Министерство цифрового развития, связи и массовых коммуникаций РФ
Уральский технический институт связи и информатики (филиал) ФГБОУ ВО
"Сибирский государственный университет телекоммуникаций и информатики" в г. Екатеринбурге
(УрТИСИ СибГУТИ)

Д.И. Бурумбаев

СЕТЕВОЕ ПРОГРАММИРОВАНИЕ

Методические указания по выполнению практических работ
студентов очной и заочной формы обучения
на базе среднего общего образования,
обучающихся по направлению подготовки бакалавра
09.03.01 «Информатика и вычислительная техника»
(профиль: «Программное обеспечение средств вычислительной техники и автоматизированных систем»),
в соответствии с требованиями ФГОС ВО 3 поколения++

Екатеринбург 2023

УДК 004.42
ББК 32.973.4

Составитель: ст. преподаватель кафедры ИСТ Бурумбаев Д.И.
Рецензент: к.п.н., доцент Зацепин В.А.

Бурумбаев Д.И. Сетевое программирование: методические указания к выполнению практических работ / Д.И. Бурумбаев -Екатеринбург: УрТИСИ СибГУТИ, 2023 – 67 с.

Методические указания по выполнению практических работ по дисциплине «Сетевое программирование» предназначены для студентов очной и заочной форм обучения на базе среднего общего образования, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника» по профилю «Программное обеспечение средств вычислительной техники и автоматизированных систем» изучающих дисциплину «Сетевое программирование».

Выполнение практических работ предусматривает дополнительное изучение материала, расширяет кругозор теоретических знаний дисциплины, получение необходимых навыков работы с литературой.

Методические указания содержат перечень литературы, задания для выполнения практической работы, а также а также список контрольных вопросов для защиты практических работ.

Рекомендовано НМС УрТИСИ СибГУТИ в качестве методических указаний по выполнению практических работ по дисциплине «Сетевое программирование» для студентов очной и заочной формы обучения на базе среднего общего образования, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника» по профилю «Программное обеспечение средств вычислительной техники и автоматизированных систем».

УДК 004.42
ББК 32.973.4

Кафедра Информационных систем и технологий

Содержание

Пояснительная записка

Методические указания к выполнению практических работ составлены в соответствии с рабочей программой дисциплины «Сетевое программирование» предназначены для направления подготовки бакалавров: 09.03.01 «Информатика и вычислительная техника» по профилю «Программное обеспечение средств вычислительной техники и автоматизированных систем».

Выполнение практических работ предусматривает закрепление теоретических знаний курса, получение необходимых навыков самостоятельно работать с литературой, изучение принципов работы с сетевым программным обеспечением. Выполнение практических работ следует производить с необходимыми пояснениями, приводит результат выполненных этапов.

Практические работы оформляются в соответствии с требованиями по оформлению и предоставляется преподавателю для проверки. По результатам проверки и защиты работ студенту выставляется оценка «зачтено» или «не зачтено».

На проведение практических работ в соответствии с программой отводится: 34 часа для очной формы обучения.

Количество часов на выполнение каждой работы, а также ее тема, указаны в таблице.

№ п/п	Наименование лабораторных работ, практических занятий	Объем в часа		
		О	З	$З_Д$
1	Работа с системой управления версиями	4	2	-
2	Разработка API на node.js	4	2	-
3	Работа с авторизацией и аутентификацией	4	-	-
4	Работа с базами данных	6	4	-
5	Разработка блок-схемы работы приложения	4	2	-
6	Разработка интерфейса программного продукта	6	4	-
7	Тест по теме «Клиент-серверная архитектура»	2	-	-
8	Тест по теме «Технологии разработки программного обеспечения»	2	-	-
9	Тест по теме «Микросервисная архитектура»	2	-	-
ВСЕГО		34	14	-

Практическая работа 1

Работа с системой управления версиями

1 Цель работы:

1.1 Научиться работать с системой управления версиями;

1.2 Закрепить знания по теме «Системы управления версиями».

2 Перечень оборудования:

2.1 Персональный компьютер;

2.2 Система управления версиями Git;

2.3 Visual Studio Code.

3.Ход работы:

3.1 Перед началом работы с Git необходимо создать папку с фамилией и номером группы, в которой будут хранится наши элементы. После этого, необходимо создать следующую структуру, как показано на рисунке 1.

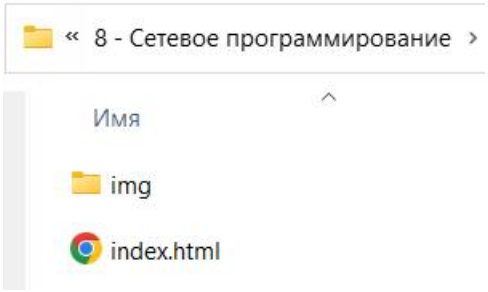


Рисунок 1 – Структура папки

Как видно из структуры, в корневой директории лежат:

1. Основная страница блога index.html
2. Папка img, в которой хранятся изображения (файлы *.jpg) сайта.

Тематика блога выбирается самостоятельно, картинки – тоже, но в рамках общественной нормы.

В качестве примера предполагается, что блог посвящен диким животным. Тогда структура файла index.html будет выглядеть как показано в листинге 1.

Листинг 1 – Структура index.html

```
<!DOCTYPE html>
<html>

  <head>
    <title>Wild Animals Blog</title>
    <meta charset="utf-8">
  </head>

  <body align=center>

    <img width=100px src="pictures/paw_print.jpg">
```

```
    <h1>Wild Animals</h1>
    <small>- Blog about nature -</small>

    <br>
    <br>
    <br>
    <br>

    <p>Let's talk about wild animals around the world:</p>
    <h2>Giraffe</h2>
    <img src="pictures/giraffe.jpg">
    <p><b>Area:</b> Africa</p>
    <p><b>Weight:</b> 900-1200kg</p>
    <p><b>Height:</b> 6m</p>
    <br>

    <h2>Elehant</h2>
    <img width=350px src="pictures/elephant.jpg">
    <p><b>Area:</b> Africa, Asia</p>
    <p><b>Weight:</b> 4000-7000kg</p>
    <p><b>Height:</b> 3m</p>
    <br>

  </body>
</html>
```

Если отрыть данный файл в браузере, то выглядит основная (и пока единственная) страница блога показана на рисунке 2:

Рисунок 2 – Страница блога в браузере

Далее требуется создать репозиторий для нашего блога. Для этого необходимо перейти в директорию будущего репозитория (или открыть при помощи терминала вашу папку)

```
$ cd название вашей папки
```

Далее необходимо прописать команду git. Если у Вас открывается сообщение, как показано на рисунке 3, то значит git установлен. Если подобное сообщение не открывается необходимо установить его в зависимости от Вашей операционной системы с сайта https://git-scm.com/book/ru/v2/Введение-Установка-Git

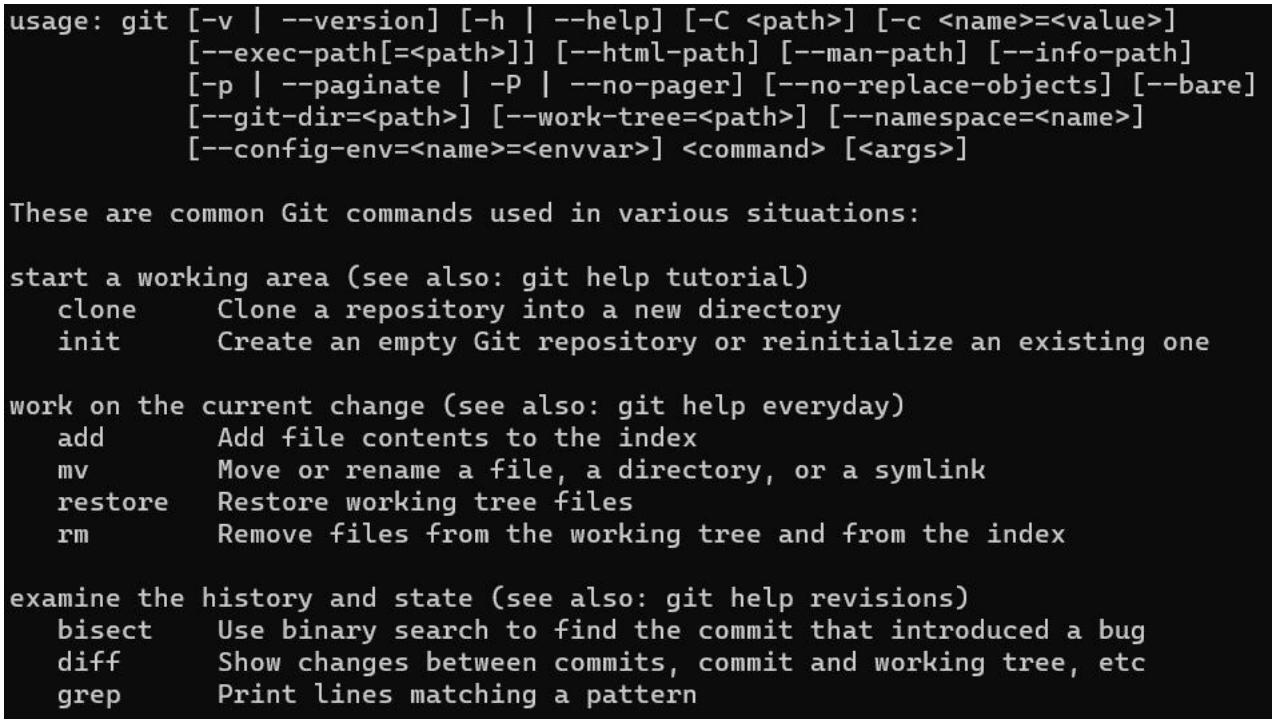


Рисунок 3 – Сообщение, если git установлен

Для создания репозитория необходимо прописать команду git init, после чего появится сообщение:

```
git init
Initialized empty Git repository in /Путь к вашей папке/.git/
```

После такого сообщения, можно проверить структуру git файла, как показано в листинге 2:

Листинг 2 – Проверка структуры файла .git

```
tree .git
.git
├── HEAD
├── config
├── description
```

```
├── hooks
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

Далее необходимо проверить текущее состояние рабочей директории, как показано в листинге 3:

Листинг 3 – Проверка текущего состояния директории

```
git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        img/
        index.html

nothing added to commit but untracked files present (use "git add" to
track)
```

Как видно из листинга, есть не отслеживаемые файлы. Как сделать отслеживаемыми – будет чуть-чуть попозже.

Далее необходимо выполнить следующее задание – настроить пользователя Git на уровне локального репозитория:

1) Изучите содержимое файла конфигурации Git для текущего репозитория (.git/config)

2) Настройте имя и email пользователя для текущего репозитория

3) Убедитесь, что файл .git/config изменился соответствующим образом

Для вывода содержимого файла конфигурации Git для текущего репозитория необходимо воспользоваться командой, как показано в листинге 4.

Листинг 4 – Вывод содержимого файла конфигурации

```
cat .git/config // команда
[core] // отсюда начинается результат
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
```

Далее необходимо задать имя и email пользователя для текущего репозитория при помощи команды, показанной в листинге 5. Значение name и your@email.ru указываются студентами самостоятельно

Листинг 5 – Добавление имени и email пользователя

```
$ git config user.name name
$ git config user.email your@email.ru
```

После этого, необходимо проверить содержимое файла конфигурации при помощи команды, которая была использована в листинге 4. Результат выполнения для тестового задания представлен в листинге 6.

Листинг 6 – Вывод содержимого файла конфигурации

```
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[user]
        name = bdi
        email = bdi@urtisi.ru
```

Следующим заданием будет необходимо создать первый коммит. Для этого необходимо посмотреть состояние рабочей директории, как объяснялось ранее.

После того, как было проверено текущее состояние директории, необходимо добавить все файлы в индекс при помощи команды **git add *.**

После этого, снова необходимо проверить статус репозитория. Если все сделано верно, то файлы попали в индекс и теперь готовы к коммиту, как показано в листинге 7.

Листинг 7 – Статус репозитория после добавления в индекс

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   img/elephant.jpg
        new file:   img/giraphe.jpeg
        new file:   img/paw_print.jpg
        new file:   index.html
```

Далее необходимо выполнить коммит при помощи команды **git commit -m "Project initial"**. После этого должен быть ответ, как показан на листинге 8.

Листинг 8 – Ответ после выполнения коммита

```
git commit -m "G-02: Initial"
[master (root-commit) 6bf1f85] G-02: Initial
 4 files changed, 36 insertions(+)
 create mode 100644 img/elephant.jpg
 create mode 100644 img/giraphe.jpeg
 create mode 100644 img/paw_print.jpg
 create mode 100644 index.html
```

Далее можно проверить последнее изменение при помощи команды **git log,** как показано в листинге 9.

Листинг 9 – Результат выполнения команды git log

```
git log
commit 6bf1f8525f7d730a00d0c93635867e2d40d74042 (HEAD -> master)
Author: bdi <bdi@urtisi.ru>
Date:   Sun Oct 15 22:31:54 2023 +0500

    G-02: Initial
```

В конце необходимо проверить статус репозитория, результат выполнения представлен в листинге 10.

Листинг 10 – Результат выполнения команды git status

```
git status
On branch master
nothing to commit, working tree clean
```

Индивидуальное задание по пункту 3.1. В существующий файл index.html необходимо добавить еще одну картинку и дополнительный текст и сделать второй коммит самостоятельно. Результат выполнения может быть представлен в форме листинга/скриншота.

3.2 Индивидуальное задание студента.

1) Зарегистрируйтесь на GitHub и подтвердите свою почту.

2) Настройте SSH или HTTPS подключение (на ваш выбор). Для того, чтобы настроить SSH, можно прочитать здесь: https://timeweb.cloud/tutorials/windows/kak-sgenerirovat-ssh-klyuch-dlya-windows

3) Отправьте ваш локальный репозиторий (созданный в задании 3.1) на аккаунт GitHub (при помощи командной строки или вспомогательного ПО, не имеет значения).

4) Внесите изменения в ваш репозиторий на локальном компьютере, сделайте коммит и отправьте его на GitHub.

5) Создайте ветку с указанием Вашей фамилии и инициалов, например, BurumbaevDI

6) Далее необходимо переключится на ветку с вашей фамилией, изменить ваш файл index.html и отправить его на GitHub (именно с второй ветки).

7) Необходимо выполнить Pull Request, указав комментарий.

8) После этого необходимо произвести слияние версий от главной ветки (master/main) и второстепенной (в случае примера, это BurumbaevDI). Данный раздел необходимо изучить самостоятельно.

Для того, чтобы выполнить индивидуальное задание №2 необходимо ознакомиться с приложением А к первой практической работе.

Приложение А

Создание репозитория на Github

До текущего момента мы работали с локальным репозиторием, который сохранялся в папке на компьютере. Если мы хотим иметь возможность сохранения проекта в интернете, создадим репозиторий на Github. Для начала нужно зарегистрироваться на сайте github.com под именем myuser (в вашем случае это может быть любое другое имя).

После регистрации нажимаем кнопочку "+" и вводим название репозитория. Выбираем тип Public (репозиторий всегда Public для бесплатной версии) и нажимаем Create.

В результате мы создали репозиторий на сайте Github. На экране мы увидим инструкцию, как соединить наш локальный репозиторий со вновь созданным. Часть команд нам уже знакома.

