Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Создание онлайн-платформы поликлиники НОМТЦ МГТУ им. Баумана

Студент _____ИУ7-65Б_____ _____ _____А.А. Лаврова_____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____Л.Л. Волкова_____
(Подпись, дата) (И.О.Фамилия)

2020 г.

Содержание

Введение

В современном мире каждая сфера деятельности связана с использованием информационных технологий, в частности, сети Интернет. Сейчас уже невозможно представить какую-либо компанию или организацию без своего сайта.

Недавно веб-сайты состояли только из HTML-страниц, но сейчас для разработки крупных проектов используется система «клиент-сервер» – веб-приложение. Главной особенностью веб-приложения является возможность обмена, обработки и изменения информации. Такое взаимодействие можно представить только с использованием баз данных, которые позволяют управлять большим количеством информации.

Целью проекта является разработка клиент-серверного приложения для НОМТЦ МГТУ им. Баумана, которое предоставляет возможность для пациентов просмотреть свою медицинскую карту и записаться на приём к врачу, а для врачей – сделать запись в медицинскую карту и просмотреть своё расписание.

Для достижения поставленной цели, необходимо решить следующие задачи:

- Формализовать задачу и определить необходимый функционал
- Провести анализ существующих СУБД
- Спроектировать базу данных
- Выбрать подходящий язык программирования, спроектировать архитектуру программы
- Реализовать пользовательский интерфейс

1. Аналитический раздел

В этом разделе проводится обзор архитектуры программного комплекса, анализ существующих моделей баз данных и выбор наиболее подходящего для решения поставленных задач.

1.1 Формализация задачи

В соответствии с поставленными целями необходимо разработать клиент-серверное веб-приложение для НОМТЦ МГТУ им. Баумана с возможностью регистрации и аутентификации.

Для каждого типа пользователя предусмотрен свой набор функций.

Пациент:

- Просмотр своей медицинской карты
- Запись на приём к врачу

Врач:

- Запись диагноза в медицинскую карту пациента
- Поиск медицинской карты пациента

Администратор:

- Регистрация нового пользователя в системе с установлением соответствующей роли
- Добавление новости
- Просмотр расписания записи на приём пациентов для каждого врача

Также предусмотрен новостной раздел, сведения о врачах, расписание приёма и контакты поликлиники.

1.2 Трёхуровневая архитектура

Трехуровневая архитектура приложений — это модульная клиент-серверная архитектура, которая состоит из уровня представления, уровня приложения и уровня данных.

Уровень представления является клиентским приложением, которое переносит задачи по обработке информации на сервер. Примером этого уровня может служить компьютер с браузером, на котором отображается веб-приложение.

Серверная часть располагается на уровне приложения, в ней сосредоточена большая часть бизнес-логики. Обработка информации также происходит на уровне сервера.

Сервер базы данных – это уровень данных, реализуется, как правило, средствами систем управления базами данных, подключение к этому компоненту обеспечивается только с уровня сервера приложений.

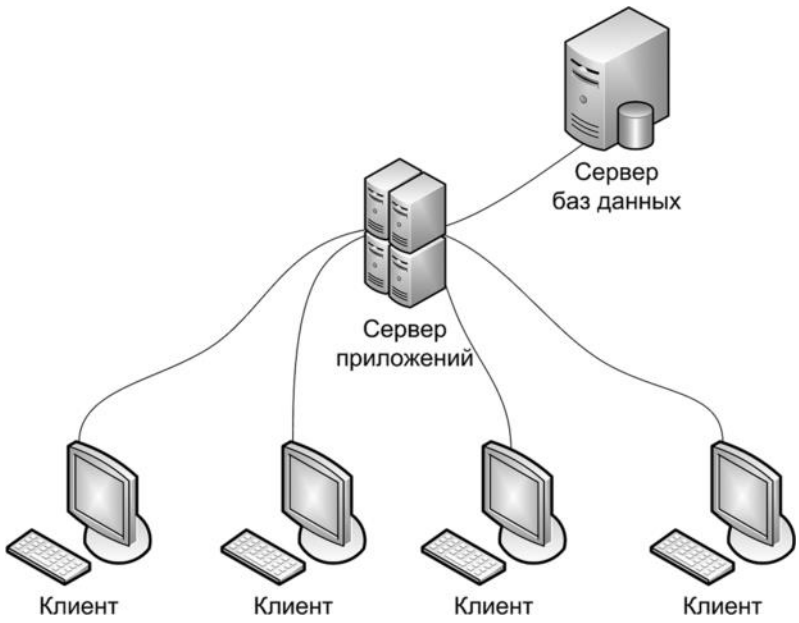

Рисунок 1 – Пример трёхуровневой архитектуры

В простейшей конфигурации сервер приложения и сервер базы данных может находится на одном компьютере. Однако с точки зрения безопасности и

масштабируемости сервер базы данных должен находится на отдельном устройстве, к которому по сети подключаются остальные серверы приложений.

Достоинства трёхуровневой архитектуры:

- Масштабируемость – при необходимости способность выдержать увеличение нагрузки
- Конфигурируемость - изолированность уровней друг от друга позволяет быстро переконфигурировать систему при возникновении сбоев
- Низкие требования к скорости сети между клиентами и сервером

Недостатки:

- Более высокая сложность создания приложения
- Высокие требования к производительности серверов приложений и базы данных

1.3 Двухзвенная клиент-серверная архитектура

В архитектуре "клиент-сервер" программное обеспечение разделено на две части - клиентскую часть и серверную часть. Задача клиентской части (программы-клиента) состоит во взаимодействии с пользователем, передаче пользовательского запроса серверу, получение запроса от серверной части (программы-сервера) и представление его в удобном для пользователя виде. Программа-сервер же обрабатывает запросы клиента и выдает ответы.

