МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**КУРСОВАЯ РАБОТА**

**МОДУЛЬ ГЕНЕРАЦИИ QR КОДА ПОСЕЩАЕМОСТИ СТУДЕНТА**

Работу выполнил К.С. Минин

(подпись)

Направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии

Курс 3

Направленность Вычислительные технологии

Научный руководитель

канд. пед. наук, доц. Н.Ю. Добровольская

(подпись, дата)

Нормоконтролер

ст. преп. А.В. Харченко

(подпись, дата)

Краснодар

2022

**РЕФЕРАТ**

В курсовой работе 31 страница, 14 рисунков, 9 источников.

Ключевые слова: ИДЕНТИФИКАЦИЯ, QR CODE, ДЕКОДИРОВАНИЕ, REST, FLASK, СИСТЕМА УЧЕТА ПОСЕЩАЕМОСТИ.

Цель курсовой работы – реализовать программный модуль, генерирующий QR код посещаемости студента.

При подготовке курсовой работы изучены основные принципы идентификации пользователей, структура и реализация UUID, алгоритм декодирования QR кода, а также архитектурный стиль REST.

В работе выполнен обзор основных методов библиотек uuid и flask в языке программирования Python.

В практической части курсовой работы реализовано приложение, генерирующее QR код студента по запросу клиента и отображающее его на локальном web сервисе.

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc103931094)

[1 Основные аспекты задачи идентификации 5](#_Toc103931095)

[1.1 Понятия идентификации и аутентификации 5](#_Toc103931096)

[1.2 Универсальный уникальный идентификатор 7](#_Toc103931097)

[1.3 Алгоритм декодирования QR кода 10](#_Toc103931098)

[2 Задача генерации QR кода посещаемости студента 15](#_Toc103931099)

[2.1 Архитектура REST 15](#_Toc103931100)

[2.2 Микрофреймворк Flask 18](#_Toc103931101)

[2.3 Структура используемого QR кода 22](#_Toc103931102)

[2.4 Генерация QR кода посещаемости студента 23](#_Toc103931103)

[3 Результаты работы приложения 27](#_Toc103931104)

[Заключение 30](#_Toc103931105)

[Список использованных источников 31](#_Toc103931106)

# ВВЕДЕНИЕ

Задача достоверной идентификации и авторизации пользователей в компьютерных системах не теряет своей актуальности. Человек каждый день проводит продолжительное время в сети интернет, пользуясь различными сервисами: социальными сетями, банковскими системами, корпоративными сетями предприятий. При входе пользователя в любую из перечисленных систем необходимо убеждаться в подлинности каждого субъекта.

Цифровые технологии предлагают различные методы идентификации субъекта: биометрическая идентификация, аутентификация по паролю, по ключевым словам. Однако такие формы идентификации позволяют только соотнести идентифицируемый субъект с некоторым кодом идентификации. Технология QR кодов позволяет кроме собственно соотнесения, хранить дополнительную информацию в коде идентификации.

Алгоритмы идентификации и аутентификации являются ядром программного учета посещаемости студентов в вузе. При помощи механизмов QR кодов возможно осуществить данные процессы, а также значительно упростить систему учета посещаемости.

В предлагаемой курсовой работе реализован второй этап исследования – модуль генерации QR кода для учета посещаемости студентом учебных занятий.

# 1 Основные аспекты задачи идентификации

Для более четкого понимания процесса работы модуля генерации QR кода посещаемости студента необходимо провести анализ основных определений, связанных с идентификацией пользователей.

## **Понятия идентификации и аутентификации**

При регистрации субъекта в некоторой информационной системе с ним ассоциируется некоторая информация, называемая идентификатором, которая позволяет однозначно определить его. В качестве идентификатора должна быть выбрана уникальная в пределах данной системы информация. Идентификатор может состоять из данных, непосредственно описывающих субъект: фамилия, имя и отчество; номер зачетной книжки; дата рождения. Также существуют различные методы, позволяющие генерировать уникальные или псевдоуникальные последовательности символов, однозначно определяющие субъект.

Для получения доступа к ресурсам системы пользователю необходимо пройти процесс первичного взаимодействия с системой, включающий в себя идентификацию, аутентификацию и авторизацию.

Идентификация – процедура распознавания пользователя по его идентификатору [1]. Этот процесс происходит в первую очередь при попытке входа субъекта в систему и заключается в получении идентификатора и проверкой его наличия в базе данных информационной сети.

Аутентификация – процедура проверки подлинности идентифицированного пользователя [1]. Данный процесс позволяет получить достоверную информацию о том, что субъект является именно тем, кем себя объявляет. Зачастую в простых системах аутентификация заключается в вводе пользователем пароля.

Авторизация – процесс предоставления пользователю прав на выполнение определенных действий в системе [1]. Очевидно, что авторизацию необходимо производить только после идентификации и аутентификации.

На рисунке 1 представлена наиболее распространенная схема предоставления пользователю доступа к некоторой компьютерной системе.