Добавляем удаленный репозиторий (по протоколу SSH или HTTPS) под именем origin (вместо origin можно использовать любое другое имя).

```
git remote add origin git@github.com:myuser/project.git
```

Можем просмотреть результат добавления с помощью команды:

```
git remote -v
```

Если все было правильно сделано, то увидим:

```
origin git@github.com:myuser/project.git (fetch)
origin git@github.com:myuser/project.git (push)
```

Для того, чтобы отменить регистрацию удаленного репозитария введите:

```
git remote rm origin
```

Это может понадобиться, если вы захотите поменять SSH доступ на HTTPS. После этого можно добавить его опять, например, под именем github и протоколом HTTPS.

```
git remote add github https://github.com/myuser/project.git
```

Следующей командой вы занесете все изменения, которые были сделаны в локальном репозитории на Github.

```
git push --set-upstream origin master
```

Ключ -u используется для того, чтобы установить связь между удаленным репозиторием github и вашей веткой master. Все дальнейшие

изменения вы можете переносить на удаленный репозиторий упрощенной командой.

Другим заданием является ветвление. Ветвление стало неотъемлемой частью командной разработки, потому что оно дает возможность работать над разными версиями исходного кода. Основной идеей ветвления является отклонение от основного кода и продолжение работы независимо от него. Также это удобно в тестировании отдельного функционала, потому что позволяет работать над новой частью кода, не беспокоясь о поломке чего-то в рабочей версии.

Под веткой принято понимать независимую последовательность коммитов в хронологическом порядке. Однако конкретно в Git реализация ветки выполнена как указатель на последний коммит в рассматриваемой ветке. После создания ветки уже новый указатель ссылается на текущий коммит.

Имя основной ветки Git-проекта по умолчанию — master (однако зачастую бывает main, например, в GitHub), она появляется сразу при инициализации репозитория. Эта ветка ничем не отличается от остальных и также ее можно переименовать, но по договоренности master принято считать главной веткой в проекте.

Команда git branch — главный инструмент для работы с ветвлением. С ее помощью можно добавлять новые ветки, перечислять и переименовывать существующие и удалять их.

Чтобы в Git добавить ветку мы используем:

```
$ git branch <name of new branch>
```

После данной операции ветка уже была создана, но вы по-прежнему находитесь в прежней ветке. Если вы планируете переместиться на другую ветку, в том числе только что созданную, необходимо написать checkout:

```
$ git checkout <name of branch>

$ git branch
```

При выполнении этой строки мы получим список существующих веток, где символом * будет отмечена ветка, где вы сейчас находитесь. Это может выглядеть так:

```
 first_branch
* master
  second_branch
```

С помощью параметра -v можно получить последний сохраненный коммит в каждой ветке.

```
$ git branch -v
  first_branch 8fa301b Fix math
* master 225cc2d Merge branch 'first_branch'
  second_branch c56ee12 Refactor code style
```

После того как вы выполнили создание ветки и изменили документ, Вам необходимо отправить Pull Request.

Pull Request — запрос на включение. На включение написанного вами кода в чужой репозиторий.

Если нужно отправить в свой удалённый репозиторий вновь созданную ветку (не сливать её с master), делаем следующее:

```
$ git push master new_branch
```

Добрались таки до ответа на поставленный вопрос: что такое pull request, зачем оно нужно и как его достичь. Как предложить владельцу репозитория свои изменения?

Для этого зайдите в свой аккаунт, выбирайте репозиторий владельца и ищите небольшую зелёную кнопку.

Перед тем как сделать запрос вы имеете возможность добавить комментарий, просмотреть то, какие файлы будут изменены, какие коммиты добавлены. В верхнем углу окна добавления запроса обратите внимание откуда куда и что вы сливаете. Если необходимо слить основные ветки выбор падёт на репозиторий username:master, если отдельную ветку (вспоминаем branch) — так и указывайте её.

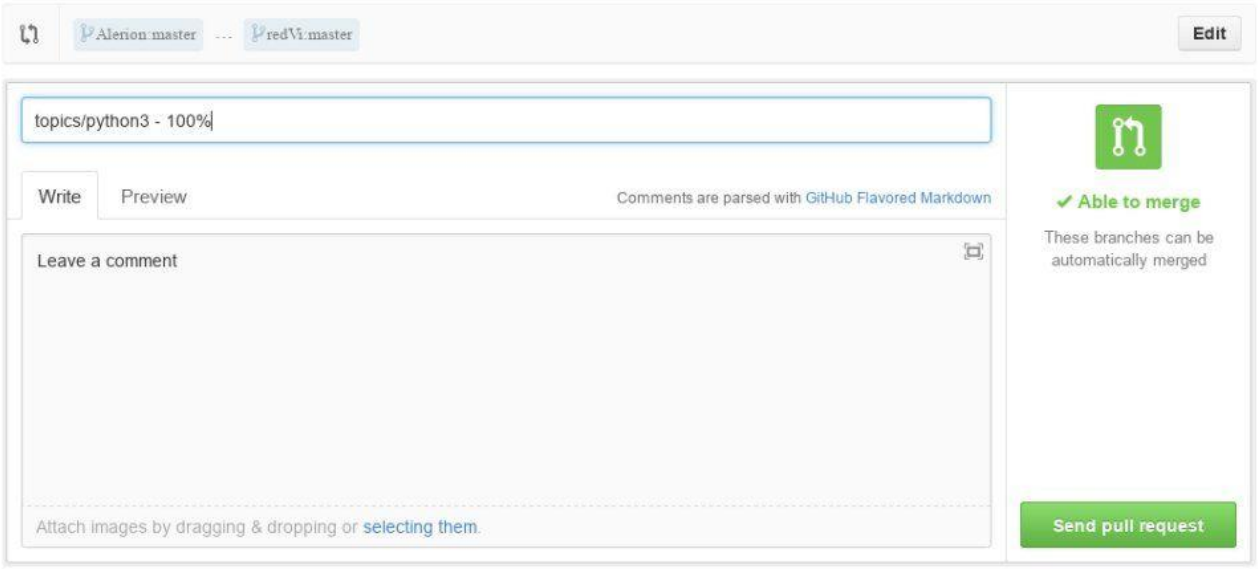


Рисунок 4 – Отправка Pull Request

Практическая работа 2

Разработка API на node.js

1 Цель работы:

1.1 Научиться работать с API;

1.2 Закрепить знания по теме «Знакомство с API».

2 Перечень оборудования:

2.1 Персональный компьютер;

2.2 Postman;

2.3 node.js;

2.4 Visual Studio Code.

3.Ход работы:

3.1 Интерфейсы прикладного программирования или API (Application Programming Interface) применяются в разработке повсеместно. Они позволяют одним программам последовательно взаимодействовать с другими – внутренними или внешними – программными компонентами. Это является ключевым условием масштабируемости, не говоря уже о возможности повторного использования приложений.

В настоящее время довольно распространены онлайн-сервисы, использующие общедоступные API. Они дают возможность другим разработчикам легко интегрировать такие функции, как авторизация через соцсети, платежи кредитной картой и отслеживание поведения.

Применяемый при этом стандарт де-факто называется «передачей состояния представления» (REpresentational State Transfer) или сокращённо REST. Простыми словами, REST API – это набор правил, по которым серверные приложения взаимодействуют с клиентскими.

Для создания простого, но безопасного бэкенда на основе REST API может быть задействовано множество платформ и языков программирования, например, ASP.NET Core, Laravel (PHP) или Bottle (Python).

В качестве примера будет использован следующий стек-технологий:

1) js — как пример распространённой кроссплатформенной среды выполнения JavaScript;

2) Express, который значительно упрощает выполнение основных задач веб-сервера в Node.js и является стандартным инструментом для создания серверной части на основе REST API;

3) Mongoose, который будет соединять наш бэкенд с базой данных MongoDB.

Чтобы понять, как работает REST API, нужно подробнее рассмотреть, что представляет собой стиль архитектуры программного обеспечения REST, представленный на рисунке 1.

REST API используются для доступа к данными и их обработки с помощью стандартного набора операций без сохранения состояния. Эти

операции являются неотъемлемой частью протокола HTTP. Они представляют собой основные функции создания («create»), чтения («read»), модификации («update») и удаления («delete») и обозначаются акронимом CRUD.

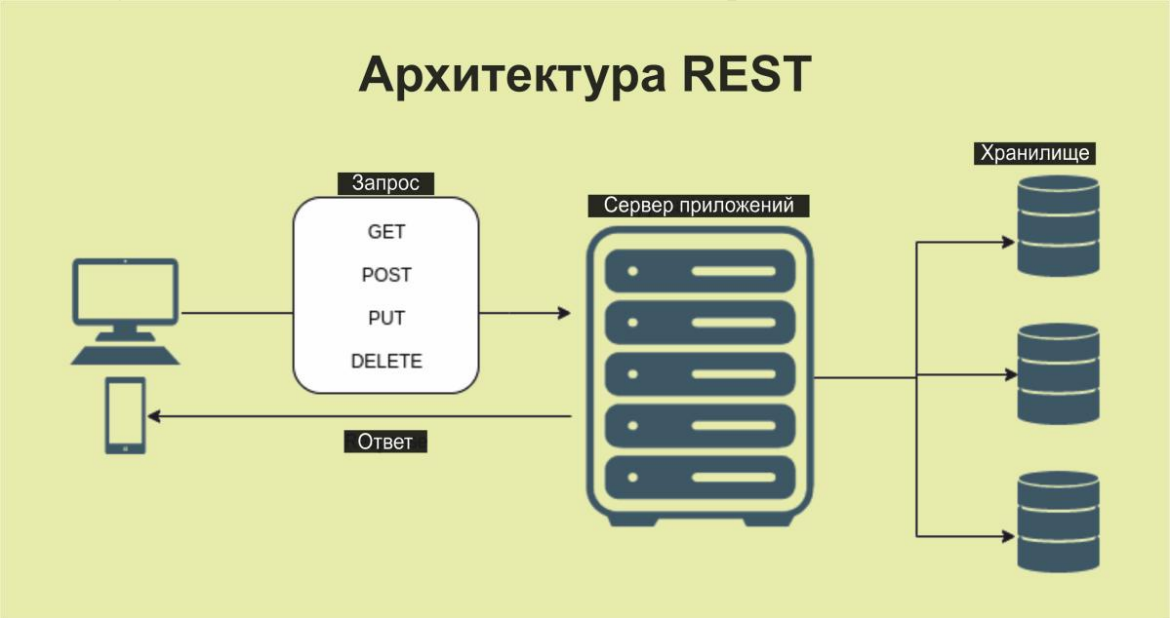


Рисунок 1 – Архитектура REST

Операциям REST API соответствуют, хотя и не полностью идентичны, следующие методы HTTP:

- POST (создание ресурса или предоставление данных в целом).
- GET (получение индекса ресурсов или отдельного ресурса).
- PUT (создание и замена ресурса).
- PATCH (обновление/изменение ресурса).
- DELETE (удаление ресурса).

С помощью этих HTTP-методов и имени ресурса в качестве адреса, мы можем построить REST API, создав конечную точку для каждой операции. В результате получается стабильная и легко понятная основа, которая позволит быстро дорабатывать код и осуществлять его дальнейшее сопровождение.

Та же основа будет применяться для интеграции сторонних функций, о которых было сказано чуть выше. Большинство из них тоже использует REST API, что ускоряет такую интеграцию.

Для хранения данных фиктивных пользователей в примере к практического задания будет использовано MySQL DB, но также можно использовать другие.

В начале необходимо создать базу данных, имя которой будет соответствовать № группы и фамилии обучающегося. В данной базе данных необходимо создать таблицу:

```
Имя базы данных – group_familia
Имя таблицы – users
```

Для создания таблицы необходимо воспользоваться SQL-командой, которая представлена на рисунке 1.

```
CREATE TABLE users (
  id int(11) NOT NULL AUTO_INCREMENT,
  name varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  email varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  password varchar(200) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY email (email)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Рисунок 1 – SQL команда для создания таблицы

В результат будет получен результат, представленный на рисунке 2.

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	id	int(11)			No	None		AUTO_INCREMENT
☐	2	name	varchar(50)	utf8mb4_unicode_ci		No	None		
☐	3	email	varchar(50)	utf8mb4_unicode_ci		No	None		
☐	4	password	varchar(200)	utf8mb4_unicode_ci		No	None		

Рисунок 2 – Результат выполнения SQL-команды

Теперь необходимо создать новую папку на вашем рабочем столе (или в другом месте) под названием familia-rest-api и открыть данную папку в node.js/Visual Studio Code.

После инициализации в среде разработки установите следующие пакеты:

```
npm i express express-validator mysql2 jsonwebtoken bcryptjs
```

После установки вышеуказанных пакетов node файл package.json выглядит так, как показано в листинге 1.

Листинг 1 – Файл package.json

```
{
  "name": "node-login-rest-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "chandan tudu",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
```

```
    "express": "^4.17.1",
    "express-validator": "^6.9.2",
    "jsonwebtoken": "^8.5.1",
    "mysql2": "^2.2.5"
  }
}
```

Далее необходимо создать структуру разрабатываемого API, которая представлена на рисунке 3.

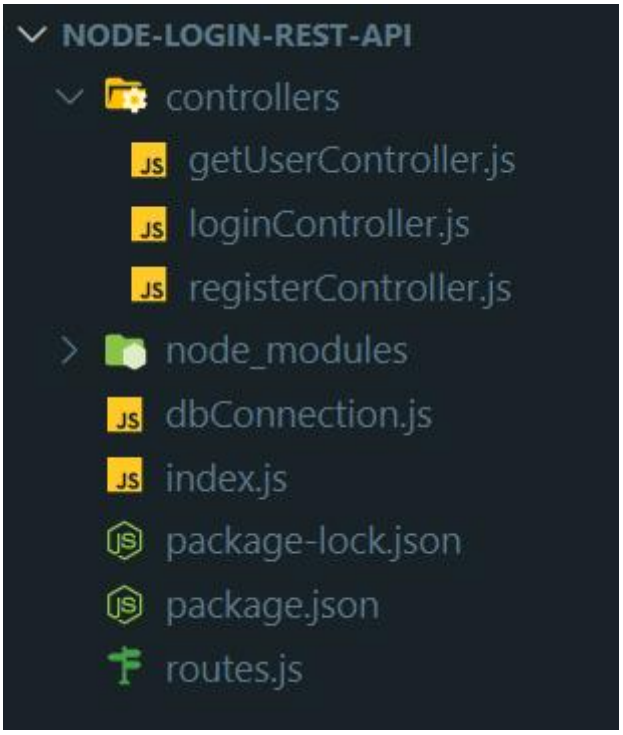


Рисунок 3 – Структура проекта

Файл dbConnection.js отвечает за связь с базой данных. Для того, чтобы произошла связка в нем необходимо прописать следующий код, представленный в листинге 2:

Листинг 2 – Файл dbConnectrion.js

```
const mysql = require("mysql2");

const db_connection = mysql
  .createConnection({
    host: "", // имя хоста
    user: "", // имя пользователя
    database: "", // имя базы данных
    password: "", // пароль от базы данных
  })
  .on("error", (err) => {
    console.log("Failed to connect to Database - ", err);
  });

module.exports = db_connection;
```

После установления соединения с БД теперь необходимо настроить маршруты, и для этого нужно в файле routes.js в корневом каталоге прописать код, представленный в листинге 3:

Листинг 3 – Код в файле routes.js

