Candy Basket Documentation

Release 1.0

Ghislain Hachey, Maya Goldman and Dan McGarry

January 30, 2015

Contents:

CHAPTER

ONE

USER GUIDE

This manual is for the pilot of the Mnemoniq software, release 1.0. It is written in three sections. The first sets out the purpose of the software and describes how it is designed to improve an organisation's ability to monitor and learn from the information it collects. It will be most useful to people using Mnemoniq in a team-based environment. The second section describes how to install Mnemoniq on an intranet. The third section is for developers who want to play around with Mnemoniq and suggest improvements to be incorporated into subsequent releases of the software.

Mnemoniq has been developed by Dan McGarry and Derek Brien of the Pacific Institute for Public Policy (PiPP), Ghislain Hachey of Nuzusys, and Louise Shaxson of ODI. Feedback on the software, and on this manual, will be welcomed: we are keen to improve it based on people's experience of using Mnemoniq, and developers' ideas about what it could do better.

1.1 Introduction

[Mnemonic: a tool designed to aid memory]

Mnemoniq is a browser-based tool for improving your organisational memory. Much of the information that organisations need to monitor their performance is ephemeral, either tacit knowledge held in people's heads or fragments of information from meetings, records of phone conversations, thoughts that come to you on the bus, internet links or bits of key reports. Traditional knowledge management systems work poorly for this, but Mnemoniq allows you to record and tag any information that can be stored electronically and retrieve it via a tag cloud.

For example, you come out of a meeting having been given a series of insights into a project you're working and a link to a useful report that others in the team ought to read. During the coffee break you had a phone conversation with a collaborator on a different project that suggests things are not going to plan. Instead of sending round a series of emails when you get back to the office (if you remember to), Mnemoniq allows you to enter these 'memories' into your team's collective memory instantaneously, commenting on what they mean for the team and tagging them to reflect how your team works and what they need to know. As long as people keep entering their memories into Mnemoniq, your organisation's collective memory will be up to date.

Mnemoniq is not just a storage tool, it is also a powerful way of retrieving memories. Because it is structured around a tag cloud, it is very easy for anyone to dive into any level of detail they want to. Someone waiting to go into a meeting can quickly review the latest relevant memories to be sure that they are fully aware of what the team collectively knows. Managers can see whether themes are emerging across different projects, review everything related to 'strategy' or refresh their memory of the detail of a project. It is particularly useful in fast-moving or political environments where information needs to be at people's fingertips. Mnemoniq is mobile; you can enter the information via tablets or smartphones meaning that you can record a memory (and others can retrieve it) as soon as you have it.

1.1.1 What memories should you record and how should you tag them?

Mnemoniq is not designed to store large pieces of information such as reports or slideshows: these should be stored in your regular knowledge management systems. And you will already have systems for recording the sort of information

that goes into quarterly and annual reports. Mnemoniq stores everything else—the information that does not get reported to others but that you and your team need to know to ensure that you are on the right track. It sits behind your own organisation's own security systems so the information you enter is secure which makes it a useful tool for storing sensitive information. You can set the access to as many or as few people as you like, though the more people who have access to it the better it functions as an information-sharing tool.

What you record it is up to you but we suggest that once Mnemoniq is installed for a team, you meet to discuss the sorts of information you will find useful and to set a few ground rules about how to tag it. The tags are predictive, meaning that as you begin to type the text for a tag Mnemoniq throws up the tags that already exist. This helps ensure that dodgy spelling does not affect how information is tagged.

You can develop as many tags as you like, but the software does require you to use the three supertags: *confirm*, *challenge* and *surprise*. These draw on Irene Guijt's work on monitoring and learning. Mnemoniq was developed to help PiPP better monitor what it was doing to foster political debate in the Pacific (see www.pacificpolicy.org). They needed to tag information according to whether it confirmed that they were broadly on the right track (confirm), indicated that there was a push back against what they were doing (challenge), or was out of left field and needed to be thought through in more detail (surprise).

Note that because it Mnemoniq is a system for recording information, what it contains will come under the remit of any data protection legislation that may be in force in your country. What you record and how you record it will need to be compliant.

1.1.2 Can I generate reports?

Although Mnemoniq's flexibility means that you can use it in pretty much any way you choose, it will be most effective when it is used honestly; sharing fragments of information that make sense to you and your team but not necessarily to others. The software therefore does not allow you to generate reports, or contain other functions which would allow the information to be viewed externally.

1.1.3 What sort of performance management systems does it require?

Mnemoniq needs to be actively managed as a tool for sharing information and making sense of it. Space needs to be made within your team's decision-making systems to ensure that the information people are putting into it is useful. It is as important to take information out as it is to put it in, which you can do by calling up your team's memories at your regular team meetings and using it to help decide what the information means in terms of what you need to start doing, stop doing, keep doing or change.

1.1.4 Why does it not have more functionality?

While the software is not just a dumping ground, it also does not do your thinking for you. You need to think about why you are putting the information in so you can tag it in ways that will make sense to your team. And because what you will be putting in will be fragments rather than whole documents (possibly photos accompanied by half-digested thoughts, or audio files) you need to collectively think about what it means when you take it out.

1.2 Using the tool

1.2.1 Logging in

In order to access Mnemoniq you will need to begin by typing in the address of the URL. A username and password will be requested: once you have entered them you will be taken to the Home page, an example of which is shown in the screenshot below. The interface is designed to work with touch, allowing you to use a mobile platform for ease of entry and constant availability. This also allows you to enter the information in an unobtrusive fashion.

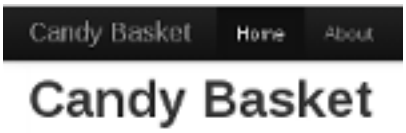
1.2.2 Toolbar

The black toolbar running along the top of the screen allows you to navigate between pages and contains four options:

- **Home** The Home option takes you back to the main display page, such as the one displayed in the screenshot above.
- **About** Contains links to the documentation.
- **Contact** Contains contact information.
- **Add new...** The Add new function allows you to add a new entry into the database.

1.2.3 Header

The page contains a header and a subtitle at the top of the page, below the toolbar, which you can define yourself.



1.2.4 Entry bar

The entry bar (or search bar) allows entries to be selected according to their tags. By entering keywords in the entry bar only those entries which have been tagged with those keywords will be selected. You can enter two or more words into

the entry bar, separated by a space, and only those entries which have been tagged with both words will be displayed. By clicking the *X* on each tag you will return to the full list of entries.

1.2.5 Word cloud

The word cloud displays all the tags which have been entered so far. The size of the words reflects the number of times the tag has been entered. The cloud is also clickable, giving you an alternative way of selecting a group of entries. By clicking on one of the words in the cloud, only those entries which have that tag will be displayed.

Once you have selected a tag, using either the entry bar or the word cloud, the word cloud updates itself to reflect the tags in the refined selection. You may then select a second tag by again clicking on the word cloud. The previous search will not be cleared by doing so.

1.2.6 Tagging

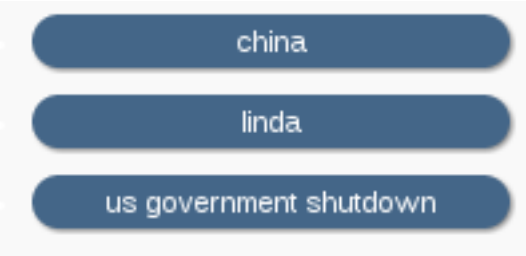
Tagging is the most important component of the software.

1.2.7 General tags

Once an entry has added, the tags associated with that entry will be displayed on the left hand side. Tags are designed to be:

- Short
- Immediately comprehensible
- Lower case
- Not categories, but flexible
- Intuitive

- Technical jargon may be useful, as shadowing the language which is spoken within the user group may facilitate precision.
- As you begin to specify a tag for an entry, tags which have already been entered will come up as suggestions. There are two reasons for this function, which is not intended to be prescriptive. For practical reasons, as it facilitates ease of entry on a mobile device and, importantly, it allows the number of equivalent tags to be reduced to reduce in order to streamline functions such as the word cloud.



1.2.8 Supertags

A supertag is a special type of tag which will be coloured in either green, yellow or red according to its content. This is a structured form of tagging. While the other tags provide context, supertags classify the information according to whether or not the entry confirms, surprises or challenges what you are currently doing and thinking beliefs. These tags float to the top of the tag list, and the bar across the top gives you a visual indication of the proportion of super tags. This is designed to show you how things appear to be developing:

- if the entire line is green (Confirm) it would indicate that you are either being complacent or ignoring information that might suggest you should be doing things differently
- if there is a high proportion of yellow, it could indicate you are working in a changing environment
- if there is a high proportion of red, it may indicate two things. One interpretation could be that what you are doing is inappropriate. Another interpretation is that what you are doing is generating a backlash—which you may in fact see as progress, particularly if you are trying to change the terms of a debate

1.2.9 Tag count function

The tag count function provides allows for a quick check-in of how things appear to be going. However, should you be leaning heavily on one category of supertag, you may want to think a little bit harder about what the reason for that is. Given that employees are inputting the information themselves, the information is easily biased in that employees select which information is considered pertinent. Should you find the majority of your supertags are:

- **Confirm**: then this would seem to indicate that you are being too complacent or ignoring challenging information (green).
- **Surprise**: this may indicate, for example, a changing environment (yellow).
- **Challenge**: this may suggest that the work being done by the organisation may be inappropriate or need revision (red).

