МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

**Кафедра современных технологий программирования**

КОХАНОВА АННА ВИТАЛЬЕВНА

**Разработка расширения “To-Do list” для Google Chrome.**

Курсовая работа

студента 2 курса специальности

1-40 01 02 «Программное обеспечение информационных технологий»

дневной формы получения образования

|  |
| --- |
| Научный руководитель:  Скращук Владислав Сергеевич, преподаватель |

Гродно, 2016

**РЕЗЮМЕ**

Тема курсовой работы

**“Разработка расширения “To-Do list” для Google Chrome.”**

Работа содержит – 28 страниц, 5 глав в основной части, 10 иллюстраций, 6 литературных источников.

Ключевые слова – To-Do List, Google Chrome расширение, пользовательский интерфейс, авторизация, аккаунт Google+, Google Drive, Google Drive API.

Цель курсовой работы – разработка расширения “To-Do list” для браузера Google Chrome с возможностью создания, редактирования, удаления и сохранения текстовых заметок на Google Drive пользователя.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc452455645)

[ГЛАВА 1. Логика приложения. 6](#_Toc452455646)

[1.1 Добавление заметки. 6](#_Toc452455647)

[1.2 Удаление заметок. 8](#_Toc452455648)

[1.2.1 Удаление одной заметки. 8](#_Toc452455649)

[1.2.2 Удаление отмеченных заметок. 9](#_Toc452455650)

[1.2.3 Удаление всех заметок. 9](#_Toc452455651)

[1.3 Редактирование заметки. 10](#_Toc452455652)

[ГЛАВА 2. Пользовательский интерфейс приложения. 11](#_Toc452455653)

[2.1 Основные элементы пользовательского интерфейса. 11](#_Toc452455654)

[2.2 Bootstrap. 12](#_Toc452455655)

[2.3 Фон. 13](#_Toc452455656)

[ГЛАВА 3. Google Chrome расширение. 14](#_Toc452455657)

[3.1 Понятие Google Chrome расширения. 14](#_Toc452455658)

[3.2 Структура Google Chrome расширения. 14](#_Toc452455659)

[ГЛАВА 4. Авторизация. 18](#_Toc452455660)

[4.1 Авторизация через социальные сети. 18](#_Toc452455661)

[4.2 OAuth протокол. 19](#_Toc452455662)

[4.2.1 Преимущества и недостатки протокола OAuth. 19](#_Toc452455663)

[4.2.2 OAuth 2.0. 20](#_Toc452455664)

[4.3 Регистрация приложения в Google API Console. 21](#_Toc452455665)

[4.4 Авторизация в расширении. 22](#_Toc452455666)

[ГЛАВА 5. Работа с Google Drive API. 24](#_Toc452455667)

[5.1 Использование API при разработке приложений. 24](#_Toc452455668)

[5.2 Описание Google Drive API. 24](#_Toc452455669)

[5.3 Работа с Google Drive пользователя. 25](#_Toc452455670)

[ЗАКЛЮЧЕНИЕ. 27](#_Toc452455671)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ. 28](#_Toc452455672)

# ВВЕДЕНИЕ

Современный высокотехнологичный мир предлагает множество уникальных приложений для различных устройств, которые способны улучшить качество повседневной жизни. Деловые программы типа “Персональный органайзер” или “Календарь” могут стать незаменимыми помощниками в различных ситуациях, когда возникает необходимость быстро записать важную информацию на своем устройстве или предварительно распланировать дела, что позволяет упростить рабочий процесс и повысить производительность.

В настоящее время все большую популярность приобретают приложения типа “To-Do List”, которые имеют понятный интерфейс и простейший функционал, и представляют собой простой список дел (задач). Существует две возможные реализации данного приложения: как десктопное приложение, или как веб-приложение.

Согласно статистике, с годами отдается все большее предпочтение web-приложениям, поскольку в настоящее время большинство людей используют свои компьютеры в основном для выхода в Интернет. Основная часть действий обычного среднестатистического пользователя – это работа в браузере. Однако не всегда пользователю может хватить базового функционала браузера для решения каких-либо задач. Поэтому существует возможность быстро и легко расширять его возможности, делая пребывание в Интернете еще более комфортным. Любой желающий может разработать свое собственное расширение с необходимым ему функционалом и встроить его в браузер. Еще одним преимуществом является быстрый доступ к функционалу расширения с помощью верхней панели.



Поэтому в качестве инструмента для реализации данного проекта было выбрано расширение для популярного браузера Google Chrome.

