Integrallis

Ideas, Implemented.

Spring Security & Oauth 2

By Joseph Faisal Nusairat

www.integrallis.com

PROBLEM

WEB SERVICES

RESTFUL WEB SERVICES

- Web services have become the best way of accessing services between a client and server.

- Many people started using SOAP based web services; however, they came with added complexity for the client and the server.

- Most public and even internal facing services started to use REST for ease.

- Many started as an easy way for internal apps to communicate

REPRESENTATION STATE TRANSFER

- RESTful web services was introduced and designed in 2000.
- REST was used to send requests with normal URLs as opposed to more complicated SOAP envelopes.
- Requests can be in the GET or POST.
- The response is then returned in XML or JSON
- Problem is there is no security by default

SECURITY

RESTFUL WEB SERVICES

- Unless you are creating the most basic application, there will often be a security apparatus that one needs to wrap around web service calls.

- Aside from security monitoring API usage and amounts by clients is also necessary

- Unlike SOAP, etc there are no set standards. People have implemented each differently.

EXAMPLES

SECURITY

- There are a variety of different types implemented
- 2 Way SSL, X.509 - enterprise like
- SAML - SOAP like
- HTTP Basic
- Home Grown
- Oauth


Google


flickr

WHAT IS OAUTH 2

BEGINNINGS

OAUTH

- The beginnings of Oauth occured in November 2006 with the discussion group created in April 2007 with an initial push from Twitter.

- In April 2010 Oauth 1.0 was published as RFC 5849

- On August 31, 2010 all 3rd party apps have been required to use Oauth to access Twitter.

- End goal was to create allow communication between 3rd party applications easier.

OAUTH OVERVIEW

OAUTH

- Is an open standard for authenticating between client and server.

- Oauth works by handing out tokens instead of credentials for authorization of specific resources on a specific site.

- Allows users to get access without sharing the permissions for a site.

- Users post with GET/POST and traditionally receive JSON as the response.

OAUTH2 OVERVIEW

OAUTH

- Oauth 2 is the next generation of Oauth that focuses on simplicity to the client developer.

- The core use is around the idea of Tokens. A client requests a token from a server and this serves as the authentication mechanism when accessing resources in subsequent requests.

- Authorization can then easily be altered on the server for the various clients and users the server exposes.

