 zadach[1][j] += '.png'

Теперь выбранные графемы сравнивается с этим массивом.

Для удобства выбора графем, мы приняли решение сортировать их в графемном поле по логике: выбранные графемы должны сочетаться со всеми графемами из графемного поля. То есть если у нас выбрана графема 001 и база иероглифов выглядит так:

4ffa:020,021

4ff1:001,010,011  
4ff5:001,013,014

4ff8:018,019

4e0d:022

В графемном поле будут выведены только графемы 001,010, 011,013,014.

Это было реализовано следующим образом: создана функция sortir, которая считывает основную базу данных и если находит совпадения в массиве графем иероглифа, то добавляет все графемы этого иероглифа.

for i in range(0, len(bd) - 1):  
 if all(elem in bd[i][1] for elem in b):  
 for elem in bd[i][1]:  
 if (elem + '.png') not in dostup:  
 dostup.append(elem + ".png")

Далее если массив dustup не пустой, он сортируется и передается на страницу для отображения, если пустой (такое может быть, если нет ни одной выбранной графемы), то выводятся все графемы:

if len(a) > 0:  
 return sorted(dostup)  
else:  
 return [f for f in os.listdir('static/graf') if os.path.isfile(os.path.join('static/graf', f))]

Так же по просьбе заказчика был создан алгоритм в этой же функции, который ограничивает ввод одной и той же графемы, для того чтоб она больше не отображалась в основном поле:

for j in range(0,len(a)-1):  
 if a.count(a[j]) >= 9:  
 clearr.append(a[j])

Теперь нужно было реализовать секундомер. Для его реализации мы использовали библиотеку time в python. Было добавлено четыре переменные в самое начало кода (для старта секундомера, обозначение идет или не идет, для записи начального времени, для форматирования времени):

start\_time = 0  
running = False  
timeofstart = time.localtime(time.time())  
formtime = time.strftime("%H.%M.%S", timeofstart)

Так же были созданы переменные в js для сохранения данных:

const startTimeElement = document.getElementById("time");  
const startBtn = document.getElementById("start-btn");  
const stopBtn = document.getElementById("stop-btn");

И функция начала секундомера при первой загрузке:

let count = 0;  
if (count === 0) {  
 localStorage.clear();  
 fetch("/start", { method: "POST" });  
 count++;  
}

И обновление страницы каждую секунду:

setInterval(async () => {  
 const response = await fetch("/time", { method: "GET" });  
 const time = await response.text();  
 startTimeElement.innerText = time;  
}, 1000);

Отображение на странице реализовано следующим образом:

<p id="time">00:00:00</p>

Вся основная логика секундомера прописана в коде python: функция start\_timer вызывается при первом открытии страницы test, в ней один раз просто записывается время начала эксперемента и больше она не используется, далее реализовали функцию остановки секундомера, в ней просто переменной running присваивается значение false. Далее создана функция для подсчета времени прошедшего с начала эксперемента:

@app.route("/time", methods=["GET"])  
def get\_time():  
 global start\_time, running, elapsed\_time  
 if running:  
 elapsed\_time = time.time() - start\_time  
 return f"{int(elapsed\_time // 3600):02d}:{int((elapsed\_time % 3600) // 60):02d}:{int(elapsed\_time % 60):02d}"  
 else:  
 return "Эксперимент остановлен!"

В ней используются все прописанные ранее функции и далее она возвращает время отформатированное в нужном формате (Час:Минута:Секунда).

После выполнения этого этапа мы решили приступить к реализации загрузке иероглифа из шаблона. Для этого была создана функция randomier() и в нее был перемещен алгоритм случайного выбора иероглифа из папки, а так же добавлен алгоритм выбора из шаблона, который работает так: текст файла считывается в массив и каждый раз, когда иероглиф собран правильно берется следующий по индексу элемент.

if os.path.isfile('uploads/shablon.txt'):  
 shablon = open('uploads/shablon.txt','r')  
 if len(ierog\_files) == 0:  
 for line in shablon:  
 ierog\_files.append(str(line.strip())+'.bmp')  
 file = ierog\_files[filenum]  
 filenum += 1

Теперь нам осталось сделать так чтоб программа обновляла правильно введенный иероглиф, а при неправильно введенном выводила ошибку.