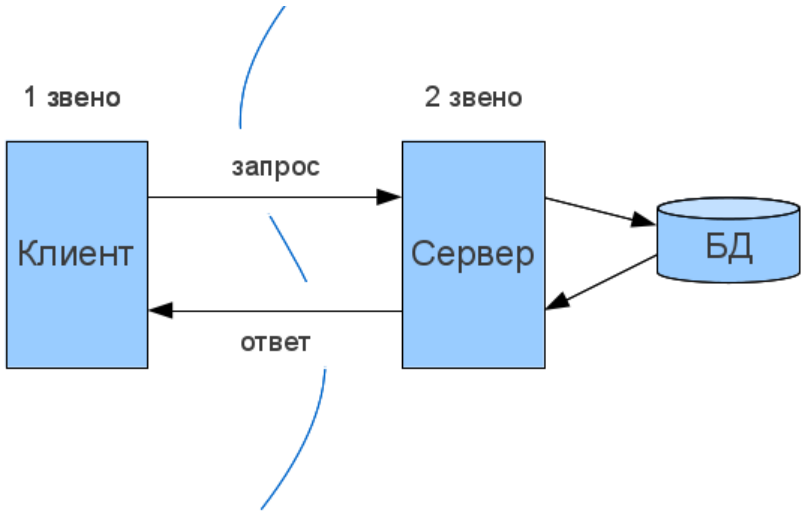

Рисунок 2 - Двухзвенная клиент-серверная архитектура

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса.

Достоинства двухзвенной архитектуры:

- Нет дублирования кода
- Безопасность персональных данных
- Нет необходимости обслуживания нескольких серверов
- Существенная часть проектных задач становится уже решенной

Недостатки:

- «Жесткая» сцепка СУБД и серверной части
- При неработоспособности сервера клиенты также не смогут продолжать работу

1.4 Виды баз данных

Существует огромное количество разновидностей баз данных, но их всех можно классифицировать по модели данных, которая определяет логическую структуру БД.

- Сетевая модель
- Иерархическая модель
- Реляционная модель
- Функциональная модель

1.4.1 Сетевая модель БД

Сетевые базы данных относятся к теоретико-графовым моделям. Сети – это естественный способ представления отношений между объектами базы данных и связей между этими объектами (таблицы баз данных или сущности). Сетевую модель можно представить в виде ориентированного графа, который состоит из узлов и ребер. Узлы направленного графа – это ни что иное, как объекты сетевой базы данных, а ребра такого графа показывают связи между объектами сетевой модели данных, причем ребра показывают не только саму связь, но и тип связи (связь один к одному или связь один ко многим).

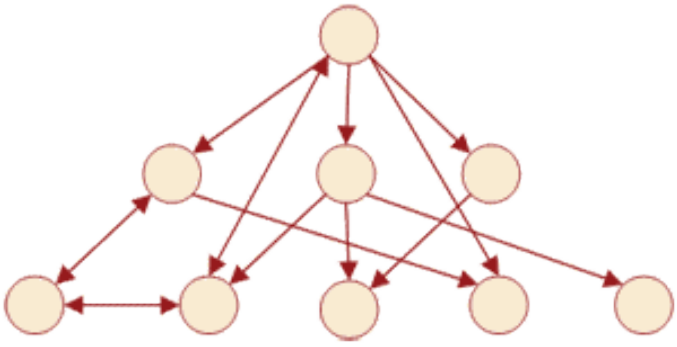

Рисунок 3 – Пример структуры сетевой модели БД

Достоинством такой модели является возможность строить множественные связи между объектами. Однако главным недостаток остается невозможность изменить структуру после ввода данных.

1.4.2 Иерархическая модель БД

Иерархическая модель БД также опирается на теорию графов, так как её можно назвать частным случаем сетевой модели. В её основе лежит древовидная структура, где между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня).

Иерархическая модель появилась раньше сетевой, однако она более простая, а вследствие менее эффективная.

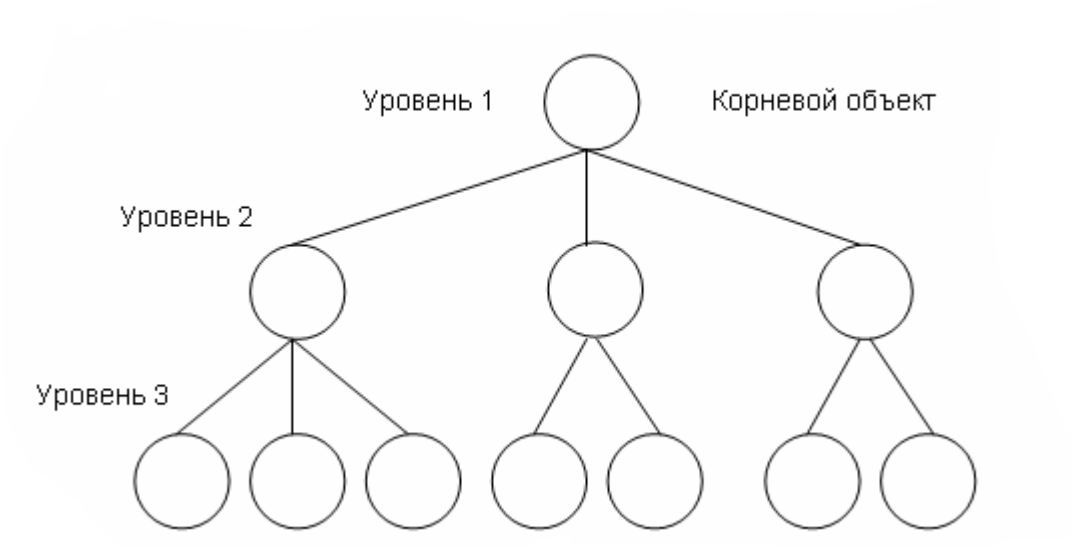

Рисунок 4 – Пример структуры иерархической модели БД

1.4.3 Реляционная модель БД

Реляционная модель есть представление БД в виде совокупности упорядоченных нормализованных отношений. Для реляционных отношений характерны следующие особенности:

- Любой тип записи содержит только простые (по структуре) элементы данных.
- Порядок кортежей в таблице несуществен.

- Упорядочение значащих атрибутов в кортеже должно соответствовать упорядочению атрибутов в реляционном отношении.
- Любое отношение должно содержать один атрибут или более, которые вместе составляют уникальный первичный ключ.
- Если между двумя реляционными отношениями существует зависимость, то одно отношение является исходным, второе - подчиненным.
- Чтобы между двумя реляционными отношениями существовала зависимость, атрибут, служащие первичным ключом в исходном отношении, должны также присутствовать в подчиненном отношении.