1.2.10 Add new entry

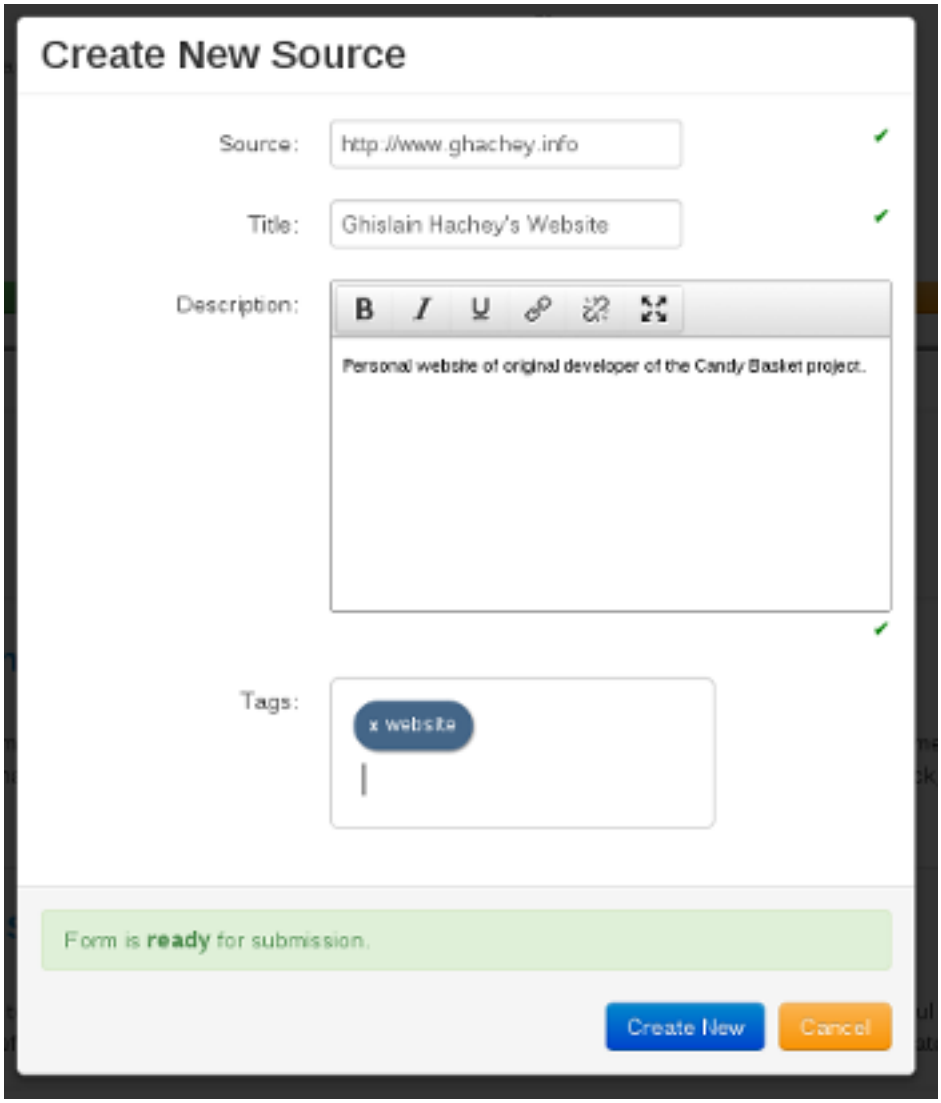
When adding a new entry, there are four fields you need to fill in:

- The *Source* entry bar allows you to specify the URL from which your information is sourced. Even if you enter information offline this field does need to be filled in, but you can enter anything such as a.com or offline.com
- The *Title* field allows you to specify a title for your entry
- The *Description* field should preferably be used to explain why you thought the entry was important.
- The *Tags* field is used to enter tags.

Once you have finished entering the fields select Save Changes and Close to finish adding the entry. The entry then floats to the top of the entry list, and the date it was created shows on the right hand side. If an entry is edited, the editing date is stored in the place of the creation date, and it again floats to the top.

Submitting a new entry (or updating an existing one) will not be accepted until the form is *ready* for submission. For example, when adding a new entry a form will look like this.

Once you start entering valid data the fields will be marked ready and you will be able to submit your entry. It will be clear when the form is ready for submission.



1.2.11 Sources of information

The information is added by individual employees. The tool is designed to comprise external information, and may consist of anything, such as:

- Photos
- Web sites
- Emails
- Short pieces of text entered offline
- Entering material which is not available online can either be manually

inputted, or the page number can be referenced. The URL can be replaced with anything (such as offline.com), whether or not such an URL exists.

1.3 Contact details

For more information, please contact:

Name	Organisation	Title	Issue	Email Address
Louise Shaxson	ODI	Research Fellow	Monitoring techniques, feedback and general support	l.shaxson@odi.org.uk
Dan McGarry	PiPP	Chief technologist	Technical support	dmcgarry@pacificpolicy.org
Derek Brien	PiPP	Executive Director and co-founder	Feedback	dbrien@pacificpolicy.org

CHAPTER

TWO

ADMINISTRATOR GUIDE

2.1 Introduction

Note: Mnemoniq is referred herein as Candy Basket and memories and candies for historical reasons. This will eventually be changed, but for now the technical terminologies used are the old ones.

This guide is meant for the person who wants to deploy Candy Basket in its own environment. Candy Basket supports single signon for authentication, and once authenticated users have full access to the application. Candy Basket was tested to work with both Windows Server 2008 and Samba 4 as Active Directory and Domain Controller.

Candy Basket can be deployed on essentially any operating system using any HTTP server. However, the only tested and supported approach is deploying Candy Basket on the latest Debian 7 with Apache and WSGI.

2.2 Pre-Built VMWare Image

The supported way of deploying Candy Basket is the pre-built VMWare image. This image comes ready to be deployed to your VMware infrastructure with minimal configuration.

The installation of the image should be a simple matter of booting it from your VMware host server. However, you will need to correctly setup networking, preferably in bridge mode where it will live side-by-side with the rest of the network. You will need to assign it a static IP address.

It is assumed you run your own internal DNS. You will need to add your chosen fully qualified host name as an A record to this new server, for example:

```
192.168.1.10     A      candy.pacificpolicy.org
```

Reverse DNS must also be working for single sign-on to work. Once the new server is part of your network and you can correctly ping it using its host name you will need to change the domain name in several places.

- All three Apache Virtual Hosts configuration in */etc/apache2/sites-available/*
- The CORS Python file in */srv/www-apps/candybasket/backend/config.py*. The allowed domains should be identified there. This should a single fully qualified host name such as *candy.pacificpolicy.org*.
- The Candy Basket RESTful service location in */srv/www-apps/candybasket/frontend/static/js/services.js*. The variable *wsUrl* should be changed to your own fully qualified host name.

Restart Apache, tail -f its */var/log/error.log* file and try pointing your browser to *https://candy.pacificpolicy.org*.

2.3 VMWare Test Environment

If you are interested in improving the VMWare image or anything in the build process these notes might come in handy to create your own test environment. We use VMware workstation with a number of VMware Virtual Machines (VM) to simulate a production environment. On my machines I create the following VMs:

- **debian.pacificpolicy.org the machine running the Candy Basket** application deployed as detailed in deploy-debian-apache
- **winserver2008.pacificpolicy.org An Active Directory and Domain** Controller running Windows Server 2008
- **samba4.pacificpolicy.org An Active Directory and Domain** Controller running Samba4 on Debian
- **win7.pacificpolicy.org** A Windows 7 workstion

2.3.1 VMWare Vitual Machines Lab Setup

- Install VMware workstation
- Create a number of vitual machines: one Windows Server 2008 (winserver2008.pacificpolicy.org), one Windows 7 Pro (win7.pacificpolicy.org), one Windows 8 Pro (win8.pacificpolicy.org), one Debian Linux to host software application (server.pacificpolicy.org).
- Create a private LAN Segment. This can be done in any VM's LAN settings.
- Configure the Windows server 2008 VM with two virtual network interfaces: one will NAT from the host and the other should be part of a the private LAN segment.
- Configure the VMs that will be part of the domain (Win7, Win8 and Debian Linux) with the private LAN segment for networking.

2.3.2 Windows Server 2008 Configuration

Open the Server Manager and add and configure necessary roles including:

- Active Directory Domain Services steps.
- DHCP Server (e.g. assign a pool of 192.168.30.100-200 on the Windows server internal network interfaces with static IP of 192.168.30.1)
- DNS Server (e.g. create A records for all machines in the test lab: win7.pacificpolicy.org, debian.pacificpolicy.org, winserver2008.pacificpolicy.org and add CNAME records for services on the debian server, something like CNAME www.pacificpolicy.org -> debian.pacificpolicy.org)
- Network Policy and Access Services (to turn it into a Gateway (Route/NAT)) step.

If you want to access private VMs using SSH from your host you could port forward traffic through the Windows Server with the following commands:

```
C:\> netsh interface portproxy add v4tov4 listenport=2222 listenaddress=172.16.228.136 connectport=22
```

2.3.3 Join Domain with Windows VMs

Boot the VMs and verify that networking is working correctly. Make sure IP addresses are assigned according to the DHCP pool configure in the previous step. Test DNS with nslookup or simple ping:

```
[root]$ nslookup winserver2008.pacificpolicy.org
[root]$ nslookup win7.pacificpolicy.org
[root]$ nslookup server.pacificpolicy.org
```

If the Windows Server was correctly setup as a gateway in "Network Policy and Access Services" access to the Internet should work. Verify that the time is correctly being synced with Internet servers. You should now be able to join the domain.