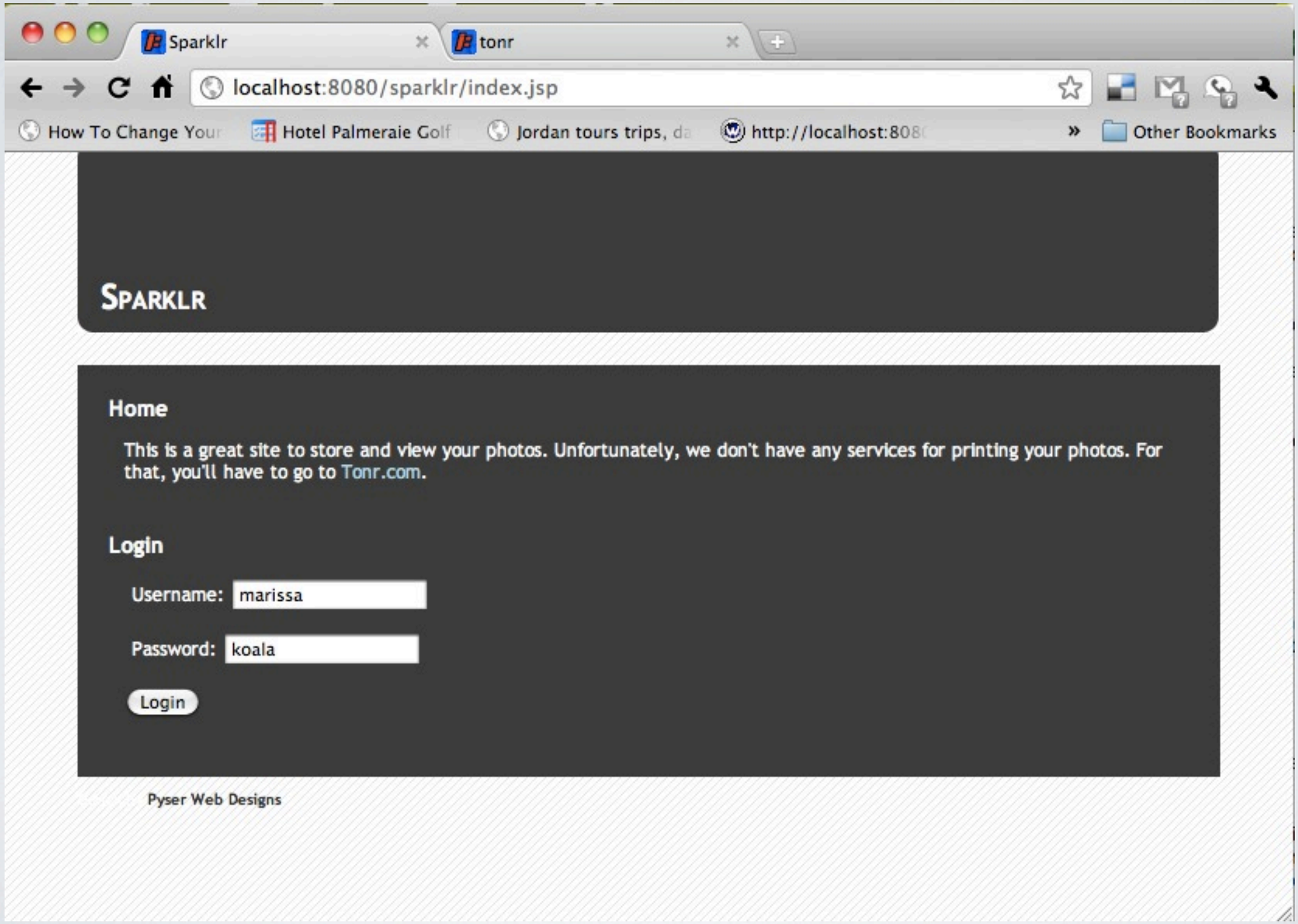
UNTRUSTED

TYPES

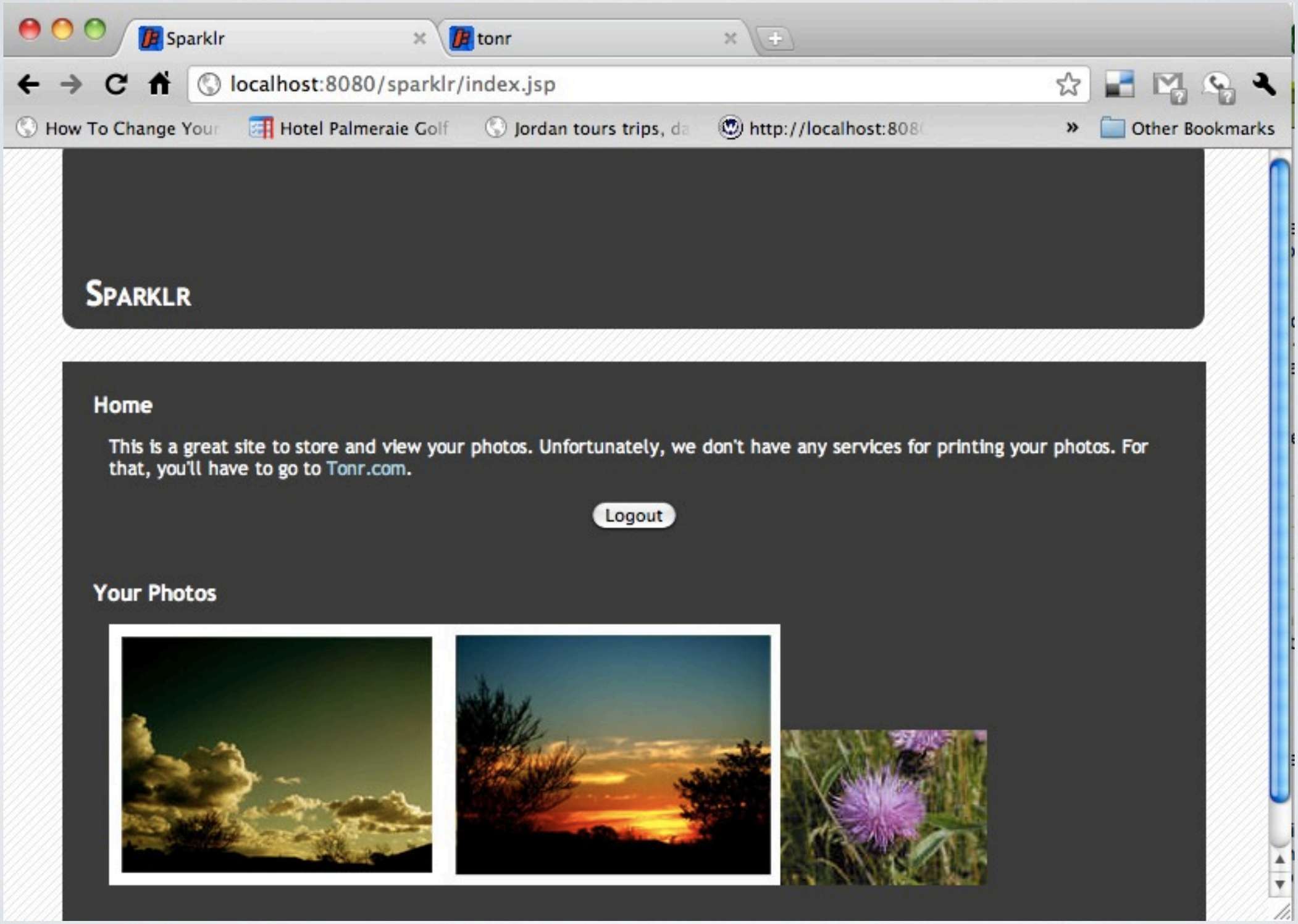
- When you want to open up your site to other people, you won't always know the person using the site. While you may want them to have access the site you also don't want to give them unfettered access.

- Untrusted access allows the user to have access to the site but won't allow them to have access to users without the user's explicit permission.

