eTarget Technical Documentation

Rob Dunne
rob.dunne@manchester.ac.uk

Anja Le Blanc
anja.leblanc@manchester.ac.uk

16th July 2021

Contents

1 Application structure

The web application is an AngularJS UI interacting with a Java REST API. The data processing for parsing CSV files into the database is done using a Python (version 3) script, run on a cronjob. A data flow diagram is included in Appendix A along with a file tree of the web application in Appendix C.

1.1 UI

The User Interface (UI) is built using AngularJS v1.6.3. Additional JavaScript libraries have been used including jQuery, and shims for IE support. These are contained in the /js directory.

The UI is built as a single page application (SPA), so all HTML is contained either within the index.jsp file, or in the /includes directory. Included files are fetched with the ng-include attribute and are checked for user permission by the API first.

The URLs for dev, test, and prod are:

- Dev: uomdevetarget.azurewebsites.net
  (old: decmttarget.azurewebsites.net)
- Test: decmtetarget.digitalecmt.com
- Prod: christieetarget.digitalecmt.com

1.2 API

The REST API consists of JSP files (data.jsp, component.jsp) which act as routers to the relevant Java classes and methods. Each API endpoint has its own class, and inherits generic methods for user permissions and database interactions from the parent API class. No Java frameworks are used and the files are structured based on a standard Tomcat application set up / OOP.

In addition to the JSP base API there is now also a native Java Rest API. This development was required as JSP can handle only GET and POST requests but not DELETE.

1.3 Data API

All API calls for data are routed through api/data.jsp, with parameters passed to determine what to fetch. This endpoint should only return JSON data. A user permission check is done when fetching data.

1.4 Component API

All API calls for a web component (a small piece of HTML) are done through api/component.jsp, with parameters passed to determine what to fetch. This endpoint should only return HTML. A user permission check is done when fetching HTML.

1.5 Java API

Endpoint: `/rest/report/{specimen_id}/{run}`
HTTP method: DELETE

Deletes the specifid report, if the user has permission otherwise returns error code 403.

Endpoint: `/rest/pdf/{name.pdf}`
HTTP method: GET

Accesses the specified PDF, if the user has permission otherwise returns error code 403.

1.6 Data processing

The data processing VM handles all incoming files and data from the clinical partners, and copies PDFs to a secure storage area, and CSV data to the database. This is a separate Linux VM from where the web application is hosted.

A Python script (targetdata.py) is called once a minute via cron that fetches files and data and redistributes them. The Python script can be found in the ./targetdata directory in the home folder of the VM.

Cron job rg_Target_Dev, TargetData VM: /var/spool/cron/crontabs (crontab -e as the user)
Shell script rg_Target_Dev, TargetData VM: /home/commander/target_data/run.sh

1.7 Database

Data is often split across multiple tables in the database (database schema in Appendix B) - the initial schema was defined based on a previous project - so check for foreign keys when tracking data. Common data requests, such as a list of patient data, are aggregated in views for more simplified SQL queries.

2 Infrastructure

This project runs on Azure cloud services provided by CRUK. Please contact Paul Fitzpatrick for access[1].

2.1 App Service

The App Service is a Tomcat server that hosts the web application.

2.2 Data processing VM

The VM is a Linux server that processes files that are added to the file storage. PDF files are copied to configured secure storage area in a configured directory (Table: 1). CSV files are processed for patient data which is added to the database. No data is stored on the VM.

2.3 File storage

The file storage is accessed by the data uploaders. Files uploaded are copied by the VM and then deleted. Access is by Azure Storage Explorer[2] using the access key found in the Azure control panel.

A second storage area is used as the premanent location for uploaded PDF files.

[1] Paul Fitzpatrick, `mailto:Paul.Fitzpatrick@digitalecmt.org`
[2] Azure Storage Explorer, `https://azure.microsoft.com/en-gb/features/storage-explorer/`

2.4 Database

Standard SQL server and SQL database. The patient ID reference is encrypted using Triple DES. The encryption key is stored in the `eTargetVault` (Key Vault) in the `DECMT_DEV_eTarget` Resource group.