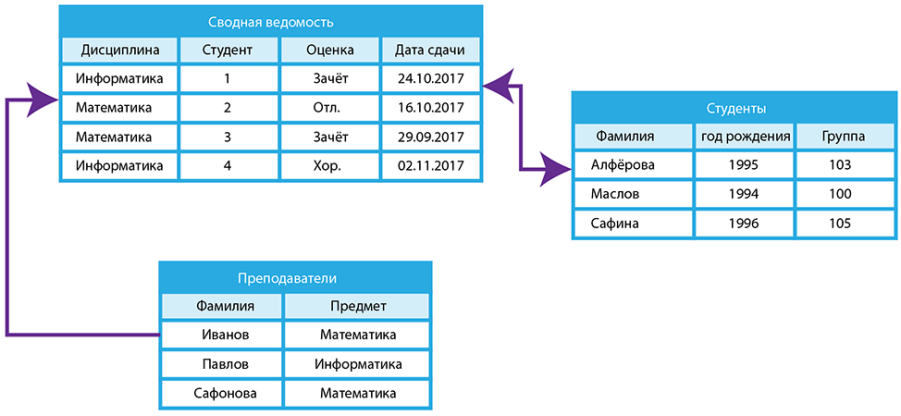

Рисунок 5 – Пример структуры реляционной модели БД

Преимуществом такой модели является простота реализации и независимость данных друг от друга. Недостатком можно считать то, что расходуется много памяти для поддержания всех данных в виде таблиц, а также низкая скорость обработки информации.

1.4.4 Функциональная модель БД

Функциональные базы данных используются для решения аналитических задач, таких как финансовое моделирование и управление производительностью. Вместо того чтобы представлять объект записью с определенным содержанием или же кортежем в дереве, функциональная модель сообщает, какие функции (или операции) определены на этом объекте. Представление объекта — это дело реализации, и оно определяется на более низком уровне абстракции.

1.5 Вывод

Исходя из поставленной задачи наиболее оптимальным решением станет комбинация реляционной модели базы данных и двухзвенной клиент-серверной архитектуры, так как это позволит реализовать поставленные цели, не затрачивая при этом большие вычислительные мощности и не усложняя программную архитектуру.

2. Конструкторский раздел

2.1 Проектирование базы данных

База данных для НОМТЦ МГТУ им. Баумана состоит из 7 основных таблиц.

- Таблица расписания приёма - Schedule
- Таблица с информацией о врачах – Doctors
- Таблица новостей – News
- Таблица всех пользователей сайта – User
- Таблица медицинских диагнозов – Record
- Таблица с информацией о записи на приём – Notes
- Таблица записи на прием к каждому врачу – Doctor_N

2.1.1 Таблица Schedule

В данной таблице хранятся данные о расписании всех врачей с указанием специальности, номера кабинета и этажа.

- id_sch – целочисленное поле, идентификационный номер записи
- sch_doc – целочисленное поле, идентификационный номер врача
- sch_spec – символьное поле, специальность врача
- sch_cab – целочисленное поле, номер кабинета
- sch_floor – целочисленное поле, этаж, на котором находится кабинет
- sch_mon – символьное поле, время работы в понедельник
- sch_tue - символьное поле, время работы во вторник
- sch_wed - символьное поле, время работы в среду
- sch_thu - символьное поле, время работы в четверг
- sch_fri - символьное поле, время работы в пятницу

2.1.2 Таблица Doctors

Хранит ознакомительную информацию о врачах, в том числе краткую биографию.

- id_doc – целочисленное поле, идентификационный номер врача
- doc_name – символьное поле, ФИО врача
- doc_spec – символьное поле, специальность врача
- doc_about – символьное поле, краткая биография врача
- doc_login - целочисленное поле, идентификационный номер врача в таблице всех пользователей

2.1.3 Таблица User

Позволяет идентифицировать пользователя сайта в соответствии с его ролью и реализовать аутентификацию.

- id_sign – целочисленное поле, идентификационный номер пользователя
- sign_login – символьное поле, логин пользователя
- sign_password – символьное поле, хэш пароля пользователя
- sign_role – целочисленное поле, роль пользователя (пациент, врач или администратор)

2.1.4 Таблица News

Хранит новости, которые может добавить администратор на специальный раздел сайта.

- id_news - целочисленное поле, идентификационный номер новости
- news_date – поле даты, хранит дату добавления записи
- news_title – символьное поле, заголовок новости
- news_text – символьное поле, текст новости

2.1.5 Таблица Record

Содержит в себе все записи диагнозов пациентов, которые впоследствии могут посмотреть как сам пациент в личном кабинете, так и врач.

- id_rec – целочисленное поле, идентификационный номер записи
- rec_login – целочисленное поле, идентификационный номер пациента
- rec_date – поле даты, хранит дату постановки диагноза
- rec_diag – символьное поле, текст диагноза

2.1.6 Таблица Notes

В эту таблицу делается запись каждый раз, когда пациент записывается на приём для того, чтобы затем администратор мог предоставить каждому врачу расписание записей.

- id_notes – целочисленное поле, идентификационный номер записи
- notes_day – символьное поле, день недели записи
- notes_time – символьное поле, время записи
- notes_user – целочисленное поле, идентификационный номер пациента
- notes_doctor – целочисленное поле, идентификационный номер врача

2.1.6 Таблица Doctor_N

Таблица записи на приём, создаётся отдельно для каждого врача. С её помощью пациент может интерактивно записаться на приём в своём личном кабинете.