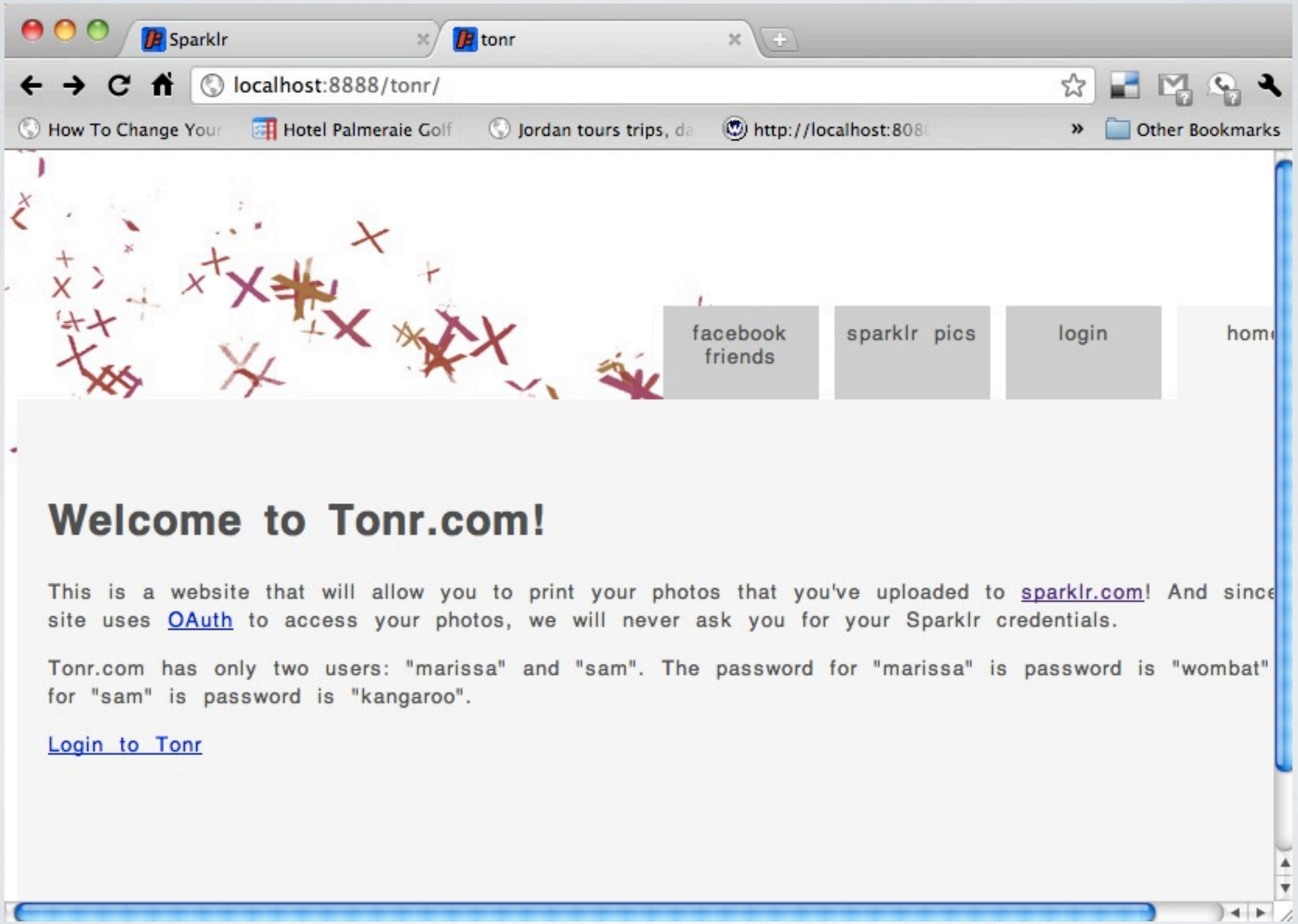
SITE I


Sparklr
tonr
localhost:8080/sparklr/index.jsp
How To Change Your
Hotel Palmeraie Golf
Jordan tours trips, da
http://localhost:808
Other Bookmarks
SPARKLR
Home
This is a great site to store and view your photos. Unfortunately, we don't have any services for printing your photos. For that, you'll have to go to Tonr.com.
Login
Username: marissa
Password: koala
Login
Pyser Web Designs