Для этого была написана функция submit\_selected\_grafems(), в которой при выполнении условия, что отсротированный массив zadach равен отсортированному массиву выбранных графем, добавляется в массив для подтверждения “1” (если неверно введен, то добавляется 0) и очищается список выбранных графем. Так же в этой функции в массив logtext записывается название иероглифа и время правильного ввода.

def submit\_selected\_grafems():  
 global elapsed\_time  
 if submitgrafems() == 1:  
 selected\_grafs.clear()  
 submitier.append("1")  
 logtext.append(str(zadach[0]) +':'+ str(int(elapsed\_time // 3600))+'.'+str(int((elapsed\_time % 3600) // 60))+'.'+str(int(elapsed\_time % 60))+'\n')  
 timetest.append(int(str(elapsed\_time).split('.')[0]))  
 return jsonify({'result': 'correct'})  
 else:  
 submitier.append("0")  
 return jsonify({'result': 'incorrect'})

Далее массив с единицами и нулям передается в главную функцию. Там уже иероглиф обновляется в соответствии с последним элементом массива единиц и нулей.

if submitier[-1] == "1":  
 submitier.append("0")  
 folder2\_image = randomier()  
 foundindex(folder2\_image)

Теперь нам нужно создать файл, в который будет записываться итоговый результат эксперемента. В функции подтверждения графем. Далее создана функция logfilewrite(). Она создает логфайл (или если он уже есть очищает его), записывает в него. Все из массива logtext, далее записывает среднее время ввода иероглифа и время начала эксперемента.

def logfilewrite(f):  
 logfile = open('logfile.txt', 'w')  
 logfile.truncate()  
 logfile.writelines(f)  
 if len(timetest) > 0:  
 logfile.writelines(averagetime(f))  
 logfile.writelines('Время проведения эксперимента - '+str(formtime))  
 logfile.close()

Следующим этапом для нас было сделать так, чтоб эту программу можно было запустить на любом компьютере с операционной системой windows.

Мы использовали множество возможных библиотек(docker, pyinstaller и тд), но вручную создать exe файл из нашего проекта не получалось, так как используется множество внешних файлов и библиотек. Нам пришлось искать решение этой проблемы в интернете. Мы нашли программу, написанную при помощи python и библиотеки pyinstaller, которая называется auto\_py\_to\_exe.

Она решила все наши проблемы с установкой нашей программы, успешное упаковав её в exe со всеми нашими характеристиками.

### Обзор полученного пользовательского интерфейса

На первой странице локального веб-портала, который мы можем увидеть на рисунке 4, находится стартовая страница, на ней мы можем увидеть две кнопки: «Начать эксперемент» и «Страница загрузки», с помощью которых можно перейти на следующие страницы локального веб-портала.