Для достижения цели данной курсовой работы было необходимо решить следующие задачи:

* Продумать логику и смоделировать интерфейс расширения.
* Изучить особенности разработки расширения для браузера Google Chrome и его структуру.
* Реализовать авторизацию.
* Реализовать возможность сохранения пользовательских данных с возможной дальнейшей синхронизацией.

# ГЛАВА 1. Логика приложения.

Логика данного приложения написана на JavaScript, поскольку этот язык ориентирован на браузеры, как язык сценариев для придания интерактивности веб-страницам. JavaScript является объектно-ориентированным С-подобным языком и используется в клиентской части веб-приложений: клиент-серверных программ, в котором клиентом является браузер, а сервером — веб-сервер, имеющих распределённую между сервером и клиентом логику. Обмен информацией в веб-приложениях происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами.

## Добавление заметки.

Чтобы добавить заметку в список, необходимо ввести нужный текст в текстовое поле и нажать кнопку + (или клавишу Enter). Данный функционал реализован с помощью обработчика событий:

$(document).ready(function () {

$("#taskInput").keyup(function (event) {

if (event.keyCode == 13 && currentId == -1) {

countTasks++;

$("#plus").click(); }

}

)};



По событию onclick вызывается функция saveTask для сохранения введенного текста в список заметок.

document.getElementById('plus').onclick = saveTask;

Проверка поля для ввода и вызов функции addToList(itemValue):

function saveTask() {

tasks[currentId].name = document.getElementById("taskInput").value;

if (itemValue !== '') {

addToList(itemValue);

}

$('.new-todo').val(null); - очистка поля ввода

$('.new-todo').focus(); - фокус на поле ввода

}

Добавление заметки в список и обновление HTML-разметки с помощью функции updateListView():

var tasks = [];

function addToList(task) {

tasks.push({

name: task,

done: false

});

updateListView();

}

Обновление HTML-разметки на основе списка tasks, динамическое создание HTML-элементов с помощью js:

function updateListView() {

var ul = document.getElementById('taskList');

ul.innerHTML = '';

countTasks = 0;

countCompletedTasks = 0;

currentId = -1;

if (tasks.length != 0) {

tasks.forEach(function (task) {

countTasks++;

var li = document.createElement("li");

li.className = "task";

li.id = tasks.indexOf(task);

var label = document.createElement("label");

label.className = "taskText";

label.textContent = task.name;

label.style.width = '70%';

var ch = document.createElement("input");

ch.className = "toggle";

ch.type = "checkbox";

ch.style.width = '10%';

ch.checked = task.done;

ch.onclick = toggleChecked;

ch.id = tasks.indexOf(task);

if (task.done) {

label.style.textDecoration = 'line-through';

countCompletedTasks++;

}

var spanRemove = document.createElement("span");

spanRemove.className = "remove glyphicon glyphicon-remove";

spanRemove.onclick = deleteFromList;

spanRemove.style.width = '10%';

var spanEdit = document.createElement("span");

spanEdit.className = "edit glyphicon glyphicon-pencil";

spanEdit.onclick = editTask;

spanEdit.style.width = '10%';

li.appendChild(ch);

li.appendChild(label);

li.appendChild(spanEdit);

li.appendChild(spanRemove);

ul.insertBefore(li, ul.firstChild);

});

}

saveData();

}

## Удаление заметок.

## Удаление одной заметки.

Обработчик события по нажатию на кнопку “delete” и вызов функции удаления из списка. 

$("#remove").click(function () {

deleteFromList();

});

Удаление заметки по ее id и вызов функций обновления HTML-разметки:

function deleteFromList(e) {

tasks.splice(e.target.parentElement.id, 1);

updateListView();

updateCompletedListArray();

}

## 1.2.2 Удаление отмеченных заметок.

По событию onclick кнопки Delete completed вызывается функция deleteCompleted для удаления отмеченных заметок из списка с последующим обновлением HTML-разметки и списка отмеченных заметок.

document.getElementById('deleteCompletedBtn').onclick = deleteCompleted;

function deleteCompleted() {

for (var i = completedTasks.length; i--; ) {

tasks.splice(completedTasks[i], 1);

}

saveData();

updateCompletedListArray();

updateListView();

}



## 1.2.3 Удаление всех заметок.

По событию onclick кнопки Delete all вызывается функция deleteAll для удаления всех заметок из списка с последующим обновлением HTML-разметки и списка заметок.