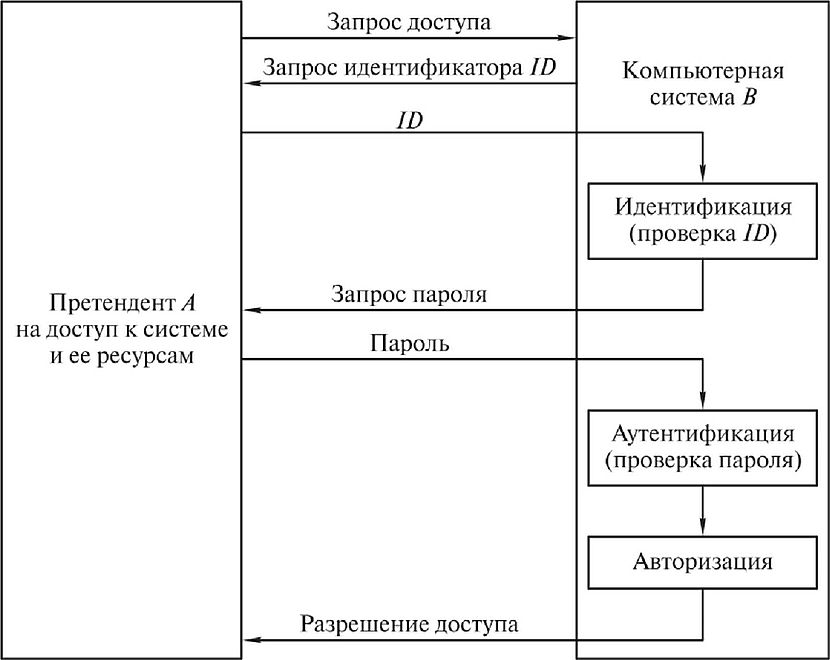


Рисунок 1 – Общая схема предоставления доступа к системе

В случае использования QR кода для отметок посещаемости студентами учебных практических занятий с точки зрения пользователей системы все перечисленные выше процессы сводятся к одному – показу студентом его личного QR кода и наведении преподавателем камеры смартфона на данный код. В качестве идентификатора студента можно использовать номер студенческого билета, который выдается ему при поступлении в университет. Идентификация будет производится программно путем поиска соответствующего идентификатора в базе данных приложения. В данном случае аутентификацией пользователя можно считать генерацию студентом через приложение его личного QR кода, так как вход в приложение осуществляется по логину и паролю, известному студенту. В авторизации пользователя нет необходимости, так как приложение должно просто отслеживать факт присутствия студента на занятии (факт сканирования его QR кода).

## **Универсальный уникальный идентификатор**

В целях обеспечения уникальности и безопасного использования генерируемого QR кода необходимо предусмотреть его динамическое изменение через определенные промежутки времени.

Для генерации псведоуникальных последовательностей, которые будут содержаться внутри символа QR кода и периодически обновляться, удобно использовать Универсальный уникальный идентификатор(UUID – universally unique identifier – «универсальный уникальный идентификатор»), основное назначение которого заключается в предоставлении распределенным системам возможности уникально идентифицировать информацию без центра координации. Общее количество уникальных ключей UUID (без учёта версий) составляет 2128 = 25616 или около 3,4 × 1038. Это означает, что, генерируя 1 триллион ключей каждую наносекунду, перебрать все возможные значения удастся лишь за 10 миллиардов лет. Таким образом разработчик любой системы может создать UUID, использовать его для идентификации чего-либо и быть достаточно уверенным в том, что данный идентификатор непреднамеренно никогда не будет использован для другого объекта в другой системе [2].

Классическое текстовое представление UUID – серия из 32 шестнадцатеричных символов (128 битное бинарное число), разделенных дефисами на пять групп по схеме «8-4-4-4-12» (Рисунок 2). Значения на позициях M и N определяют версию и вариант UUID соответственно.



Рисунок 2 – Общий вид UUID

Всего выделяется 5 версий UUID имеющих различные реализации и особенности, подходящие для определенных задач (Рисунок 3) [2].

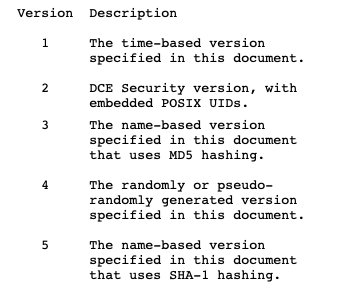


Рисунок 3 – Версии UUID

Версия 1 основывается на использовании 48 битного MAC-адреса узла, на котором происходит генерация UUID и 60 битной метки времени (число 100-наносекундных интервалов, прошедших с полуночи 15 октября 1582 года по UTC – даты начала применения григорианского календаря) [3]. Также временная метка дополняется 13 или 14 битным clock sequence в тех случаях, когда системные часы обновляются недостаточно быстро, или на многопроцессорных системах, вследствие чего метка времени у разных UUID может оказаться одинаковой. Поскольку метка времени и clock sequence вместе составляют 74 бита, всего может быть сгенерировано 274 = 1,8×1022 уникальных UUID версии 1 на одном узле.