```
    const router = require('express').Router();
    const {body} = require('express-validator');
    const {register} = require('./controllers/registerController');
    const {login} = require('./controllers/loginController');
    const {getUser} = require('./controllers/getUserController');

    router.post('/register', [
        body('name',"The name must be of minimum 3 characters
length")
        .notEmpty()
        .escape()
        .trim()
        .isLength({ min: 3 }),
        body('email',"Invalid email address")
        .notEmpty()
        .escape()
        .trim().isEmail(),
        body('password',"The Password must be of minimum 4
characters length").notEmpty().trim().isLength({ min: 4 }),
    ], register);


    router.post('/login',[
        body('email',"Invalid email address")
        .notEmpty()
        .escape()
        .trim().isEmail(),
        body('password',"The Password must be of minimum 4
characters length").notEmpty().trim().isLength({ min: 4 }),
    ],login);

    router.get('/getuser',getUser);

    module.exports = router;
```

Внутри папки controllers необходимо создать три контроллера:

1) registerController.js – для вставки нового пользователя.
2) loginController.js – для пользователя, входящего в систему.
3) getUserController.js – для получения сведений о пользователе с использованием токена JWT.

Для первого файла листинг кода будет иметь следующий вид (листинг 4):

Листинг 4 – Код для registerController.js

```
    const {validationResult} = require('express-validator');
    const bcrypt = require('bcryptjs');
    const conn = require('../dbConnection').promise();

    exports.register = async(req,res,next) => {
```

```
        const errors = validationResult(req);

        if(!errors.isEmpty()){
            return res.status(422).json({ errors: errors.array() });
        }

        try{

            const [row] = await conn.execute(
                "SELECT `email` FROM `users` WHERE `email`=?",
                [req.body.email]
              );

            if (row.length > 0) {
                return res.status(201).json({
                    message: "The E-mail already in use",
                });
            }

            const hashPass = await bcrypt.hash(req.body.password,
12);

            const [rows] = await conn.execute('INSERT INTO
`users`(`name`,`email`,`password`) VALUES(?,?,?)',[
                req.body.name,
                req.body.email,
                hashPass
            ]);

            if (rows.affectedRows === 1) {
                return res.status(201).json({
                    message: "The user has been successfully
inserted.",
                });
            }

        }catch(err){
            next(err);
        }
    }
```

Для файла loginController.js листинг кода будет иметь следующий вид (листинг 5):

Листинг 5 – Код для loginController.js

```
    const jwt = require('jsonwebtoken');
    const bcrypt = require('bcryptjs');
    const {validationResult} = require('express-validator');
    const conn = require('../dbConnection').promise();


    exports.login = async (req,res,next) =>{
        const errors = validationResult(req);

        if(!errors.isEmpty()){
            return res.status(422).json({ errors: errors.array() });
```

```
        }

        try{

            const [row] = await conn.execute(
                "SELECT * FROM `users` WHERE `email`=?",
                [req.body.email]
              );

            if (row.length === 0) {
                return res.status(422).json({
                    message: "Invalid email address",
                });
            }

            const            passMatch            =            await
bcrypt.compare(req.body.password, row[0].password);
            if(!passMatch){
                return res.status(422).json({
                    message: "Incorrect password",
                });
            }

            const  theToken  =  jwt.sign({id:row[0].id},'the-super-
strong-secrect',{ expiresIn: '1h' });

            return res.json({
                token:theToken
            });

        }
        catch(err){
            next(err);
        }
    }
```

Для getUserController.js код представлен в листинге 6:

```
    const jwt = require('jsonwebtoken');
    const conn = require('../dbConnection').promise();

    exports.getUser = async (req,res,next) => {

        try{

            if(
                !req.headers.authorization ||
                !req.headers.authorization.startsWith('Bearer') ||
                !req.headers.authorization.split(' ')[1]
            ){
                return res.status(422).json({
                    message: "Please provide the token",
                });
            }

            const   theToken   =   req.headers.authorization.split('
')[1];
```

```
            const decoded = jwt.verify(theToken, 'the-super-strong-
secrect');

            const [row] = await conn.execute(
                "SELECT `id`,`name`,`email` FROM `users` WHERE
`id`=?",
                [decoded.id]
            );

            if(row.length > 0){
                return res.json({
                    user:row[0]
                });
            }

            res.json({
                message:"No user found"
            });

        }
        catch(err){
            next(err);
        }
    }
```

В заключении необходимо создать главный JS-файл с названием index.js, код для которого представлен в листинге 7:

```
    const express = require('express');
    const routes = require('./routes');
    const app = express();

    app.use(express.json());
    app.use(routes);
    // Handling Errors
    app.use((err, req, res, next) => {
        // console.log(err);
        err.statusCode = err.statusCode || 500;
        err.message = err.message || "Internal Server Error";
        res.status(err.statusCode).json({
          message: err.message,
        });
    });

    app.listen(3000,() => console.log('Server is running on port
3000'));
```

В заключении, индивидуальным заданием каждого студента является тестирование разработанного API при помощи программного обеспечения Postman.

Для этого необходимо при помощи GET и POST запросов произвести тестирование на: регистрацию, авторизацию, получение данных о пользователе и получение данных о токене. Количество созданных

пользователей в базе данных должно быть равно 3, первым из которых является студент, выполняющий практическую работу.

4. Контрольные вопросы:

4.1 Что такое API и какую роль оно играет в разработке программного обеспечения?

4.2 Какие принципы проектирования API следует учитывать при его разработке?

4.3 Что такое RESTful API и какие основные принципы он соблюдает?

4.4 Какую роль играет формат данных (например, JSON или XML) при разработке API?

4.5 Какие механизмы аутентификации и авторизации могут использоваться в API?

4.6 Как можно предоставить документацию для API, чтобы облегчить работу разработчикам?

4.7 Какие инструменты и технологии используются для разработки и тестирования API?

5. Содержание отчета:

5.1 Титул

5.2 Цель работы

5.3 Ход реализации работы

5.4 Ответы на контрольные вопросы.

Практическая работа 3

Работа с авторизацией и аутентификацией

1 Цель работы:

1.1 Научиться работать с регистрацией, авторизацией и аутентификацией;

1.2 Закрепить знания по теме «Аутентификация и авторизация пользователей в клиент-серверных приложениях».

2 Перечень оборудования:

2.1 Персональный компьютер;

2.2 Postman;

2.3 node.js;

2.4 Visual Studio Code.

3.Ход работы:

3.1 Для реализации системы регистрации, авторизации и аутентификации необходимо выполнить ряд действий. Первым делом необходимо настроить базу данных. Для примера, можно использовать базу данных MySQL при помощи XAMPP, WAMP или другого ПО. Внутри необходимо создать базу данных familia_login. Затем внутри этой базы данных нужно создать таблицу с именем users при помощи SQL-запроса, который представлен в листинге 1.

Листинг 1 – SQL-запрос для создания таблицы

```
CREATE TABLE users (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  name varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  email varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  password varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY email (email)
 )    ENGINE=InnoDB    AUTO_INCREMENT=2    DEFAULT    CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Далее в качестве примера представлена реализация при помощи node.js и JavaScript. Для начало необходимо установить NPM при помощи команды:

Листинг 2 – Инициализация NPM

```
npm i express ejs express-session express-validator bcryptjs mysql2
```

А также изменить созданный файл package.json

Листинг 3 – Package.json

```
{
  "name": "nodejs-login-registration",
  "version": "1.0.0",
  "description": "Node JS Login And Registration System",
  "main": "index.js",
```

```
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "w3jar.com (chandan)",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "ejs": "^3.1.6",
    "express": "^4.17.1",
    "express-session": "^1.17.2",
    "express-validator": "^6.11.1",
    "mysql2": "^2.2.5"
  }
}
```

Далее необходимо создать структуру для реализации системы как показано на рисунке 1.

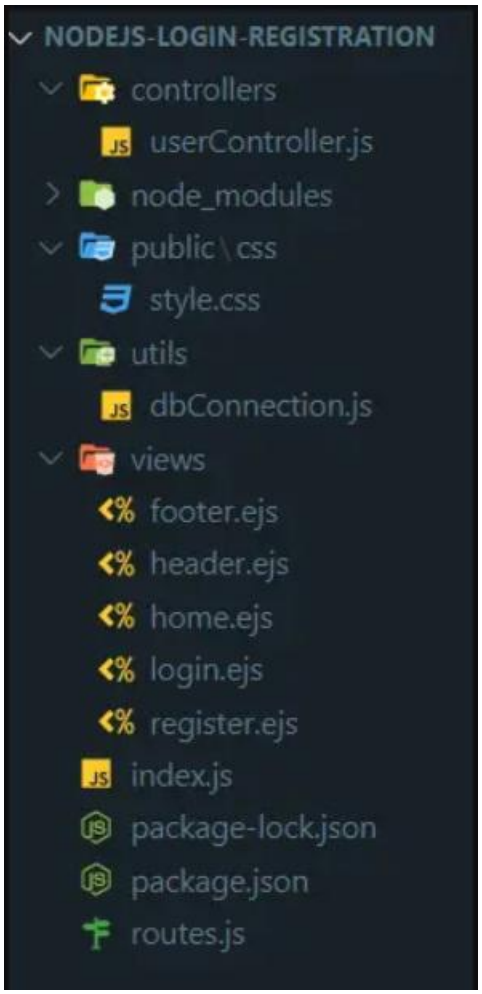


Рисунок 1 – Структура проекта

Далее для подключения базы данных к проекту необходимо в файле dbConnection.js необходимо прописать код, представленный в листинге 4.

Листинг 4 – Код в файле dbConnection

```
const mysql = require('mysql2');

const dbConnection = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'nodejs_login'
});

module.exports = dbConnection.promise();
```

Далее необходимо наполнить каталог views. Описанные далее файлы будут находиться в этой папке. Содержимое файлов представлено в листингах 5-10.

Листинг 5 – Код в файле header.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title><%= (typeof(title) !== 'undefined') ? title : 'Node JS
Login and Registration System - W3jar' %></title>
    <link rel="preconnect" href="https://fonts.gstatic.com">
    <link
href="https://fonts.googleapis.com/css2?family=Ubuntu:wght@400;700&d
isplay=swap" rel="stylesheet">
    <link rel="stylesheet" href="./style.css">
</head>
<body>
```

Листинг 6 – Код в файле footer.ejs

```
</body>
</html>
```

Листинг 7 – Код в файле login.ejs

```
<%- include('header',{
    title:'Login'
}); -%>

<div class="container">
    <h1>Login</h1>
    <form action="" method="POST">
        <label for="user_email">Email</label>
        <input type="email" class="input" name="_email"
id="user_email" placeholder="Enter your email">
        <label for="user_pass">Password</label>
        <input type="password" class="input" name="_password"
id="user_pass" placeholder="Enter new password">
        <% if(typeof error !== 'undefined'){ %>
        <div class="err-msg"><%= error %></div><% } %>
```

```
        <input type="submit" value="Login">
        <div class="link"><a href="./signup">Sign Up</a></div>
    </form>
</div>

<%- include('footer'); -%>
```

Листинг 8 – Код в файле register.ejs

```
<%- include('header',{
    title:'Signup'
}); -%>

<div class="container">
    <h1>Sign Up</h1>
    <form action="" method="POST">
        <label for="user_name">Name</label>
        <input type="text" class="input" name="_name" id="user_name"
placeholder="Enter your name">
        <label for="user_email">Email</label>
        <input      type="email"      class="input"      name="_email"
id="user_email" placeholder="Enter your email">
        <label for="user_pass">Password</label>
        <input    type="password"    class="input"    name="_password"
id="user_pass" placeholder="Enter new password">
        <% if(typeof msg != 'undefined'){ %>
        <div class="success-msg"><%= msg %></div><% }
        if(typeof error !== 'undefined'){ %>
        <div class="err-msg"><%= error %></div><% } %>
        <input type="submit" value="Sign Up">
        <div class="link"><a href="./login">Login</a></div>
    </form>
</div>

<%- include('footer'); -%>
```

Листинг 9 – Код в файле home.ejs

```
<%- include('header'); -%>
<div class="container">
    <div class="profile">
        <div class="img">&#128100;</div>
        <h2><%= user.name %></h2>
        <span><%= user.email %></span>
        <a href="/logout">Log Out</a>
    </div>
</div>
<%- include('footer'); -%>
```

Листинг 10 – Код в файле routes.js

```
const router = require("express").Router();
const { body } = require("express-validator");

const {
    homePage,
    register,
    registerPage,
    login,
```

```
    loginPage,
} = require("./controllers/userController");

const ifNotLoggedin = (req, res, next) => {
    if(!req.session.userID){
        return res.redirect('/login');
    }
    next();
}

const ifLoggedin = (req,res,next) => {
    if(req.session.userID){
        return res.redirect('/');
    }
    next();
}

router.get('/', ifNotLoggedin, homePage);

router.get("/login", ifLoggedin, loginPage);
router.post("/login",
ifLoggedin,
    [
        body("_email", "Invalid email address")
            .notEmpty()
            .escape()
            .trim()
            .isEmail(),
        body("_password",  "The  Password  must  be  of  minimum  4
characters length")
            .notEmpty()
            .trim()
            .isLength({ min: 4 }),
    ],
    login
);

router.get("/signup", ifLoggedin, registerPage);
router.post(
    "/signup",
    ifLoggedin,
    [
        body("_name",  "The  name  must  be  of  minimum  3  characters
length")
            .notEmpty()
            .escape()
            .trim()
            .isLength({ min: 3 }),
        body("_email", "Invalid email address")
            .notEmpty()
            .escape()
            .trim()
            .isEmail(),
        body("_password",  "The  Password  must  be  of  minimum  4
characters length")
            .notEmpty()
            .trim()
            .isLength({ min: 4 }),
```

```
    ],
    register
);

router.get('/logout', (req, res, next) => {
    req.session.destroy((err) => {
        next(err);
    });
    res.redirect('/login');
});

module.exports = router;
```

Далее необходимо в корневом каталоге приложения создать папку controllers и внутри этой папки создать файл userController.js, содержание которого представлено в листинге 11.

Листинг 11 – Код в файле userController.ejs

```
const { validationResult } = require("express-validator");
const bcrypt = require('bcryptjs');
const dbConnection = require("../utils/dbConnection");

// Home Page
exports.homePage = async (req, res, next) => {
    const [row] = await dbConnection.execute("SELECT * FROM `users` 
WHERE `id`=?", [req.session.userID]);

    if (row.length !== 1) {
        return res.redirect('/logout');
    }

    res.render('home', {
        user: row[0]
    });
}

// Register Page
exports.registerPage = (req, res, next) => {
    res.render("register");
};

// User Registration
exports.register = async (req, res, next) => {
    const errors = validationResult(req);
    const { body } = req;

    if (!errors.isEmpty()) {
        return res.render('register', {
            error: errors.array()[0].msg
        });
    }

    try {

        const [row] = await dbConnection.execute(
            "SELECT * FROM `users` WHERE `email`=?",
```

```
            [body._email]
        );

        if (row.length >= 1) {
            return res.render('register', {
                error: 'This email already in use.'
            });
        }

        const hashPass = await bcrypt.hash(body._password, 12);

        const [rows] = await dbConnection.execute(
            "INSERT      INTO      `users`(`name`,`email`,`password`)
VALUES(?,?,?)",
            [body._name, body._email, hashPass]
        );

        if (rows.affectedRows !== 1) {
            return res.render('register', {
                error: 'Your registration has failed.'
            });
        }

        res.render("register", {
            msg: 'You have successfully registered.'
        });