2.3.4 Join Domain with the Linux Server VM

The Linux server can make use of a static IP address. The appropriate A record should be added to the Windows Server 2008 DNS zone file. Test networking (DNS and Internet access) and join the domain as a samba client.

Regarding Linux Guest VMs, VMWare specifically recommends to use NTP on the guest instead of the VMware's time syncing feature (see here). Since time sync is critical for the proper functioning of the single signon authentication it is better to be safe and follow best practices. Installing ntp on Debian is easy:

```
[root]$ aptitude install ntp
```

The default values in /etc/ntp.conf are fine, but it is recommended by VMware to add the following line to make sure ntp always syncs regarless of any large time jump it observes between Internet NTP servers and local OS time:

```
tinker panic 0
```

then restart ntp:

```
[root]$ service ntp restart
```

You might find it useful to allow yourself to SSH inside the Debian VMware from your host terminal; otherwise, getting in and out of the VM's terminal is annoying and you loose the ability to copy/paste from host to VM. Apparently setting up port forwarding on Windows Server 2008 R2 through the GUI is cmpletely broken. I have not tested this myself, but it is easy to do on the command line as detailed here.

2.4 Deployment on Debian with Apache HTTP server

The steps to deploy Candy Basket on a production Debian server are very similar to setting up Candy Basket in a development environment.

2.4.1 Oerating System

Download an ISO of the latest Debian and do a bare installation with only the standard utilities and SSH. You may also install any other useful packages you will most likely eventually need such as rsync, ntp, curl, wget.

```
[root]$ aptitude install rsync ntp sudo curl wget vim locate screen zip git
```

Create a user to 'own' the Candy Basket application

```
[root]$ adduser candy
```

2.4.2 Dependencies

Install the Apache HTTP server with Python support.

```
[root]$ aptitude install apache2 libapache2-mod-wsgi python-dev
```

Install Python and preferably virtualenv to cleanly isolate the application and its dependencies. This process is exactly as defined in **python-and-virtualenv_**. The only difference in production will be the creation of an BASELINE virtual environment for Apache: this is an empty virtual environment with its own clean Python installation meant to power Python web applications. The BASELINE virtual environment could be owned by any user but in this case we will make use of the *candy* user.

Create the BASELINE.

```
[candy]$ mkvirtualenv BASELINE
```

And finally, tell Apache about it by adding the following line in */etc/apache2/conf.d/wsgi.conf*.

```
WSGIPythonHome  /home/candy/.virtualenvs/BASELINE
```

At this point, you should have two virgin virtual environments, test it before going further.

```
[candy]$ lsvirtualenv
BASELINE
========

candy.pacificpolicy.org
=======================
```

Install CouchDB in a similar way as you would in a development environment. In production, CouchDB can also be owned by *candy* instead of *root*. The new Linux Filesystem Hierarchy Standard [FHS] recommends installing such "non-distro provided" or optional software in */opt/*. As user *root*, make a nice place for it.

```
[root]$ mkdir /opt/candy/
[root]$ chown candy:candy -R /opt/candy/
```

As user *candy*, install CouchDB as detailed in **couchdb_**. Couchdb dependencies will have to be installed as *root*, of course. In production it would be a good idea to have an init script. On Debian you can simply edit the distribution provided */etc/init.d/skeleton*. Test you correctly created the init script:

```
[root]$ /etc/init.d/couchdb start
[root]$ ps -ef | grep couch
(couchdb processes running)
[root]$ /etc/init.d/couchdb stop
[root]$ ps -ef | grep couch
(no couchdb process output)
[root]$ /etc/init.d/couchdb restart
```

Once the init script is working you can instruct the system to start it on boot:

```
[root]$ update-rc.d couchdb default
```

You should not be able to login CouchDB by pointing your browser to *http://localhost:5984/_utils* or *http://ip.address:5984/_utils* if you are connecting from a remote machine although by default CouchDB listens on localhost so this would involved changing the configuration. From the administrative interface create a database with the name *candybasketng* and add the design documents (i.e. views) located in *db/views/docs/*.

2.4.3 Candy Basket as Apache Virtual Host

The final step to deploy the Candy Basket application. Create a directory where the application will be served from:

```
[root]$ mkdir -p /srv/www-apps/candybasket/
```

More work will be done to improve the development to production workflow cycle but for now simply *rsync* the whole source tree into */srv/www-apps/candy.pacificpolicy.org/*. Let's assume you have the latest source checked out in */home/candy/*:

```
[root]$ rsync -avg /home/candy/tagging-tool/ /srv/www-apps/candybasket/
[root]$ chown candy:www-data -R /srv/www-apps/candybasket/
```

Create three Apache virtual hosts: one for the Candy Basket REST service and two for the Candy Basket application (HTTP and HTTPS). Sample configuration are included below.

candybasket.http:

```
<VirtualHost *:80>
        ServerName candy.pacificpolicy.org
        ServerAlias candy candy.pacificpolicy.org.vu
        ServerAdmin admin@localhost

        RewriteEngine on
        ReWriteCond %{SERVER_PORT} !^443$
        RewriteRule ^/(.*) https://%{HTTP_HOST}/$1 [NC,R,L]

        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

candybasket.https:

```
<IfModule mod_ssl.c>
<VirtualHost *:443>
        ServerName candy.pacificpolicy.org.vu
        ServerAlias candy candy.pacificpolicy.org
        ServerAdmin admin@localhost

        DocumentRoot /srv/www-apps/candybasket/frontend/

        <Directory /srv/www-apps/candybasket/frontend/>
                Order allow,deny
                Allow from all
        </Directory>

        Alias /help /srv/www-apps/candybasket/docs/build/html/

        ProxyPass /basket http://candy-restapi-v1.pacificpolicy.org.vu/basket
        ProxyPassReverse /basket http://candy-restapi-v1.pacificpolicy.org.vu/basket

        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/ssl_access.log combined

        SSLEngine on
        SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

        <FilesMatch "\.(cgi|shtml|phtml|php)$">
                SSLOptions +StdEnvVars
        </FilesMatch>
```

```
        <Directory /usr/lib/cgi-bin>
                SSLOptions +StdEnvVars
        </Directory>

        BrowserMatch "MSIE [2-6]" \
                nokeepalive ssl-unclean-shutdown \
                downgrade-1.0 force-response-1.0
        # MSIE 7 and newer should be able to use keepalive
        BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
</VirtualHost>
</IfModule>
```

candybasket-restapi-v1:

```
<VirtualHost *:80>
        ServerName candy-restapi-v1.pacificpolicy.org
        ServerAlias candy-restapi-v1 candy-restapi-v1.pacificpolicy.org.vu
        ServerAdmin admin@localhost

        WSGIDaemonProcess runservice user=www-data group=www-data processes=1 threads=5
        WSGIScriptAlias / /srv/www-apps/candybasket/backend/runservice.wsgi

        <Directory /srv/www-apps/candybasket/backend/>
#               Header set Access-Control-Allow-Origin "*"
#               Header set Access-Control-Allow-Credentials true
                WSGIProcessGroup runservice
                WSGIApplicationGroup %{GLOBAL}
                WSGIScriptReloading On
                Order deny,allow
                Allow from all
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Enable the needed modules and the new virtual hosts and then restart Apache:

```
[root]$ a2ensite candybasket.http
[root]$ a2ensite candybasket.https
[root]$ a2ensite candybasket-restapi-v1
[root]$ a2enmod ssl
[root]$ a2enmod rewrite
[root]$ a2enmod proxy
[root]$ a2enmod proxy_http

[root]$ service apache2 restart
```

Make sure name resolution is working for the domains used in the Apache Virtual Hosts. If you you do not have internal DNS adding the records in the servers' */etc/hosts* file will work:

```
127.0.0.1       candy.pacificpolicy.org
127.0.0.1       candy-restapi-v1.pacificpolicy.org
127.0.0.1       candy.pacificpolicy.org.vu
127.0.0.1       candy-restapi-v1.pacificpolicy.org.vu
```

Connect to the *candy* virtualenv and install Candy Basket's Python dependencies:

```
[candy]$ workon candy.pacificpolicy.org
(candy.pacificpolicy.org)[candy]$ cd /srv/www-apps/candybasket/backend/
(candy.pacificpolicy.org)[candy]$ pip install -r requirements.pip
```

As a final step to make sure that all the bits connect together the WSGI script */srv/www-apps/candybasket/backend/runservice.wsgi* should be verified. It is mostly also preconfigured except that the following two lines will depend on your own environment: what did you call the Python virtualenv (it's *candy* here) and what Python version is running on your OS. If steps herein were closely followed the following two lines should be edited and uncommented to look like:

```
# If using virtualenv, add the virtualenv's site-packages to sys.path as well
VENV_PATH = "/home/candy/.virtualenvs/candy.pacificpolicy.org/"
site.addsitedir(os.path.join(VENV_PATH,'lib/python2.7/site-packages/'))
```

Restart Apache, tail -f its */var/log/error.log* file and try pointing your browser to *https://candy.pacificpolicy.org*. At this point you should have a fully working albeit insecured installation of Candy Basket.

2.5 Windows Active Directory and Apache Kerberos Single Sign-on

Candy Basket can be securely deployed in a Windows environment with users authenticating to it using single signon (SOO). In other words, members of the domain that are logged in the network should be able to access the web application securely without providing credentials.