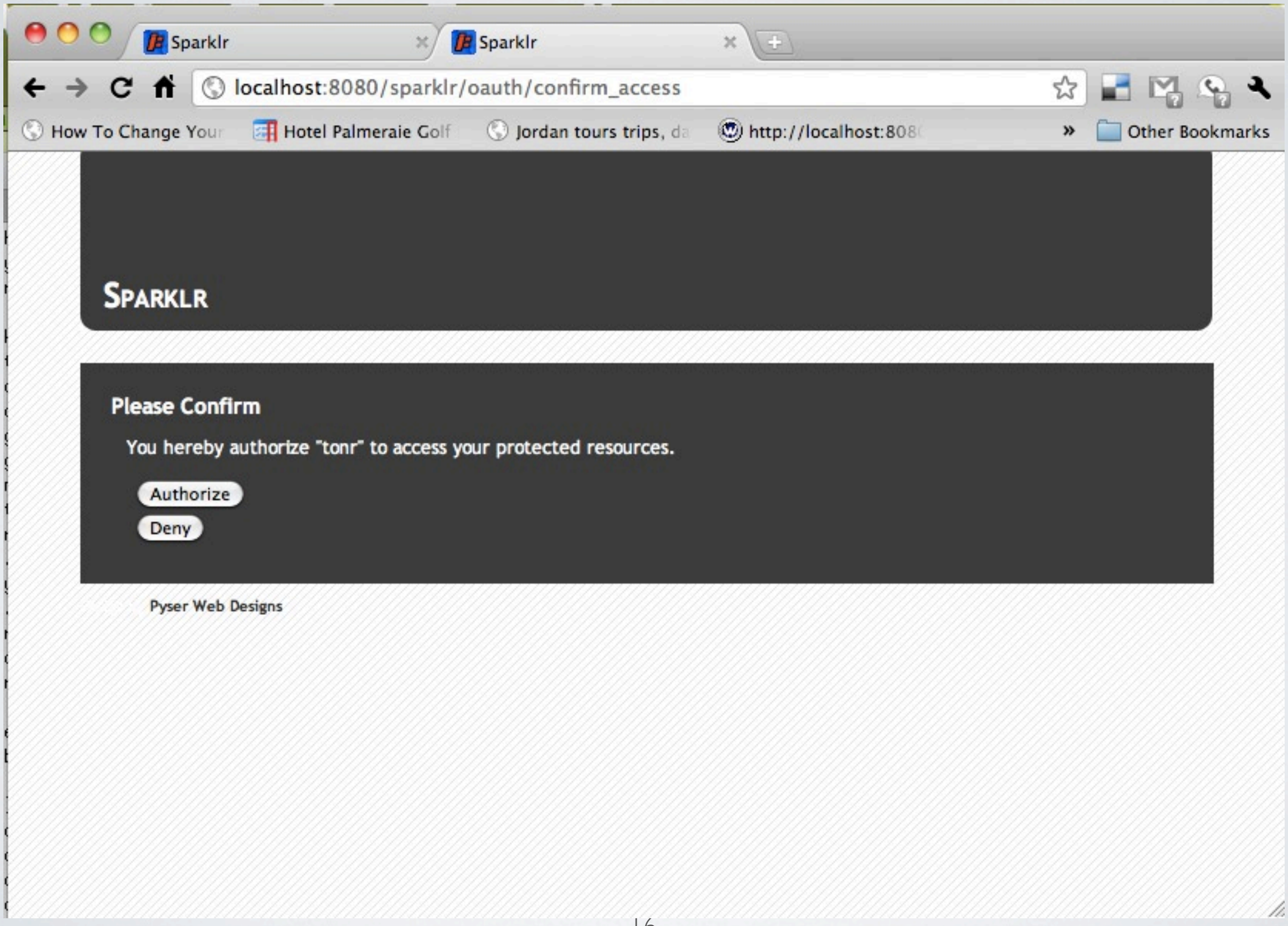
SITE 1


Sparklr
tonr
localhost:8080/sparklr/index.jsp
How To Change Your
Hotel Palmeraie Golf
Jordan tours trips, da
http://localhost:808
Other Bookmarks
SPARKLR
Home
This is a great site to store and view your photos. Unfortunately, we don't have any services for printing your photos. For that, you'll have to go to Tonr.com.
Logout
Your Photos

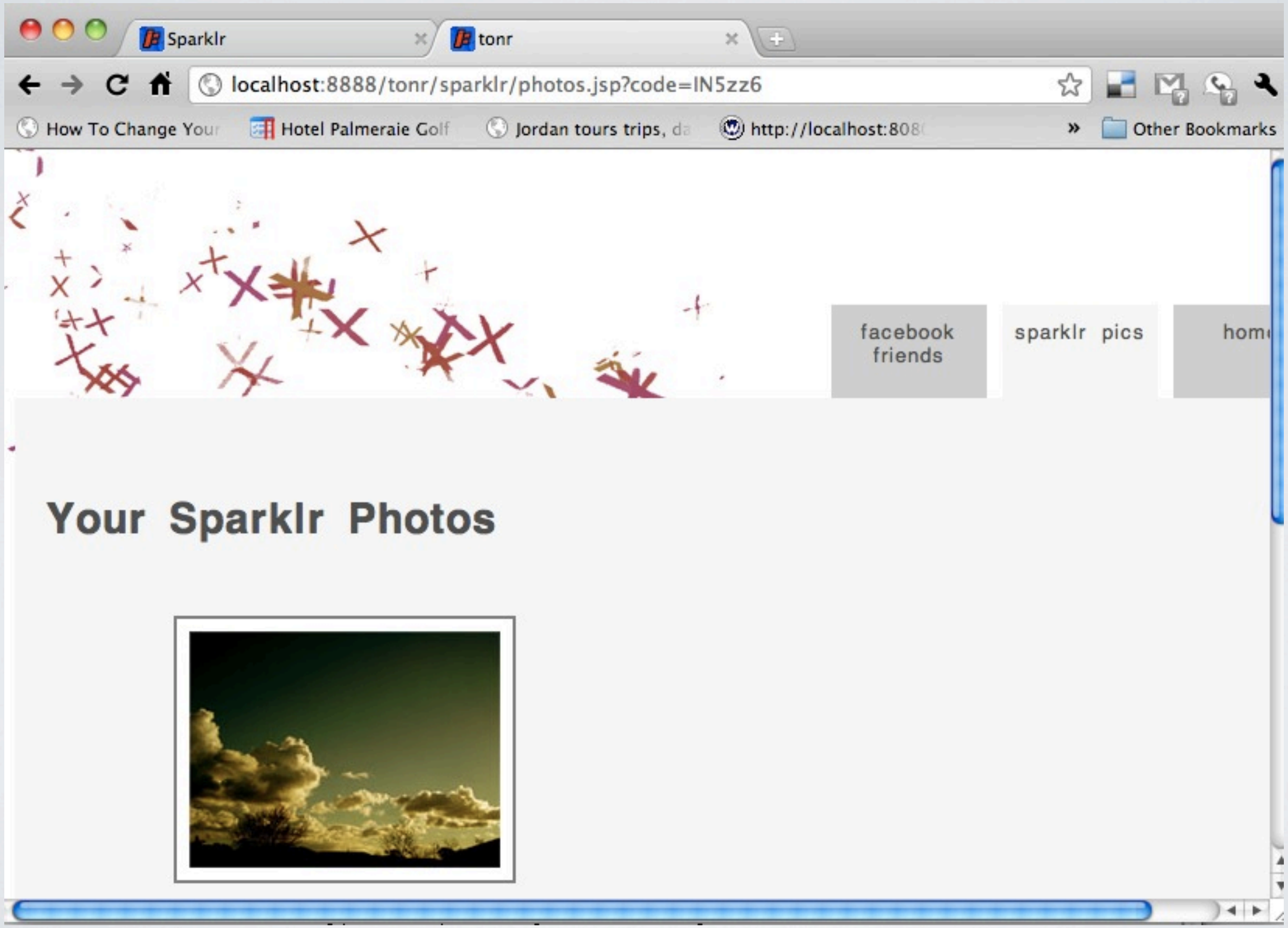
SITE 2 - 3RD PARTY



SITE 2 - 3RD PARTY



SITE 2 - 3RD PARTY



TRUSTED

TYPES

- Conversely, you want to give transparent access to the site. This is useful when you are developing a mobile or desktop client piece.
- Also be used when having a trusted partner.
- The software is developed by you so you want to make access to the site transparent. You can trust yourself when accessing the website.