    } catch (e) {
        next(e);
    }
};

// Login Page
exports.loginPage = (req, res, next) => {
    res.render("login");
};

// Login User
exports.login = async (req, res, next) => {

    const errors = validationResult(req);
    const { body } = req;

    if (!errors.isEmpty()) {
        return res.render('login', {
            error: errors.array()[0].msg
        });
    }

    try {

        const  [row]  =  await  dbConnection.execute('SELECT  *  FROM
`users` WHERE `email`=?', [body._email]);

        if (row.length != 1) {
            return res.render('login', {
                error: 'Invalid email address.'
            });
```

```
        }

        const checkPass = await bcrypt.compare(body._password,
row[0].password);

        if (checkPass === true) {
            req.session.userID = row[0].id;
            return res.redirect('/');
        }

        res.render('login', {
            error: 'Invalid Password.'
        });


    }
    catch (e) {
        next(e);
    }

}
```

Для добавления стилистики нашего приложения необходимо добавить файл style.css внутри общей папки приложения. Код представлен в листинге 12.

Листинг 12 – Код в файле style.css

```
*,
*::before,
*::after {
    box-sizing: border-box;
}

html {
    -webkit-text-size-adjust: 100%;
    -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
    font-size: 16px;
}

body {
    background-color: #f7f7f7;
    font-family: "Ubuntu", sans-serif;
    margin: 0;
    padding: 0;
    color: #222222;
    overflow-x: hidden;
    overflow-wrap: break-word;
    -moz-osx-font-smoothing: grayscale;
    -webkit-font-smoothing: antialiased;
    padding: 50px;
}

.container {
    background-color: white;
    max-width: 450px;
    margin: 0 auto;
```

```
    padding: 40px;
    box-shadow: 0 1rem 3rem rgba(0, 0, 0, 0.175);
    border-radius: 3px;
}

.container h1 {
    margin: 0 0 40px 0;
    text-align: center;
}

input,
button {
    font-family: "Ubuntu", sans-serif;
    outline: none;
    font-size: 1rem;
}

.input {
    padding: 10px;
    width: 100%;
    margin-bottom: 10px;
    border: 1px solid #bbbbbb;
    border-radius: 3px;
}

.input:hover {
    border-color: #999999;
}

.input:focus {
    border-color: #0d6efd;
}

[type="submit"] {
    background: #0d6efd;
    color: white;
    border: 1px solid rgba(0, 0, 0, 0.175);
    border-radius: 3px;
    padding: 12px 0;
    cursor: pointer;
    box-shadow: 0 0.125rem 0.25rem rgba(0, 0, 0, 0.075);
    margin-top: 5px;
    font-weight: bold;
    width: 100%;
}

[type="submit"]:hover {
    box-shadow: 0 0.5rem 1rem rgba(0, 0, 0, 0.15);
}

label {
    font-weight: bold;
}

.link {
    margin-top: 10px;
    text-align: center;
}
```

```
.link a {
    color: #0d6efd;
}

.success-msg,
.err-msg {
    color: #dc3545;
    border: 1px solid #dc3545;
    padding: 10px;
    border-radius: 3px;
}

.success-msg {
    color: #ffffff;
    background-color: #20c997;
    border-color: rgba(0, 0, 0, 0.1);
}

.profile {
    text-align: center;
}

.profile .img {
    font-size: 50px;
}

.profile h2 {
    margin-bottom: 3px;
    text-transform: capitalize;
}

.profile span {
    display: block;
    margin-bottom: 20px;
    color: #999999;
}

.profile a {
    display: inline-block;
    padding: 10px 20px;
    text-decoration: none;
    border: 1px solid #dc3545;
    color: #dc3545;
    border-radius: 3px;
}

.profile a:hover {
    border-color: rgba(0, 0, 0, 0.1);
    background-color: #dc3545;
    color: #ffffff;
}
```

Последним действием является оформление главного файла index.js, представленного в листинге 13.

Листинг 13 – Код в файле index.js

```
const express = require('express');
const session = require('express-session');
const path = require('path');
const routes = require('./routes');
const app = express();

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(express.urlencoded({ extended: false }));
app.use(session({
    name: 'session',
    secret: 'my_secret',
    resave: false,
    saveUninitialized: true,
    cookie: {
        maxAge: 3600 * 1000, // 1hr
    }
}));

app.use(express.static(path.join(__dirname, 'public')));
app.use(routes);

app.use((err, req, res, next) => {
    // console.log(err);
    return res.send('Internal Server Error');
});

app.listen(3000, () => console.log('Server is runngin on port
3000'));
```

3.2 Далее в качестве индивидуального задания 1 необходимо проверить работоспособность полученного приложения. Результаты занести скриншотами из базы данных в отчет

3.3 В качестве индивидуального задания 2 необходимо интегрировать дополнительный способ авторизации через Google и Github.

Для авторизации через Google необходимо инициализировать дополнительные модули через команду, представленную в листинге 14.

Листинг 14 – Установка дополнительных модулей

```
npm install express passport passport-google-oauth2 cookie-session
```

Далее необходимо добавить в корневую структуру добавить файл passport.js, содержащий код, представленный в листинге 15.

Листинг 15 – Код файла passport.js

```
const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth2').Strategy;

passport.serializeUser((user , done) => {
    done(null , user);
})
```

```
passport.deserializeUser(function(user, done) {
    done(null, user);
});