2.5.1 Install Necessary Software

Some kerberos, Apache and samba packages are needed:

```
[root]$ aptitude install apache2-mpm-prefork libapache2-mod-auth-kerb
[root]$ aptitude install krb5-config krb5-user krb5-clients samba-client ntp
```

2.5.2 Time Synchronization

This setup is high sensitive to clocks being in sync. The network time protocol is the best approach to make things work:

```
[root]$ aptitude install ntp
```

The default values in */etc/ntp.conf* are fine. However, ntp will stop syncing if it detects a large enough jump in time as it assumes you are getting time from an invalid source. Syncing using the provided default OS' ntp servers should be safe. If you are on VMware or anywhere the time may drift easily it would be a good idea to always sync regarless of any large time jump; it can be achieved by adding the following line at the top of */etc/ntp.conf*:

```
tinker panic 0
```

then restart ntp:

```
[root]$ service ntp restart
```

2.5.3 DNS Configuration

The setup here as three machines: a Windows Server 2008 RC2 with Active Directory and Domain Controller, a Debian werver running the web service and a Windows 7 workstation. Forward and reverse DNS should be configured

something like this:

```
winserver2008.pacificpolicy.org    A    192.168.30.1
debian.pacificpolicy.org    A    192.168.30.10
www.pacificpolicy.org    CNAME    debian.pacificpolicy.org
test.pacificpolicy.org    CNAME    debian.pacificpolicy.org
```

Make sure everything resolves as it should from within win7.pacificpolicy.org:

```
[root]$ nslookup debian.pacificpolicy.org
Server:        192.168.30.1
Address:    192.168.30.1#53
Name:    debian.pacificpolicy.org
Address: 192.168.30.10

[root]$ nslookup www.pacificpolicy.org
Server:        192.168.30.1
Address:    192.168.30.1#53
www.pacificpolicy.org    canonical name = debian.pacificpolicy.org.
Name:    www.pacificpolicy.org
Address: 192.168.30.10

[root]$ nslookup 192.168.30.10
Server:        192.168.30.1
Address:    192.168.30.1#53
10.30.168.192.in-addr.arpa    name = debian.pacificpolicy.org.
```

2.5.4 Kerberos configuration

Back on the Debian server, backup the original and create your own:

```
[root]$ sudo cp /etc/krb5.conf /etc/krb5.conf.bak

[libdefaults]
        default_realm = PACIFICPOLICY.ORG
        # The following krb5.conf variables are only for MIT Kerberos.
        krb4_config = /etc/krb.conf
        krb4_realms = /etc/krb.realms
        kdc_timesync = 1
        ccache_type = 4
        forwardable = true
        proxiable = true

[realms]
        PACIFICPOLICY.ORG = {
                kdc = winserver2008.pacificpolicy.org
                master_kdc = winserver2008.pacificpolicy.org
                admin_server = winserver2008.pacificpolicy.org
                default_domain = pacificpolicy.org
        }

[domain_realm]
        .pacificpolicy.org = PACIFICPOLICY.ORG
        pacificpolicy.org = PACIFICPOLICY.ORG

[login]
        krb4_convert = true
        krb4_get_tickets = false
```

Test Kerberos by getting a ticket-granting ticket (TGT) for the domain controller's Administrator user:

```
[root]$ kinit Administrator
Password for Administrator@PACIFICPOLICY.ORG:
[root]$ klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: Administrator@PACIFICPOLICY.ORG

Valid starting    Expires           Service principal
24/10/2013 15:13  25/10/2013 01:13  krbtgt/PACIFICPOLICY.ORG@PACIFICPOLICY.ORG
    renew until 25/10/2013 15:13
```

2.5.5 Configure Samba to Join the Domain

Backup the original configuration and use the minimal configuration below:

```
[root]$ cp /etc/samba/smb.conf /etc/samba/smb.conf.bak

[global]
        netbios name = debian
        realm = PACIFICPOLICY.ORG
        workgroup = PACIFICPOLICY
        server string = %h server
        dns proxy = no
        log file = /var/log/samba/log.%m
        max log size = 1000
        panic action = /usr/share/samba/panic-action %d
        security = ADS
        password server = winserver2008.pacificpolicy.org
        encrypt passwords = true
        passdb backend = tdbsam
        obey pam restrictions = yes
        unix password sync = yes
        passwd program = /usr/bin/passwd %u
        passwd chat = *Enter\snew\s*\spassword:* %n\n *Retype\snew\s*\spassword:* %n\n *password\supd
        pam password change = yes
        map to guest = bad user
        kerberos method = dedicated keytab
```

2.5.6 Join the domain

The server should be a member of the domain; this is easy with Samba:

```
[root]$ net ads join -U Administrator
Enter Administrator's password:
Using short pacificpolicy.org -- PACIFICPOLICY
Joined 'SERVER' to realm 'pacificpolicy.org'
```

If the domain is joined successfully a new Active Directory account will be created. That machine account could be used but I opted to create a specific user to handle authentication of the service. On the windows server add a new AD user account (e.g. I add a user HTTP Service with user httpservice) and make sure the password can not be reset and will last foreever.

Now you need to create the 'principle': someone or something to authenticate or authenticate to (e.g. users, services). This can be a little tricky and there are a few ways to achieve this. Use the kpass utility to create the keytab with the

principal; it will both add the service principal to the user and create a keytab which can later be used by a service such as Apache:

```
C:\> ktpass -princ HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG
-mapuser httpservice@PACIFICPOLICY.ORG
-crypto RC4-HMAC-NT
-ptype KRB5_NT_PRINCIPAL
-pass somepassword
-out c:\Temp\krb5.keytab
```

Copy the file *c:\Temp\krb5.keytab* on the Debian server somewhere appropriate (e.g. */etc/krb5.keytab*). Assign correct ownership and permissions:

```
[root]$ chown root.www-data /etc/krb5.keytab
[root]$ chmod 0640 /etc/krb5.keytab
```

This should be it, but some testing will help. Get a Ticket-Granting Ticket (TGT) for the service principal:

```
[root]$ kinit HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG
```

View the ticket from the cache:

```
[root]$ klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG

Valid starting    Expires           Service principal
30/10/2013 16:20  31/10/2013 02:20  krbtgt/PACIFICPOLICY.ORG@PACIFICPOLICY.ORG
    renew until 31/10/2013 16:20
```

Get a service ticket for the principal:

```
[root]$ kvno HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG
HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG: kvno = 4
```

List what is in the ticket cache and make sure you show the encryption type usig the '-e' flag:

```
[root]$ klist -e
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG

Valid starting    Expires           Service principal
30/10/2013 16:20  31/10/2013 02:20  krbtgt/PACIFICPOLICY.ORG@PACIFICPOLICY.ORG
    renew until 31/10/2013 16:20, Etype (skey, tkt): aes256-cts-hmac-sha1-96, aes256-cts-hmac-sha1-96
30/10/2013 16:22  31/10/2013 02:20  HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG
    renew until 31/10/2013 16:20, Etype (skey, tkt): arcfour-hmac, arcfour-hmac
```

Compare the service principal ticket above with the one from the keytab file which will be used by Apache:

```
[root]$ klist -e -k -t /etc/krb5.keytab
Keytab name: FILE:/etc/krb5.keytab
KVNO Timestamp        Principal
---- ---------------- ------------------------------------------------------
   4 01/01/1970 11:00 HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG (arcfour-hmac)
```

The KVNO (password version number) , the encryption type and the service principal name (i.e. HTTP/debian.pacificpolicy.org@PACIFICPOLICY.ORG) must all match. Apache VirtualHost Configuration

Add appropriate lines in the virtualhost to enable kerberos authentication:

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName test.pacificpolicy.org

        DocumentRoot /srv/www-apps/test-single-signon

        <Directory /srv/www-apps/test-single-signon>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all

                # Kerberos Single Signon
                AuthType Kerberos
                AuthName "Kerberos Login"
                KrbAuthRealms PACIFICPOLICY.ORG
                KrbServiceName HTTP
                KrbMethodNegotiate On
                KrbMethodK5Passwd On
                Krb5KeyTab /etc/krb5.keytab
                Require valid-user

        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel debug

        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

2.5.7 Test from Client Workstations

Try login from a workstation that is joined to the domain and logged in with a user; you should automatically be authenticated. You migvht have to indicate to the Internet Explorer that the site you are accessing is part of the Intranet. For example, add *http://*.pacificpolicy.org* to Internet *Options->Security->Intranet->Sites*.

Try with another workstion not joined to the domain; you should be prompt to enter credentials.

CHAPTER

THREE

DEVELOPER GUIDE

3.1 Introduction

This guide documents high level development standards, policies and procedures without going into lower level details of the source code and APIs.

3.1.1 Programming Language

The chosen programming language is Javascript both on the backend and the frontend. Javascript is relatively easy to get started with, it is increasingly popular to develop web applications and has a growing wealth of libraries to use to build systems faster. NodeJS is the Javascript backend platform and a decent starting point and reference for the Javascript language is provided by the Mozilla Developer Network.

3.1.2 Web Stack

No web application is built without a web framework (or library). The prototype was built using Bottle/Flask in Python but the production system will move to NodeJS and ExpressJS. Both are good choices but moving to the NodeJS has a number of advanges which were important to us including pervasive use of asynchronous programming on backend and frontend making it easier to scale with same amount of resources. A road to unify backend and frontend languages, libraries and devleopment tooling.

A micro framework was prefered to a full blown and much less flexible framework such as Django or Ruby on Rails. Applications can be fine crafted much better with small composable libraries. In addition to Express, a number of Express plugins are used when needed and Express middleware pluggins can easily be written when none appropriate are available.