You will need to add your IP address to the firewall whitelist in Azure to access the database.

3 First set up

3.1 Virtual Machine

This section assumes a Azure Administrator has created a barebone Linux VM. The only task for the VM is to run a python script in regular intervals (every minute). The set up consists of installing all required Python libraries, setting up the configuration file, and creating a cron job for the regular runs.

3.1.1 Setup SSH access to VM

SSH access to the VM is required firstly to set it up, but also to keep the VM updated. Steps:

1. Log into Azure
2. Find the VM in 'Resource Groups' or 'Virtual Machines'
3. In the section for 'Support and Troubleshooting' select 'Reset Password'
4. Choose 'Reset SSH public key'
5. Choose a user name and copy your public key into 'SSH public key' field.

For ease of use I have set up an alias in .bash_aliases with the IP address of the VM found on the Azure portal.

To gain root access on the VM do `sudo bash`.

It is advisable to run an update of the operating system before continuing to install new software.

The service VMs are set up, so that you need to open the firewall for certain time before you can ssh into them. In the Security Centre of the Azure Portal you find the 'Advanced Cloud Defense' with an item 'Just in time VM access'. Find your VM in there, tick the checkbox and click the button 'Request access'; select port 22 and click 'Open ports'. A short time late you'll gain access.

3.1.2 Create a service account for running python

Idealy you want an service account (`useradd -r commander`) for the user running the python script in a cron job. But this was not done on any of our environments and for consistency reasons we go for a normal user account which can't be used for login.

```
useradd commander
usermod -L commander
```

This means the user still got a home directory `/home/comander` and an associated shell (`/bin/bash`).

3.1.3 Install Python libraries

Install python libraries as the user who should run them, i.e. commander. That means only this user can execute those libraries.

```
pip3 install 'azure-storage-blob==1.1.0'
pip3 install 'azure-storage-file==1.1.0'
pip3 install pymssql
#pip install crypto
pip3 install pycryptodome
pip3 install 'openpyxl==2.6.4'
pip3 install pycel
pip3 install ftpretty
pip3 install zlib
pip3 install jsonschema

yum install zlib-devel
```

As an admin install

```
apt-get install python3-crypto
```

3.1.4 Configure python script

The python script requires a `.config` file for its configuration. This file is not part of the git repository as it contains sensitive information. The basic structure is: `key; value`.

For the list of keys see Table 1.

3.1.5 Setup cron job for python script

The python script should run as the user created in section 3.1.2.

```
sudo bash
su commander
crontab -e
```

Add the line (adapt to the correct location):

```
*/1 * * * * bash /home/commander/target-data/run.sh /dev/null 2>&1
```

and save. This runs the run.sh script every minute. The script itself makes sure it is not running more than once, i.e. if the ingestion takes longer than a minute.

3.2 Database

There is currently no SQL script for setting up an empty eTarget database. Therefore you currently have to copy an existing eTarget database (from the Dev or Test environment) to the new Database Service and clean out the existing data.

3.3 Storage

Create a file share `data` on Azure portal or via a Storage Explorer.

Create a blob share `log` on Azure portal or via a Storage Explorer.

Table 1: Table of required keys for the configuration

Key	Description	Where to find
remotehostname	DNS name of database server	Azure portal; SQL database; Overview; Server name
remoteusername	User name to access the DB	Azure portal; DECMT_DEV_eTarget resourcegroup; eTargetVault; Secrets; find the right db
remotepassword	password to remoteusername	Azure portal; DECMT_DEV_eTarget resourcegroup; eTargetVault; Secrets; find the right db
remotedbname	Name of the database	Azure portal; SQL database; SQL databases
fileuser	Username for storage where users upload data files to (i.e. the data directory)	Azure portal; Storage account; Settings; Access keys; storage account name
filekey	Access key for that storage account	Azure portal; Storage account; Settings; Access keys; key1 or key2 key
patientkey	Is used to encrypt and decrypt the Christie patient number	Azure portal; DECMT_DEV_eTarget resource group; eTargetVault; Secrets; Encryption Key
storageurl	Connection string to the storage account	Azure portal; Storage Account; Settings; Access keys; use connection string
containername	Name directory in blob usually 'reports'	will be created if not existent
logblob	Name of the blob container for log files	create with the file explorer or in the Azure portal

3.4 Application

Please read the sections 5, 6.