- id_doc - целочисленное поле, идентификационный номер записи
- doc_time – символьное поле, промежутки времени для записи
- doc_mon – целочисленное поле, хранит информацию о возможности записи на приём в данный промежуток времени в понедельник

- doc_tue - целочисленное поле, хранит информацию о возможности записи на приём в данный промежуток времени во вторник
- doc_wed - целочисленное поле, хранит информацию о возможности записи на приём в данный промежуток времени в среду
- doc_thu - целочисленное поле, хранит информацию о возможности записи на приём в данный промежуток времени в четверг
- doc_fri - целочисленное поле, хранит информацию о возможности записи на приём в данный промежуток времени в пятницу

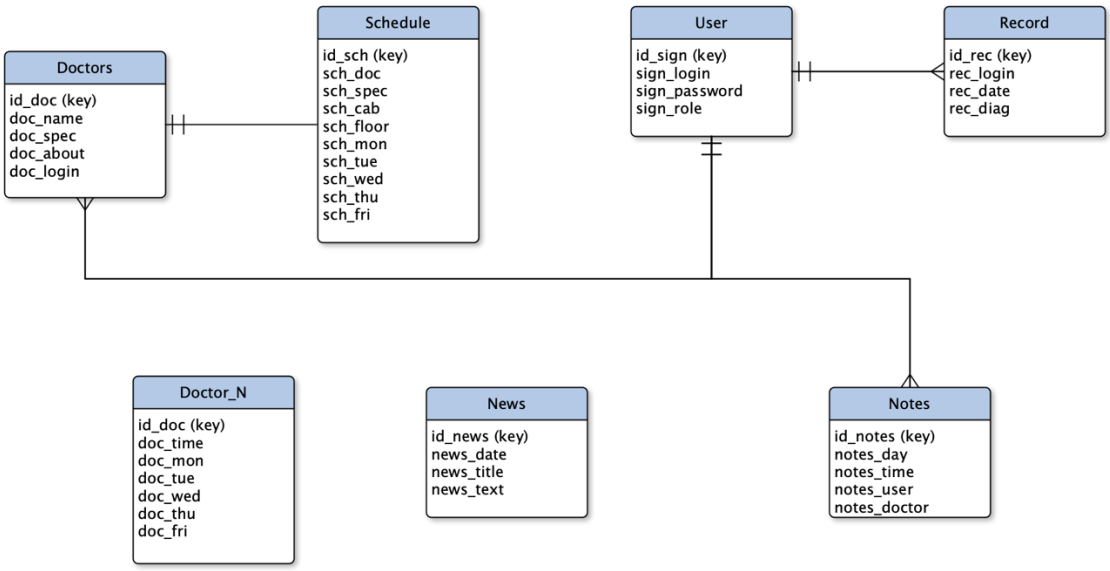

Рисунок 6 – Схема БД

2.1 Проектирование схемы MVC

Паттерн программирования MVC (Model-View-Controller) включает в себя 3 компонента: модель, представление, контроллер. Каждый из компонентов выполняет свою роль и является взаимозаменяемым. Это значит, что компоненты связаны друг с другом лишь некими четкими интерфейсами, за которыми может лежать любая реализация. Такой подход позволяет подменять

и комбинировать различные части, обеспечивая необходимую логику работы или внешний вид приложения.

Модель отвечает за внутреннюю логику работы программы. Здесь мы можем скрыть способы хранения данных, а также правила и алгоритмы обработки информации.

Представление отображает данные модели. На этом уровне мы лишь предоставляем интерфейс для взаимодействия пользователя с моделью. Смысл введения этого компонента тот же, что и в случае с предоставлением различных способов хранения данных на основе нескольких моделей. В случае веб-приложения представлением являются html-страницы.

Контроллер предоставляет связь между моделью и действиями пользователя, полученными в результате взаимодействия с представлением. Координирует моменты обновления состояний модели и представления. Принимает большинство решений о переходах приложения из одного состояния в другое. В данном проекте контроллер отвечает за обработку запросов GET и POST и следит за состоянием модели, где по запросу GET как правило предоставляет страницу по умолчанию, а по запросу POST – производит обработку и выдачу информации.

3. Технологический раздел

3.1 Выбор среды разработки и языка программирования

Для разработки программы выбран язык Python по причине совмещения нескольких парадигм программирования, а также из-за большого разнообразия представленных библиотек и фреймворков для создания веб-приложения.

В качестве среды разработки использован PyCharm, так как с его помощью удобно работать и с python-файлами, и html-страницами. Среда обладает кроссплатформенностью и большим выбором настроек проекта.

Для создания данного проекта в качестве инструмента, который облегчит процесс создания веб-приложения, был выбран фреймворк Flask. Благодаря его простоте и гибкости, разработчик может сам выбрать способ реализации тех или иных задач.

3.2 Выбор СУБД

Для управления базой данных из приложения пользователи Flask могут выбрать Flask-SQLAlchemy - расширение, предоставляющее обертку для проекта SQLAlchemy, который является ORM (Object-relational mapping) или объектно-реляционным отображением. ORM позволяет приложениям БД работать с объектами вместо таблиц или SQL. Операции выполняются над объектами, а потом прозрачно транслируются в команды БД при помощи ORM.

В качестве СУБД был выбрана компактная и встраиваемая SQLite. Она не требует отдельного процесса сервера или системы для работы, а также подходит для данного проекта, так как ожидается обработка средних по размеру данных.

3.3 Реализация моделей хранения данных

Flask-SQLAlchemy обладает функционалом для интеграции СУБД приложениями в объектно-ориентированном стиле. Необходимые классы наследуются от модели базового класса Model.