3.1.3 Database

The CouchDB document database was chosen for its flexibility and simplicity. CouchDB is one of the several NoSQL database types built specifically for the web. If you are not familiar with CouchDB have a look at its website at https://couchdb.apache.org/ and documentation at http://docs.couchdb.org/en/latest/..

3.2 Setting Up the Development Environment

It is pretty easy to setup your own development environment to work on the Candy Basket tool. Here you will find the necessary steps to get you started. Essentially, the following subsections can be followed in order and everything should work.

The latest NodeJS will need to be installed on your operating system (OS); binaries are available for all popular OSes. Instructions are given in a platform-agnostic fashion to the extent that it is possible.

3.2.1 Development Tools

You will need you typical development tools: a command line, a text editor or IDE, a web browser with good development plugins such as the Google Chrome Javascript console or Firefox's web developer extension and firebug. It does not matter much which tool, choose the ones you're most confortable with.

3.2.2 NodeJS and NPM

NodeJS is the Javascript platform for writing Javascript software on the backend. The NodeJS package manager in use is npm and the canonical way to install npm package is locally to whatever software you are developing as opposed to globally on your operating system. Therefore, use:

```
[user]$ npm install express
```

and not:

```
[user]$ npm install -g express
```

And say you are adding a new dependency to your branch make sure you save it to the package.json definition with:

```
[user]$ npm install --save new-dep
```

Or if it is a dependency only used during development:

```
[user]$ npm install --save-dev new-dep
```

Some packages are typically better installed globally, however. Once you have NodeJS installed on your machine you should install bower and grunt-cli globally:

```
[user]$ npm install -g bower
[user]$ npm install -g grunt-cli
```

This is all that should be needed as a foundation development setup. However, if you plan to test the production deploy on your development machine—and you probably should before doing any pull request—you will also need a few more global packages.

[user]$ npm install -g forever [user]$ npm install -g http-server

3.2.3 CouchDB

CouchDB is used as the database for this tool. The easiest way to install CouchDB is to use the OS' package manager (Debian's apt-get, Mac OS X's brew, Red Hat's yum). CB makes use of three databases: *candy_basket_test*, *candy_basket_development*, *candy_basket* (for production). On the development machine only the candy_basket_development must be created in advanced (and *candy_basket* if you plan on testing deploys locally). The integration tests will create and destroy the *candy_basket_test* database automatically, in fact, tests will fail if this database is already present.

Sample data must be added to the development database. This can be done in a number of ways, either programmatically importing old candies or manually entering some sample data using the running development application. The views have to be created manually at the moment. You can easily just copy and paste the views definition from the integration tests (i.e. *backend/test/specs/controllers.js*) in the CouchDB admin web UI.

In the test database sample data is automatically generated as part of unit tests and the views are also programmatically created before they are used.

3.2.4 Grunt

Grunt is use to automate a number of time consuming tasks. It takes a while to learn but is well worth the efforts. Included here is a short list of things you will use grunt for.

Grunt on the backend

First, from the *backend* directory you can execute tests and serve the backend application.

To run the tests from the backend you will need two terminals since these contain also integration tests in addition to unit tests. In both terminals you should set the NODE_ENV to 'test' like this:

```
[user]$ export NODE_ENV=test
```

In one terminal serve the backend in test mode:

```
[user]$ grunt serve
```

And the other terminal you run the tests. Those tests will run against a test environment (with a test database as configured in *backend/config.js*):

```
[user]$ grunt test
```

When simply developing you should only need one terminal to serve the backend application. But you need to switch the environment to development wih the following:

```
[user]$ export NODE_ENV=development
```

And then you can serve the backend in development mode:

```
[user]$ grunt serve
```

The nice thing is that you can do all the above at the same time and it want interfere as test, development and production environments all use different ports.

Building the backend will clean any previous build, JSHint all the code, run the tests and prepare all files for production into *backend/dist* and can be done with the following:

```
[user]$ grunt
```

Grunt on the frontend

In the frontend, things are very similar, but simpler. There is no need to set the environment variable from the command line; this is done within the grunt processes. So, all you really need in the frontend currently is to run a development web server (this will be in development mode automatically):

```
[user]$ grunt serve
```

To run your tests you can:

```
[user]$ grunt test
```

The build process will first clean any previous build, make sure all the code JSHints, run all tests and then do an impressive number of optimisations to the application and its dependencies and package it in *frontend/dist*. The build process is done with:

```
[user]$ grunt
```

Grunt globally in app root

Finally, work has also commenced on automating some other tasks in the root of the candy-basket application. Currently, it can already do some really useful things which will be described here.

It can automatically create new releases following some modern conventions similar to the ones used by the AngularJS team which would typically be a repetitive number of boring tasks, but not with the following:

```
[user]$ grunt release:patch
[user]$ grunt release:minor
[user]$ grunt release:major
[user]$ grunt release:prerelease
```

Which one to execute will depend on the work on the recent branches pulled into master (see *On-going Development*). For example, let's say you pulled 3 branches that address bugs then you could cut a release with:

```
[user]$ grunt release:patch
```

But if you add new features you might want to cut a release with:

```
[user]$ grunt release:minor
```

And for major upgrades such as those containing backward incompatible changes:

```
[user]$ grunt release:major
```

For more information on this process you can refer to url{https://github.com/geddski/grunt-release} and url{http://semver.org/}.

It can automatically generate a CHANGELOG.md file. This is a little tricky as the *from* and *to* commit hashes must be setup manually in the *candy-basket/Grunfile.js*'s changelog property to generate to new part of the CHANGELOG.md and automatically append it to CHANGELOG.md. The good thing is that if a mistake is done then you can simply *git checkout CHANGELOG.md* and try again. You can use *git log* to identify the hash of the last CHANGELOG.md commit which will be your *from* and the most recent release cut which will be you *to* and then:

```
[user]$ grunt history
```

Verify that the CHANGELOG.md looks good, do any manual changes you see fit and commit this CHANGELOG.md to master directly.

It can build the docs if you make any amendments to these source files:

```
[user]$ grunt docs
```

It can Bbuild the backend, build the frontend, build the docs and move everything into *candy-basket/dist* ready to be deployed on remote server:

```
[user]$ grunt build
```

And even do a full deploy of the application which will do a complete *grunt build* as above and then start the backend and frontend using forever. For this to work there are a couple of things you need to have working. First, you will need authbind to enable a non-privileged OS user to start network services on ports below 1024 (i.e. 443), you must emph{not} have anything listening on 4443 and 443 and forever must be installed. But then it's just a matter of:

```
[user]$ grunt deploy
```

To see services you can:

```
[user]$ forever list
```

And before you try to re-deploy you must stop the services first:

```
[user]$ forever stopall
```

For the installation and configuration of authbind refer to url{http://www.debian-administration.org/article/386/Running_network_services_as_a_non-root_user.} or any number of easy tutorials on how to use authbind.

3.2.5 Dependencies

This application has a number of dependencies but they can all easily be installed from within the root of your own clone repository and from the *backend* and *frontend* directories. The production backend libraries and the development and test libraries are typically always npm packages with the dependencies clearly defined the *packages.json* files, one in the backend, one in the frontend and one in the root directory. In other words, everywhere you see a package.json file you must change to that directory and install dependencies like this:

```
[user]$ npm install
```

Frontend dependencies, those that will run in the client browser powering the web UI are installed using the Bower package management tool. From within the frontend directory you can simply do:

```
[user]$ bower install
```

Those commands are idempotent and it does not matter how often you execute them. Installing new dependencies for development can be done with the same tool.

Backend dependencies and frontend development and test libraries:

```
[user]$ npm install new-grunt-plugin new-backend-library
```

Though to save the dependency in the package.json you would do:

```
[user]$ npm install --save-dev new-grunt-plugin new-backend-library
```

Frontend dependencies:

```
[user]$ bower install new-angular-third-party-directive
```

and the same to persist the dependency if you end up keeping it:

```
[user]$ bower install --save new-angular-third-party-directive
```

Some of the packages may have additional lower level dependencies of their own in which case you would typically have to install some package on your OS such as xml headers from the development package. This should be made clear from failures to install dependencies and is typically quickly addressed by installing from the OS' software repository (apt, yum, brew, etc.)

3.3 Development Work-flow

The CB project constantly strives to improve its development operations in order to produce software of higher quality at a more efficient rate. This part of the developer guide will constantly evolve and should be kept close at hand when

developing on the CB project.

3.3.1 Software Configuration Management

All software is managed through Git (Source Control Management) and Github (Issue tracking, collaboration, etc.) in a publicly accessible repository. Its location is currently at https://github.com/ghachey/candy-basket/ but it will likely eventually change to the owning organization Nasara. Until then you can retrieve your own full local clone of the project with Git installed on your machine:

```
[user]$ git clone git@github.com:ghachey/candy-basket.git
```

However, never publish work to master (at least as rarely as possible). The following section describes the procedures to develop on CB.

3.3.2 On-going Development

New development work on a software project is either of maintenance (fixing bugs, addressing security issues) or construction nature (adding new features). Regardless of the type of work, all new work should be done in a branch, not on master. For example, let's say we're tackling issue #3 from the issue tracking system (Trac, Github Issues, etc.) you should create a branch like this [PRO-GIT]:

```
[user]$ git checkout -b issue3
```

Work on the issue, add relevant tests so it does not occur again, all the while only committing locally on your branch. Discuss with team members the fix if not sure about something. Get team members to review and refactor code if needed. After all this is done you can go ahead with publishing your new fix following our defined standard procedure.