document.getElementById('clearTasksBtn').onclick = deleteAll;

function deleteAll() {

if( tasks.length > 0 ) {

var ul = document.getElementById('taskList');

ul.innerHTML = '';

tasks = [];

completedTasks = [];

document.getElementById('json').value = JSON.stringify(tasks);

countCompletedTasks = 0;

countTasks = 0;

document.getElementById('count').innerHTML = countCompletedTasks + '/' + countTasks;

}

}

## Редактирование заметки.

При нажатии на кнопку редактирования заметки, текст пересылается в поле ввода, а id заметки записывается в переменную currentId. 

function editTask(e) {

currentId = e.target.parentElement.id;

document.getElementById("taskInput").value = e.target.parentElement.innerText;

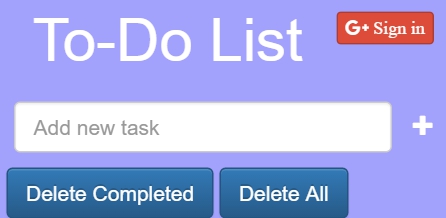
$('.new-todo').focus();

}

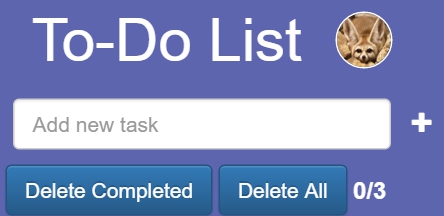
# ГЛАВА 2. Пользовательский интерфейс приложения.

## 2.1 Основные элементы пользовательского интерфейса.

Чтобы запустить расширение, необходимо кликнуть по иконке расширения в верхней панели. Если пользователь еще не авторизован, то появится окно фиксированного размера с кнопкой “Sign in”.



После нажатия на кнопку появится окно запроса на использование личной информации о пользователе, при подтверждении которого кнопка авторизации пропадет, а вместо нее появится аватар авторизованного пользователя (image).



Заметки появляются в виде ненумерованного списка в основном блоке приложения. Когда высота списка заметок начинает превышать высоту блока – появляется возможность скроллить заметки, хоть самого скроллера и не видно (display: none).

Кнопки редактирования и удаления заметки по умолчанию не видны и появляются только тогда, когда пользователь наведет курсор на заметку.



## 2.2 Bootstrap.

Для оформления элементов использовался Bootstrap (фреймворк) – свободный набор инструментов для создания веб-приложений. Был выбран именно он, так как основными преимуществами Bootstrap’a являются:

* Высокая скорость масштабирования.
* Простота в использовании.
* Гармоничный дизайн.
* Хорошая совместимость с браузерами.

В проекте были использованы элементы:

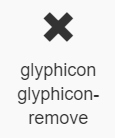
* Для добавления:



* Для редактирования:



* Для удаления:



Для оформления кнопки авторизации был использован Bootstrap-класс btn-google и fa-google-plus:

<button id="authorizeButton" class="btn btn-xs btn-google"><i class="fa fa-google-plus">Sign in</i>

Для оформления кнопок удаления использовался класс btn-primary:

<button id="clearTasksBtn" class="btn btn-primary">Delete All</button>

Анимация ожидания также реализована с помощью классов данного фреймворка:

<i class="fa fa-spinner fa-pulse fa-3x fa-fw margin-bottom”id="loading"></i>



## 2.3 Фон.

В качестве фона расширения используются 6 специально подобранных изображений котиков в определенной цветовой гамме. Фон меняется динамически каждый раз при запуске приложения. Цвет верхнего блока также меняется каждый раз в зависимости от выбранной картинки.

Для создания зависимостей была использована структура данных Map с возможностью добавления связанной пары: ключа и значения.

var backgroundMap = new Map();

backgroundMap.set('cats/1.jpg', '#494b73');

backgroundMap.set('cats/2.jpg', '#5c65ad');

backgroundMap.set('cats/3.jpg', '#b3d0db');

backgroundMap.set('cats/4.jpg', '#9fbdd2');

backgroundMap.set('cats/5.jpg', '#a9cbdf');

backgroundMap.set('cats/6.jpg', '#a2a1fb');

# ГЛАВА 3. Google Chrome расширение.

## 3.1 Понятие Google Chrome расширения.

В настоящее время большинство людей используют свои компьютеры в основном для выхода в Интернет, поэтому браузеры являются наиболее часто запускаемыми программами. Для более комфортной работы пользователей существует возможность создания расширения, поскольку создатели браузеров предоставляют всю необходимую информацию для разработки.