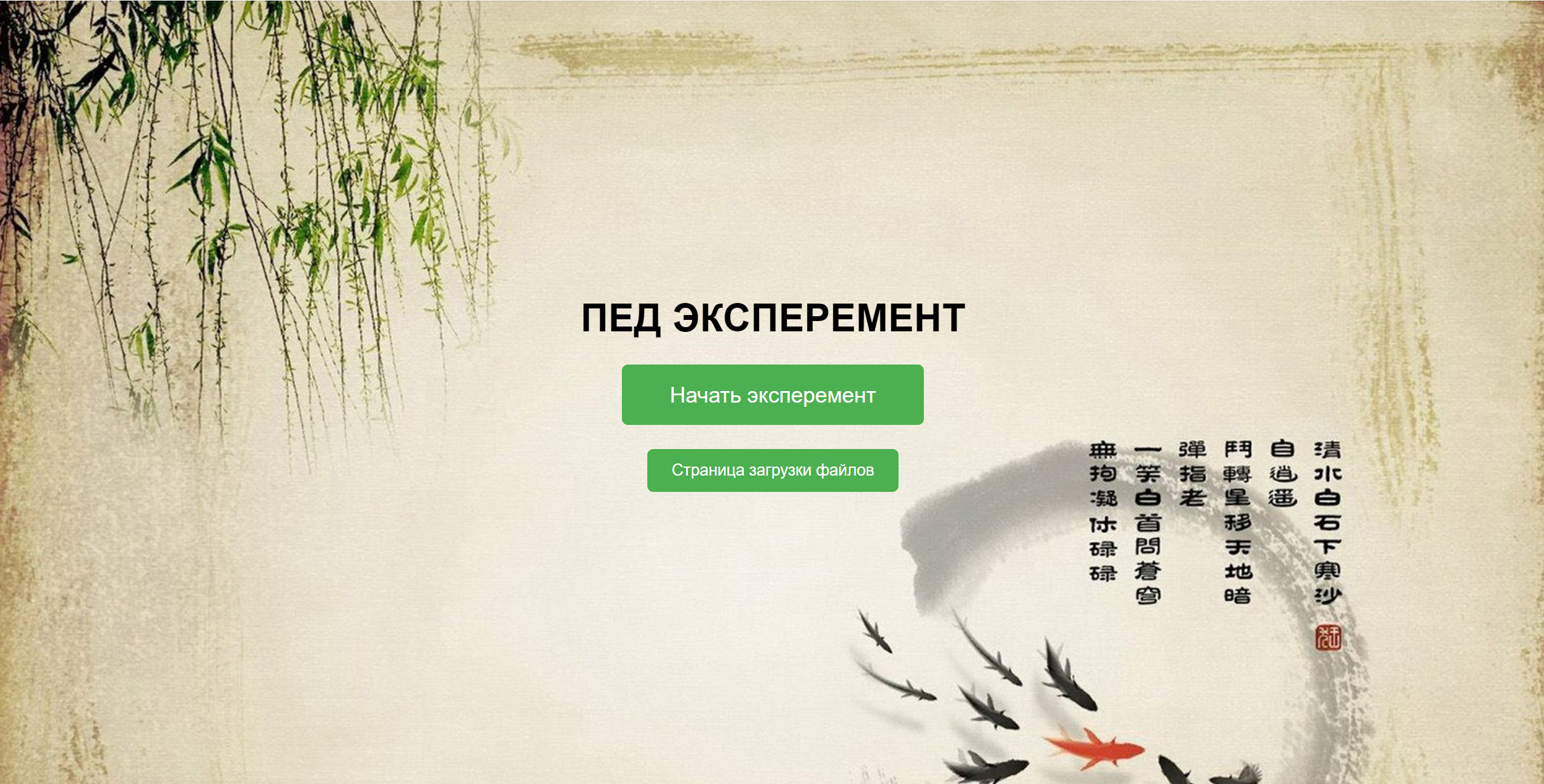


Рисунок 4 – стартовая страница локального веб-портала

На второй странице пользовательского интерфейса локального веб-портала, который мы можем увидеть на рисунке 5, осуществляется загрузка файлов, нужных для эксперемента. К ним относятся:

1. Графемы (Графемы zip)
2. Иероглифы (Иероглифы zip)
3. База графем и иероглифов (database.txt)
4. Шаблон (shablon.txt)

После загрузки файлов пользователь нажимает кнопку «Загрузить». Если при загрузке была допущена ошибка, то пользователю нужно нажать кнопку «Очистить папки» и добавить файлы заново.