It is desirable to keep the history of master's commits as clean as possible for more effective code review. The established way of achieving this is to squash all your local commits from your *issue3* branch into a single properly formatted commit before publishing changes and doing a pull request to master.

Squashing commits in git is straight forward [PRO-GIT]. However, the consolidated commit must follow the following conventions adapted from Google project AngularJS which will greatly enhanced the historical information on master and allow for automatic generation of the changelog. The format of the commit message must follow the following convention:

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Any line of the commit message must not be longer than 100 characters. This allows the message to be easier to read on github as well as in various git tools.

<type>

Should be either of the following:

- feat (when working on new feature)
- fix (when fixing a bug or addressing a security vulnerability)
- docs (when working on documentation)
- style (improving formatting, missing semi colons, indentation, etc.)
- refactor (when doing minor or major refactoring work)

- test (when adding missing tests)
- chore (maintain)

<scope>

Should specify the location of the commit as succinctly and completely as possible (e.g. $location, $rootScope, ngHref, ngClick, ngView)

<subject>

Subject line contains succinct description of the change. Remember it must not be longer than 100 characters and this *includes* both the <type>(<scope>) identified before. Here are some convensions:

- use imperative, present tense: "change" not "changed" nor "changes"
- don't capitalize first letter
- no period full stop (.) at the end

<body>

[Optional] Slightly more elaborated description possibly spanning over several lines never more than 100 characters each.

- just as in <subject> use imperative, present tense
- includes motivation for the change and contrasts with previous behavior

<footer>:

[Optional] should include either breaking changes and/or references of what issues were resolved if any. All breaking changes have to be mentioned in footer with the description of the change, justification and migration notes.

The following includes several examples of properly formatted squashed commit messages.

A new feature commit:

```
feat($browser): onUrlChange event (popstate/hashchange/polling)

Added new event to $browser:
* forward popstate event if available
* forward hashchange event if popstate not available
* do polling when neither popstate nor hashchange available

Breaks $browser.onHashChange, which was removed (use onUrlChange instead)
```

A fix for browser compatibility commit:

```
fix($compile): couple of unit tests for IE9

Older IEs serialize html uppercased, but IE9 does not...
Would be better to expect case insensitive, unfortunately jasmine does
not allow to user regexps for throw expectations.

Closes #392
Breaks foo.bar api, foo.baz should be used instead
```

A new feature request from issue #351 commit:

```
feat(directive): ng:disabled, ng:checked, ng:multiple, ng:readonly, ng:selected

New directives for proper binding these attributes in older browsers (IE).
Added coresponding description, live examples and e2e tests.
```

```
Closes #351, #456
```

Some cleanup commit:

```
style($location): add couple of missing semi colons
```

Some documentation work commit:

```
docs(guide): updated fixed docs from Google Docs

Couple of typos fixed:
* indentation
* batchLogbatchLog -> batchLog
* start periodic checking
* missing brace
```

A new feature with major breaking changes:

```
feat($compile): simplify isolate scope bindings

Changed the isolate scope binding options to:
* @attr - attribute binding (including interpolation)
* =model - by-directional model binding
* &expr - expression execution binding

This change simplifies the terminology as well as
number of choices available to the developer. It
also supports local name aliasing from the parent.

BREAKING CHANGE: isolate scope bindings definition has changed and
the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

scope: {
  myAttr: 'attribute',
  myBind: 'bind',
  myExpression: 'expression',
  myEval: 'evaluate',
  myAccessor: 'accessor'
}

After:

scope: {
  myAttr: '@',
  myBind: '@',
  myExpression: '&',
  // myEval - usually not useful, but in cases where the
  // expression is assignable, you can use '='
  myAccessor: '=' // in directive's template change myAccessor() to myAccessor
}

The removed `inject` wasn't generaly useful for directives so there should be no code using it.
```

For example, you've been working on your branch and made three commit with vague non-useful messages such as "Work in progress", "Small fix", etc. You want to wrap up the work with a nice single squashed commit following the

above format. You can use Git's rebase tool:

```
[user]$ git rebase -i HEAD~3
```

This will pull open an editor with something like the following:

```
pick f7f3f6d Work on docs
pick 310154e Work in progress
pick a5f4a0d Small fix

# Rebase 710f0f8..a5f4a0d onto 710f0f8
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

To squash the three commits into one you would edit the script to look like this:

```
pick f7f3f6d Work on docs
squash 310154e Work in progress
squash a5f4a0d Small fix

# Rebase 710f0f8..a5f4a0d onto 710f0f8
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

When saving this you will return to a text editor where you can merge the commit messages seeying something like this

```
# This is a combination of 3 commits.
# The first commit's message is:
Work on docs

# This is the 2nd commit message:
```

```
Work in progress

# This is the 3rd commit message:

Small fix
```

Those commits are practically useless in the grand scheme of things. You want to replace it with a single properly formatted message following above conventions. In this case you would remove the above from the text editor and replace it with something like the following:

```
docs(developer-guide.rst): update docs with new code base refactory

What's changed in details:
* Change backend section to reflect migration to NodeJS
* Refactor various part of guide with new content
* Introduce new conventions and standards
```

Save this nicely formatted commit and then you're ready to publish your work and do a pull request:

```
[user]$ git push
```

Although if you were working entirely on a detached local branch like I do you would need to push like this instead:

```
[user]$ git push --set-upstream origin replace-this-with-branch-name
```

Do the pull request from github and use the last commit as the message.

3.3.3 Application Deployment

Automation for optimized deployment is currently in the works and nearly working. The *backend* can be grunt deployed. The *frontend* can be grunt deployed in a highly optimized fashion following Google's best practice for making the web faster. The optimized frontend deployment works *almost*. There remains a couple of tricky bits to address but it is mostly working except a couple of noticeable things: keystrokes with the Timeline are not working, angular-bootstrap templates are not accessible and so the modal and slider are not working as expected.

The only requirements for Candy Basket to work in production are the NodeJS platform, grunt-cli Grunt's command line interface, forever to run node applications reliably and http-server small light weight and fast HTTP server:

```
[user]$ npm install -g grunt-cli
[user]$ npm install -g forever
[user]$ npm install -g http-server
```

The process to build the backend can be done individually (not yet executing tests first):

```
[user]$ cd candy-basket/backend/
[user]$ grunt
```

The process to build the frontend can be done individually also:

```
[user]$ cd candy-basket/frontend/
[user]$ grunt
```

And the whole Candy Basket application can be deployed including executing test, building docs, building backend, building frontend and copying all files to *candy-basket/dist*:

```
[user]$ cd candy-basket/
[user]$ sudo su
[root]# grunt deploy
```

Currently, the application must be started as user root. The next step would be either to use iptables to redirect 80 to 8080 and start user as non-privileged one or use authbind. Optionally, another tasks could be added to move it to the desired location on the server.

Services are started on port 4443 (backend) and 443 (frontend) so those port must not be taken. The application only functions on https with currently no redirect from http.

3.4 High Level Architecture

Briefly, this application is composed of two main parts: a computer consumable service on the backend (i.e. runs on the server) and a human consumable service on the frontend (i.e. runs in the browser). The backend is a NodeJS powered RESTful service and the frontend is an HTML, CSS and Javascript Web User Interface (UI) capable of talking to the backend.

- README.md – A brief introduction and pointers
- LICENSE.md – GNU General Public License version 3
- CHANGELOG.md – Automatically generated change logs
- backend – The NodeJS RESTful service
- frontend – The AngularJS Web application
- docs – The documentation for this project
- package.json – Root meta data JSON file
- Gruntfile.js – Grunt task automation file common to backend and frontend

3.4.1 Backend – The RESTful Service

The backend is written entirely in the Javascript programming language implementing a simple RESTful service. The backend is a RESTful service following a Resource Oriented Architecture (ROA) as defined in [REST-SERV]. The following tables describe its service. Note that no API version number is included in the URI; it will be included in the host as http://candy-restapi-v1.pacificpolicy.org.vu/.

User Account Service

Each organisation can have a number of users using the tool. However, user management is usually done using an external service such as Active Directory or another LDAP service like OpenLDAP. Candies do not yet have ownership and are globally accessible by the organisation once authenticated.

The URI design goes like this. A “basket” refers to the whole organisation. In other words, organisations have their private basket of candies. An organisation (and therefore a basket) can have many users; the organization and its users can be represented as /basket/users/, but this will not be used at first. All candies are associated to a user and are (at least at first) accessible to any authenticated staff.

The services offer no CRUD operations on users at the moment as this is considered to be done using an external service (Active Directory, OpenLDAP).

Source (Candies) Service

This is the main service of candy basket: users can add “source(s)” and tag them. A source can have a URL, file(s), title description and tags. In the technical world of Candy Basket (such as in the source code) sources are typically referred to as candies; they are exactly the same thing. In the UI the term source is used.

Operation	HTTP Method and URI
Create a source	POST /basket/candies
View a source	GET /basket/candies/{uuid}
Modify a source	PUT /basket/candies/{uuid}
Delete a source	DELETE /basket/candies/{uuid}

Utilities Service

Only a couple of utility aggregates are needed at the moment.

Operation	HTTP Method and URI
Fetch all sources	GET /basket/candies
Fetch all tags	GET /basket/candies/tags
Fetch all tags by candies	GET /basket/candies/tags-by-candies

When developing it is often useful to use the RESTful API directly. Here are some example usage.