passport.use(new GoogleStrategy({
    clientID:"YOUR ID", // Данные из вашего аккаунта.
    clientSecret:"YOUR SECRET", // Данные из вашего аккаунта.
    callbackURL:"http://localhost:4000/auth/callback",
    passReqToCallback:true
  },
  function(request, accessToken, refreshToken, profile, done) {
    return done(null, profile);
  }
));
```

Для получения данных с платформы Google необходимо:

1) Перейти на сайт https://console.cloud.google.com/ и зайти в раздел API's & Services

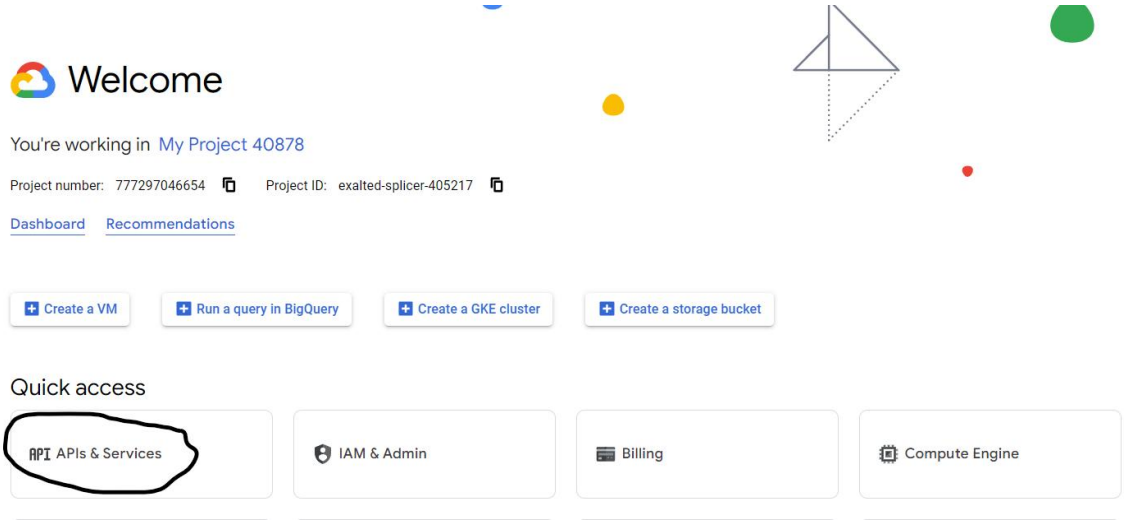


Рисунок 2 – Главная страница Google Cloud

Далее необходимо зайти во вкладку Credentials, а затем Create Credentials и выбрать Create OAuth client ID. Далее необходимо выбрать тип приложения "Web application" (при условии разработки веб-приложения). В ячейке «Authorized redirect URIs» нажать вкладку «Add URL» и добавить две ссылки, как показано на рисунке 3 и нажать кнопку «Create».

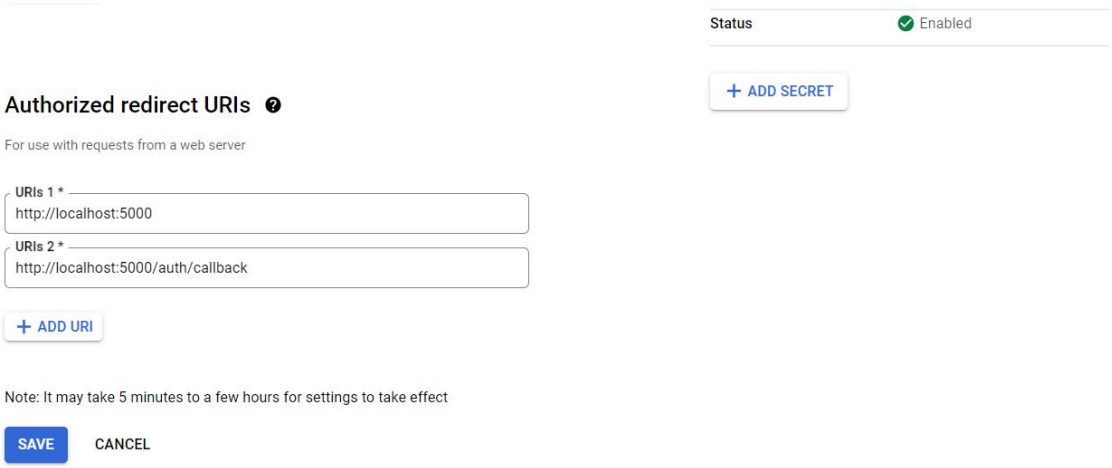


Рисунок 3 – Добавление ссылок

После этого появится ID и Secret, как показано на рисунке 4, которые необходимо добавить в код.

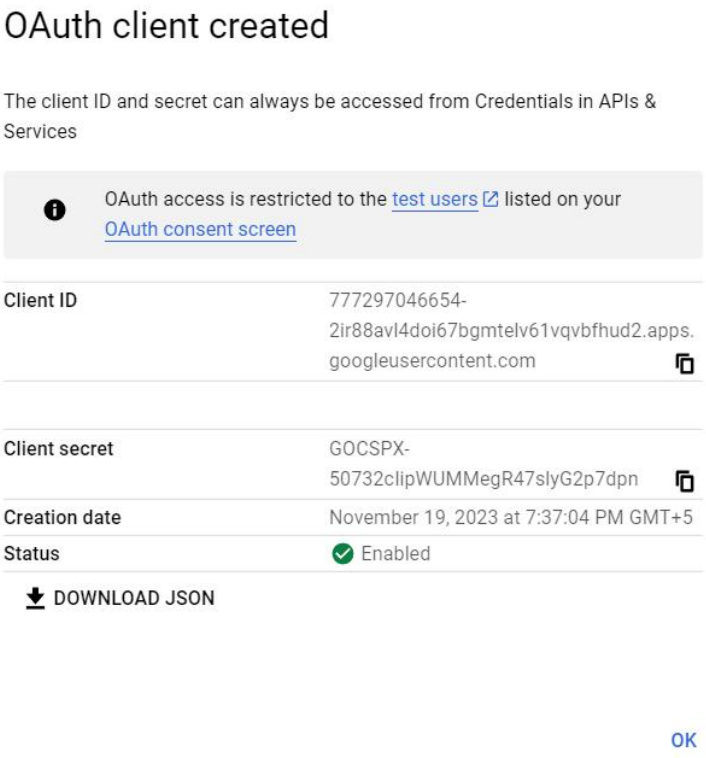


Рисунок 4 – Полученные результаты аутентификации

Авторизацию через github необходимо изучить самостоятельно и интегрировать в ваше решение.

4. Контрольные вопросы:

4.1 Какова роль аутентификации и авторизации в клиент-серверных приложениях?

4.2 Какие методы аутентификации можно использовать в клиент-серверных приложениях?

4.3 Как обеспечить безопасность процесса аутентификации в клиент-серверных приложениях?

4.4 Какие технологии и протоколы используются для обеспечения безопасности при аутентификации в клиент-серверных приложениях?

4.5 Что такое многофакторная аутентификация и каковы ее преимущества в клиент-серверных приложениях?

4.6 Какие основные принципы должны соблюдаться при осуществлении процесса авторизации в клиент-серверных приложениях?

4.7 Как обеспечить защиту от несанкционированного доступа к ресурсам в клиент-серверных приложениях?

4.8 Каким образом можно реализовать сессионную аутентификацию в клиент-серверных приложениях?

4.9 Как обеспечить безопасность при передаче учетных данных между клиентом и сервером в клиент-серверных приложениях?

4.10 Какие методы могут использоваться для обработки ошибок аутентификации и авторизации в клиент-серверных приложениях?

5. Содержание отчета:

5.1 Титул

5.2 Цель работы

5.3 Ход выполнения работы

5.4 Ответы на контрольные вопросы.

Практическая работа №4

Работа с базами данных

1 Цель работы:

1.3 Научиться работать с базами данных;

1.4 Закрепить знания по теме «Основы работы с базами данных».

2 Перечень оборудования:

2.1 Персональный компьютер;

2.2 СУБД;

2.3 Visual Studio Code.

3.Ход работы:

3.1 Перед началом выполнения работы необходимо ознакомиться с материалами, представленными в приложении А.

3.2 После изучения теоретического материала необходимо выполнить индивидуальное задание, которое получается у преподавателя.

4. Контрольные вопросы:

4.1 Пояснить термин «реляционная база данных».

4.2 Для чего необходима система разграничения прав пользователей?

4.3 Какие основные привилегии разрешают доступ к БД?

4.4 Для чего применяются ключевые поля?

4.5 Сколько ключевых полей и полей ссылок может содержать таблица?

4.6 Каким способом можно поменять структуру таблицы без применения команды ALTER?

4.7 В чем различие операторов INSERT и LOAD DATA?

5. Содержание отчета:

5.1 Титул

5.2 Цель работы

5.3 Ход реализации работы

5.4 Ответы на контрольные вопросы.

Приложение А

Пример проектирования базы данных MySQL (пример с библиотекой)

С точки зрения клиента — библиотека является местом, где можно получить книгу, а затем сдать ее. Некоторые клиенты пользуются возможностью самостоятельного подбора литературы в информационной системе библиотеки. Пользователь не задумывается о том, откуда в системе появляются новые книги, но их туда вносит библиотекарь. Также, информационная система позволяет ему находить читателей с задолженностями и маловостребованные книги. От обычного посетителя библиотеки полностью скрыта роль администратора информационной системы.

Реальная информационная библиотечная система представляет собой большую и сложную систему, предусматривающую возможность параллельной работы тысяч пользователей и интегрирующуюся с другими библиотечными системами.

А.1 Инфологическое проектирование

А.1.1 Анализ предметной области и информационных задач пользователей

Основная задача любой библиотеки – обработка книжного фонда. Нетрудно выделить три основные группы пользователей системы: читатель, библиотекарь, администратор. Деятельность каждого из них показана на диаграмме вариантов использования, представленная на рисунке А.1.

Уже сейчас можно выделить некоторые сущности и отношения будущей базы данных.

При таком подходе не понятно, как именно связать читателя с книгой. Если книга имеет несколько экземпляров – то она может быть выдана нескольким читателям. Даже если же под книгой понимать один экземпляр – то при сохранении в таблице книг текущего читателя приведет к невозможности получения информации о том, кто (и сколько раз) брал эту книгу ранее.

Решением может быть введение дополнительной переменной – карточки о выдаче книги. При выдаче книги читателю заводится карточка, а при сдаче книги – в нее ставится соответствующая пометка. С помощью этих карточек определяются задолженности каждого пользователя и вычисляется статистика использования книг.

При бронировании литературы читателем – также заводится карточка, если забронированная литература не взята читателем в определенный срок – карточка уничтожается. Существует ограничение на количество книг, которые может забронировать читатель.

При подборе литературы пользователь просматривает каталог литературы с возможностью фильтрации результатов поиска по автору, названию, году издания.

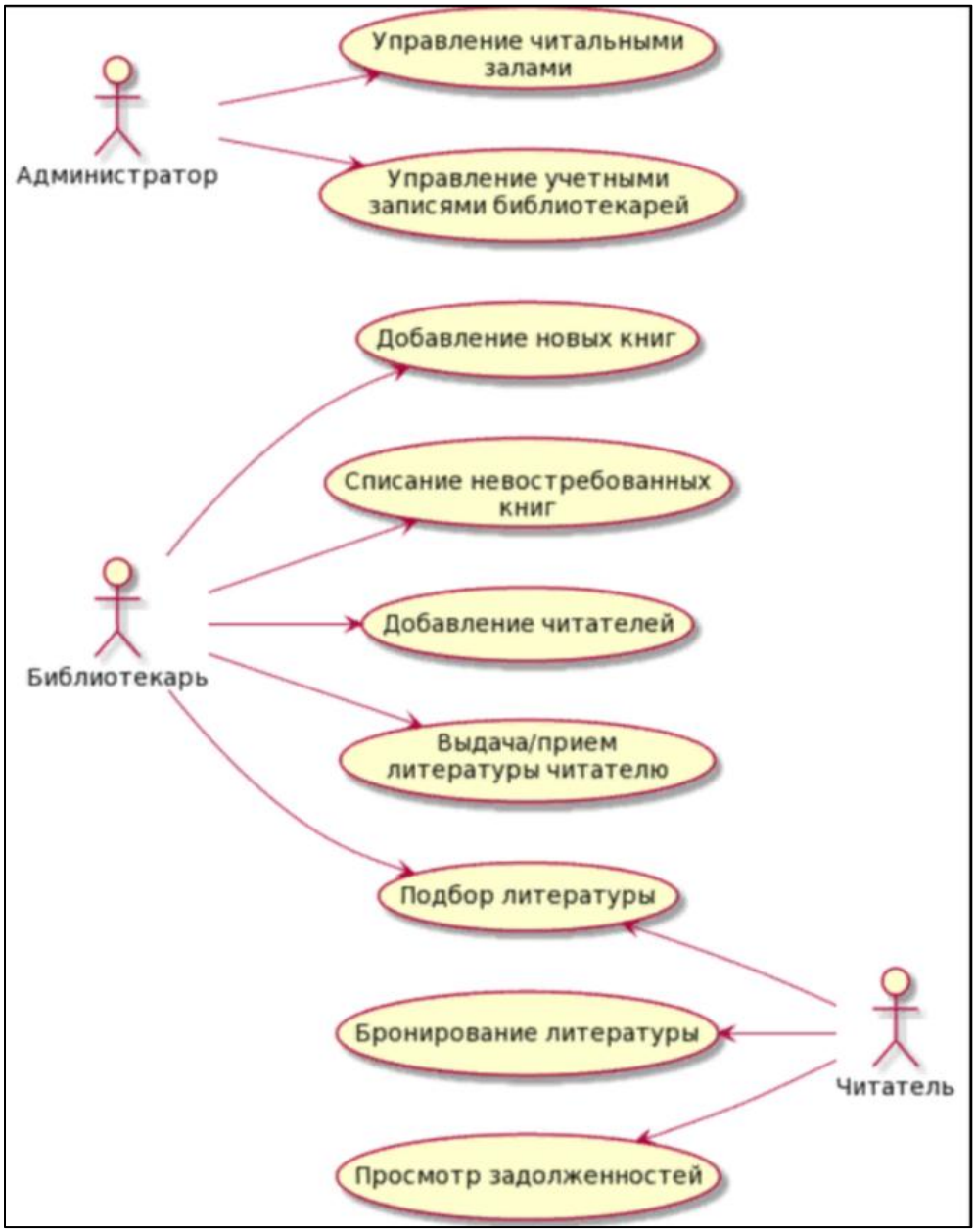


Рисунок А.1 – Диаграмма вариантов использования

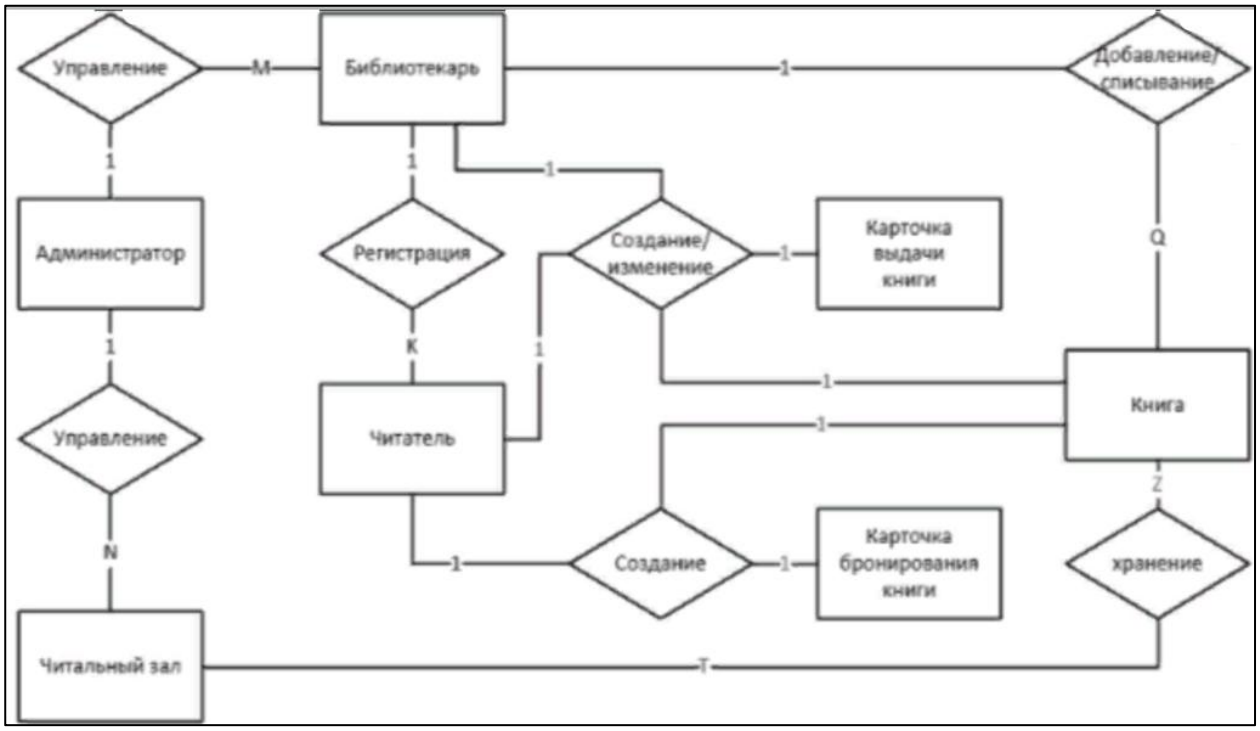
Есть возможность расчета статистики по всем книгам библиотеки, при этом количество выданных экземпляров книги за заданный период времени. Также можно задать минимальное число экземпляров книг, для которых выполняется расчет. На основании этой статистики производится списание неиспользуемых книг из библиотеки.

Можно выделить следующие основные переменные данной предметной области:

-пользователь (библиотекари и администраторы);

-читатель;

-читальный зал;
-книга;
-карточка выдачи книги;
-карточка бронирования книги.



Решением может быть введение дополнительной переменной – карточки о выдаче книги. При выдаче книги читателю заводится карточка, а при сдаче книги – в нее ставится соответствующая пометка. С помощью этих карточек определяются задолженности каждого пользователя и вычисляется статистика использования книг.

При бронировании литературы читателем – также заводится карточка, если забронированная литература не взята читателем в определенный срок – карточка уничтожается. Существует ограничение на количество книг, которые может забронировать читатель.

Есть возможность расчета статистики по всем книгам библиотеки, при этом количество выданных экземпляров книги за заданный период времени. Также можно задать минимальное число экземпляров книг, для которых выполняется расчет. На основании этой статистики производится списание неиспользуемых книг из библиотеки.

В соответствии с прецедентами, показанными на рисунке, база данных должна реализовывать, следующие запросы:

-отобразить книги, соответствующие заданным условиям;

-отобразить пользователей, имеющих незакрытые вовремя карточки выдачи книг (библиотекарь ищет должников);

-отобразить все книги, соответствующие незакрытым вовремя карточкам выдачи книг заданного пользователя (пользователь пришел в библиотеку за

новыми книгами — надо посмотреть является ли он должником и сообщить ему об этом);

-удалить все карточки бронирования, созданные более чем N секунд назад;

-отобразить все книги, соответствующие незакрытым карточкам бронирования книг заданного пользователя (читатель заказал книги и пришел в библиотеку за ними — библиотекарю надо получить этот список чтобы выдать).

А.1.2 Формирование схемы данных

Для формирования схемы данных необходимо сначала дополнить ER-диаграмму реквизитами переменных. Иногда, при этом удается найти ошибки построения ER-диаграммы — в этой задаче было обнаружено, что книгу необходимо «как-то» связать с залом библиотеки. Сделать это можно поместив в книгу реквизит «номер зала», однако при таком подходе одну и ту же книгу придется описывать в базе несколько раз (если она встречается в разных залах). Более правильный подход заключается во введении дополнительной переменной «размещение книги». На рисунке А.3 показана ER-диаграмма с добавленной сущностью и реквизитами.

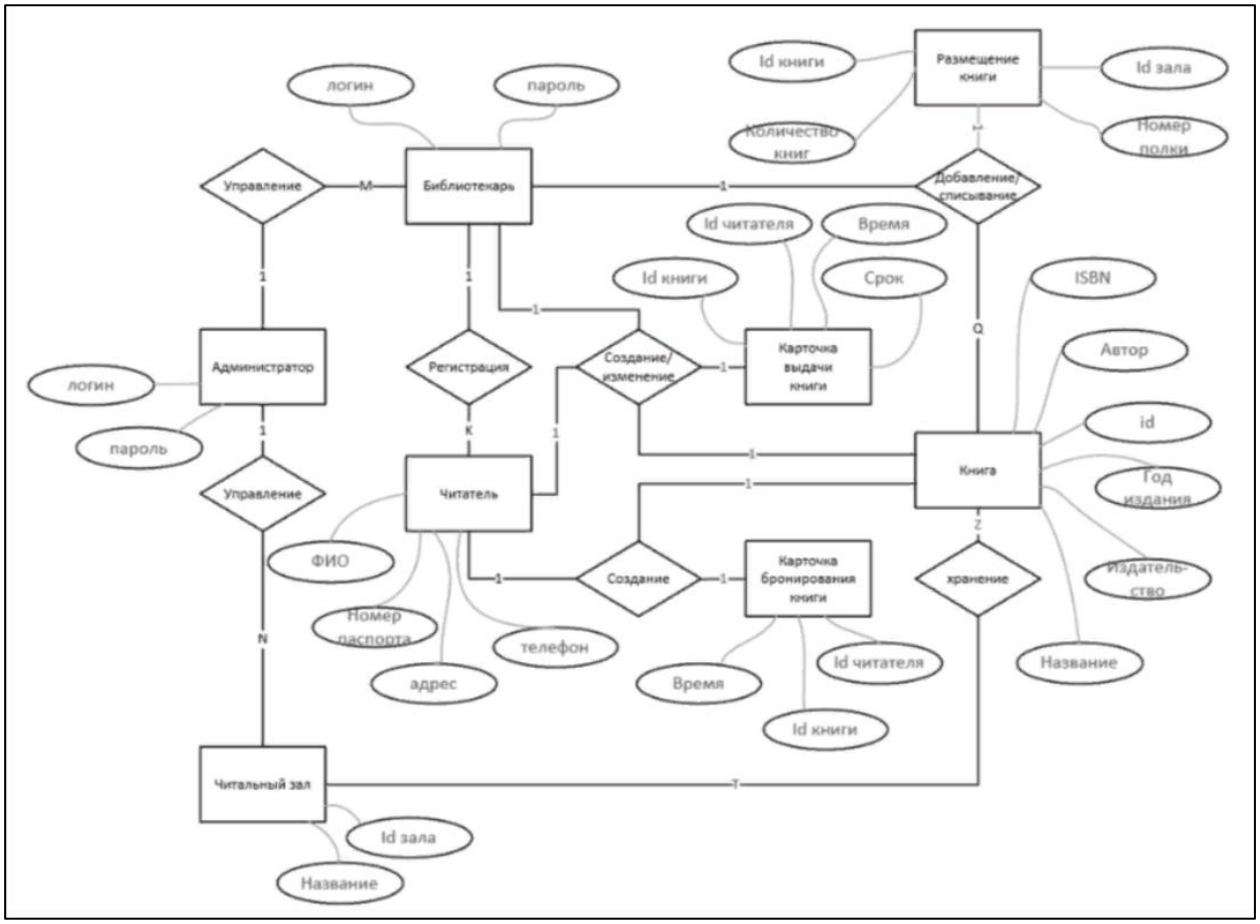


Рисунок А.3 – ER-диаграмма

Приведенная ER-диаграмма отражает основные таблицы, связи и атрибуты, на ее основе можно построить модель БД. Однако, в ходе построения такой диаграммы обязательно нужно выделить ключевые поля (внешние и внутренние), иногда – индексы и типы данных.

При разработке этой модели возникало желание объединить таблицу администраторов с таблицей библиотекарей – добавить таблицу users, однако:

- администратор не связан с конкретным залом (пришлось бы заполнять соответствующее поле null-значениями);

- вероятно, это осложнило бы распределение прав доступа — сейчас доступ к таблице administrators имеет только администратор базы данных (работающий через специальную панель СУБД и не имеющий учетной записи в разрабатываемой системе). Однако при соединении таблиц пользовательские запросы требовали бы доступа к новой таблице.

При построении этой диаграммы был найден и исправлен недочет ER-диаграммы – добавлена таблица librarians_rooms, объединяющая библиотекарей и залы. Это нужно, так как один библиотекарь может работать в нескольких залах, но несколько библиотекарей могут работать в одном и том же зале.

А.2 Физическое проектирование

А.2.1 Выбор СУБД и других программных средств

Реализовать разрабатываемую систему можно с использованием любой СУБД, в том числе — нереляционной (NoSQL). NoSQL базы данных, в свою очередь делятся на несколько типов:

- колоночные базы и базы «ключ-значение» призваны ускорить обработку данных за счет реализации особых схем хранения данных в памяти;

- документные базы позволяют хранить данные с разными полями (у разных объектов) и лучше подходят для параллельной обработки данных. Однако, медленно выполняют обновление данных.

В примере база будет использоваться внутри библиотеки и, скорее всего, не будет требовать очень высокой производительности. Кроме того, структура таблиц меняться не должна. Поэтому будем использовать реляционные базы данных.

Рекомендуется выбирать СУБД работающие в облаке если сложно предсказать будущую нагрузку, однако, в настоящее время почти все популярные СУБД доступны в качестве облачного сервиса. Так например, Google Cloud SQL предоставляет PostgreSQL, SQL Server и MySQL. Яндекс предоставляет такой же функционал, а также облачный доступ к ClickHouse, Redis, Kafka, MongoDB.