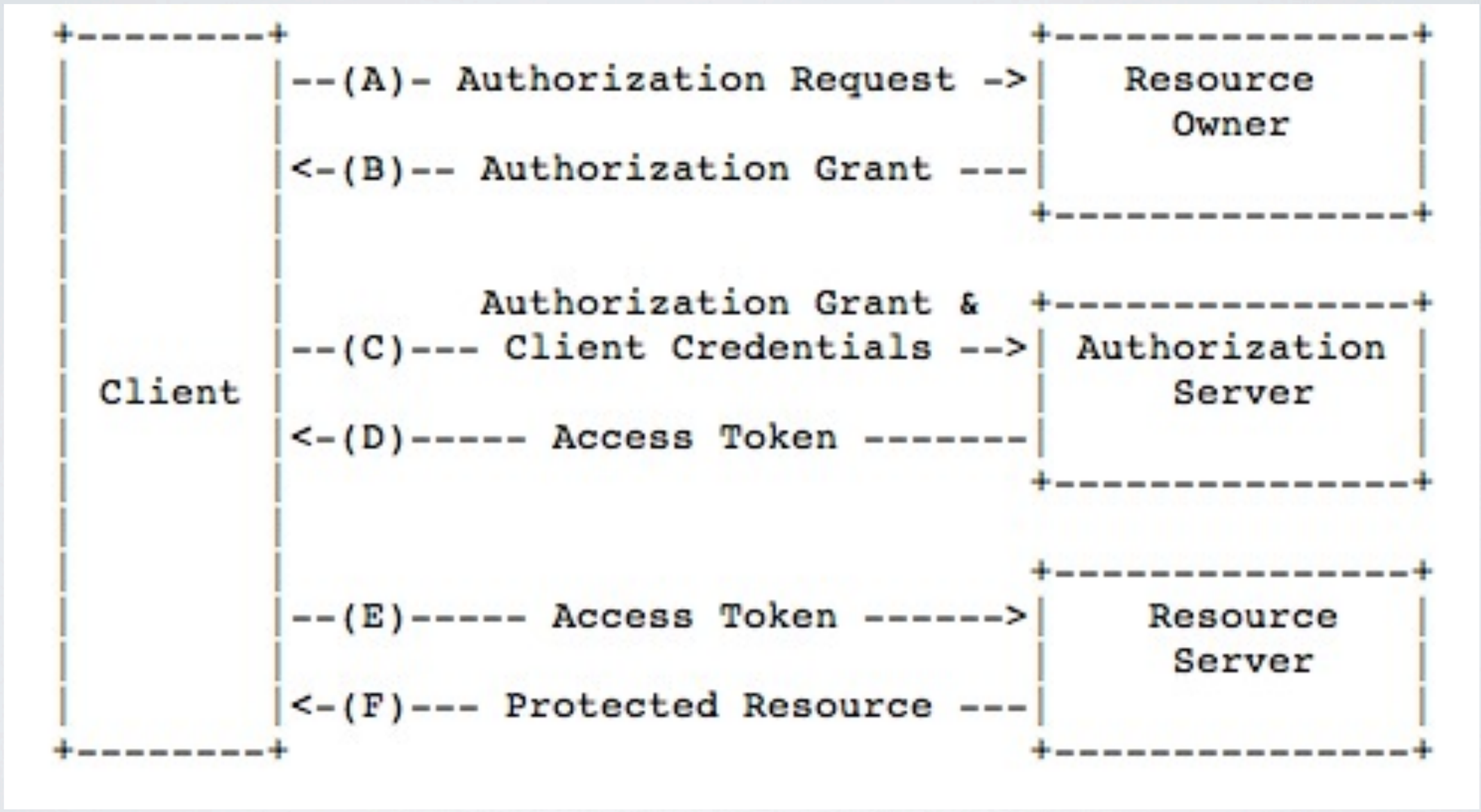
COMMUNICATION

TYPES

- Communication occurs by first requesting a token from the server.
- This token is then used for subsequent requests to the resources on the server.
- If the token is about to expire you can refresh it.