Fetching all candies:

```
[user]$ curl --user candy:P@55word -X GET http://localhost:3003/basket/candies
```

Fetching a candy:

```
[user]$ curl --user candy:P@55word -X GET http://localhost:3003/basket/candies/03c0b670e5c56bfb461a7
```

Creating a candy:

```
[user]$ curl --user candy:P@55word
             -X POST \
             -H "Accept: application/json"  \
             -H "Content-Type: application/json" \
             -d @candy.json \
             http://localhost:3003/basket/candies
```

Where candy.json would be the JSON candy in a file named candy.json accessible within the directory from which curl command is being executed. Routes only accept JSON at the moment. It could look something like this:

```
{
  "source": "http://www.ghachey.info",
  "title": "Ghislain Hachey Website",
  "description": "A bit updated",
  "tags": ["gh","ict","website"]
}
```

Or an invalid candy (dangeous scripts):

```
{
  "source": "http://www.ghachey.info",
  "title": "Ghislain Website",
  "description": "<script>alert(\"Hacked onced, shame on you.\");</script>",
  "tags": [
    "Website",
    "Ghislain Hachey"
  ]
}
```

If you want to test uploading the easiest is to use the frontend directly. Otherwise, you could build a request yourself with curl by setting the *Content-Type* to *multipart/form-data* and the additional JSON data which would be something like this:

```
"files":[{"name":"0bf6198aac462ddbb12add63fff0d8c2.pdf",
          "originalName":"Artificial Intelligence Search Algorithms.pdf"},
         {"name":"7fde008c066d3ed6226d5a88b2f1e7ef.png",
          "originalName":"linkedin.png"}]
```

Where the name is a UUID generated by the frontend upload code and the original name is also kept. The file would be sent to the ownCloud with the unique name but could be listed and retrived using the original name.

Updating a candy:

```
[user]$ curl --user candy:P@55word
             -X PUT \
             -H "Accept: application/json"  \
             -H "Content-Type: application/json" \
             -d @candy-update.json \
             http://localhost:3003/basket/candies/id-of-candy-in-couchdb
```

Where id-of-candy-in-couchdb is the id automatically created on POST and returned in the Location header for latter retrieval. It can be retrieved in a number of ways. Looking at data in the DB is fairly easy and quick. The newly updated candy could look like this:

```
{
  "_id": "id-of-candy-in-couchdb"
  "source": "http://www.ghachey.info",
  "title": "Ghislain Hachey Website",
  "description": "A bit updated--oups, I meant a bit outdated",
  "tags": ["gh","ict","website"]
}
```

This would completely replace the previous document. For example, if you had a *files* data in the JSON document and none in the update then that data would no longer be present. A complete update on a document containing also files could be achieved with a minimum couple of async curl requests. First the file upload(s):

```
[user]$ curl --user candy:P@55word
             -X POST \
             -H "Content-Type: multipart/form-data; boundary=----------------------------119366476258
             --data-binary @test.txt \
             http://localhost:3003/files
```

And then the actual candy:

```
[user]$ curl --user candy:P@55word
             -X PUT \
             -H "Accept: application/json"  \
             -H "Content-Type: application/json" \
             -d @candy-update.json \
             http://localhost:3003/basket/candies/id-of-candy-in-couchdb
```

3.4.2 Frontend – Web UI Application

The Frontend is developed using the AngularJS web framework with community Angular modules and our own code. The frontend code based is organised following Angular community best practices.

- *frontend/app/scripts/app.js*: this is where the application is bootstrapped. It contains some configuration and some routes definitions.
- *frontend/app/scripts/services/*: this directory contains application services.

- *frontend/app/scripts/controllers/*: this is where the business logic resides; no DOM manipulation should happen here.
- *frontend/app/scripts/filters/*: this where filters are stored often used has a final filtering step before presenting the data (e.g. money and date conversions formatters). It can also include data filtering code.
- *frontend/app/scripts/directives/*: this is where you can manipulate the DOM as you wish. Think of directives as a means to extend HTML and browser capabilities for web *applications*.
- *frontend/app/index.html*: is the base HTML file for the whole application
- *frontend/app/views/*: contains all the other HTML partials that make up the rest of the application.
- *frontend/app/styles/*: contains custom styles
- *frontend/app/images/*: contains images
- *frontend/test/specs/*: where unit tests resides. Directories in there mirrors the content of the *frontend/app/scripts/*

3.5 Low Level Documentation and API

The lower level documentation about software design, application programming interfaces, small gotchas and all other nitty-gritty details about the source code is written directly inside the source code. It can be extracted and exported to hard copy formats such as HTML or PDF and eventually may be integrated with this documentation also. But currently the place to access it is directly inside the source code for two main reasons: the JSDoc generators by default generate incomplete mostly useless and ugly HTML output and since this is not intended to be used by others as a public API it's not worth the effort of extracting these lower level docs.

3.6 Documentation

Higher level documentation is prepared using an excellent tool developed in the Python world called Sphinx http://sphinx-doc.org which uses the reStructuredText markup language http://sphinx-doc.org/rest.html. Sphinx outputs to HTML and PDF but could also output to other formats.

In the docs folder there is a `source` directory which contains the source files with the markup content; this is where the documentation is written. The `build` directory is where the documentation is produced either in PDF, HTML or other supported format.

If you plan on producing documentation you will need to install Sphinx. Sphinx is written in Python can the easiest way to install it is to install Python and pip and then execute the following to install globally:

```
[user]$ sudo pip install sphinx
```

Outputs are generated using a simple make command from within the `docs` directory:

```
[user]$ make latexpdf
[user]$ make html
```

Or simply type `make` to get a list of other options. If you wish you generate PDF you will need to install the Tex type setting system along with LaTeX, but this is optional. How to do this will largely depend on your OS. There is usually a very large all in one package available for popular OSes either packaged as binary or directly available through the OS' package manager.

However, if you have Sphinx installed there is no longer any need to manually build the docs. You can simply use Grunt from the candy-basket root directory like this:

```
[user]$ grunt docs
```

All source code including the application programming interface is documented in a modern Javascript fashion using a jsdoc style with AngularJS additional conventions on the frontend. This has a number of advantages including keeping the documentation directly with the code and more in sync, preparation of AngularJS style documentation with the ability to add example usage, online discussions and a number of others things not readibly available when simply using Sphinx. When writting source code simply document it following AngularJS and jsdoc styles and the production of the online documentation is currently not being done as it provides little added value. If you're interested in lower level development details the place to look at now is the source.

3.7 Security Considerations

Since Candy Basket will be used as a tool by organizations with varying degrees of security requirements it must be designed and evolve with a number of security considerations in mind and the aim of constantly improving its security status quo.

If you are interested in helping contribute code to Candy Basket we provide some mininum security related recommendations, guidelines and procedures to follow.

3.7.1 Authentication

The backend currently supports only HTTP Basic Authentication on every single endpoint. It is critical to properly setup SSL/TLS to encrypt all communication between the client (frontend) and the server (backend). It makes use of a single user called *candy* to authenticate the frontend with the backend which is configurable in *backend/config.js*. Therefore, users authenticated to the frontend through some LDAP single sign-on mechanism will then automatically have access to data from backend. In other words, no access to frontend, no access to backend either.

3.7.2 SSL/TLS Encryption

This application is moving towards a strict and mandatory use of encryption throughout all its the various components . Self-signed keys and certificates are used for development and test and the equivalent of curl's –insecure flag is set when executing requests in those modes. This insecure flag is off by default in production.

ownCloud

Candy Basket uses ownCloud as file storage. Connections to ownCloud must be encrypted. Developers can use their own local ownCloud server for development and test though will have to include their own certificate in the *backend/certificates* directory and change the config.js. The certificate to make use of the pacificpolicy.org ownCloud server is also included. Casre must be taken with the configuration of the ownCloud server to enforce secure connections at all times.

Nasara backend

The backend now also supports encryption. In fact, it only listens on https, period (port 3003 for test and development and 443 for production). A set of development keys was generated which can be use for just development and test without change in the *backend/config.js*. Both the private key and public certificate are committed to the repo for development and test convenience. Needless to say they should not be used in production. A new set should be used, either self signed or both from a CA depending on the context and target users.

Nasara frontend

To access the backend with the self-signed certificate in development from AngularJS the browser needs to confirm the insecure connection (like curl's –insecure or NodeJS's process.env.NODE_TLS_REJECT_UNAUTHORIZED = '0'). Only once you will have to point the browser directly to the backend by putting the address *https://localhost:3003/* in the URL address bar.

3.7.3 Latest Top 10 Security Risks

An initial security assessment determined that this application was designed with all the security basics in mind although tighthening security should always remains an objectif as he project evolves. Candy Basket was measured against OWASP's most up-to-date Top 10 Security Risks. It is important to re-assess Candy Basket towards this top 10 list every year (or whenever it is revised). Any change should be carefully examine to make sure Candy Basket still covers all of them with details reflected in this documentation.

Injection

General information can be found at A1 – Injection. Candy Basket makes all necessary efforts to validate data both in the frontend and the backend to prevent any injection through its communication with its data store.

Automated scanners can do a good job but should be combined with manual code review for completeness.

Broken Authentication and Session Management

General information can be found at A2 – Broken Authentication and Session Management. All authentication and session management of the application (i.e. frontend) is left at the Active Directory level through Kerberos and Apache. This setup should be verified at each upgrade to make sure it is updated and working as expected. Also make sure that only the frontend can communicate with the backend at the web server configuration level.

Strict adherence to recommendations in A2 – Broken Authentication and Session Management is a good start. Anybody working on Candy Basket should have in their possession the VMware network lab: a Windows 2008 Server (AD, DNS...), A Debian server (Apache, Kerberos to host Candy Basket), a Windows 7 workstation, a Windows 8 workstation, any other network node useful in testing authentication and sessions.