Версия 2 похожа на версию 1, но младшие 8 бит clock sequence заменены на номер «локального домена» («local domain» number), а младшие 32 бита метки времени (timestamp) заменены целочисленным идентификатором, значимым в пределах указанного локального домена. С одной стороны, 40 битов домена допускают около 1 триллиона значений идентификаторов для одного узла. С другой стороны, с урезанным значением метки времени до 28 старших значащих бит по сравнению с 60 битами в версии 1 обновление UUID становится возможным только один раз в 429,49 секунды в отличие от 100 наносекунд в версии 1. Таким образом UUID версии 2 не подходит, если требуется генерировать UUID чаще чем раз в 7 минут.

UUID версий 3 и 5 образуются путём хэширования идентификатора пространства имён и имени. Версия 3 использует алгоритм хеширования MD5, версия 5 – SHA-1. Суть UUID версий 3 и 5 в том, что одна и та же пара из пространства имён и имени будет отображаться в один и тот же UUID. При этом ни пространство имён, ни имя не могут быть обратно получены из UUID, кроме как методом перебора.

UUID версии 4 генерируется случайным образом. Как и в других версиях UUID, 4 бита используются для указания версии, 2 или 3 бита указывают на вариант. Для варианта 1 на случайно сгенерированную часть приходится 122 бита, что даёт 2122, или 5,3×1036 возможных вариантов UUID версии 4 варианта 1. UUID версии 4 варианта 2 имеет вдвое меньше возможных вариантов, так как ещё один бит используется для обозначения варианта.

UUID имеет несколько вариантов, обозначаемых одним, двумя или тремя битами (Рисунок 4).

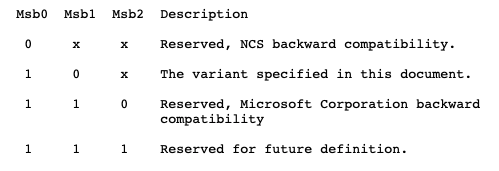


Рисунок 4 – Варианты UUID

UUID имеет несколько вариантов, обозначаемых одним, двумя или тремя битами.

Вариант 0 присутствует для обеспечения обратной совместимости с устаревшим форматом UUID Apollo Network Computing System 1.5. Обозначение варианта 0 как 0xxx позволяет избежать конфликтов с историческими UUID NCS, если они всё ещё существуют в базах данных.

Варианты 1 и 2 используются в текущих спецификациях UUID. Вариант 1 (обозначается двумя битами 10xx) является основным. Вариант 2 (обозначается тремя битами 110x) описан в стандарте как зарезервированный для обратной совместимости с ранними GUID из Microsoft Windows [3]. В каноническом текстовом представлении варианты 1 и 2 одинаковы, за исключением битов вариантов.

В RFC 4122 вариант 3 (111x) зарезервирован для использования в будущем [3].

## **Алгоритм декодирования QR кода**

В ГОСТ Р ИСО/МЭК 18004-2015 определен рекомендованный алгоритм, с помощью которого происходит обнаружение символов QR кодов и их декодирование [4].

В первую очередь происходит определение положения шаблонов поиска на изображении (Рисунок 5). Каждый модуль определяется последовательностью темный-светлый-темный-светлый-темный, относительные значения ширины каждого элемента которой находятся в соотношении 1:1:3:1:1.

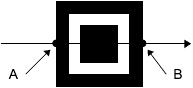


Рисунок 5 – Линия сканирования в шаблоне поиска

Если не было найдено ни одной линии, которая удовлетворяла бы такому условию, происходит изменение пикселей на инвертированные светлые и темные.

После нахождения всех трёх шаблонов поиска определяется угловая ориентация символа посредством анализа координат центров найденных шаблонов поиска для уточнения, какой из шаблонов находится в левом верхнем углу символа, и таким образом определяется угол поворота символа.

Далее происходит вычисление версии символа QR кода. Для этого определяются величины D, W­UL и WUR - расстояние на линии, пересекающей символ и проходящей через центры левого верхнего и правого верхнего шаблонов и ширину двух этих шаблонов соответственно (Рисунок 6).

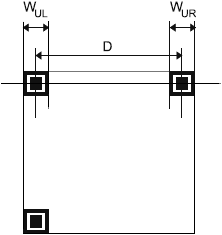


Рисунок 6 – Линия сканирования в шаблоне поиска

Номинальный размер символа вычисляется по формуле:

X = (W­UL + WUR)/14

Приблизительная версия символа вычисляется по формуле:

V = [(D/X)-10]/4

Если приблизительная версия символа 6 или менее, ее принимают в качестве окончательной версии. Если приблизительная версия 7 или более, декодируют информацию о версии согласно рисунку 7.

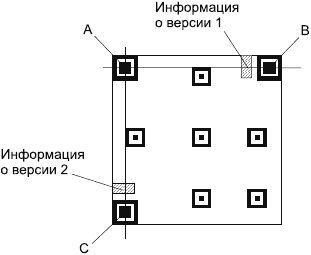


Рисунок 7 – Шаблоны поиска и информация о версии

После определения версии символа QR кода определяют координаты центров всех направляющих шаблонов, а затем происходит формирование сетки выборки пикселей для сканирования, которые проведены на равном удалении от этих точек (Рисунки 8-9).