Расширение — это компьютерная программа, работающая в связке с браузером и наделяющая его новыми возможностями. На самом базовом уровне расширение Chrome – это набор HTML, CSS, JavaScript-файлов, позволяющий добавить некоторую функциональность в Chrome через JavaScript API, который предоставляет браузер. По сути, расширение – это веб-страница в Google Chrome, имеющая доступ к некоторым дополнительным API. Расширения также могут работать только на определенных страницах, выполняться в фоне или модифицировать содержание страницы. [[1]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ.)

## 3.2 Структура Google Chrome расширения.

Первое, что необходимо сделать, это создать базовую структуру проекта и папку со всеми необходимыми файлами, такими как .html, .css, .js, .ico. Типичное расширение для Chrome состоит из файла манифеста и некоторой комбинации из фоновой страницы, страниц пользовательского интерфейса и сценариев. Сначала необходимо заполнить файл манифеста с именем manifest.json. Этот файл содержит информацию, необходимую Chrome для того, чтобы загрузить расширение — заголовок, описание, язык, необходимые разрешения, иконки и т.п. [[2]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ.)

Для начала необходимо заполнить обязательные поля в файле манифеста:

{

"manifest\_version": 2,

"name": " To-Do List",

"description": "This extension will show To-Do List",

"version": "1.0"

}

Далее - рекомендуемые поля:

“default\_locale”: “en”; // язык по умолчанию

“icons”: {…},

“short\_name”: ”Short Name”;

Заполнение “permissions” - массив с дополнительными правами расширения на выполнение тех или иных действий:

"permissions": [

"identity",

"https://accounts.google.com/o/oauth2/token"

]

"oauth2": {

"client\_id" : "CLIENT\_ID.apps.googleusercontent.com",

"scopes": [

"https://www.googleapis.com/auth/drive",

"https://www.googleapis.com/auth/plus"

]

}

Необходимые настройки политики безопасности контента (Content Security Policy).

CSP — это механизм обеспечения безопасности, с помощью которого можно защищаться от атак с внедрением контента. CSP описывает безопасные источники загрузки ресурсов, устанавливает правила использования встроенных стилей, скриптов. Для работы с Google API в расширении необходимо внести скрипт https://apis.google.com/ в список разрешенных скриптов с ключевым словом self (соответствует текущему источнику). [[3]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ.)

"content\_security\_policy":"script-src 'self' <https://apis.google.com/>;”

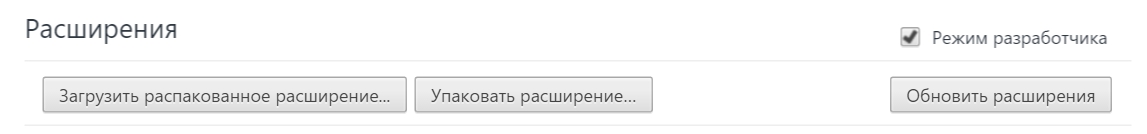
Установка иконки приложения:

"browser\_action": {

"default\_icon": "icon.png"

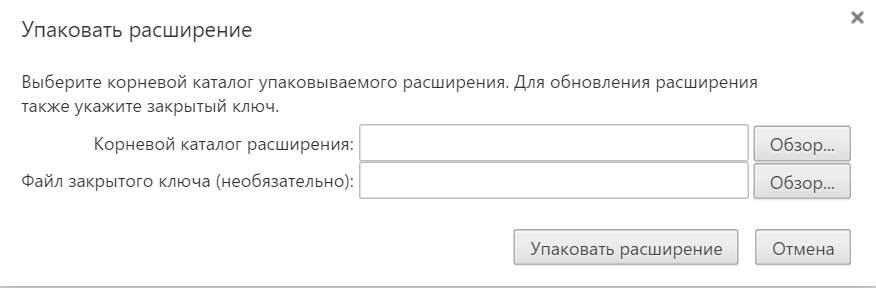
}

Чтобы протестировать разработанное расширение необходимо подключить его в браузере. Для этого надо включить режим разработчика в браузере, перейти в **Меню Chrome -> Инструменты -> Расширения** и загрузить распакованное расширение.