XSS

General information can be found at A3 – Cross-Site Scripting (XSS). Candy Basket covers this one much like protecting agains injections: frontend and backend data validation, automatic sanitization of rich content, and appropriate escaping of untrusted data.

A mix of automated tools and manual code review can be employed for Integrated Penetration Testing.

Security Misconfiguration

General information can be found at A5 – Security Misconfiguration. There is a lot to keep track of here: OS configuration, Web server and modules configuration, Candy Basket application configuration, third party libraries default security related configuration. A simple change to the Candy Basket code base making use of the configuration variables could open up an easy security hole. For example, in DEBUG mode the application accepts a non existant Origin header to make testing of backend with curl straight forward. If this were left unchanged in production an attacker could execute cross-domain requests simply by removing the Origin completely in its reponses. It's all too easy to write a single line of code that can result in this; I wrote one myself and left it there for about 15 minutes until I realised the consequence.

Regular overview of all the configuration from low to high level should be integrated into the develop, test and deploy cycle. A grep on DEBUG on the whole Candy Basket code base might help in identifying unsafe code. The use of scanners on the OS and Web server (e.g. Nessus) can be useful. The important thing is the have solid and fast development operations in place with the ability to deploy in new secure environments that can be quality tested in quick cycles.

Sensitive Data Exposure

General information can be found at A6 – Sensitive Data Exposure. Most sensitive user data is handled at the Active Directory level. Securing this aspect means keeping the Windows (or Samba4) server updated and properly configured. The information in the database (aka. the candies) can often also be considered sensitive information and is secured through a combination of all the security mechanisms in place. Otherwise, data can be accessed in a number of ways:

```
* CouchDB only listens on the local interface but this could further
```

be tightened.

- The backend has access to the data so should be secure. At the moment, its access is restricted to the frontend through Apache directives.
- The frontend can access data through the backend but the user must be authenticated with an Active Diretory to access the frontend.

Missing Function Level Access Control

General information can be found at A7 – Missing Function Level Access Control. Candy Basket does not do much in terms of Access Control. Either the user has access to the application or not. This significantly reduces the complexity and therefore the attack vectors.

Make sure all other risks are properly addressed and this one should be covered.

Cross-Site Request Forgery (CSRF aka. XSRF)

General information can be found at A8 – Cross-Site Request Forgery (CSRF aka. XSRF). A number of mechanisms exist with varying degrees of strength to protect against CSRF. The present status quo with Candy Basket can best be explained by summarising an email dialogue between Dan McGarry and Ghislain Hachey:

```
> It is possible to protect against CSRF by checking the origin header,
> but since this could be spoofed it would only be a first line of
> defence. From my understanding, the highest form of security against CSRF
> is making use of secret tokens first generated by the server and sent
> on each request from the application (which is the number one
> recommendation of OWASP. However,
> according to Angular developers the above scenario is typical of
> non-CORS applications where cookie-based authentication is used
> (i.e. cases most vulnerable to CSRF attacks).  We have a different use
> case: one, we are CORS enabled (`http://www.mnemonic.com` talking to
> `http://rest.mnemonic.com`); two, we do not make use of cookie-based
> authentication but make use of authentication at the Apache level
> using kerberos and AD. While CORS alone is not a protection against
> CRSF, the first line of defence herein (i.e. checking origin) would
> make it quite hard for an attacker who would have to *both* spoof the
> origin *and* trick the user into clicking a malicious page executing
> it when logged into AD at work or on a VPN.
>
> I looked into alternatives to further secure the CSRF weakness and
```

```
> found out about the use of XSRF-TOKEN. In short, the server generates
> a secret token which is passed to Angular on the first request as a
> cookie which is then returned by Angular on each request in a header
> (i.e. X-XSRF-TOKEN). Server then verifies header matches cookie on
> each request and if so considers the user legitimate (since only
> Javascript running in the user browser could know the original
> token). However, according to angular this is typically a non-CORS use
> case and Angular does not bother returning the token I created on the
> server in the request header because we do cross-domain requests
> making our use case a bit more painful when it comes to using this type
> of protection. See
> <http://docs.angularjs.org/api/ng/service/$http> and
> <https://github.com/angular/angular.js/issues/5122>.
>
> My take on it is that we have a relatively non-typical use case:
> de-coupled REST server with single page web application authenticating
> with Apache/Kerberos/AD. I see two possible paths we could take:
>
> 1) To secure this to my taste I would make it "impossible" to talk to
>    the REST server from anything but the frontend application
>    (essentially what CORS aims except it does not offer protection
>    against spoofing). At the moment, this is enforced at the web
>    server level but does not protect against sophisticated
>    spoofing. The angular application would make use of a user which
>    would authenticate to the backend through robust use of token-based
>    authentication. Token-based authentication has a number of
>    advantages over the currently prevalent use of cookie-based
>    authentication (good reads here
>    <http://www.jamesward.com/2013/05/13/securing-single-page-apps-and-rest-services>,
>    <http://blog.auth0.com/2014/01/07/angularjs-authentication-with-cookies-vs-token/>). Another
>    advantage of doing this would be to take Candy Basket one step
>    closer to a "public offering" and not just an "enterprise
>    offering".
>
> 2) Another more hackish method is to force Angular to send the
>    XSRF-TOKEN by intercepting and adding headers on each XHR. However,
>    the angular folks specifically deactivated this as they essentially
>    say it should not be done like this and stated it was causing
>    problems with the CORS pre-flights (CORS makes use of pre-flight
>    OPTIONS requests to check whether non-safe requests such as POST,
>    PUT and DELETE are allowed by the origin). This approach would
>    secure the backend and frontend integration against spoofing but is
>    not my preferred options.
```

In conclusion, Candy Basket's current CSRF protection of checking the Origin on the server side and only allowing the frontend access to the backend seems adequate. Even if the attacker manages to spoof the origin *and* trick the user into clicking a malicious link disguised as cute kittens, the backend would refuse the request—even when the user is authenticated—based on restrictions at the Apache level (i.e. only the frontend application can talk to the backend).

Using Components with Known Vulnerabilities

General information can be found at A9 – Using Components with Known Vulnerabilities.

Candy Baskets is based on a number of libraries each of which could potentially have security vulnerabilities. While it is often impractical to constantly assess all third party libraries it is easy to subscribe to some kind of communication channels and observe the evolution of all the components used in your software. Communication channels could be either mailing lists, social networks or the github issues tracker.

If there are discovered security vulnerabilities—those that are of actual real life concern—they will often be announced through the project's communication channels. You should at the very least follow announcements of the following projects:

- Angular (and all its modules which are usually upgraded in sync)
- D3 and D3 Cloud
- JQuery
- MomentJS
- UI Bootstrap
- UndercoreJS
- CouchDB
- The hosting Platform (OS, Web server, Modules...)

Whenever any of the above project announces a security vulnerability there should be an upgrade in process. Typically, very little change will be required, sometimes a simple matter of executing a *bower upgrade* and a re-deploy. At times, you may be faced with breaking changes which will require you to also upgrade the Candy Basket code.

All the above third libraries take security seriously. If you plan on integrating a new library to add features to Candy Basket a good deal of consideration must be given to the security aspect of the new library. Adopting a project with little regard to security should be *always* avoided.

Unvalidated Redirects and Forwards

General information can be found at A10 – Unvalidated Redirects and Forwards. Candy Basket makes almost no use of redirects and forwards and no use of dangerous redirects and forwards (using destination parameters based on users or other dynamic variables).

Avoid using all but the most simple forwards and redirects. For example, a redirect to the list view on save or cancel operation is fine but avoid anything else for the moment. This will depend on the future direction of Candy Basket.

Miscellaneous and Application Specific

There are a number of security considerations that were not part of the top 10 list but that do apply to our specific use case. Those should be documented here:

- We make use of JSON as the data interchange format. JSON contains a subtle vulnarability when returning data as an array. Angular offers a way to address this issue by prefixing all JSON requests with the string ")]}',n" as described here. We simple always return an JSON object instead. For example, if we want to return an array of Candies we would send something like {"data" : ["candy1", "candy2"...]} and transform the request in Angular to process the array.

3.7.4 Integrated Penetration Testing

The above guidelines and procedures should offer an excellent starting point to ensure a secure web application. Of course, securing a web application should not stop here. We would like to see a more integrated penetration testing process. There are a number of tools that can be used to help support this process. Most of those tools have a relatelively steep learning curve but they are worth the time investment.

After some time evaluating several free software tools that were either recommended by OWASP or considered promising projects we have come up with a short list of tools to watch:

- OWASP Zed Attack Proxy Project (ZAP)

- OWASP CSRFTester
- OWASP WebScarab
- Vega a fork of Google Researchers' Skipfish backed up by commercial support. A younger but promising project which seem easier to use at first glance.

One or more of those tools should eventually be integrated into the development process. At first only making use of simple features such as automated scans and slowly integrating more complicated robust testing processes one by one. As these new processes come to live they should be clearly documented here with instructions on how to use the tools.

Apress.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

BIBLIOGRAPHY

[REST-SERV] Leonard Richardson and Sam Ruby, RESTful Web Services, O'Reilly, May 2007.

[FHS] Rusty Russell, Daniel Quinlan and Christopher Yeoh, Filesystem Hierarchy Standard 2004, freestandards.org

[PRO-GIT] Scott Chacon, Pro Git,Available at http://www.git-scm.com/book,