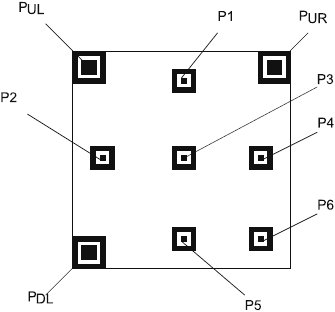


Рисунок 8 – Шаблоны поиска и направляющие шаблоны

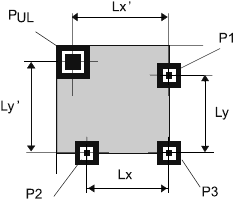


Рисунок 9 – Верхняя левая область символа

Аналогичным образом определяются области сканирования вокруг других двух шаблонов поиска, а также во всех неохваченных областях символа.

На следующем этапе просматривают области изображения размером 33 пикселей с центрами на пересечениях линий сетки выборки и определяют, является этот модуль светлым или темным на основании глобального порога. Формируют битовую матрицу, присваивая двоичную 1 темным модулям и двоичный 0 светлым модулям. Также декодируют информацию о формате, смежную с верхним левым направляющим шаблоном для получения уровня исправления ошибок и указателя шаблона маски, примененной к символу. Если число обнаруженных ошибок превышает способность их исправления, повторяют эту процедуру для декодирования информации о формате, смежную с верхним правым и нижним левым шаблонами поиска.

Если невозможно получить из двоичной строки надлежащую информацию о формате, предпринимают попытки определения, является ли правильной последовательность битов, если их прочитать в обратном порядке. В случае успеха продолжают декодирование изображения зеркально отображенного символа, меняя местами координаты строк и столбцов.

На заключительном этапе декодирования происходит выполнение процедуры маскирования, путем применения операции XOR шаблона маски данных к области кодирования символа, и восстановление знаков символа, представляющих данные и кодовые слова исправления ошибок. Эта процедура является обратной процессу маскирования данных, применяемому в процессе кодирования. Также происходит определение кодовых слов символа согласно правилам их размещения.

После этого перегруппировывают кодовые слова по блокам, как требуется для данной версии символа и уровня исправления ошибок и проводят декодирующую процедуру обнаружения и исправления ошибок, чтобы исправить такое число ошибок и стираний, которое максимально возможно для данной версии символа и уровня исправления ошибок. Затем происходит восстановление первоначального двоичного потока сообщения, посредством сбора блоков данных в последовательность и декодируют каждый сегмент из битового формата в соответствии с указанным режимом.

# 2 Задача генерации QR кода посещаемости студента

На данном этапе исследования задача генерации QR кода посещаемости студента может быть разделена на следующие этапы: получение информации о студенте; генерация временного UUID для QR кода; отображение полученной информации на символе QR кода. Использование разработанного по такой схеме модуля возможно внутри программного продукта, который по результатам сканирования символа QR кода и получения из него информации вносит информацию о посещении данным студентом конкретного занятия в университете.

## **2.1 Архитектура REST**

REST (англ. Representational State Transfer) является архитектурным стилем для обеспечения стандартов между компьютерными системами в сети, что облегчает для систем обмен данными друг с другом [5]. Системы, отвечающие требованиям REST и часто называемые RESTful, характеризуются тем, что не имеют сохранения состояния и разделяют интересы клиента и сервера.

В архитектурном стиле REST реализация клиента и сервера могут быть выполнены независимо друг от друга. Это означает, что код на стороне клиента может быть изменён в любое время без ущерба для работы сервера, также и код на стороне сервера может быть изменён без влияния на работу клиента.

До тех пор, пока каждая сторона знает, какой формат сообщений следует направлять другой стороне, они могут храниться модульно и раздельно. Посредством отделения задач пользовательского интерфейса от задач хранения данных достигается повышение гибкости интерфейса между платформами и улучшение расширяемости за счёт упрощения компонентов сервера. Кроме того, разделение позволяет каждому компоненту развиваться независимо.

Используя интерфейс REST, различные клиенты попадают в одни и те же конечные точки, выполняют те же действия и получают одинаковые ответы, что положительно влияет на чистоту и позволяет избежать многократного дублирования кода [6].

REST определяет, как компоненты распределенной системы должны взаимодействовать друг с другом. В общем случае этот происходит посредством запросов-ответов. Система может считаться RESTful, если при ее проектировании и реализации были учтены следующие ограничения:

1) приведение архитектуры к модели клиент-сервер: необходимо отделять потребности клиентского интерфейса от потребностей сервера, хранящего данные;

2) отсутствие состояния: все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса;

3) кэширование: у ответов сервера должно быть явное или неявное обозначение как кэшируемых или некэшируемых;

4) единообразие интерфейса: клиент должен всегда понимать, в каком формате и на какие адреса ему нужно слать запрос, а сервер, в свою очередь, также должен понимать, в каком формате ему следует отвечать на запросы клиента;

5) слои: применение промежуточных серверов способно повысить масштабируемость за счёт балансировки нагрузки и распределённого кэширования;

6) код по требованию (необязательное ограничение): данное ограничение подразумевает, что клиент может расширять свою функциональность, за счет загрузки кода с сервера в виде апплетов или сценариев.