Листинг 1 – реализация моделей

```
1. class Doctors(db.Model):
2.     __tablename__ = 'doctors'
3.     id_doc = db.Column(db.Integer, primary_key = True)
4.     doc_name = db.Column(db.String(100), nullable = True)
5.     doc_spec = db.Column(db.String(100), nullable=True)
6.     doc_about = db.Column(db.Text, nullable=True)
7.     doc_login = db.Column(db.Integer, db.ForeignKey('user.id_sign'))
8.
9.     def __repr__(self):
10.         return '<Doctors %r>' % self.id_doc
11.
12. class Schedule(db.Model):
13.     __tablename__ = 'schedule'
14.     id_sch = db.Column(db.Integer, primary_key = True)
15.     sch_doc = db.Column(db.Integer, db.ForeignKey('doctors.id_doc'))
16.     sch_cab = db.Column(db.Integer, nullable=True)
17.     sch_floor = db.Column(db.Integer, nullable=True)
18.     sch_mon = db.Column(db.String(100), nullable=True)
19.     sch_tue = db.Column(db.String(100), nullable=True)
20.     sch_wed = db.Column(db.String(100), nullable=True)
21.     sch_thu = db.Column(db.String(100), nullable=True)
22.     sch_fri = db.Column(db.String(100), nullable=True)
23.
24.
25.     def __repr__(self):
26.         return '<Schedule %r>' % self.id_sch
27.
28. class User(db.Model, UserMixin):
29.     __tablename__= 'sign_up'
30.     id_sign = db.Column(db.Integer, primary_key=True)
31.     sign_login = db.Column(db.String(100), nullable = True, unique =
    True)
32.     sign_password = db.Column(db.String(100), nullable=True)
33.     sign_role = db.Column(db.Integer, nullable=True)
34.
35.     def get_id(self):
36.         return (self.id_sign)
37.
38.
39. class Record(db.Model):
40.     __tablename__ = 'record'
41.     id_rec = db.Column(db.Integer, primary_key = True)
42.     rec_login = db.Column(db. Integer, nullable = True,
    db.ForeignKey('user.id_sign'))
43.     rec_date = db.Column(db.Date, default=datetime.utcnow)
44.     rec_diag = db.Column(db.Text, nullable = True)
45.
46.     def __repr__(self):
47.         return '<Record %r>' % self.id_rec
48.
49. class News(db.Model):
50.     __tablename__ = 'news'
51.     id_news = db.Column(db.Integer, primary_key = True)
```

```
52.      news_date = db.Column(db.Date, default=datetime.utcnow)
53.      news_title = db.Column(db.String(300), nullable = True)
54.      news_text = db.Column(db.Text, nullable = True)
55.
56.      def __repr__(self):
57.          return '<News %r>' % self.id_news
58.
59.  class Notes(db.Model):
60.      __tablename__ = 'notes'
61.      id_notes = db.Column(db.Integer, primary_key = True)
62.      notes_day = db.Column(db.String(50), nullable = True)
63.      notes_time = db.Column(db.String(50), nullable=True)
64.      notes_user = db.Column(db.Integer, nullable=True,
   db.ForeignKey('user.id_sign'))
65.      notes_doctor = db.Column(db.Integer, nullable=True,
   db.ForeignKey('doctors.id_doc'))
66.
67.      def __repr__(self):
68.          return '<Notes %r>' % self.id_notes
```

3.4 Реализация контроллера

Атрибут method указывает метод HTTP-запроса, который должен использоваться при отправке формы на сервер. По умолчанию он отправляется с запросом GET, но почти во всех случаях использование POST запрос, улучшающий взаимодействие с пользователем, поскольку запросы этого типа могут отправлять данные формы в тело запроса, в то время как запросы GET добавляют поля формы к URL-адресу, загромождая адресную строку обозревателя.

Листинг 2 – Реализация обработки POST и GET запросов на примере добавления нового диагноза

```
1. @app.route('/add_new_diagnosis', methods = ['GET', 'POST'])
2. def add_new_diagnosis():
3.     if request.method == 'POST':
4.         login_patient = request.form['login_patient']
5.         diagnosis = request.form['diagnosis']
6.
7.         new_diagnosis = Record(rec_login = login_patient, rec_diag =
   diagnosis)
8.
9.         try:
10.             db.session.add(new_diagnosis)
11.             db.session.commit()
12.             return redirect(url_for('doctor_space'))
13.          except:
14.             return "Произошла ошибка!"
15.
16.      else:
17.          return render_template('add_new_diagnosis.html', messages =
   messages)
```

3.5 Реализация регистрации и аутентификации пользователей

Регистрация пользователя в приложении является добавлением в базу данных (в таблицу User) записи, содержащей необходимую информацию для аутентификации. Для этого пользователь вводит соответствующие данные в поля регистрационной формы.

Благодаря использованию фреймворка Flask мы имеем базовый набор инструментов для реализации регистрации и аутентификации.

Немаловажным фактом является то, что хранить пароли пользователей необходимо в зашифрованном формате из соображений безопасности. Для этого была задействована библиотека werkzeug.security, позволяющая хэшировать все поступающие пароли.

Листинг 3 – реализация регистрации пользователя

```
1. @app.route('/registration', methods=['GET', 'POST'])
2. def registration():
3.     login = request.form.get('reg_login')
4.     password1 = request.form.get('reg_password1')
5.     password2 = request.form.get('reg_password2')
6.     role = request.form.get('role')
7.
8.     if request.method == 'POST':
9.         if not (login or password1 or password2):
10.             flash('Заполните все поля!')
11.         elif password1 != password2:
12.             flash('Пароли не совпадают')
13.         else:
14.             hash_password = generate_password_hash(password1)
15.             new_user = User(sign_login = login, sign_password =
    hash_password, sign_role = role)
16.             db.session.add(new_user)
17.             db.session.commit()
18.
19.      return render_template('registration.html', messages = messages)
```

Листинг 4 – реализация аутентификации пользовтеля

```
1. @app.route('/sign_up', methods=['GET', 'POST'])
2. def sign_up():
3.     login = request.form.get('login')
4.     password = request.form.get('password')
5.
6.     if request.method == 'POST':
```

```
        if login and password:
            user = User.query.filter_by(sign_login = login).first()
            if user and check_password_hash(user.sign_password, password):
                login_user(user)
                if user.sign_role == 1:
                    global username
                    username = login
                    return redirect(url_for('user_space'))
                elif user.sign_role == 2:
                    return redirect(url_for('doctor_space'))
                elif user.sign_role == 3:
                    return redirect(url_for('admin_space'))
            else:
                flash('Неправильный логин или пароль!')
        else:
            flash('Нет логина или пароля!')
    return render_template('sign_up.html')
```

3.5 Реализация HTML-шаблона

Для исключения дублирования кода необходимо использовать шаблоны, которые предоставляет фреймворк Flask. В процессе работы приложения динамическая разметка заменяется, а затем генерируется статическая HTML-страница. В Flask существует встроенный движок шаблонов Jinja, который и занимается тем, что конвертирует шаблон в статический HTML-файл.