MySQL хорошо подходит если объем данных не превышает 2Гб, иначе – лучше взять более сложный в настройке PostgreSQL. Если бы речь шла о

крупной библиотеке – то MySQL не подошел бы, например, библиотека МГУ хранит более 10 миллионов книг, если предположить, что одна книга в нашей базе описывается 200 байтами (хранит строки) — то только таблица с описанием книг заняла бы 1,86 Гб и MySQL не справился бы. Однако, в более простой базе, как наша – его вполне хватит. Для разработки будет использована MySQL.

А.2.2 Определение требований к операционной обстановке

В пункте А.2.1 выполнялся выбор СУБД, однако при этом точно не известен объем памяти, необходимый для хранения таблиц. Очевидно, в библиотеке основной объем памяти будут занимать книги, пользователи и карточки выдачи/бронирования книг.

Предполагается, в библиотеку в месяц будет поступать 100 новых (разных) книг и записываться 200 пользователей. Тысяча пользователей возьмет по 3 книги. Сколько книг будет забронировано — не важно, т.к. карточки бронирования уничтожаются. Учитывая, что для хранения записи об одной книге требуется 45*4+4*2 = 188 байт, для хранения читателя 184 байта, а одна карточка выдачи книги занимает 32 байта можно определить примерный объем памяти, необходимый для базы данных библиотеки в течении одного месяца работы:

100*188 + 200*184 + 1000*3*32 = 18800 + 36800 + 96000 = 151600 байт = 148 Кб

Значит, за год объем базы не должен превысить 1,73Мб.

А.2.3 Описание групп пользователей и прав доступа

Администратор базы данных взаимодействует с базой посредством исполнения SQL-запросов. При этом он имеет доступ ко всем данных, может изменять структуру БД, устанавливает права доступа для остальных групп.

Администратор зала библиотеки имеет доступ по чтению и записи к отношениям librarians, rooms, librarians_rooms. При необходимости работы с фондами библиотеки администратор входит в систему с учетной записью библиотекаря.

Библиотекарь имеет доступ:

- по чтению к отношениям: readers, issue_cards, librarians_rooms и rooms;
- по чтению и записи к отношениям: readers, booking_cards, book_places, books, issue_casrds.

Читатель библиотеки может взаимодействовать с системой через программу-клиент, установленную в зале библиотеки или извне библиотеки через веб-интерфейс. При этом, он имеет доступ по чтению к отношениям: books, book_places, rooms, booking_cards, issue_cards. При работе через программу-клиент читатель имеет также доступ по записи к отношению issue_cards — он может из читального зала забронировать книгу.

Практическая работа 5

Разработка блок-схемы работы приложения

1 Цель работы:

1.1 Научиться составлять блок-схемы работы приложений;

1.2 Закрепить знания по теме «Разработка блок-схемы работы приложения».

2 Перечень оборудования:

2.1 Персональный компьютер;

2.2Microsoft Office;

2.3 Графический редактор.

3.Ход работы:

3.1 Перед началом выполнения работы необходимо ознакомиться с материалами, представленными в приложении А.

3.2 После изучения теоретического материала необходимо выполнить индивидуальное задание, которое получается у преподавателя.

4. Контрольные вопросы:

4.1. Что такое блок-схема и для чего она используется в разработке приложений?

4.2. Какие основные элементы используются при создании блок-схемы, и что они представляют?

4.3. Какие стандартные символы используются для обозначения различных операций, принятия решений и потоков данных в блок-схеме?

4.4. Каковы основные этапы создания блок-схемы для работы приложения?

4.5. Как можно использовать блок-схему для документации и обсуждения логики работы приложения с другими участниками проекта?

4.6. Какие преимущества предоставляет использование блок-схемы при разработке приложений?

4.7. Каковы основные правила построения понятной и эффективной блок-схемы для работы приложения?

4.8. Какие инструменты могут использоваться для создания блок-схем, и как выбрать подходящий инструмент для конкретного проекта?

4.9. Как блок-схема может быть использована для выявления и исправления ошибок в логике работы приложения до начала разработки кода?

4.10. Каким образом блок-схема может помочь понять последовательность выполнения операций и потенциальные точки улучшения в работе приложения?

5. Содержание отчета:

5.1 Титул

5.2 Цель работы

5.3 Ход реализации работы

5.4 Ответы на контрольные вопросы.

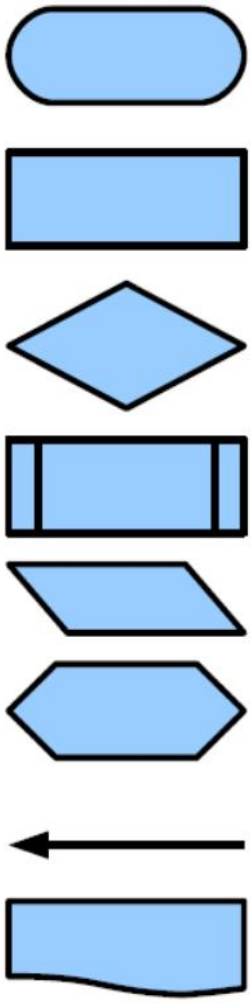
Приложение А

Схема алгоритма (блок-схема) в программировании – графическое представление программы или алгоритма с использованием стандартных графических элементов (прямоугольников, ромбов, трапеций и др.), обозначающих команды, действия, данные и т. п.

Блок-схема – условное изображение алгоритма, программы для ЭВМ, процесса принятия решения, документооборота и т.п., предназначенное для выявления их структуры и общей последовательности операций.

Правила построения блок-схем определяются ГОСТ 19.701 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения» [1]. Стандарт распространяется на условные обозначения (символы) в схемах алгоритмов, программ, данных и систем и устанавливает правила выполнения схем, используемых для отображения различных видов задач обработки данных и средств их решения.

Обозначения в блок-схемах (основные элементы) согласно ГОСТ 19.701-90:



1. Начало или конец. Внутри фигуры пишут или «конец» соответственно.

2. Прямоугольником обозначается операция. присваивание. Внутри блока пишут операции, выполняются на данном шаге алгоритма.

   обозначается оператор ветвления. Внутри ромба проверяемые условия. Например, «a < b».

   подпрограммы. Внутри блока пишут имя подпрограммы и передаваемые ей параметры.

5. Параллелограмм обозначает операции ввода-данных.

   известным числом итераций. Внутри обычно счетчик цикла, начальное, конечное значение и шаг последнем ГОСТе цикл заменен на другой блок.

   обозначают направление процесса

   данных на печать

Вычислительные процессы, используемые для решения различного рода задач на ЭВМ, в общем виде могут быть разделены на три большие группы: линейные, разветвляющиеся и циклические.

Линейным принято называть вычислительный процесс, в котором этапы вычислений выполняются в линейной последовательности и каждый этап

выполняется только один раз. На схеме блоки размещаются сверху вниз в порядке их выполнения (рисунок А.1). Для таких процессов характерно, что направление вычислений не зависит от исходных данных или промежуточных результатов. Линейные процессы имеют место, например, при вычислении арифметических выражений.

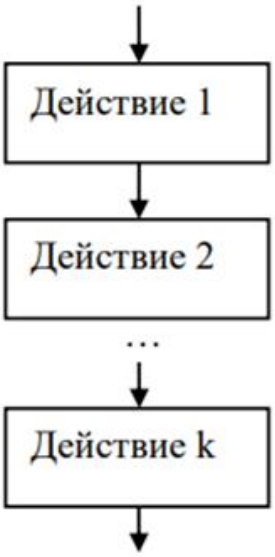


Рисунок А.1 – Блок-схема линейного процесса

Разветвляющийся вычислительный процесс реализуется по одному из нескольких заранее предусмотренных направлений в зависимости от выполнения некоторого условия (логического выражения). Каждое направление вычислений называется ветвью. В любом конкретном случае процесс реализуется только по одной ветви, а выполнение остальных исключается. Ветвящийся процесс, включающий в себя две ветви, называется простым, более двух ветвей – сложным. Сложный ветвящийся процесс можно представить с помощью простых ветвящихся процессов. Изображения полного и неполного вариантов ветвлений в виде блок-схем представлены на рисунках А.2, А.3.

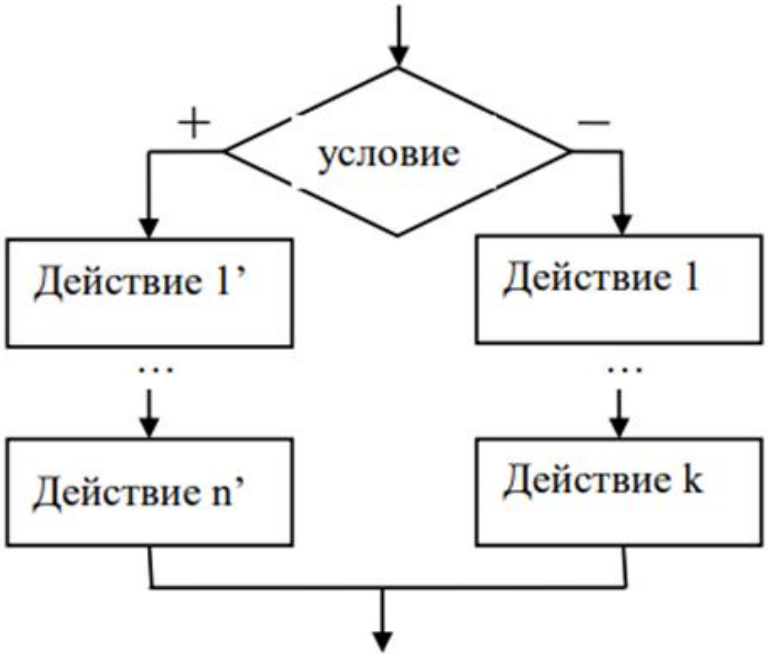


Рисунок А.2 – Блок-схема полного варианта ветвления «если-то-иначе»

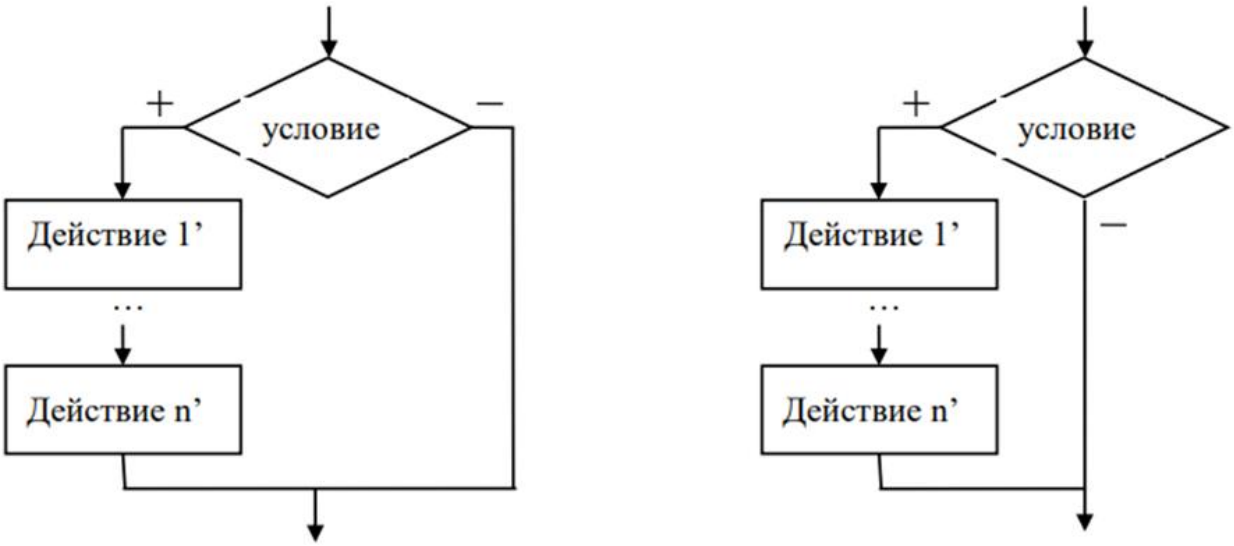


Рисунок А.3 – Блок-схемы неполного варианта ветвления «если-то»

Циклический вычислительный процесс включает участки, на которых вычисления выполняются многократно по одним и тем же математическим формулам, но при разных значениях исходных данных. Такой многократно повторяющийся участок вычислений называется циклом. Для организации цикла необходимо предусмотреть:

– задание начального значения параметра цикла – переменной, которая будет изменяться при его повторении;

– изменение значения этой переменной перед каждым новым повторением цикла;

– проверку условия окончания цикла по значению его параметра и порядок перехода к началу цикла, если он не окончен.

Цикл называется детерминированным (цикл с параметром), если число повторений тела цикла заранее известно или определено. Цикл называется итерационным (с пред- и постусловием), если число повторений тела цикла заранее неизвестно и зависит от значений параметров (некоторых переменных), участвующих в вычислениях.

Выполнение цикла «пока» начинается с проверки условия, поэтому такую разновидность циклов называют циклы с предусловием. Переход к выполнению действия осуществляется только в том случае, если условие выполняется, в противном случае происходит выход из цикла. Можно сказать, что условие цикла «пока» – это условие входа в цикл. В частном случае может оказаться, что действие не выполнялось ни разу. Условие цикла необходимо подобрать так, чтобы действия, выполняемые в цикле, привели к нарушению его истинности, иначе произойдет зацикливание. Зацикливание – это бесконечное повторение выполняемых действий. Блок-схема цикла с предусловием (цикла «пока») представлена на рисунке А.4.

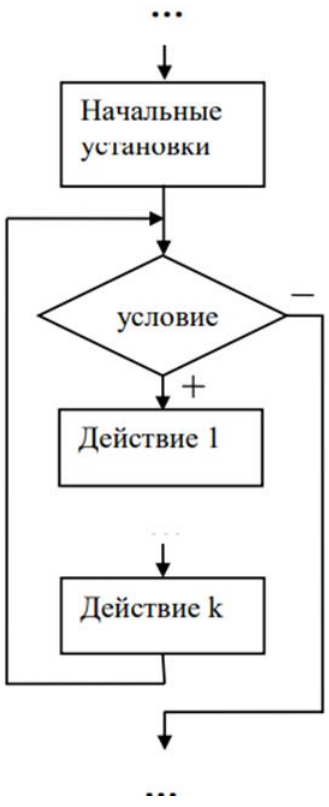


Рисунок А.4 – Блок-схема цикла с предусловием (цикл «пока»)

В качестве примера будет рассмотрена задача:

Разработать блок-схему алгоритма вычисления факториала натурального числа n.

Факториал натурального числа вычисляется по формуле: $n!=1\cdot 2\cdot 3\cdot \ldots \cdot n$. Для решения задачи будет использоваться цикл со счетчиком i. Тогда правило произведения выглядит следующим образом:

– начальное значение произведения S =1 ;

– в теле некоторой циклической конструкции выполнить команду S =S·*< множитель > .

Тогда блок-схема алгоритма решения задачи с использованием цикла «пока» представлены на рисунке А.5.

Шаг 1. Ввод n

Шаг 2. $S = 1;\ i = 1$

Шаг 3. Цикл «пока» $i \le n$

3.1 $S = S \cdot i$

3.2 $i = i + 1$

Конец цикла «пока»

Шаг 4. Вывод S

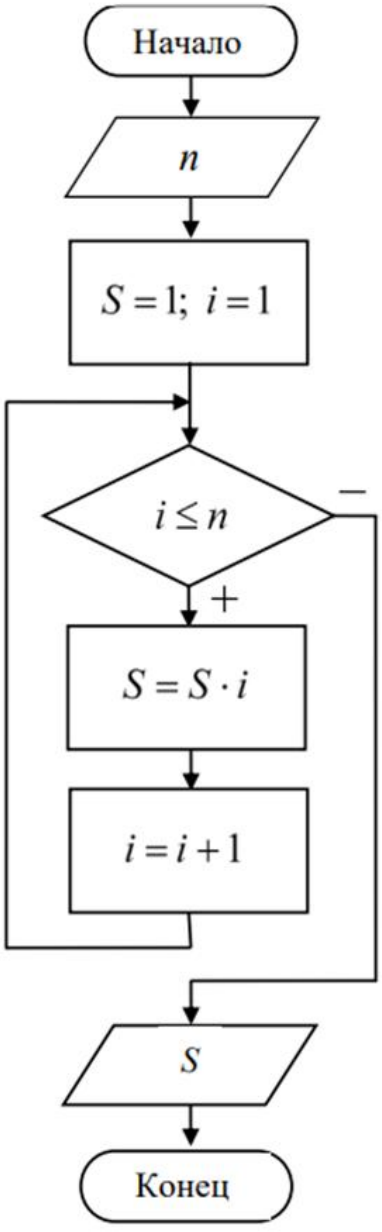


Рисунок А.5 – Блок-схема алгоритма решения задачи

Другим примером, который может быть рассмотрен является процесс авторизации, пример блок-схемы которого представлен на рисунке А.6

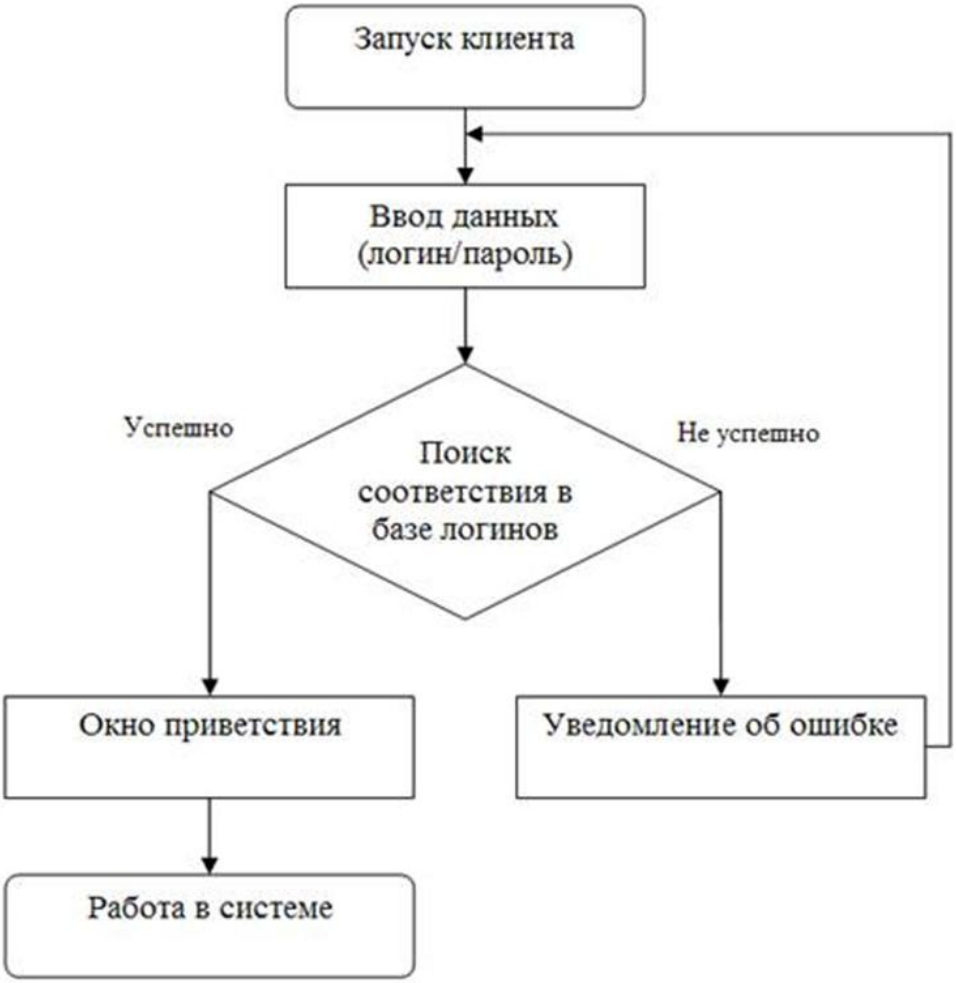


Рисунок А.6 – Блок-схема процесса авторизации

Практическая работа 6

Разработка интерфейса программного продукта

1 Цель работы:

1.1 Закрепить теоретические знания по разработке пользовательского интерфейса;

1.2 Получить практические навыки по проектированию и разработке интерфейса пользователя.

2 Перечень оборудования:

2.1 Персональный компьютер;

2.2 ПО для дизайна и разработки интерфейса.

3 Задание:

3.1 Перед началом выполнения работы необходимо ознакомиться с материалами, представленными в приложении А;

3.2 Выполнить этапы предварительного и высокоуровневого проектирования при разработке пользовательского интерфейса приложения для предметной области, соответствующей варианту задания;

3.3 Разработать главное меню в среде разработки приложения с анализом и обоснованием его различных состояний.

4. Контрольные вопросы:

4.1. Какие основные принципы лежат в основе разработки пользовательского интерфейса?

4.2. Какие этапы включает в себя процесс разработки интерфейса программного продукта?