Системы, основанные на парадигме REST, не имеют сохранения состояния, следовательно, для сервера нет необходимости знать о состоянии клиента и наоборот. Таким образом, и сервер, и клиент могут понять любое новое полученное сообщение без использования информации из предыдущих сообщений. Такой подход позволяет RESTful-приложениям достигать надёжности, быстрой производительности и расширяемости, как компонентам, которые могут быть управляемы, обновлены и повторно использованы, не затрагивая систему в целом даже во время её работы.

В данной архитектуре клиенты отправляют запросы на поиск или изменение ресурсов, а серверы отправляют ответы на эти запросы. Такой запрос обычно состоит из:

1) НТТР-метода, который определяет вид операции;

2) заголовка, который позволяет клиенту передавать информацию о запросе (Accept);

3) пути к ресурсу;

4) необязательного тела сообщения, содержащего данные.

Основные методы HTTP, которые могут быть использованы для взаимодействия с ресурсами в системе REST:

1) GET — получение конкретного ресурса (по id) или коллекции ресурсов;

2) POST — создание нового ресурса;

3) PUT — обновление конкретного ресурса (по id);

4) DELETE — удаление конкретного ресурса (по id).

В заголовке запроса клиент отправляет MIME (Multipurpose Internet Mail Extensions) тип контента, который он может получить с сервера (поле Accept). Например, клиент может указать текстовый файл с расширением css как «text/css», изображение с форматом gif – «image/gif» и т.п.

В случаях, когда сервер отправляет клиенту полезную нагрузку, он должен включать MIME тип контента в заголовок ответа. Это поле заголовка контента предупреждает клиента о типе данных, которые он посылает в теле ответа. Тип контента, который сервер возвращает обратно в ответе, должен быть одним из параметров, указанных клиентом в поле принятия запроса.

Ответы от сервера также содержат коды состояния для оповещения клиента об успешном или неудачном выполнении операции. Основные коды, которые отправляются в случае успеха в ответ на методы HTTP:

1) GET — return 200 (OK);

2) POST — return 201 (CREATED);

3) PUT — return 200 (OK);

4) DELETE — return 204 (NO CONTENT).

Таким образом, успешное проектирование системы в парадигме REST предоставляет разработчикам разделить ее на независимые серверную и клиентскую части, а также производить их модернизацию независимо друг от друга.

## **2.2 Микрофреймворк Flask**

Flask – фреймворк для создания веб-приложений на языке программирования Python [7]. Относится к категории «микрофреймворков», то есть предоставляет лишь самые базовые возможности, которых будет достаточно для реализации REST парадигмы в данном программном продукте.

Flask реализует две зависимости: Jinja2 и Werkzeug. Первая зависимость связывает фреймворк с движком шаблонов, который позволяет использовать механизмы условного рендеринга и шаблонного моделирования web-страницы. Вторая зависимость необходима для подключения к проекту набора инструментов WSGI, который является стандартным инструментом языка Python для развертывания веб-приложений и взаимодействия между ними и различными серверами разработки.

Установка данного фреймворка довольно тривиальна и происходит посредством ввода в терминале команды «pip install flask» и включением данного пакета в текущий проект.

Проверка правильности выполнения HTTP запросов и получение JSON данных на странице в браузере не является практичным приемом отладки. Для этого удобно использовать плагин CURL для командной строки. Он позволяет выполнять различные виды запросов, получать расширенную информацию о их выполнении, об авторизованных пользователях и многое другое. Данный плагин используется в командной строке.

Например, информация при запросе существующего студента будет выглядеть как показано ниже.

C:\Users\kiril>curl –i

http://127.0.0.1:5000/qr/api/v0.0/students/s0153478

HTTP/1.1 200 OK

Server: Werkzeug/2.1.2 Python/3.9.5

Date: Fri, 20 May 2022 07:05:45 GMT

Content-Type: application/json

Content-Length: 88

Connection: close

{

"student": {

"fullName": "Minin Kirill Sergeevich",

"id": "s0153478"

}

}

Далее отображен ответ при запросе клиентом студента по несуществующему идентификатору.

C:\Users\kiril>curl –i

http://127.0.0.1:5000/qr/api/v0.0/students/s0

HTTP/1.1 404 NOT FOUND

Server: Werkzeug/2.1.2 Python/3.9.5

Date: Fri, 20 May 2022 07:07:07 GMT

Content-Type: application/json

Content-Length: 27

Connection: close

{

"error": "Not found"

}

В реализованном приложении предусмотрено несколько HTTP методов типа ‘GET’. На данном этапе разработки данные о студентах либо находятся на развернутом локальном сервере, либо передаются в качестве переменной в запросе клиента (в частности, это ID студента, который выдается каждому при поступлении в университет и получении зачетной книжки).