For a new environment add a server section as described in 5.

In `pom.xml` create a new profile section. See table 2 for explainations of the `<properties>`.

3.4.1 Configurations

Configurations are split into three config files:

1. application.properties – contains application relevant settings
   (a) `edit.timeout` setting of the edit timeout in minutes; after that number of minutes an edit-locked patient will be released.
   (b) `application.version` sets ver version string for the browser tab name; value taken from `pom.xml`.
   (c) `application.title` Name of the application as shown on the browser tab

Table 2: Table of required properties in pom.xml <profile> section

property	**Description**	**Where to find**
serverid	given id in .m2/settings.xml	see that file
postfix	name of the war file is `etarget`+postfix`.war`	set here
azureFtpUrl	in case the deploy to the azure environment happens via sftp (wagon plug-in), this is the URL used	Azure portal; App service; Overview; FTP hostname + /sites/wwwroot
webapp.include.path	where to find the include directory on the web server; note: this changes between Windows and Linux hosts	Azure portal; App service; Advanced Tools/SSH; than look for location
databaseURL	JDBC url for connecting with the DB	`jdbc:jtds:sqlserver://` + Server name + `/` + db name; Server name and db name can be found in Azure portal; SQL database; Overview
dbuser	User name which the web-app should use	Azure portal; DECMT_DEV_eTarget resourcegroup; eTargetVault; Secrets; find the right db
dbpassword	Password for dbuser	Azure portal; DECMT_DEV_eTarget resourcegroup; eTargetVault; Secrets; find the right db
storageURL	Location of PDF files	Azure portal; Storage account; Access key; Connection string
RESOURCE GROUP_NAME	Info about your Azure subscription; used for azure maven deploy	az webapp list
WEBAPP_NAME	Name for this webapp in Azure; used for azure maven deploy	az webapp list
REGION	Region in which the webapp is deployed; used for for azure maven deploy	az webapp list
web.path	Path to the web app on the server; if on root use \	your choice

(d) `application.imageURL` Footer image URLs, comma separated, either relative URLs into the eTarget application or absolute URLs to other web resources

(e) `application.imageAlt` Alternative text for imates, displayed for screen readers or when images fail to laod

(f) `application.support` Email address on site to request help

(g) `data.blood` whether blood data is available

(h) `data.tumour` whether tumour data is available

(i) `page.additional_reports` display additional reports tab

(j) `page.ihc` display IHC tab

(k) `page.fmblood` display FM Bloods tab

(l) —verb—page.fmtumour— display FM Tumour tab

(m) `page.tumourngs` display Tumour NGS tab

(n) `page.ctdnangs` display CtDNA NGS tab

(o) `page.ctdnaexploratory` display CtDNA Exploratory tab

(p) `page.pdxcdx` display PDX CDX tab

(q) `page.name` comma separated list of providers as found in the eTarget database table `CONCEPT_DATASOURCES` column `panel_name`

(r) `page.name.[name1]` name which should be displayed for this provider

(s) `page.name.[name1].blood` whether blood tab should be displayed

(t) `page.name.[name1].tumour` whether tumour tab should be displayed

(u) `page.name.[name1].code` two letter code which will be used for this provider in reports

2. config.properties – contains connection settings

(a) `serverid` Maven setting in `.m2/settings.xml` which contains the login data (username + password) the server

(b) `azureFtpUrl` The URL to use in case deployment is done using FTP (caused problems and generally moved to azure maven deploy)

(c) `toWarFile` The location for which the war file is to be build; needs to match where to deploy in Apache

(d) `databaseURL` URL to the database; value taken from `pom.xml`.