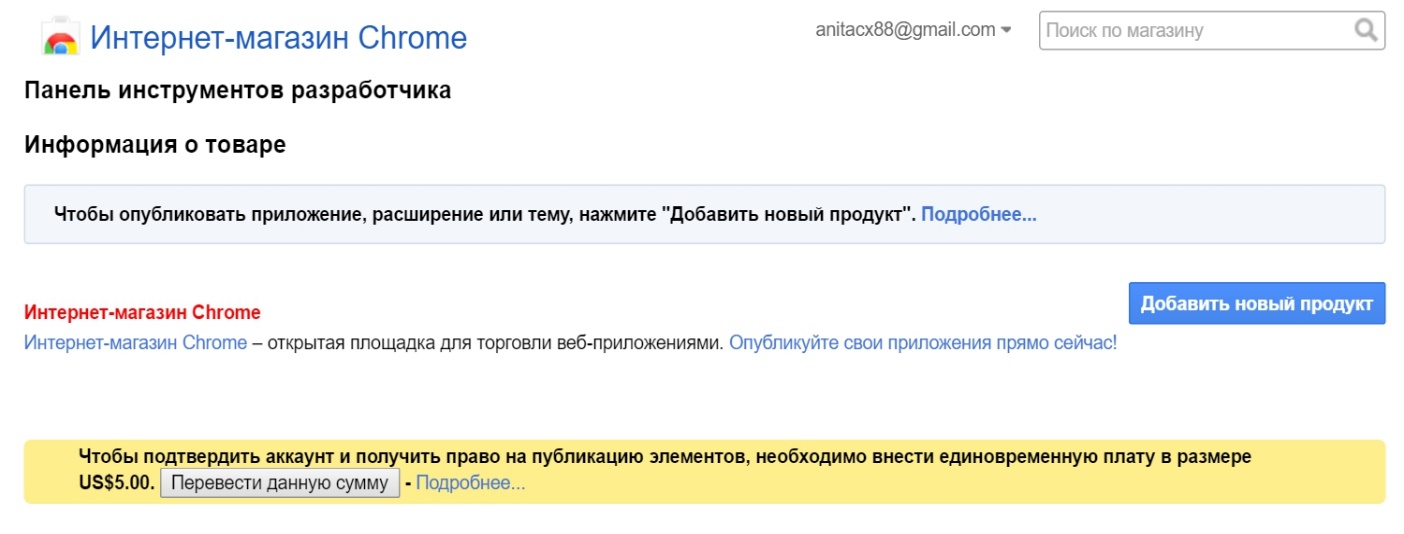


Если файл манифеста будет содержать ошибку – то расширение будет отключено или просто не загрузится. Установленный значок в виде иконки должен появиться на правой панели. После этого можно начинать работу с расширением.

Также существует возможность обратно упаковать расширение для дальнейшего его распространения. Chrome запросит каталог с распакованным расширением и файл секретного ключа. Упаковав расширение, его можно распространять любым способом — по электронной почте, через свой Web-сайт, посредством программы установки или же с помощью сервиса Chrome Web Store.



Чтобы опубликовать разработанное расширение в интернет-магазин Chrome, необходимо зарегистрироваться в качестве разработчика и добавить упакованное расширение через панель разработчика.



# ГЛАВА 4. Авторизация.

## 4.1 Авторизация через социальные сети.

Подавляющее большинство современных сайтов и web-приложений вместе с основной системой авторизации предоставляют пользователям возможность авторизоваться через социальные сети. Такой подход очень удобен для посетителя сайта, поскольку ему вообще не нужно заполнять регистрационные формы. Вместо этого достаточно выбрать кнопку социальной сети, в которой у него уже есть профиль и подтвердить доступ. По статистике, обязательная регистрация отталкивает пользователей от дальнейшего пользования продуктом, поскольку возникает необходимость придумывать и запоминать пароль, тратить время на авторизацию и подтверждение на почте.

Также, такой подход удобен и для разработчика, поскольку избавляет от необходимости писать свой собственный сервер авторизации, хранить данные о пользователях (например, в базах данных или в обычном текстовом файле), предусматривать шифрование данных, реализовывать работу с сессиями или с cookie.

При реализации входа на сайт (в приложение) с помощью социальных сетей нужно отметить, что пароль от аккаунта пользователя остается неизвестным для сайта (приложения), передается только необходимая информация из профиля. Но нужно обращать внимание на то, что иногда могут запрашиваться разрешения не только на регистрационные данные из социальных сетей, но и на доступ к дополнительной информации, такой как список контактов или дата рождения.

При этом, существует возможность давать пользователям действовать от своего аккаунта, то есть писать комментарии от своего имени или отмечать публикации с сайта у себя на странице в социальной сети.

## 4.2 OAuth протокол.

## 4.2.1 Преимущества и недостатки протокола OAuth.

Не так давно, каждый веб-сервис требовал отдельной собственной регистрации, но все большая часть из них дает возможность пользователю использовать для авторизации свой аккаунт в социальных сетях. Реализуется эта возможность благодаря протоколу OAuth. Это открытый протокол для авторизации, который позволяет предоставить третьей стороне ограниченный доступ к защищённым ресурсам пользователя без необходимости передавать ей (третьей стороне) логин и пароль.