Листинг 5 – HTML-шаблон

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="shortcut icon" href="static/image/logo.ico" type="image/x-icon">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
    <link rel = "stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
    <title>{% block title %}{% endblock %}</title>
    <style>{% block style %}{% endblock %}</style>
 </head>
 <body>
     <div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-white border-bottom shadow-sm">
     <h5 class="my-0 mr-md-auto font-weight-normal">НОМТЦ МГТУ им. Баумана</h5>
     <nav class="my-2 my-md-0 mr-md-3">
         <a class="p-2 text-dark" href="{{ url_for('hello_world') }}">О нас</a>
         <a class="p-2 text-dark" href="{{ url_for('news') }}">Новости</a>
```

```
17.           <a class="p-2 text-dark" href="{{ url_for('doctors')
   }}">Врачи</a>
18.           <a class="p-2 text-dark" href="{{ url_for('schedule')
   }}">Расписание при❓мов</a>
19.           <a class="p-2 text-dark" href="{{ url_for('contacts')
   }}">Контакты</a>
20.       </nav>
21.       <a class="btn btn-outline-primary" href="{{ url_for('sign_up')
   }}">Найти в личный кабинет</a>
22.       </div>
23.
24.       {% block body %}
25.       {% endblock %}
26.
27.       <footer class="container pt-4 my-md-5 pt-md-5 border-top">
28.       <div class="row">
29.         <div class="col-12 col-md">
30.           <img class="mb-2" src="/docs/4.5/assets/brand/bootstrap-
   solid.svg" alt="" width="24" height="24">
31.           <small class="d-block mb-3 text-muted">© 2020</small>
32.         </div>
33.         <div class="col-6 col-md">
34.           <h5>Информация для пациентов</h5>
35.           <ul class="list-unstyled text-small">
36.             <li><a class="text-muted" href="{{ url_for('hello_world')