OAUTH2 FLOW

OAUTH



TYPES OF SECURITY

VARIOUS SECURITY

OAUTH

- Client Password
- Client Credentials
- Password
- Refresh Token

CLIENT PASSWORD

CLIENT - SERVER AUTHORIZATION

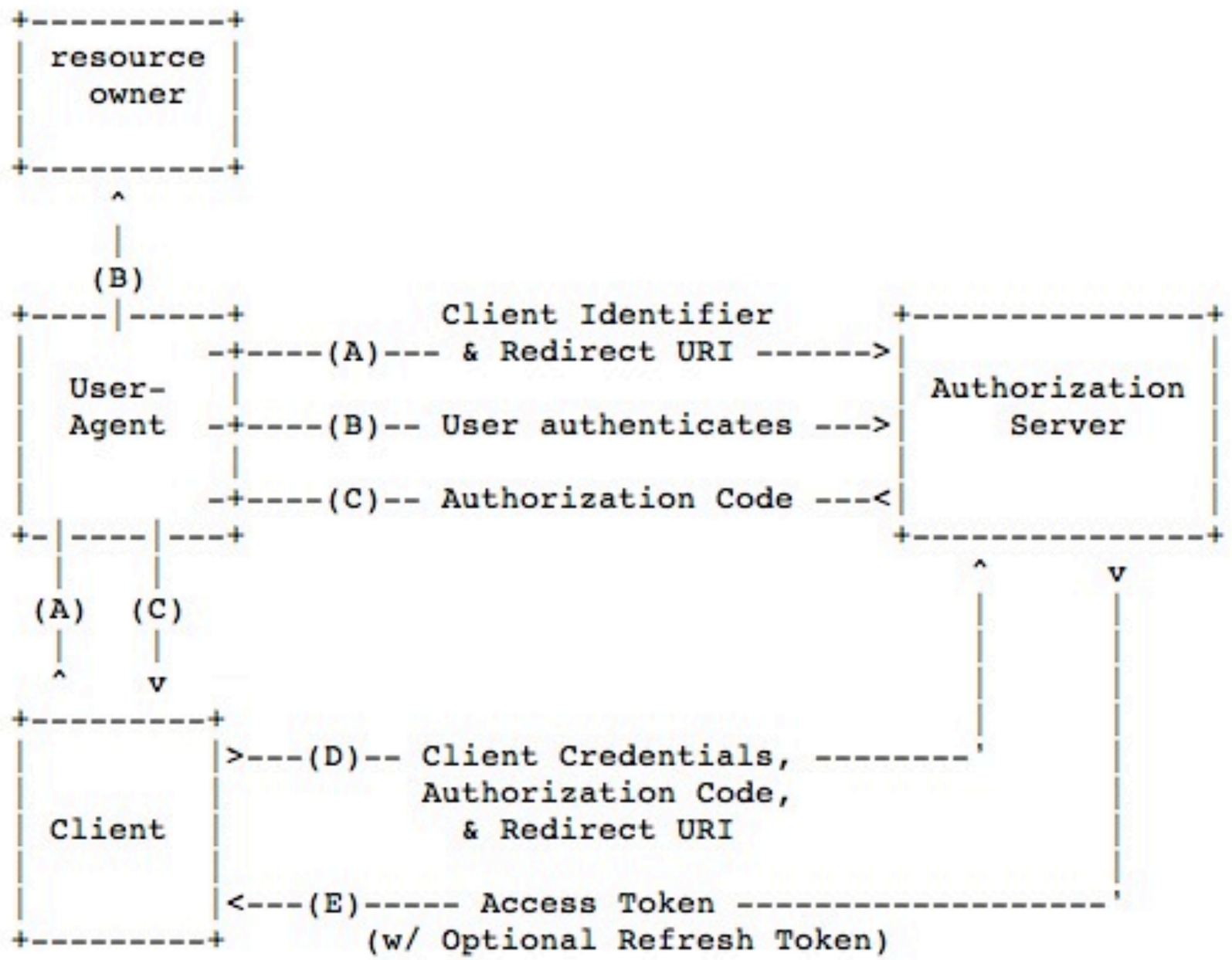
- Typically the access you see when logging into a third party application to access Facebook.

- This is used when you want to grant an untrusted source access to your site.

- This will have the client redirect to your site to enter the user name and password to grant access, thus the 3rd party application will never know the user name and password

DIAGRAM

CLIENT PASSWORD



CLIENT CREDENTIALS

CLIENT - SERVER AUTHORIZATION

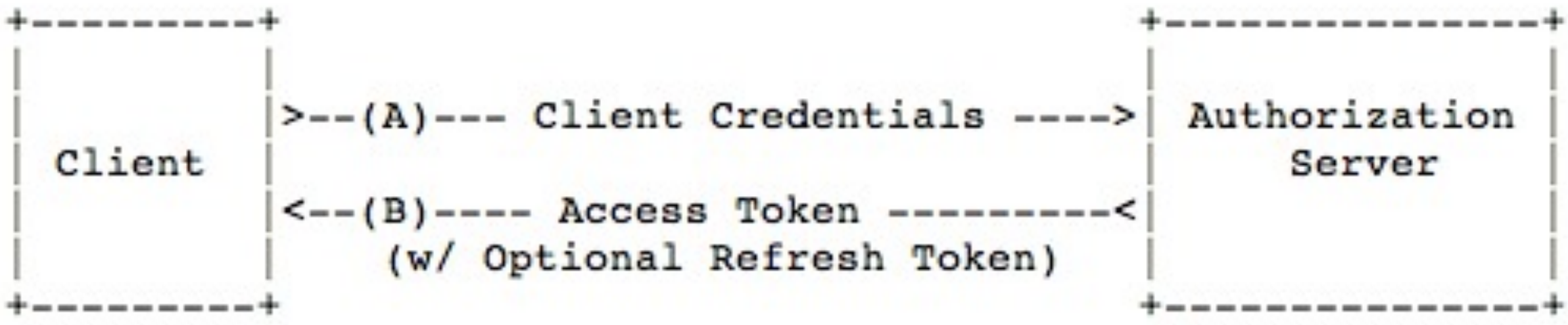
- This is for authorizations that only require access to non user specific resources on the site.

- This is used when you want to control access to 3rd party users or your own access but NOT access to individual user data.

- There would be no "acceptance" page for this criteria.