Структура, представляющая собой список студентов, выглядит так, как показано ниже.

students = [  
 {**'id'**: **"s0153478"**,  
 **'fullName'**: **'Minin Kirill Sergeevich'**},  
 {**'id'**: **"s0153477"**,  
 **'fullName'**: **'Malyga Gleb Igorevich'**},  
 {**'id'**: **"s0153612"**,  
 **'fullName'**: **'Minina Darya Aleksandrovna'**}]

Метод получения JSON файла, содержащего всех студентов, указанных в данном списке, представлен далее. В нем используется метод jsonify из пакета Flask, который преобразовывает переданный в качестве аргумента словарь в стандартный формат JSON. Как видно из рисунка 11, ответ сервера может быть передан на различные запросы клиента несколько раз, причем по разным URL адресам.

@app.route(**'/'**)  
@app.route(**'/qr/api/v0.0/students'**, methods=[**'GET'**])  
**def** get\_students():  
 **return** jsonify({**'students'**: students})

Далее представлен метод получения конкретного студента по его идентификатору. Основное отличие состоит в расширении URL адреса запроса клиента, а также использовании метода filter() для поиска подходящих записей в имеющемся списке студентов. Преобразование полученного результата к структуре list() обуславливается тем, что с объектом типа filter нет возможности проводить необходимые преобразования в JSON формат.

@app.route(**'/qr/api/v0.0/students/<string:student\_id>'**, methods=[**'GET'**])  
**def** get\_student(student\_id):  
 student = list(filter(**lambda** t: t[**'id'**] == student\_id, students))  
 **if** len(student) == 0:  
 abort(404)  
 **return** jsonify({**'student'**: student[0]})

Ниже отображен метод получения QR кода студента по его идентификатору, передаваемому в запросе клиента. В нем используется метод, генерирующий QR код заданного текста. Также особенность данного запроса состоит в том, что ответ на него выдается непосредственно на WEB странице. Выбор данного подхода обосновывается наличием бинарного файла (изображения), содержащего в себе символ сгенерированного QR кода.

@app.route(**'/qr/api/v0.1/students/<string:student\_id>'**, methods=[**'GET'**])  
**def** get\_student\_v0\_1(student\_id):  
 generate\_qr.get\_qr\_by\_id(student\_id)  
 **return** render\_template(**"index.html"**,  
 title = **'Cur Student'**,  
 student\_id = student\_id,  
 filename = **f'{**os.path.dirname(os.path.abspath(\_\_file\_\_))**}\my\_qr.png'**)

Также для правильной работы отображения изображения QR кода на странице веб-сервиса, сгенерированное изображение QR кода также необходимо отправлять с сервера в виде ответа на запрос и в MIME формате. Для этого был реализован отдельный метод, который представлен далее.

@app.route(**'/qr/api/v0.1/get\_image'**, methods=[**'GET'**])  
**def** get\_image\_v0\_1():  
 **return** send\_file(**'../my\_qr.png'**, mimetype=**'image/png'**)

## **2.3 Структура используемого QR кода**

QR Code – матричная символика. Символ состоит из массива номинально квадратных модулей, структурированных в регулярную квадратную матрицу, включая уникальные шаблоны поиска, размещенные в трех углах символа (в Micro QR Code – только в одном углу) и предназначенные для упрощения определения места нахождения, размера и наклона символа. Предусмотрен широкий диапазон размеров и четыре уровня исправления ошибок. Размер символа в модулях устанавливает пользователь для обеспечения производства символов различными методами.

Основываясь на необходимости применения UUID внутри символа QR кода, а также наличия в нем идентификатора студента было принято решение использовать следующую структуру для кодирования информации в QR код (Рисунок 10).

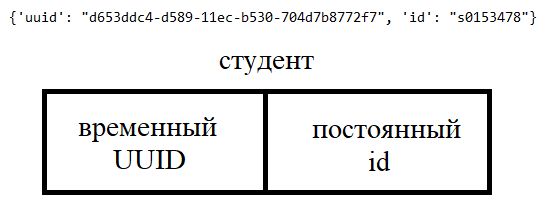


Рисунок 10 – Структура информации в QR коде

Информация, находящаяся внутри QR кода, представлена в формате JSON т.к. этого требует технология REST для возможности разделения на модули клиента и сервера, обменивающихся данными между собой.

## **2.4 Генерация QR кода посещаемости студента**

Основной метод генерации QR кода посещаемости студента представлен на рисунках 17-19. В языке Python есть встроенная библиотека uuid, которую можно легко использовать для генерации различных версий и вариантов UUID рассмотренных ранее. В данной реализации был выбран UUID на основе MAC адреса и временной метки, который позволяет генерировать уникальный идентификатор для различных студентов даже если временные метки их обращений по каким-то причинам совпадают. Далее происходит объединение всех данных в одну строку для более удобного кодирования информации. Реализовано байтовое кодирование, кодировка UTF-8 выбрана для корректного отображения кириллицы.