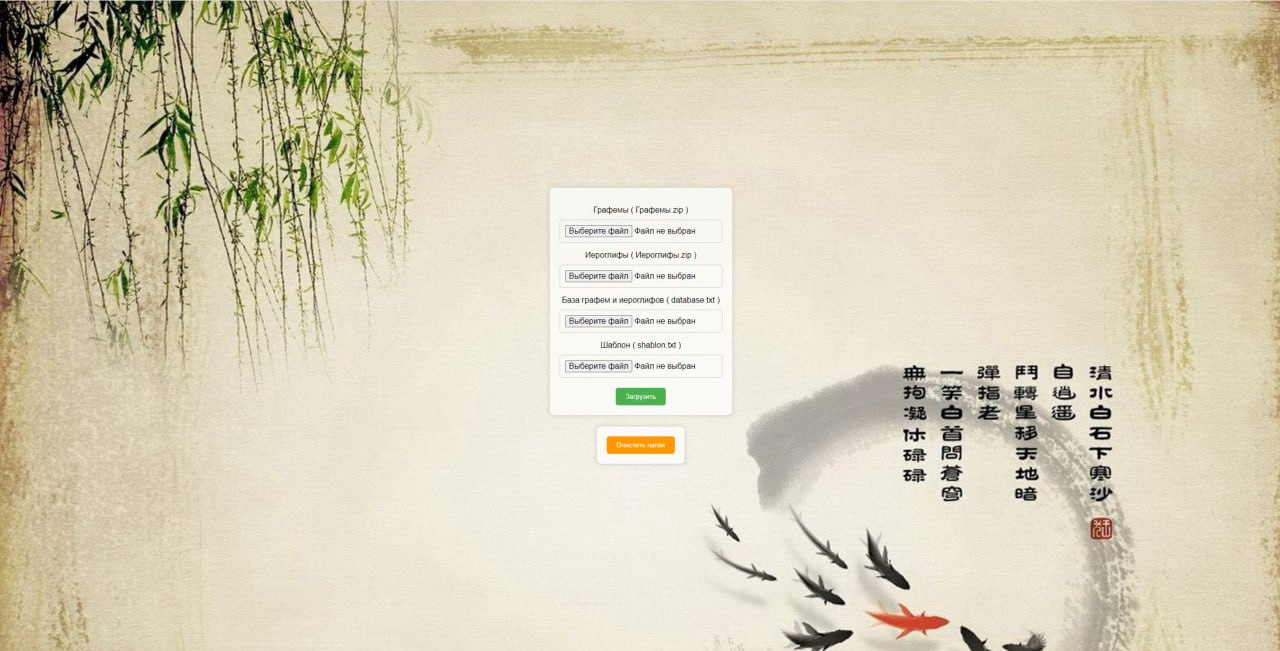


Рисунок 5 – Страница загрузки локального веб-портала

На второй странице локального веб-портала находится функционал, который нужен для проведения эксперемента:

1. Поле графем, которое занимает большую часть пространства и находится посередине данной страницы.
2. Секундомер, который позволяет узнать время введения предложенного иероглифа и находится в правом верхнем углу.
3. Поле с иероглифом, которое требуется ввести испытуемому. Поле находится под таймером и обозначено словом «Задание».
4. Поле выбранных графем, на котором отображаются графемы, которые нажал испытуемый, в ходе эксперемента. Находятся под полем с иероглифом.
5. Кнопки «Подтвердить иероглиф» и «Остановить эксперемент». Позволяют испытуемому или обозначить, что он ввел иероглиф и его нужно проверить на правильность написания, или остановить эксперемент. Находятся под полем выбранных графем.



Рисунок 6 – Основная страница локального веб-портала

## 

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Официальная документация Python – URL: https://www.python.org/doc/
2. Документация Java – URL:
3. <https://docs.oracle.com/en/java/>
4. Документация HTML - URL: <https://developer.mozilla.org/ru/docs/Web/HTML>
5. Документация Flask – URL:  
   <https://flask.palletsprojects.com/en/3.0.x/>
6. Документация auto\_py\_to\_exe – URL:

<https://pypi.org/project/auto-py-to-exe/>

## ПРИЛОЖЕНИЕ

https://github.com/sunflowerlmao/pedex1