DIAGRAM

CLIENT CREDENTIALS



PASSWORD

CLIENT - SERVER AUTHORIZATION

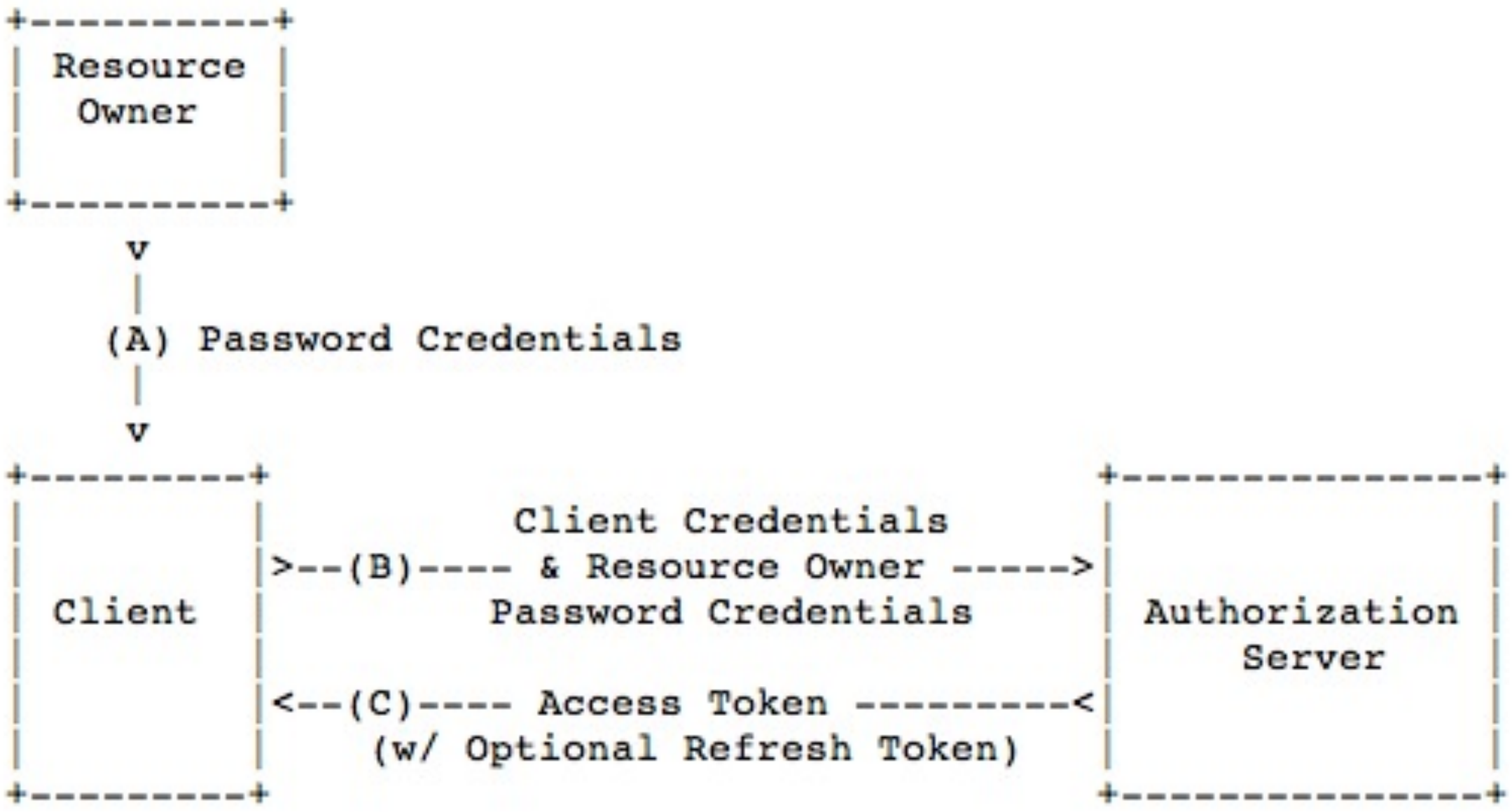
- This is similar to client - password authorization; however, the difference is major and this is for a trusted source.

- The difference is we allow the trusted client application to take the username and password and pass them to the service directly.

- This allows for a more seamless application.

DIAGRAM

PASSWORD



REFRESH TOKEN

CLIENT - SERVER AUTHORIZATION

- While each authorization returns a token, the tokens have an expiration time.

- This is used by any of the other types of authorizations to request longer use of the token.

RETRIEVING RESOURCES

ACCESSING A RESOURCE

RETRIEVING RESOURCES

- Accessing a resource should be simple. This is where the RESTful model will come back into play. In this model we will access whatever resources the server has exposed.

- One passes in the authorization token to the server and that will allow the authorization of what was authorized in the authentication piece.

REFRESHING THE TOKEN

REFRESHING TOKEN

REFRESH TOKEN

- Accessing a resource should be simple. This is where the RESTful model will come back into play. In this model we will access whatever resources the server has exposed.

- One passes in the authorization token to the server and that will allow the authorization of what was authorized in the authentication piece.

SPRING SECURITY

SPRING SECURITY

THE BASICS

- Spring Security is Springs defacto standard for securing a web site.
- Now on version 3.x it has namespace support in Spring application and Grails support via plugins.
- Can be used to easily secure Controllers, views, parts of the page, etc.
- Can authenticate with LDAP, OpenID, Facebook, etc

USER DETAILS

AUTHENTICATION

- The User class is the base security class for your application. This is a spring class and this is what will use to tie your authorizations, username, and back end domain object to.

- Grails extends this file.

- It can be extended to save the domain object, customize the salt usage, etc

- `org.springframework.security.core.userdetails.User`

SPRING SECURITY OAUTH2

OAUTH2 SECURITY

- A side project of Spring Security originally started to support OAuth; however, OAuth 2 support is nearing final completion.

- http://static.springsource.org/spring-security/oauth/

- The plugin provides support for server and client support for Oauth 2 integrating with Spring Security.

- Integration can be done with namespaces provided by OAuth2.

SERVER MODE

SPRING SECURITY OAUTH2

- Server mode is the ability to receive incoming OAuth2 requests.
- The code will add filters and providers that allow the clients to access your existing Spring Security via OAuth2.

CLIENT MODE

CLIENT - SERVER AUTHORIZATION

- Client mode is for when your server pieces need to interact with outside OAuth2 provided applications.

- The code will encapsulate the security calls to the OAuth 2 provider and allow the developer to only worry about handling the response of the calls.

FILTERS

SPRING SECURITY FILTERS

- Filters are the mechanism Spring Security uses to control the application.
- Filters are used for authorization requests, resource protection, and even exception handling of requests.
- OAuth adds 3 filters to our Spring Filter set up

FILTERS

SPRING SECURITY FILTERS

- OAuth2AuthorizationFilter
  - Filter used to authorize the request
- OAuth2ExceptionHandlerFilter
  - Filter used to handle any errors from the request
- OAuth2ProtectedResourceFilter
  - Filter used to process Secured items

PROVIDERS

SPRING SECURITY PROVIDERS

- Spring Security providers provide the actual work in authenticating.

- The providers will call out to your client services to verify the user, they will create the token, and also process the saving of the authentication object.

- Several providers are given with Spring Security OAuth 2

GIVEN PROVIDERS

SPRING SECURITY PROVIDERS

- refreshAuthenticationProvider (AM)
- accessGrantAuthenticationProvider (CS)
- verificationCodeAuthenticationProvider (AM,VCS)
- clientPasswordAuthenticationProvider (CS)
- clientCredentialsAuthenticationProvider (AM)

THE CLIENT

CLIENT

- The client is the central part of authenticating a client to the API.