(e) `dbuser` user name for the database; taken from `pom.xml`.

(f) `dbpassword` password for dbuser; taken from `pom.xml`.

(g) `storageURL` URL to Azure blob storage (location of the PDFs); taken from `pom.xml`.

(h) `storageContainerName` Name of the blob 'folder' for reports (PDFs)

(i) `trialContainerName` Name of the blob 'folder' for trial HTML documents (generated from the trail finder)

(j) `webapp.include.path` location of the include folder on the web server; taken from `pom.xml`.

(k) `RESOURCEGROUP_NAME` Name of the Azure resource group for Maven Azure deploy; taken from Azure portal

(l) `WEBAPP_NAME` Name of the Azure web application; taken from Azure portal

(m) `REGION` Name of the deploy region; note: do not take from the Azure portal, required is the short name `az` client provides, the one without spaces and catital letters.

(n) `PRICING` Pricing of the web application; taken from the Azure portal

(o) `SUBSCRIPTION_ID` Id of the subscription; taken from the Azure portal

(p) `web.path` location of where it should be deployed

3. logging.properties – standard logging properties file; set logging levels

3.5 Active Directory

Authentication is done using Microsoft's Active Directory service. This needs to be configured on the Azure portal. This part is best done by the administrator of Azure.

Enable Authentication: Azure Portal: go to the correct App Service; Authentication/Authorization; switch on 'App Service Authentication'.
Use Azure Active Directory: 'Action to take when request is not authenticated': 'Log in with Azure Active Directory'.
'Authentication Providers': configure Azure Active Directory. Management Mode: Advanced.

Client ID: Copy from Azure portal; Azure Active Directory; App registration; Application ID
Issuer URL: `https://sts.windows.net/`+ Azure portal; Azure Active Directory; Properties; Directory ID

4 Code repository

A copy of the web application code is hosted on the UoM Research IT Github repository[3]. A separate repository exists for the data processing code, also on the UoM Research IT Github[4].

Robert Haines[5] at Research IT can provide access to these repositories.

5 Build

Building is done using maven. There are currently three profiles defined: dev, test, prod. Before building check the content of those environments. The project settings are expected in a directory 'profiles' with a sub-directory named as the profile name.

```
<servers>
  <server>
    <id>etarget_dev_app</id>
    <username>USERNAME_AZURE</username>
    <password>PASSWORD_AZURE</password>
  </server>
  <server>
    <id>etarget_test_app</id>
    <username>USERNAME_AZURE</username>
    <password>PASSWORD_AZURE</password>
  </server>
  <server>
    <id>etarget_prod_app</id>
    <username>USERNAME_AZURE</username>
    <password>PASSWORD_AZURE</password>
  </server>
</servers>
```

Information about server location, usernames, and passwords can be found in the Azure Portal ($AppService \rightarrow GetPublishProfile$.)

[3] eTarget, `https://github.com/UoMResearchIT/TARGET`
[4] Data processing, `https://github.com/UoMResearchIT/target-data`
[5] Robert Haines, `mailto:robert.haines@manchester.ac.uk`

`id` is a identifier you choose and will refer to in `pom.xml`.

`username` is a combination of *msdeploySite\userName*

`password` is *userPWD*

Command to build i.e. a dev web application:

```
mvn clean package -Denv=dev
```

Assuming everything runs OK this produces etarget_dev.war file in the target directory. Ready to deploy!

5.1 Building with Java Overlays

Java Overlays is a way to overwrite part of your application with different files. This is of advantage when building for several clients with slightly diverging needs, i.e. diverging naming conventions for elements in the UI, different logos and such. Currently the configuration files are kept in an overlay structure so not to compromise access to DB details.

Creating an Overlay requires a separate project. Files which should be overwritten by this overlay need to be located at the same directory location as in the original project. So if you want to overwrite the file `\profiles\prod\config.properties` you need to create that file in that location in the new project.