При использовании OAuth-протокола пользователь не передает свой логин и пароль к защищенным ресурсам напрямую в приложение. Поэтому у пользователя больше оснований доверять приложению, поскольку пользователь может быть уверен, что несанкционированный доступ к его личным данным невозможен. Не владея логином и паролем пользователя, приложение сможет выполнять только те действия с данными, которые разрешил пользователь, и никакие другие. Так же при разработке приложения не нужно заботиться об обеспечении конфиденциальности логина и пароля пользователя, например, шифруя его. [[4]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ.)

Однако существуют и недостатки, для которых, к сожалению, нету универсального решения:

* Когда пользователь меняет пароль, приложение больше не может получить доступ к защищенным ресурсам.
* Единственный способ запретить приложению доступ к защищенным ресурсам - изменить пароль, что одновременно запретит доступ к ресурсам и другим приложениям, которые ранее его имели.

## 4.2.2 OAuth 2.0.

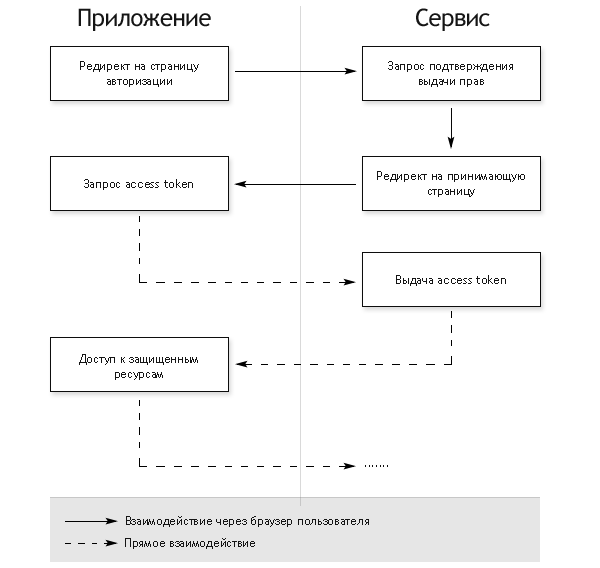
OAuth 2.0 основан на использовании базовых веб-технологий: HTTP-запросах, редиректах и т. п. Поэтому использование OAuth возможно на любой платформе с доступом к интернету и браузеру: на сайтах, в мобильных и desktop-приложениях, плагинах для браузеров.

Общая схема работы приложения, использующего OAuth, такова:

1. Получение авторизации.
2. Обращение к защищенным ресурсам.

Результатом авторизации является **access token**— некий ключ (обычно просто набор символов), предъявление которого является пропуском к защищенным ресурсам. Обращение к ним в самом простом случае происходит по HTTPS с указанием в заголовках или в качестве одного из параметров полученного **access token'а**.

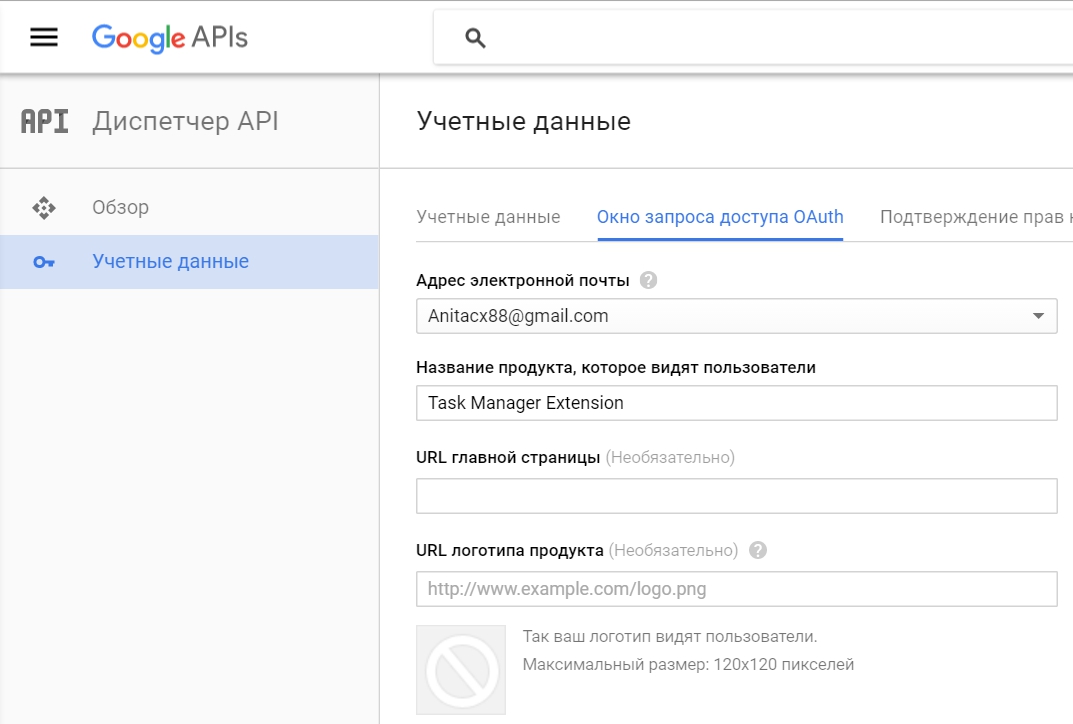
Процесс авторизации:



Перед инициализацией протокола OAuth, разработчик должен зарегистрироваться на сервере авторизации, предоставляя информацию о типе клиента, URL-адресе перенаправления (куда он должен быть перенаправлен для авторизации, после того, как сервер ресурса выдаст ему разрешение или же откажет в доступе) и другую информацию, необходимую серверу. Взамен он получает уникальный идентификатор клиента (**client\_id**) и секретный код клиента (**client\_secret**). Этот процесс называется регистрацией клиента. После регистрации клиент сможет взаимодействовать с сервером.

## 4.3 Регистрация приложения в Google API Console.

Для разработки приложения с поддержкой авторизации с использованием открытого протокола OAuth, необходимо зарегистрировать приложение в Google API Console. Это делается через регистрационную форму в Google Console, где необходимо предоставить информацию о своём приложении (название приложения, сайт приложения (или id Google Chrome расширения), URL-адрес перенаправления). [[5]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ.)



Как только приложение будет зарегистрировано, сервер будет выдавать учетные данные клиента в виде идентификатора клиента и секретного кода клиента. Идентификатор клиента - это публичная строка, которая используется API сервиса для идентификации приложения, а также используется для построения авторизационных URL’ов, которые представляются пользователям. Секретный код клиента используется для установки подлинности приложения для API сервиса, когда приложение запрашивает доступ к аккаунту пользователя и должен быть сохранен в секрете между приложением и API.

На основе этих данных, будет сформирована ссылка кода авторизации, которая выглядит следующим образом:

<https://account.google.com/o/oauth/auth?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=https://www.googleapis.com/auth/userinfo.email>

* https://account.google.com/o/oauth/auth — конечная точка авторизации
* response\_type = code — предусматривает что приложение запрашивает предоставление кода авторизации
* client\_id = CLIENT\_ID — идентификатор клиента приложения (значение, по которому API определяет (идентифицирует) приложение).
* redirect\_uri = CALLBACK\_URL — место, куда сервис перенаправляет пользователя после того, как предоставлен код авторизации.
* Scope = read — определяет уровень доступа.

## 4.4 Авторизация в расширении.

Процесс авторизации в расширении состоит из трех взаимосвязанных частей:

Функция вызывается, когда приложение полностью загружено и готово начать процесс авторизации:

function handleClientLoad() {

window.setTimeout(checkAuth, 1);

}

Функция проверки, авторизован ли текущий пользователь в приложении:

function checkAuth() {

gapi.auth.authorize({'client\_id': CLIENT\_ID, 'scope': https://www.googleapis.com/auth/drive, 'immediate': true}, handleAuthResult);

}

Функция вызывается, если сервер авторизации ответил на запрос:

function handleAuthResult(authResult) {

if (authResult && !authResult.error) {

gapi.client.load('drive', 'v2', function () {

getFile();

});

} else {

authButton.onclick = function () {

gapi.auth.authorize({ 'client\_id': CLIENT\_ID, 'scope': SCOPES[0], 'immediate': false}, handleAuthResult);

};

}

# ГЛАВА 5. Работа с Google Drive API.

## 5.1 Использование API при разработке приложений.

**API (Application Programming Interface)** - интерфейс для программирования приложений. Представляет собой набор готовых классов, процедур, функций, структур, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. Интерфейс позволяет разработчикам использовать готовые блоки для построения приложения, избавляя от необходимости писать основные части кода своими руками, при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована.

## 5.2 Описание Google Drive API.

Google предлагает широкий спектр интерфейсов прикладного программирования, чаще всего базирующихся вокруг актуальных продуктов Google. Интерфейс прикладного программирования – это интерфейс создания приложений, основанный на использовании готовых решений (например, функций).