- The client contains is the entry way into authenticating the client to access the resources on the system.

- This will also require you to write an external web piece for users to register for accessing the site, this is where that access piece can go.

CLIENT DETAIL SERVICE

LOADING THE CLIENT

- This is the one service you will HAVE to write.
- To initially test your system you can use the in memory version that comes with OAuth 2 implementation; however, this is not production suitable.
- Class should implement `ClientDetailsService`

```
ClientDetails
    loadClientByClientId(String clientId)
    throws OAuth2Exception { }
```

CLIENT DETAIL OBJECT

LOADING THE CLIENT

- This will store the details about the client stored in the database.
- clientId : unique client id
- authorizedGrantTypes : password, client_credentials, authorization_code, refresh_token
- authorities : Spring authority role types
- webserverRedirectUri : URI to be redirected to when using

CLIENT DETAIL OBJECT

LOADING THE CLIENT

- webserverRedirectUri : URI to be redirected to when using client-password authentication

- isSecretRequired : whether a secret key is needed to authenticate this client

- scope : various scopes to limit the item to.

AUTHENTICATION

SPRING SECURITY OAUTH2

- `org.springframework.security.oauth2.provider.OAuth2Authentication`
- This is the class that is created when authenticated
- Will contain your username, if authenticated, granted authorities
- If logged in with user it will be their credentials, if not will be your client credentials.

TOKEN

TOKENS

- The token and the using of the token is the key to using OAuth 2. The token is what is returned by the authorization filters, and the token is what is used when accessing the resources on the site.

- When authorizing a request a Spring Authentication object is created. The Spring Security OAuth2 token services will then serialize the authentication object and store it to the database.

- When accessing a protected resource this object is retrieved.

TOKEN SERVICES

PROVIDED BY OAUTH

- Spring Security OAuth provides 2 out of the box mechanisms for storing the Authentication objects.

- `InMemoryOAuth2ProviderTokenServices`
- `JdbcOAuth2ProviderTokenServices`
- `RandomValueOAuth2ProviderTokenServices`
  - Used to create your own custom implementation

VERIFICATION CODE

VERIFICATION CODE SERVICE

- The verification code mechanism is used when performing an authorization code request.

- The authorization object is stored in the database along with the code, the code is then returned to the client so that they may access the server.

HOW AUTHORIZATION WORKS

- First retrieve your authorizaton token:
  - /oauth/authorize
- Params:
  - grant_type / client_id / client_secret
  - username / password

WHAT HAPPENS

AUTHORIZING

- Authorization will be picked up by your Authorization Filters
- Authorization filter can call out to AccessGrantAuthenticationProvider which will contain your Client Details Service.
- The Spring Security Authentication manager is passed in
- A Success handler is also provided

WHAT HAPPENS

RESOURCE CONFIGURING

```
oAuth2AuthorizationFilter
(org....oauth2.provider.OAuth2AuthorizationFilter) {
  authenticationManager = ref('authenticationManager')
  // Need to set it cause by default it instantiates a blank one
  authenticationSuccessHandler =
    ref ("oauth2AuthorizationSuccessHandler")
```

```
oauth2AuthorizationSuccessHandler
(org....oauth2.provider.OAuth2AuthorizationSuccessHandler) {
    tokenServices = ref("mongoOAuth2ProviderTokenService")
  }
```

WHAT HAPPENS

AUTHORIZE RETURN

- If authorization is successful the return will contain two items: access_token / expires_in

- The access token will be used as our token to interact with the server.

- The expires in will be the amount of time this token is valid for before it needs to be refreshed. Default is 43,199 seconds (~12hrs)

ACCESSING A RESOURCE

RESOURCE ACCESSING

- Accessing a resource is done by adding a token to the request of the resource.

- The token can be added either on the GET/POST stream or by passing the token in the header.

- One the Get its defined with the param "oauth_token"

- In the header its "Authorization" = "oauth2 $token"

WHAT HAPPENS?

RESOURCE ACCESSING

- The **`oAuth2ProtectedResourceFilter`** intercepts the call. This resource will then use the token provided to look up the account object from the server and populate that in the current security context holders account setting.

- The user will be able to access the resource assuming the resource assuming that resource has the proper authorizations.

REFRESHING THE TOKEN

RESOURCE ACCESSING

- Tokens will have to be refreshed if they are set to expire. Its best for the client to keep track of the expiration.

- This is more critical for when using password authorization as a client credential authorization has less worry since its always active.

NAMESPACES

SERVER CONFIGURATION

```
<oauth:provider client-details-service-ref="clientDetails"
  token-services-ref="tokenServices" >
    <oauth:verification-code
      user-approval-page="/oauth/confirm_access"/>
</oauth:provider>

<oauth:client-details-service id="clientDetails">
    <oauth:client clientId="my-trusted-client"
authorizedGrantTypes="password,authorization_code,refresh_token"/>
    <oauth:client clientId="my-trusted-client-with-secret"
authorizedGrantTypes="password,authorization_code,refresh_token"
secret="somesecret"/>
    <oauth:client clientId="my-less-trusted-client"
authorizedGrantTypes="authorization_code"/>
    <oauth:client clientId="tonr"
authorizedGrantTypes="authorization_code"/>
</oauth:client-details-service>
```

NAMESPACES

SERVER CONFIGURATION

```
<oauth:resource id="facebook"
 type="authorization_code" clientId="162646850439461"
 clientSecret="560ad91d992d60298ae6c7f717c8fc93"
 bearerTokenMethod="query"
 accessTokenUri="https://graph.facebook.com/oauth/access_token"
 userAuthorizationUri="https://www.facebook.com/dialog/oauth"/>
```