In the `pom.xml` you need to add etarget as and dependency of the new project:

```
<dependencies>
        <dependency>
                <groupId>org.digitalecmt</groupId>
                <artifactId>etarget</artifactId>
                <version>2.0-SNAPSHOT</version>
                <classifier>classes</classifier>
        </dependency>
        <dependency>
                <groupId>org.digitalecmt</groupId>
                <artifactId>etarget</artifactId>
                <version>2.0-SNAPSHOT</version>
                <type>war</type>
        </dependency>
</dependencies>
```

Note: there are two dependencies, one for the Java classes, and one for the war file. Make sure that you build against the right version of etarget!

Otherwise you should define your profiles in the same way as in the base project and you need to define your build plugins and resources as needed.

Building has to be done in two steps:

```
mvn install -Denv=[env]
```

on the TARGET repository

```
mvn package azure-webapp:deploy -Denv=[env]
```

on the new Overlay repository.

The first command installs the built base eTarget package into the local Maven repository, so that the second build process can find and use it.

6 Deployment

Deployment to all environments can be done via maven, i.e. for the dev environment:

```
mvn install -Denv=dev
```

This uses the `wagon-maven-plugin` to upload the war file to the configured server using ftp. Running this goal for the first time on an empty App Service will cause an error as it tries to delete directories which do not yet exist. One way around this is to disable the install goal remove part of the plugin for the first run. Alternatively just create the directories so that the plug-in has something to delete.

On a Linux based App Service the deploy by ftp does not work. In this cases there is a second way to install a war file. Please check for spelling issues carefully as this will create Azure services if it can't find them!

```
mvn package azure-webapp:deploy -Denv=dev
```

Uses the `azure-webapp-maven-plugin` to deploy the web app. (After the packageing goal a specific deploy goal is called.) There are different configurations required for a Windows based Application server than for a Linux one!

Alternatively you could use any other deploy method provided by Azure. The aim is to get the etarget_$env.war file into the webapps folder of the correct server. Tomcat takes over the deployment of the war file on the server.

Deployment of code to the processing server VM is via SSH / SCP / git.

7 Access

Please contact Paul Fitzpatrick for a copy of the spreadsheet with login details for test and production.

Paul will also need to create accounts for you in Azure to access the dev VM (SSH) and Azure AD (to login to the frontend).

SFTP login details can be found in the Azure control panel for the relevant App Service, or in the application code config files - please email me to request these as they are not in the repo.