4.3. Как можно оценить удобство и эффективность интерфейса для конечного пользователя?

4.4. Какие факторы следует учитывать при проектировании интерфейса для различных категорий пользователей (например, опытных и начинающих)?

4.5. Какие технологии и инструменты могут быть использованы для создания пользовательского интерфейса?

4.6. Какие методы тестирования пользовательского интерфейса могут быть применены для обеспечения его качества и соответствия требованиям?

4.7. Какие принципы доступности следует учитывать при разработке интерфейса для людей с ограниченными возможностями?

4.8. Каким образом использование шаблонов и стандартов дизайна может улучшить пользовательский интерфейс?

4.9. Как влияет мобильная адаптивность на разработку интерфейса программного продукта?

4.10. Каким образом сбалансировать эстетические и функциональные аспекты разработки пользовательского интерфейса?

5. Содержание отчета:

5.1 Титул

5.2 Цель работы

5.3 Ход реализации работы

5.4 Ответы на контрольные вопросы.

Приложение А

На практике высокоуровневое проектирование пользовательского интерфейса предваряет первоначальное проектирование, которое позволяет выявить требуемую функциональность создаваемого приложения, а также особенности его потенциальных пользователей. Указанные сведения можно получить, анализируя информацию, поступающую от пользователей. С этой целью производят опрос целевой аудитории и формируют профили пользователей. Профилями называют описания главных категорий пользователей. Одна из таких категорий может быть принята за основной профиль. Следует отметить, что набор характеристик, подробно описывающий пользователя, зависит от предметной области и контекста решаемых им задач. Поэтому работа по определению целей и задач пользователей и работа по формированию их профилей ведется параллельно.

Наиболее общий шаблон профиля содержит в себе следующие разделы:

- социальные характеристики;
- навыки и умения работы с компьютером;
- мотивационно-целевая среда;
- рабочая среда;
- особенности взаимодействия с компьютером (специфические требования пользователей, необходимые информационные технологии и др.).

Профили пользователей могут по необходимости расширяться за счет добавления других (значимых с точки зрения проектировщика) характеристик пользователей.

После выделения одного или нескольких основных профилей пользователей и после определения целей и задач, стоящих перед ними, переходят к следующему этапу проектирования. Этот этап связан с составлением пользовательских сценариев. Как правило, начинают с персонификации профилей (присваивания каждому профилю условного имени), затем формулируют сценарии. Сценарий - это описание действий, выполняемых пользователем в рамках решения конкретной задачи на пути достижения его цели. Очевидно, что достигнуть некоторой цели можно, решая ряд задач. Каждую их них пользователь может решать несколькими способами, следовательно, должно быть сформировано несколько сценариев. Чем больше их будет, тем ниже вероятность того, что некоторые ключевые объекты и операции будут упущены.

В то же время, у разработчика имеется информация, необходимая для формализации функциональности приложения. А после формирования сценариев становится известным перечень отдельных функций. В приложении функция представлена функциональным блоком с соответствующей экранной формой (формами). Возможно, что несколько функций объединяются в один функциональный блок. Таким образом, на этом этапе устанавливается необходимое число экранных форм. Важно определить навигационные взаимосвязи функциональных блоков. На практике установлено наиболее подходящим число связей для одного блока равное трем. Иногда, когда

последовательность выполнения функций жестко определена, между соответствующими функциональными блоками можно установить процессуальную связь. В этом случае их экранные формы вызываются последовательно одна из другой. Такие случаи имеют место не всегда, поэтому навигационные связи формируются либо исходя из логики обработки данных с которыми работает приложение, либо основываясь на представлениях пользователей (карточная сортировка). Навигационные связи между отдельными функциональными блоками отображаются на схеме навигационной системы. Возможности навигации в приложении передаются через различные навигационные элементы.

Основным навигационным элементом приложения является главное меню. Роль главного меню велика еще и потому, что оно осуществляет диалоговое взаимодействие в системе «пользователь-приложение». Кроме того, меню косвенно выполняет функцию обучения пользователя работе с приложением.

Формирование меню начинается с анализа функций приложения. Для этого врамках каждой из них выделяют отдельные элементы: операции, выполняемые пользователями, и объекты, над которыми осуществляются эти операции. Следовательно, известно какие функциональные блоки должны позволять пользователю осуществлять какие операции над какими объектами. Выделение операций и объектов удобно проводить на основе пользовательских сценариев и функционала приложения. Выделенные элементы группируются в общие разделы главного меню. Группировка отдельных элементов происходит в соответствии с представлениями об их логической связи. Таким образом, главное меню может иметь каскадные меню, выпадающие при выборе какого либо раздела. Каскадное меню ставит в соответствие первичному разделу список подразделов.

Одним из требований к меню является их стандартизация, целью которой выступает формирование устойчивой пользовательской модели работы с приложением. Существуют требования, выдвигаемые с позиций стандартизации, которые касаются места размещения заголовков разделов, содержания разделов часто используемых в разных приложениях, формы заголовков, организации каскадных меню и др. Наиболее общие рекомендации стандартизации следующие:

- группы функционально связанных разделов отделяют разделителями (черта или пустое место);
- не используют в названиях разделов фраз (желательно не больше 2 слов);
- названия разделов начинают с заглавной буквы;
- названия разделов меню, связанных с вызовом диалоговых окон заканчивают многоточием;
- названия разделов меню, к которым относятся каскадные меню, заканчивают стрелкой;
- используют клавиши быстрого доступа к отдельным разделам меню. Их выделяют подчеркиванием;

- допускают использовать «горячие клавиши», соответствующие комбинации клавиш отображают в заголовках разделов меню;

- допускают использовать включение в меню пиктограмм;

- измененным цветом показывают недоступность некоторых разделов меню в ходе работы с приложением;

- допускают делать недоступные разделы невидимыми.

Недоступность некоторых разделов меню обуславливается следующим. Главное меню является статическим и присутствует на экране в течение всего времени работы с приложением. Таким образом, при работе с разными экранными формами (взаимодействии с разными функциональными блоками) не все разделы меню имеют смысл. Такие разделы принято являются недоступными. Поэтому в зависимости от контекста решаемых пользователем задач (иногда от контекста самого пользователя) главное меню приложения выглядит различным образом. О подобных различающихся внешних представлениях меню принято говорить как о различных состояниях меню. В отличие от схемы навигационной системы, составленной ранее и необходимой, в основном, разработчику, с меню пользователь входит в непосредственное взаимодействие. Поэтому следует составить граф состояния меню. Вершинами этого графа являются различные состояния меню (внешние представления одного и того же меню с доступными и недоступными разделами). Каждая вершина имеет пояснения о соответствии данного состояния меню отдельным экранным формам. Дуги графа состояний соответствуют операциям (командам меню), переводящим его из одного состояние в другое.

Подобный граф используют при формировании тестовых заданий на последних стадиях проектирования интерфейса. В связи с этим, важно при его формировании выполнить проверку соответствия пользовательских сценариев возможным переходам по графу.

Далее будет рассмотрен пример. Пусть предметная область представлена информационной системой, отображающую деятельность мелкой фирмы, которая связана с изготовлением и/или поставкой ряда товаров.

Первым шагом необходимо сформировать профили потенциальных пользователей программного обеспечения информационной системы. Потенциальными пользователями приложения являются, например, менеджеры по направлению товара, торговые представители, представители обслуживающего персонала и т.д. Примерные профили некоторых из названных категорий пользователей могут выглядеть следующим образом (таблица 1).

Таблица 1 – Примерные профили

Пользователи	Менеджер по направлению товара	Представители обслуживающего персонала
Социальные характеристики	Мужчины, женщины Взрослые	Женщины Взрослые

	Русскоязычные Средний уровень владения компьютером	Русскоязычные Низкий уровень владения компьютером
Мотивационно-целевая среда	Прямая производственная необходимость, удобство Мотивация к обучению высокая	Производственная необходимость, Престиж Мотивация к обучению низкая
Навыки и умения	Должны иметь значительный тренинг работы с программой	Прошли предварительный тренинг работы с программой
Требования к ПО ИС	Возможность использования ПО ИС в локальной сети Отсутствие жестких ограничений по времени Обеспечение текущей информацией по содержанию заказов Обеспечение текущей информацией по товарам Возможность проводить обобщение информации по заказам	Возможность использования программы одновременно с телефонным общением с клиентом Время реакции ПО ИС, допустимое для ожидания клиента Обеспечение текущей информацией по содержанию заказов Обеспечение текущей информацией по товарам Возможность формирования новых заказов
Задачи пользователя	Просмотр/фильтрация информации по заказам/клиентам/товарам Сортировка информации по заказам/клиентам/товарам Агрегирование информации по заказам/клиентам/товарам	Просмотр данных по товарам Создание/поиск/модификация заказа Сохранение/печать заказа Формирование счета по заказу
Рабочая среда	Стандартизированные ПК, локальная сеть	Стандартизированные ПК, специализированное телефонное обслуживание

Вторым шагом является определение функциональности приложения, исходя из целей и задач пользователей.

Определение функциональности на примере одного из профилей: представители обслуживающего персонала. Исходя из задач этой категории

пользователей, можно сформировать следующий перечень функций необходимых в приложении:

- создать новый заказ (1);
- сложный поиск заказа (2);
- редактирование заказа (3);
- добавление клиента из списка клиентов в заказ (4);
- ввод/редактирование клиента в списке клиентов (5);
- выбор товара из списка товаров (6);
- сложный поиск товаров в списке товаров (7);
- просмотр подробных данных о товаре (8);
- добавление товара из списка товаров в заказ (9);
- сохранение заказа (10);
- печать заказа (11);
- формирование счета (12).

Третьим действием необходимо сформировать множество пользовательских сценариев для выделенных профилей пользователей.