**def** get\_qr\_by\_id(student\_id):  
 my\_uuid = uuid.uuid1()  
 encoding\_string = **"{uuid: \""** + str(my\_uuid) + **"\", id: \""** + str(student\_id) + **"\"}"** bytes\_encoded = encoding\_string.encode(encoding=**'utf8'**, errors=**'replace'**)  
 binary\_encoded = binary\_encoding(bytes\_encoded)  
  
 bit\_amount\_data = len(binary\_encoded)  
 byte\_amount\_data = int(len(binary\_encoded) / 8)  
  
 version = get\_version(max\_data\_amount\_h\_corr, bit\_amount\_data)  
 max\_count\_bytes\_for\_ver = (max\_data\_amount\_h\_corr[version - 1] + 16) / 8  
  
 correction\_level = **'H'** encoding\_method = **"0100"** length\_amount\_data\_field = get\_length\_amount\_data\_byte\_code(version)  
 binary\_byte\_amount\_data = binary\_encoding\_amount\_data(byte\_amount\_data, length\_amount\_data\_field)  
 temp\_data\_flow = encoding\_method + binary\_byte\_amount\_data + binary\_encoded  
 temp\_data\_flow = fill\_data(temp\_data\_flow, max\_count\_bytes\_for\_ver)  
 byte\_amount\_data = int(len(temp\_data\_flow) / 8) *# В байтах количество данных закодированных  
  
 # байтовое представление данных (список)* binary\_encoded\_bytes = divide\_by\_bytes(temp\_data\_flow)  
 *# байтовое представление данных (строка)* binary\_encoded\_bytes\_str = divide\_by\_bytes\_str(temp\_data\_flow)

В результате выполнения первой части метода, представленной выше, получается байтовое и битовое представление исходных данных в закодированном виде. Далее полученная строка в бинарном представлении разбивается на блоки для реализации механизма исправления ошибок при чтении символа QR кода. Происходит построение байтов коррекции для данных информационных блоков и все данные преобразуются в один непрерывный байтовый и бинарный потоки.

*# кол-во блоков для нашего случая:*amount\_blocks = blocks\_amount\_corr\_h[version - 1]  
amount\_data\_in\_block = int(byte\_amount\_data / amount\_blocks)  
amount\_data\_in\_block\_remains = byte\_amount\_data % amount\_blocks  
blocks = get\_blocks(amount\_data\_in\_block, amount\_blocks, amount\_data\_in\_block\_remains)  
data\_block\_divided = get\_data\_by\_blocks(blocks, binary\_encoded\_bytes)  
  
corr\_bytes\_by\_block\_h = [17, 28, 22, 16, 22, 28, 26, 26, 24, 28,  
 24, 28, 22, 24, 24, 30, 28, 28, 26, 28,  
 30, 24, 30, 30, 30, 30, 30, 30, 30, 30,  
 30, 30, 30, 30, 30, 30, 30, 30, 30, 30]  
*# кол-во байтов коррекции для нашего случая:*amount\_corr\_bytes\_by\_block = corr\_bytes\_by\_block\_h[version - 1]  
*# полином для нашего случая:*gen\_poly = generic\_polys[amount\_corr\_bytes\_by\_block]  
correction\_bytes = get\_corr\_bytes(data\_block\_divided, amount\_corr\_bytes\_by\_block, gen\_poly)  
  
*# Объединение блоков:*data\_flow\_bytes = combine\_sys\_data\_corr\_bytes(data\_block\_divided, correction\_bytes)  
  
data\_flow\_bit = combine\_flow\_bit(data\_flow\_bytes)  
  
flat\_signs\_coords = [**...**]  
  
flat\_coords = flat\_signs\_coords[version - 1]  
  
count\_modules = 8 + 7 + flat\_coords[len(flat\_coords) - 1]

По результатам выполнения второй части метода, которая отображена выше, реализуется поиск координат для выравнивающих узоров. Также составляется матрица уже «использованных» блоков (которые были закрашены белым или черным цветом) и дополняется в процессе отрисовки символа QR кода.

def\_str = [not\_used] \* count\_modules  
used\_c\_1 = []  
**for** i **in** range(count\_modules):  
 temp\_str = def\_str.copy()  
 used\_c\_1.append(temp\_str)  
  
indents = [0, 1, 2, 3, count\_modules - 4, count\_modules - 3, count\_modules - 2, count\_modules - 1]  
indent = 4  
color\_str\_matr(used\_c\_1, white, indents)  
color\_col\_matr(used\_c\_1, white, indents)  
draw\_left\_up\_search(used\_c\_1)  
draw\_right\_up\_search(used\_c\_1, count\_modules)  
draw\_left\_down\_search(used\_c\_1, count\_modules)  
draw\_vert\_sync(used\_c\_1)  
draw\_horiz\_sync(used\_c\_1)  
draw\_flat\_signs(used\_c\_1, flat\_coords, version, indent)  
  
*# список со всеми кодами версий (начиниая с 7 версии)*version\_codes = [...]  
version\_code = []  
**if** version > 6:  
 version\_code = version\_codes[version - 7]  
 draw\_ver\_code(used\_c\_1, version\_code, count\_modules)  
mask\_cor\_code = [0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1]  
draw\_masc\_cor\_code(used\_c\_1, mask\_cor\_code, count\_modules)  
  
*# записываем данные в qr код*draw\_data(used\_c\_1, data\_flow\_bit, count\_modules)  
print\_matr\_file(used\_c\_1)  
draw\_qr(used\_c\_1, count\_modules)

В представленной выше третьей части метода происходит отрисовка остальных шаблонов, а также размещение двоичного потока в оставшихся блоках символа QR кода. Результат заполнения изображения черными и белыми пикселями сохраняется внутри метода draw\_qr() в формате png.