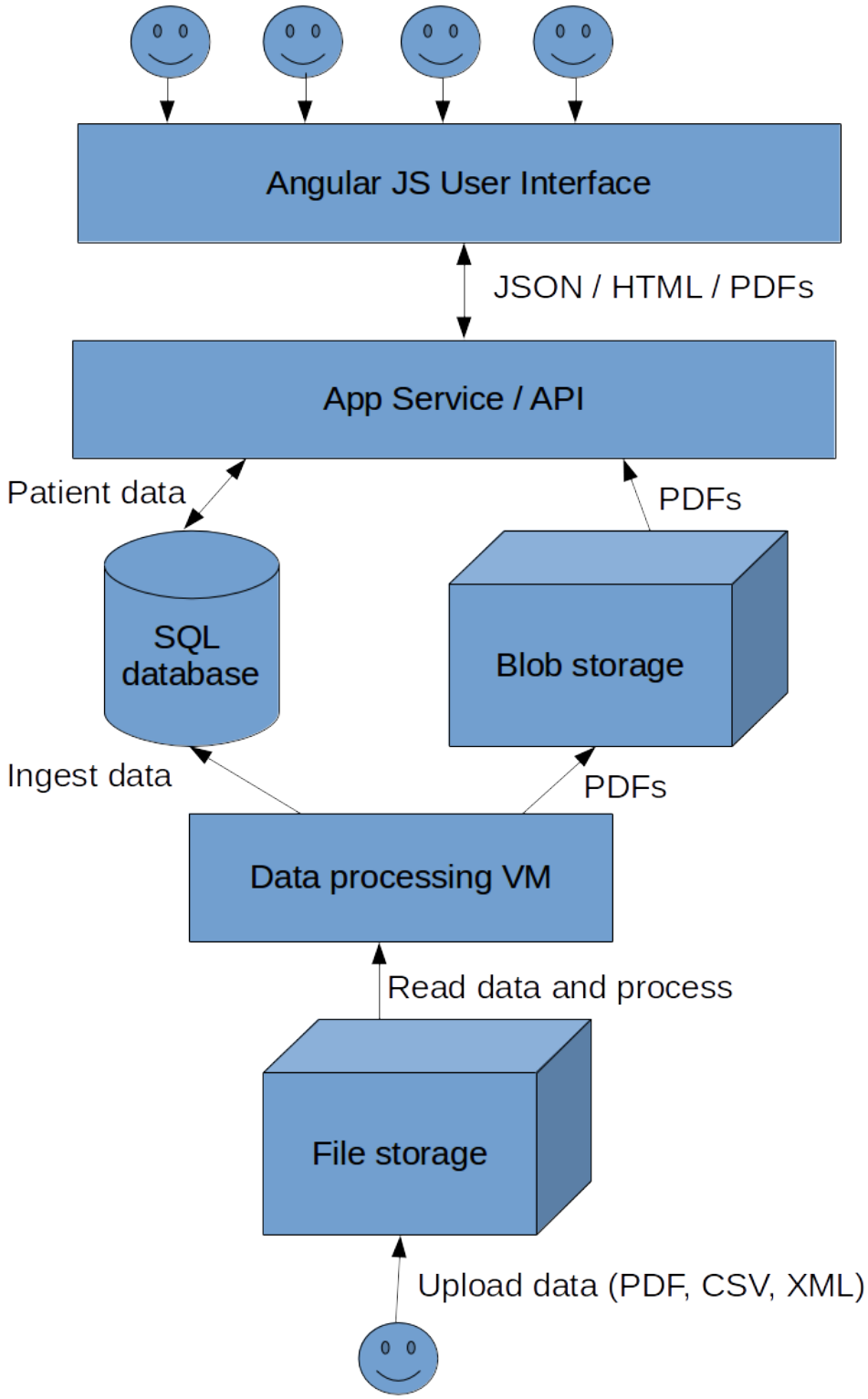
8 Known issues

The following are a list of known issues in that may not be documented in JIRA.