   }}">О нас</a></li>
37.             <li><a class="text-muted" href="{{ url_for('news')
   }}">Новости</a></li>
38.             <li><a class="text-muted" href="{{ url_for('doctors')
   }}">Врачи</a></li>
39.             <li><a class="text-muted" href="{{ url_for('schedule')
   }}">Расписание приёмов</a></li>
40.             <li><a class="text-muted" href="{{ url_for('contacts')
   }}">Контакты</a></li>
41.             <li><a class="text-muted" href="{{ url_for('sign_up')
   }}">Войти в личный кабинет</a></li>
42.           </ul>
43.         </div>
44.         <div class="col-6 col-md">
45.           <h5>Мы в социальных сетях</h5>
46.           <ul class="list-unstyled text-small">
47.             <li><a class="text-muted"
   href="https://vk.com/club158860516">Вконтакте</a></li>
48.             <li><a class="text-muted" href="#">Instagram</a></li>
49.             <li><a class="text-muted" href="#">Facebook</a></li>
50.           </ul>
51.         </div>
52.
53.       </div>
54.     </footer>
55.
56.  </body>
57.  </html>
```

3.6 Интерфейс приложения

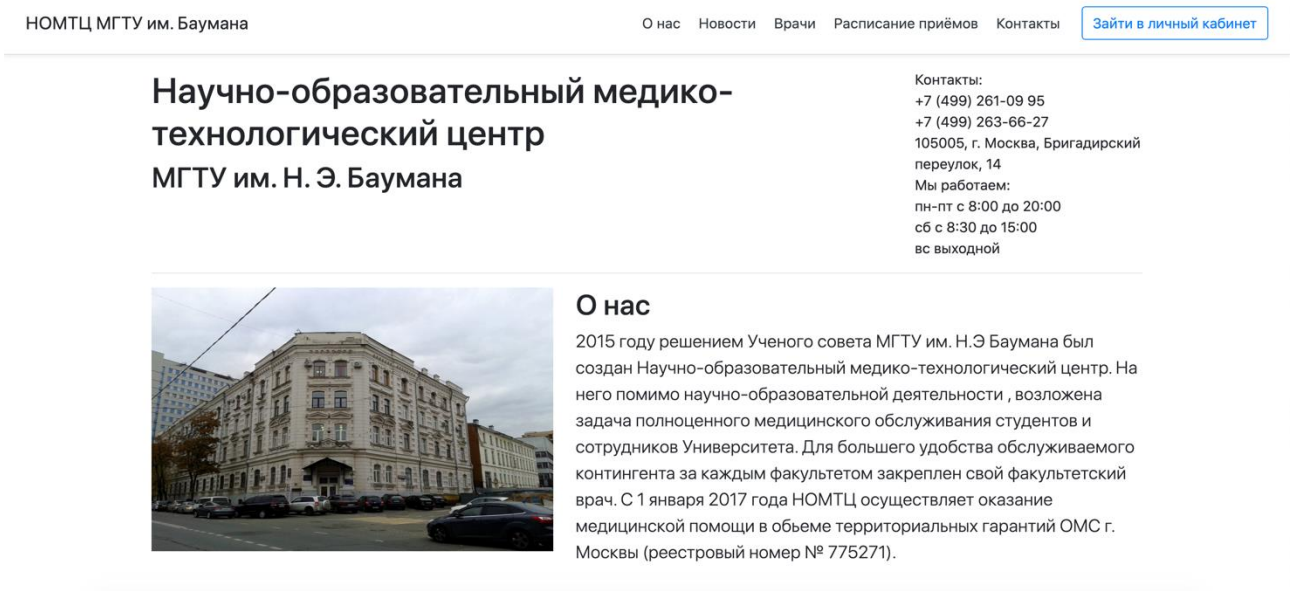

Рисунок 7 – Начальная страница сайта

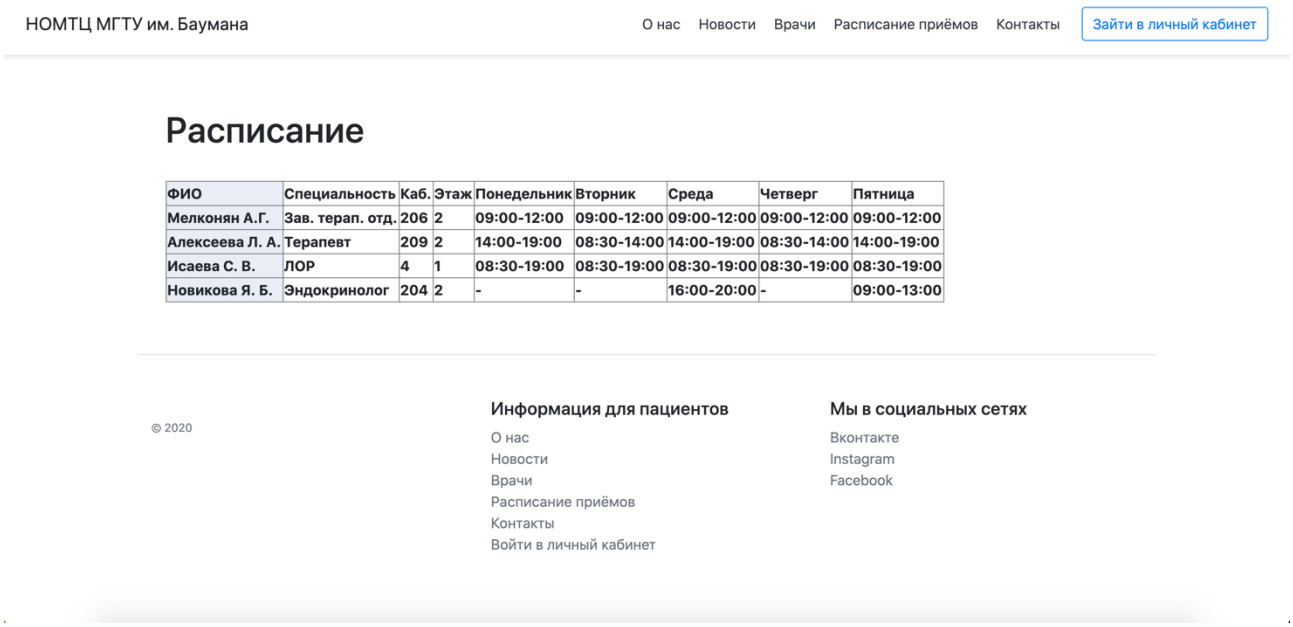

Рисунок 8 – Ознакомительное расписание врачей

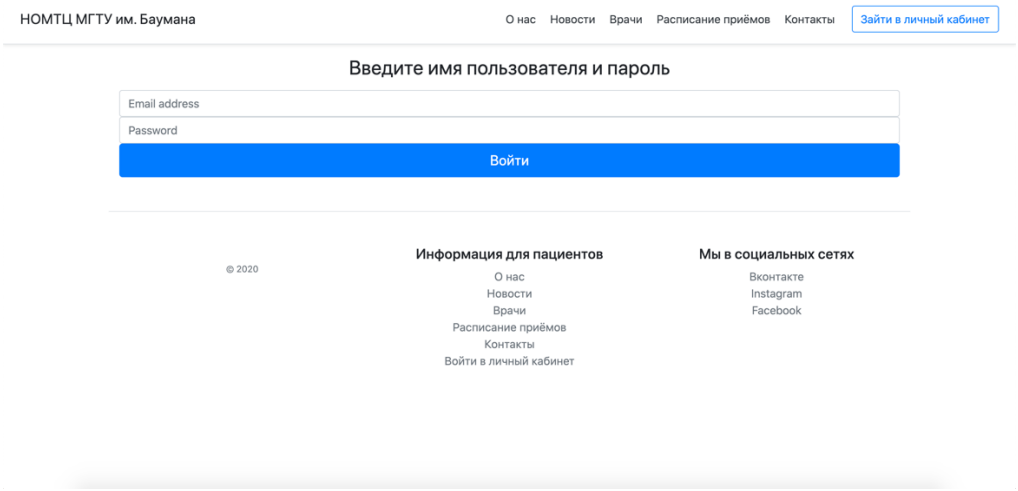

Рисунок 9 – Форма аутентификации пользователя

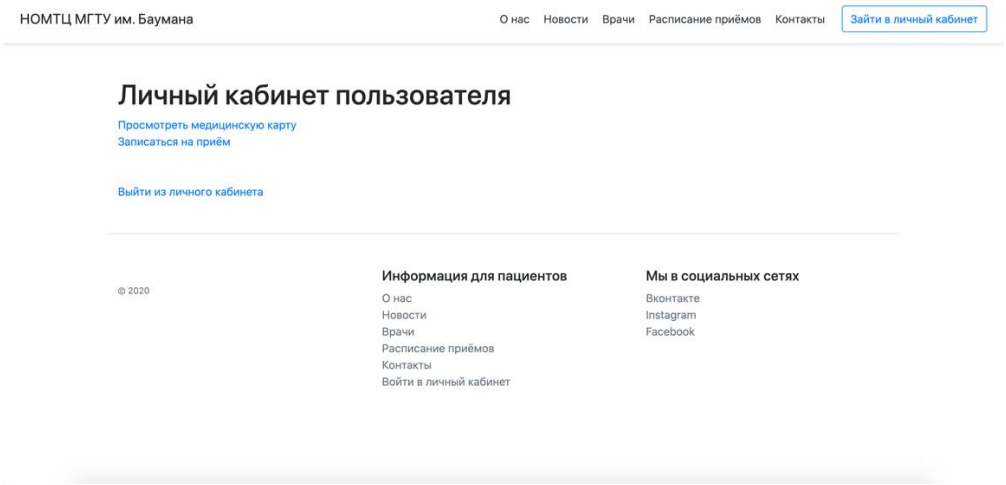

Рисунок 10 – Личный кабинет пациента

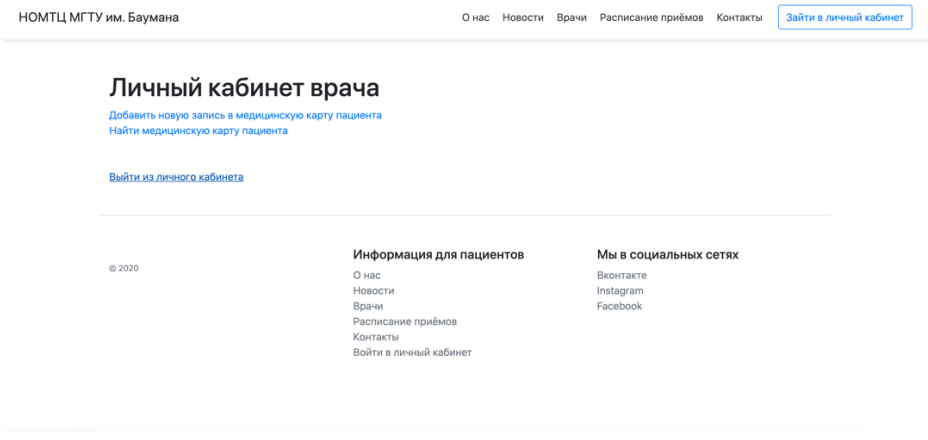

Рисунок 11 – Личный кабинет врача

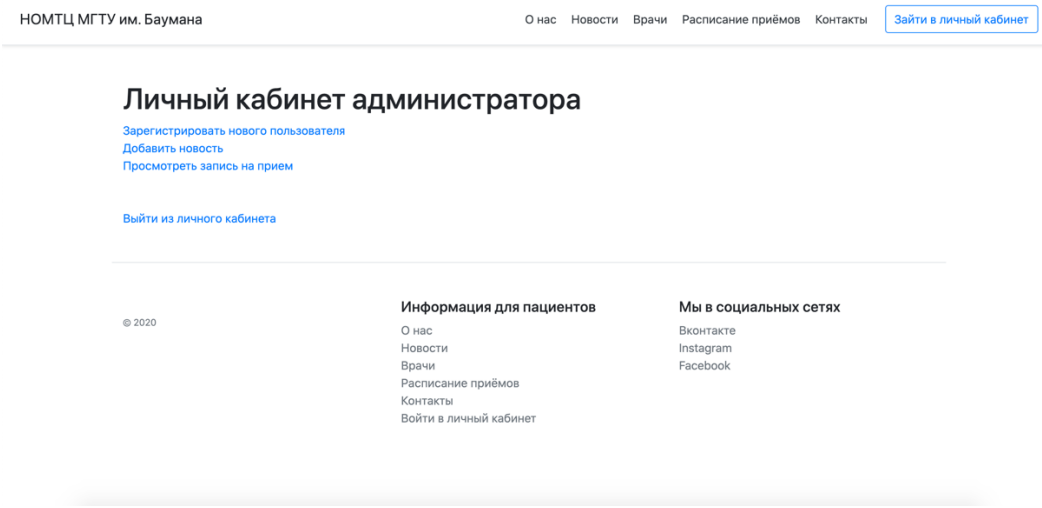

Рисунок 12 – Личный кабинет администратора

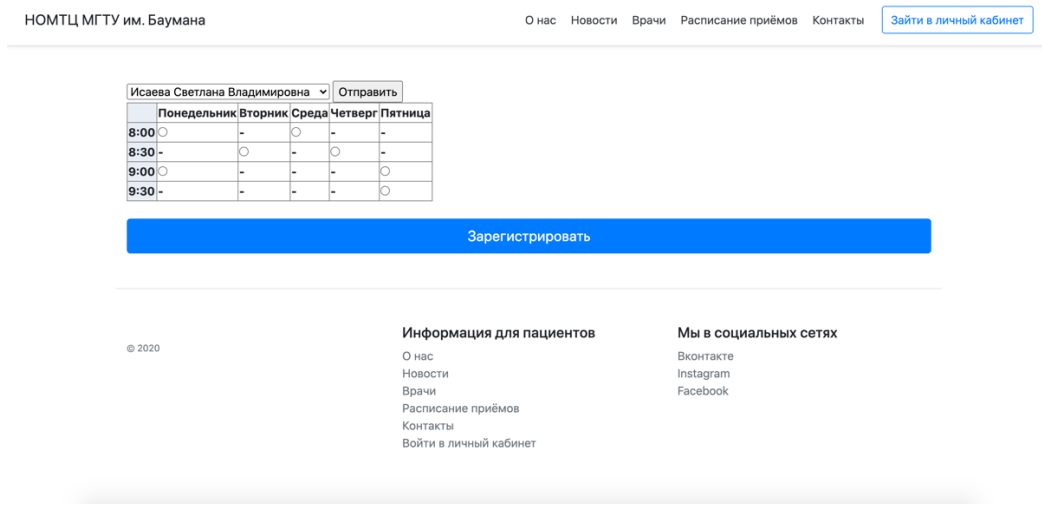

Рисунок 13 – Запись на приём к врачу

Заключение