# 3 Результаты работы приложения

В предлагаемой курсовой работе представлено приложение, позволяющее генерировать QR коды для отметок посещаемости студентов на основе байтового кодирования. Разработано на языке программирования Python с использованием библиотеки PIL и фреймворка Flask для реализации парадигмы REST и создания собственного API.

Ниже приведены примеры работы приложения – сгенерированные QR коды и их размещение на созданном веб сервисе (Рисунки 11-12).



Рисунок 11 – Результат работы приложения для первого студента

Рисунок 12 – Результат работы приложения для второго студента

В качестве демонстрации возможностей выдачи информации JSON формата на созданную WEB страницу приведены рисунки 13 и 14.

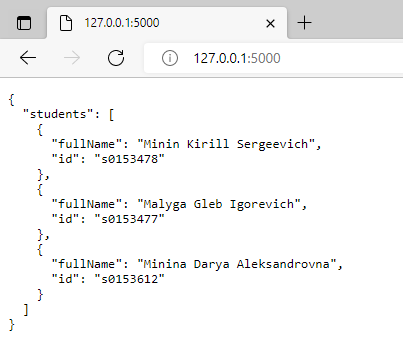
****

Рисунок 13 – Результат работы приложения при запросе всех студентов

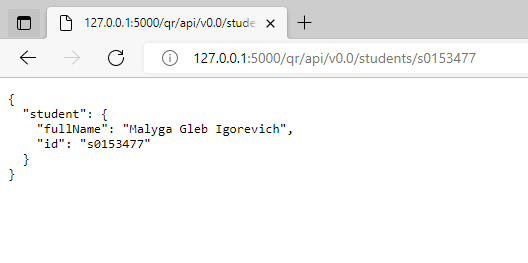
****

Рисунок 14 – Результат работы приложения при запросе одного студента

Таким образом, разработанное приложение протестировано с использованием локального сервера. QR коды, загруженные на web-страницу, могут быть отсканированы через приложение на смартфоне или модуля-сканера, расположенного на другом web-ресурсе.

# ЗАКЛЮЧЕНИЕ

Цель курсовой работы – реализовать приложение, генерирующее QR код учета посещаемости студента – достигнута.

В результате выполнения курсовой работы изучена архитектура REST распределенных web приложений, фреймворк Flask на языке программирования Python для создания web сервисов, а также основные аспекты идентификации и аутентификации пользователей.

В работе выполнен анализ понятий идентификации и аутентификации, структуры REST приложений, проведено тестирование разработанного программного продукта и представлены его результаты.

В практической части курсовой работы реализовано приложение на языке программирования Python с использованием модуля PIL и фреймворка Flask для генерации изображения QR кода и его отображения на web странице для последующего сканирования и учета посещаемости студента. Реализованы некоторые аспекты парадигмы REST.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 ГОСТ Р 58833-2020. Защита информации. Идентификация и аутентификация. Общие положения. Дата введения 2020-05-01.

2 A Universally Unique IDentifier (UUID) URN Namespace: [электронный ресурс] – URL: https://datatracker.ietf.org/doc/html/rfc4122#section-4.1.3 (дата обращения 05.04.2022).

3 UUID – стандарты, формат, кодирование,   
версии и варианты: [электронный ресурс] – URL: https://ru.wikipedia.org/wiki/UUID#%D0%92%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82\_0 (05.04.2022).

4 ГОСТ Р ИСО/МЭК 18004-2015. Информационные технологии. Технологии автоматической идентификации и сбора данных. Спецификация символики штрихового кода QR Code. Дата введения 2016-02-01.

5 Wilde, E. REST: From Research to Practice / E. Wilde, C. Pautasso – London: «Springer New York», 2014. – 524 с. – ISBN 978-1-4419-8302-9.

6 Архитектура REST: [электронный ресурс] – URL: https://habr.com/ru/post/38730/ (29.04.2022).

7 Flask documentaition and user’s guide: [электронный ресурс] – URL: https://flask.palletsprojects.com/en/2.1.x/ (дата обращения 01.05.2022).

8 Перспективы использования QR-кодировки в академической сфере // Социология науки и технологий. 2015. №2 [электронный ресурс] – URL: https://cyberleninka.ru/article/n/perspektivy-ispolzovaniya-qr-kodirovki-v-akademicheskoy-sfere (дата обращения 17.05.2022).

9 Анализ современных методов влияния QR-кодов на жизнь человека в современном мире // ЕГИ. 2020. №3 (29) [электронный ресурс] – URL: https://cyberleninka.ru/article/n/analiz-sovremennyh-metodov-vliyaniya-qr-kodov-na-zhizn-cheloveka-v-sovremennom-mire (дата обращения 17.05.2022).