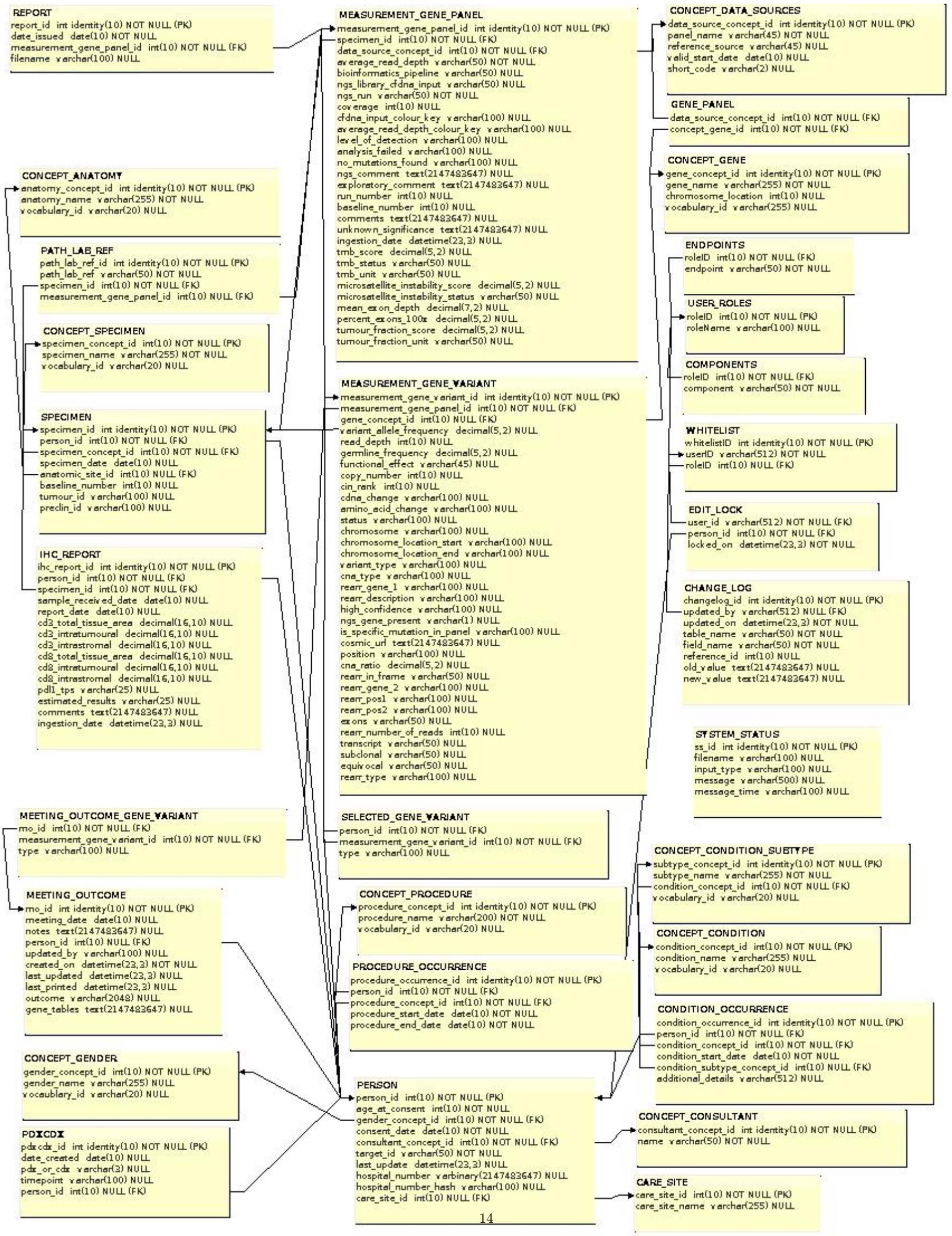
- Distro mismatch for the data processing VMs. The previous sysadmin set up the wrong distro for test and production. Dev is running Ubuntu, whereas test and prod are running RHEL. Please be aware of this when installing / updating packages, deploying code, or debugging.

9 Appendix

A Application data flow diagram



B DB schema eTarget version 1.0



C Web application structure

```
/webapps
└── /ROOT
    ├── index.jsp
    ├── logout.jsp
    ├── /css
    │   ├── pikaday.css
    │   ├── style.css
    │   └── mobile.css
    ├── /images
    │   └── example.png
    ├── /js
    │   ├── angular.min.js
    │   ├── moment.js
    │   ├── notice.js
    │   ├── pikaday.js
    │   ├── inactivity.js
    │   └── app.js
    ├── /includes
    │   ├── ctdnangscheckbox.html
    │   ├── admin.html
    │   ├── adminlink.html
    │   ├── reportextraction.html
    │   ├── tumourcheckbox.html
    │   ├── patienttable.html
    │   ├── meetingoutcome.html
    │   ├── ctdnaexpcheckbox.html
    │   └── ...
    ├── /WEB-INF
    │   ├── /classes
    │   │   ├── /org
    │   │   │   └── /digitalecmt
    │   │   │       └── /etarget
    │   │   │           └── compiled_class_files
    │   │   ├── config.properties
    │   │   ├── application.properties
    │   │   └── logging.properties
    │   └── /lib
    │       ├── gson-2.8.0.jar
    │       ├── jtds-1.3.1.jar
    │       ├── jollyday-0.5.1.jar
    │       └── ...
    └── /api
        ├── component.jsp
        └── data.jsp
```