В результате проделанной работы была формализована задача, определен необходимый функционал. Проведён анализ популярных архитектур веб-приложений, рассмотрены существующие виды моделей баз данных.

В ходе выполнения проекта была разработана структура базы данных, внедрен паттерн Model-View-Controller.

С помощью выбранных инструментов был реализован интерфейс веб-приложения, а также необходимые функции, такие как регистрация и аутентификация пользователей, запись на приём к врачу, просмотр медицинской карты, добавление записей в медицинскую карту, просмотр записи на приём каждого отдельного врача, добавление новости.

Список источников

1. Miguel Grinberg, Мега-Учебник Flask [Электронный ресурс]

   Режим доступа:

   https://habr.com/ru/post/193242/ (дата обращения – 05.05.202)
2. Flask documentation [Электронный ресурс]

   Режим доступа:

   https://flask.palletsprojects.com/en/1.1.x/ (дата обращения – 07.05.2020)
3. НОМТЦ МГТУ им. Баумана [Электронный ресурс]

   Режим доступа:

   http://nomtc.bmstu.ru/ (дата обращения – 20.05.2020)
4. Python documentation [Электронный ресурс].

   Режим доступа:

   https://www.python.org/doc/ (дата обращения - 15.05.2020)
5. НОУ Интуит [Электронный ресурс].

   Режим доступа:

   https://www.intuit.ru/ (дата обращения – 26.05.2020)
6. PythonRu – Основы Flask [Электронный ресурс].

   Режим доступа:

   https://pythonru.com/uroki/3-osnovy-flask (дата обращения - 10.05.2020)
7. SQLAlchemy documentation [Электронный ресурс].

   Режим доступа:

   https://www.sqlalchemy.org/ (дата обращения – 21.05.2020)
8. Bootstrap documentation [Электронный ресурс].

   Режим доступа:

   https://getbootstrap.com/ (дата обращения – 10.05.2020)
9. Werkzeug.security documentation [Электронный ресурс]

   Режим доступа:

   https://werkzeug.palletsprojects.com/en/0.15.x/utils/ (дата обращения – 15.05.2020)