**Google API** – это набор инструментов (клиентская библиотека), позволяющий создавать приложения, работающие с данными сервисов Google, в случае Google Drive API, с данными Google Drive пользователя. Появляется возможность использовать функции для создания\удаления файлов с диска, модификации файлов, создания и группировки файлов в папки.

При разработке приложения, разработчик может использовать данный API для реализации взаимодействия программы и Google Drive пользователя, обмена данными между ними. Таким образом, при работе с программой, конечный пользователь сможет либо вручную сохранять данные на Диске, которые после этого будут доступны на любых других устройствах, либо может быть реализовано автоматическое сохранение (например, через определенный промежуток времени без участия пользователя). [[6]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ.)

## 5.3 Работа с Google Drive пользователя.

В данной программе реализовано несколько функций для работы c Google Drive:

Функция для создания файла с заметками на диске пользователя.

function createFile() {

console.log('Creating a new file (' + metadata.title + ') without data:');

const boundary = '-------314159265358979323846264';

const delimiter = "\r\n--" + boundary + "\r\n";

const close\_delim = "\r\n--" + boundary + "--";

var base64Data = btoa(appData);

var multipartRequestBody = delimiter +'Content-Type: application/json\r\n\r\n' + JSON.stringify(metadata) + delimiter +'Content-Type: ' + contentType + '\r\n' + 'Content-Transfer-Encoding: base64\r\n' + '\r\n' + base64Data + close\_delim;

var request = gapi.client.request({

'path': '/upload/drive/v2/files',

'method': 'POST',

'params': {'uploadType': 'multipart'},

'headers': {

'Content-Type': 'multipart/mixed; boundary="' + boundary

},

'body': multipartRequestBody});

request.execute(function (res) {

fileDefault.id = res.id;

fileDefault.title = res.title;

updateFile(fileDefault.id);

});

}

Функция для перезаписи файла по ID файла.

function updateFile(fileId) {

if (!fileId) { return; }

console.log('Updating file (' + metadata.title + ') with data');

……

request.execute(function (res) {

console.log('File successfully updated!');

});

}

Функция поиска файла по имени на Google Drive пользователя.

function findFile(fname) {

var req = gapi.client.drive.files.list({q: "title='tasks'"});

req.execute(function (r) {

if (r.items && r.items.length && r.items[0].id) {

var request = gapi.client.drive.files.get({

'fileId': r.items[0].id

});

request.execute(function (resp) {

downloadFile(resp);

});

}

else {

createFile();

}

});

}

Функция скачивания данного файла с диска.

function downloadFile(file) {

if (file.downloadUrl) {

var accessToken = gapi.auth.getToken().access\_token;

var xhr = new XMLHttpRequest();

xhr.open('GET', file.downloadUrl);

xhr.setRequestHeader('Authorization', 'Bearer ' + accessToken);

xhr.onload = function () {

console.log('Data downloaded:')

appData = xhr.responseText;

};

}

}

Функция для проверки существования данного файла на Google Drive пользователя:

function checkFileExists(fname, callback) {

var req = gapi.client.drive.files.list({q: "title='tasks'"});

req.execute(function (r) {

if (r.items && r.items.length && r.items[0].id) {

callback(r.items[0].id);

} else { callback(); }

}); }

# ЗАКЛЮЧЕНИЕ.

Цель, определенная в начале данной курсовой работы, достигнута. Были решены необходимые задачи:

* Продуман и реализован функционал программы средствами JavaScript и библиотеки jQuery.
* Смоделирован удобный пользовательский интерфейс с помощью HTML и CSS.
* Изучены особенности разработки расширения для браузера Google Chrome и его структура.
* Реализована авторизация средствами Google Drive API.
* Реализована возможность сохранения заметок на Google Drive пользователя.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.

1. Создание расширения для Google Chrome - <http://programmict.ru/delaem-rasshirenie-dlya-google-crome/>. Дата доступа: 26.02.2016г.
2. Manifest File Format - <https://developer.chrome.com/extensions/manifest>. Дата доступа: 03.03.2016г.
3. Content Security Policy - <https://habrahabr.ru/company/nixsolutions/blog/271575/>. Дата доступа: 08.04.2016г.
4. OAuth tutorial - <https://developer.chrome.com/extensions/tut_oauth>. Дата доступа: 08.04.2016г.
5. Creating a Google Developers Console project and client ID - <https://developers.google.com/identity/sign-in/web/devconsole-project>. Дата доступа: 14.04.2016г.
6. JavaScript QuickStart Drive API - <https://developers.google.com/drive/v3/web/quickstart/js>. Дата доступа: 17.04.2016г.