Примером могут служить приведенные ниже сценарии действий пользователей.

1) Анна Петровна общается с клиентами по телефону. По просьбе клиента она предварительно просматривает данные о запрошенных им товарах, затем приступает к формированию нового заказа. Она вводит данные клиента, после чего выбирает указанный(ые) товар(ы) из списка и добавляет его (их) в заказ и сохраняет заказ.

2) Анна Сергеевна общаясь с клиентами по телефону, создает новые заказы. При формировании нового заказа, она выбирает клиента из списка, если его там нет, то вводит клиента в список клиентов. Затем добавляет в заказ необходимые товары, используя сложный поиск. Она распечатывает информацию заказа, после этого она сохраняет ее.

3) Анна Михайловна выполняет поиск указанного заказа по данным клиента. Она просматривает и при необходимости редактирует данные клиента, добавляет в заказ новые или удаляет из заказа прописанные там товары, при необходимости редактирует в заказе информацию по некоторым товарам, сохраняет информацию и формирует счет заказа.

4) Анна Николаевна просматривает данные о товаре, выполняет поиск заказа по товару, редактирует в заказе информацию по некоторым товарам, сохраняет информацию и распечатывает ее.

Четвертым шагом необходимо определить функциональные блоки приложения, составить схему навигационной системы.

Очевидно, что отдельные функциональные блоки соответствуют работе пользователей с информацией:

- по заказам (функции 1,2,3,4,9,10,11,12): по общему журналу заказов и по конкретному (текущему) заказу;
- по клиентам (функции 4,5): по списку клиентов в целом и по конкретному клиенту;

- по товарам (функции 6,7,8,9): по списку товаров и по данному товару подробно.

Таким образом, можно вести речь о наличии в приложении трех функциональных блоков и шести экранных форм:

- журнал заказов;
- текущий заказ;
- список клиентов;
- карта клиента;
- список товаров;
- карта товара.

В этом случае, с учетом пользовательских сценариев схема навигации по формам может выглядеть следующим образом (рисунок 1).

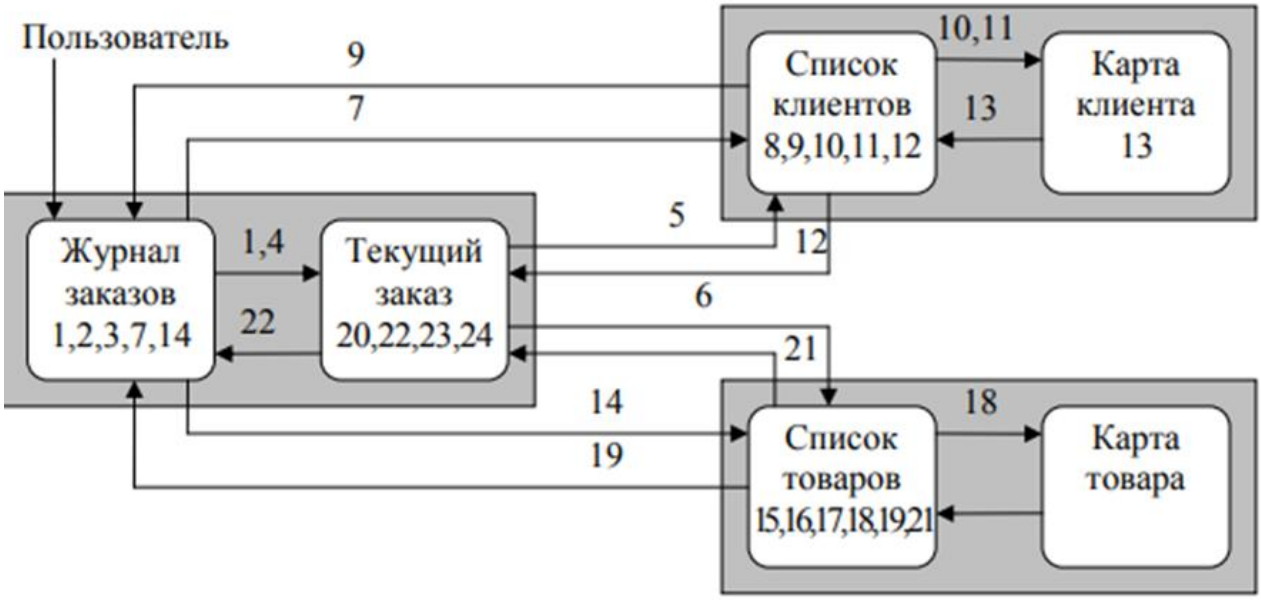


Рисунок 1 - Схема навигации

Цифрами на рисунке обозначены отдельные операции, выполняемые пользователями, которые необходимо установить на пятом этапе.

Операции, которые должен выполнять пользователь в рамках возможностей, предоставляемых ему приложением (функций приложения):

1) создать новый заказ;
2) задать атрибуты поиска заказа;
3) найти заказ по текущим атрибутам поиска;
4) открыть текущий заказ на редактирование;
5) открыть список клиентов для добавления в текущий заказ;
6) открыть список товаров для добавления в текущий заказ;
7) просмотреть список клиентов;
8) выбрать клиента из списка клиентов;
9) добавить атрибуты текущего клиента к поиску заказа;
10) ввести данные нового клиента в текущий заказ;
11) редактировать данные текущего клиента в списке клиентов;

12) добавить текущего клиента в текущий заказ;
13) сохранить данные о текущем клиенте;
14) просмотреть список товаров;
15) задать атрибуты поиска товаров;
16) найти товар по текущим атрибутам;
17) выбрать товар из списка товаров;
18) просмотреть подробные данные текущего товара;
19) добавить атрибуты текущего товара к поиску заказа;
20) редактировать данные по текущему товару в текущем заказе;
21) добавить данные текущего товара в текущий заказ;
22) сохранить текущий заказ;
23) распечатать информацию по текущему заказу;
24) сформировать счет по текущему заказу.

Соответствие приведенных операций функциональным блокам, экранным формам и навигационным переходам указано на рисунке 1.

Далее, необходимо сгруппировать операции таким образом, чтобы их группы соответствовали пунктам главного меню. В рассматриваемом примере предлагается сформировать следующие группы.

1.Действия над объектами. В качестве объектов выступают заказ, клиент, товар, представленные на рисунке 2.

Действия	Объект	Примечания
Создать	Заказ Клиент	1 10
Открыть	Заказ Клиент Товар	4 11 18
Сохранить	Заказ Клиент	22 13
Выбрать (отобрать для добавления)	Клиент Товар Атрибуты клиента Атрибуты товара	12 (в заказ) 21 (в заказ) 9 (к поиску) 19 (к поиску)
Печать	Заказ	23
Счет	Заказ	24

Рисунок 2 – Группа действий

2. Поиск. Специфическое действие, выделено отдельно; объекты – заказ (3), товар (16).

3. Работа со списками. Объекты – клиент, заказ (таблица 3)

Таблица 3 – Работа со списками

Списки	Операции	Примечания

Клиенты	Просмотреть Открыть для выбора (добавления) в заказ	7 5
Товары	Просмотреть Открыть для выбора (добавления) в заказ	14 6

4. Стандартными являются такие разделы как Файл и Справка. Их тоже следует включить в главное меню приложения.

Последним шагом является создание графа состояния меню и проведение проверки возможных переходов по графу в соответствии с пользовательскими сценариями.

Необходимо рассмотреть состояния меню для приведенного примера. Для простоты не будем учитывать состояния меню, связанные доступностью стандартных разделов Файл и Справка и их подразделов. Различные состояния прототипа меню можно представить таблицами 4-11. Разделы меню и команды, недоступные в данном состоянии выделены серым цветом. Для доступных команд в скобках указаны номера соответствующих операций.

Таблица 4 – Журнал заказов (состояние М1)

Действия	Поиск	Списки
Создать (1) Открыть (4) Сохранить Выбрать Печать Счет	Найти (3)	Клиенты (7) Товары (14)

Таблица 5 – Текущий заказ (состояние М2)

Действия	Поиск	Списки
Создать Открыть Сохранить (22) Выбрать Печать (23) Счет (24)		Клиенты (5) Товары (6)

Таблица 6 – Список клиентов (состояние М3, переход по команде 7)

Действия	Поиск	Списки
Создать (10) Открыть (11) Сохранить Выбрать (9)		

Печать Счет		

Таблица 7 – Список клиентов (состояние М4, переход по команде 5)

Действия	Поиск	Списки
Создать (10) Открыть (11) Сохранить Выбрать (12) Печать Счет		

Таблица 8 – Карта клиента (состояние М5)

Действия	Поиск	Списки
Создать Открыть Сохранить Выбрать (13) Печать Счет		

Таблица 9 – Список товаров (состояние М6, переход по команде 14)

Действия	Поиск	Списки
Создать Открыть (18) Сохранить Выбрать (19) Печать Счет	Найти (16)	

Таблица 10 – Список товаров (состояние М7, переход по команде 6)

Действия	Поиск	Списки
Создать Открыть (18) Сохранить Выбрать (21) Печать Счет	Найти (16)	

Таблица 11 – Карта товара (состояние М8)

Действия	Поиск	Списки

Граф состояний меню можно представить следующим образом (рисунок 3).

Следует подчеркнуть, что прототип меню в данном примере создается только под одну определенную категорию пользователей. Кроме того, в примере не предусмотрено развитие программного продукта, следовательно, отсутствует расширяемость функций.

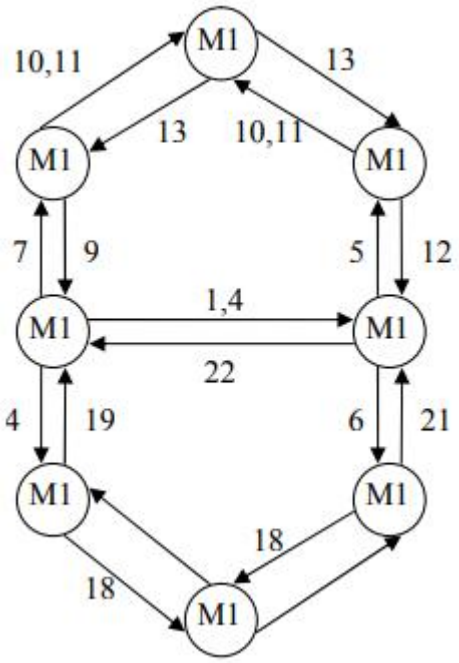


Рисунок 2 – Граф состояний меню

Практическая работа 7

Тест по теме «Основы маршрутизации. Клиент-серверная архитектура»

1 Цель работы:

1.1 Закрепление знаний по теме «Клиент-серверная архитектура».

2 Подготовка к работе:

2.1 Изучить теоретический материал по теме «Основы маршрутизации. Клиент-серверная архитектура».

3 Задание:

3.1 Ответить письменно на вопросы тестового задания.

4. Обобщенные вопросы тестового задания:

4.1 На каком уровне используется Http протокол?

4.2 Где обрабатывается информация (обрабатывается запрос) в архитектуре клиент-сервер?

4.3 На какой сервер требуется отправить запрос чтобы узнать адрес нужного ресурса?

4.4 В какой топологии сети все хосты подключены последовательно (последний соединяется с первым)?

4.5 Какой протокол, использующийся на транспортном уровне, требует подтверждения доставки пакетов?

4.6 Какие HTTP-коды информируют об ошибке на стороне сервера?

4.7 Клиент-серверная архитектура: описание, виды пользовательских интерфейсов.

4.8 Клиент-серверная архитектура: назначение блоков, описание технических устройств клиентской и серверной части.

4.9 Модель TCP/IP: назначение уровней, протоколы.

4.10 Маршрутизация: назначение, классификация, функции.

4.11 Нарисуйте маршрутизацию с установлением соединения

4.12 Какие группы кодов состояния бывают?

Практическая работа 8

Тест по теме «Технологии разработки программного обеспечения»

1 Цель работы:

1.1 Закрепление знаний по теме «Технологии разработки программного обеспечения».

2 Подготовка к работе:

2.1 Изучить теоретический материал по теме «Технологии разработки программного обеспечения».

3 Задание:

3.1 Ответить письменно на вопросы тестового задания.

4. Обобщенные вопросы тестового задания:

4.1 Что такое предметная область?

4.2 Что включает в себя анализ предметной области?

4.3 Для чего предназначено техническое задание?

4.4 Что такое архитектура программного средства?

4.5 Какие методологии используются при моделировании диаграмм вариантов использования?

4.6 Каково различие между тестированием и отладкой программы?

4.7 Что понимается под тестирование программного обеспечения?

4.8 Что понимается под требованиями к программному обеспечению?

4.9 Что понимается под требованиями к программному обеспечению?

4.10 Что такое версионный контроль?

4.11 Каким образом Agile-методологии влияют на процесс разработки программного обеспечения?

4.12 Какие этапы включают в себя жизненный цикл разработки программного продукта?

Практическая работа 9

Тест по теме «Микросервисная архитектура»

1 Цель работы:

1.1 Закрепление знаний по теме «Микросервисная архитектура».

2 Подготовка к работе:

2.1 Изучить теоретический материал по теме «Микросервисная архитектура».

3 Задание:

3.1 Ответить письменно на вопросы тестового задания.

4. Обобщенные вопросы тестового задания:

4.1 Что такое микросервисы и в чем их отличие от монолитной архитектуры?

4.2 Какие преимущества предоставляет микросервисная архитектура в сравнении с монолитной?

4.3 Какие основные принципы следует учитывать при проектировании микросервисной архитектуры?

4.4 Какие вызовы возникают при разработке и масштабировании микросервисных систем?

4.5 Какие технологии можно использовать для реализации связи между микросервисами?

4.6 Какие компоненты обычно включаются в микросервисную архитектуру?

4.7 Как микросервисы обеспечивают автономность и независимость отдельных компонентов системы?

4.8 Какие подходы к обеспечению безопасности можно использовать в микросервисной архитектуре?

4.9 Каким образом контейнеризация помогает в развертывании и управлении микросервисами?

4.10 Как можно обеспечить высокую доступность и отказоустойчивость в микросервисной архитектуре?

Библиография

1. Кауфман В.Ш. Языки программирования. Концепции и принципы [Электронный ресурс] / В.Ш. Кауфман. – Электрон. текстовые данные. – Саратов: Профобразование, 2017. – 464 c. – Режим доступа: http://www.iprbookshop.ru/64055.html
2. Е.В. Петров. Разработка клиент-серверных приложений на JavaScript, 2016 г. — 258.

Дополнительная:

3. Иван Портянкин. Сетевое программирование. Примеры и задачи, 2016 г. — 147 с
4. Выпускная квалификационная работа: Методические указания по содержанию оформлению. /Гниломедов Е.И., Бурумбаев Д.И. – Екатеринбург: УрТИСИ СибГУТИ, 2023. – 47 с. Электронные данные.- Режим доступа: https://aup.uisi.ru/cixfiles/4224375/mu_po_oformlenijue_vkr_2023.pdf.
5. В.С. Герасимов. Методы и модели разработки программного обеспечения, 2013 г. – 254 с.