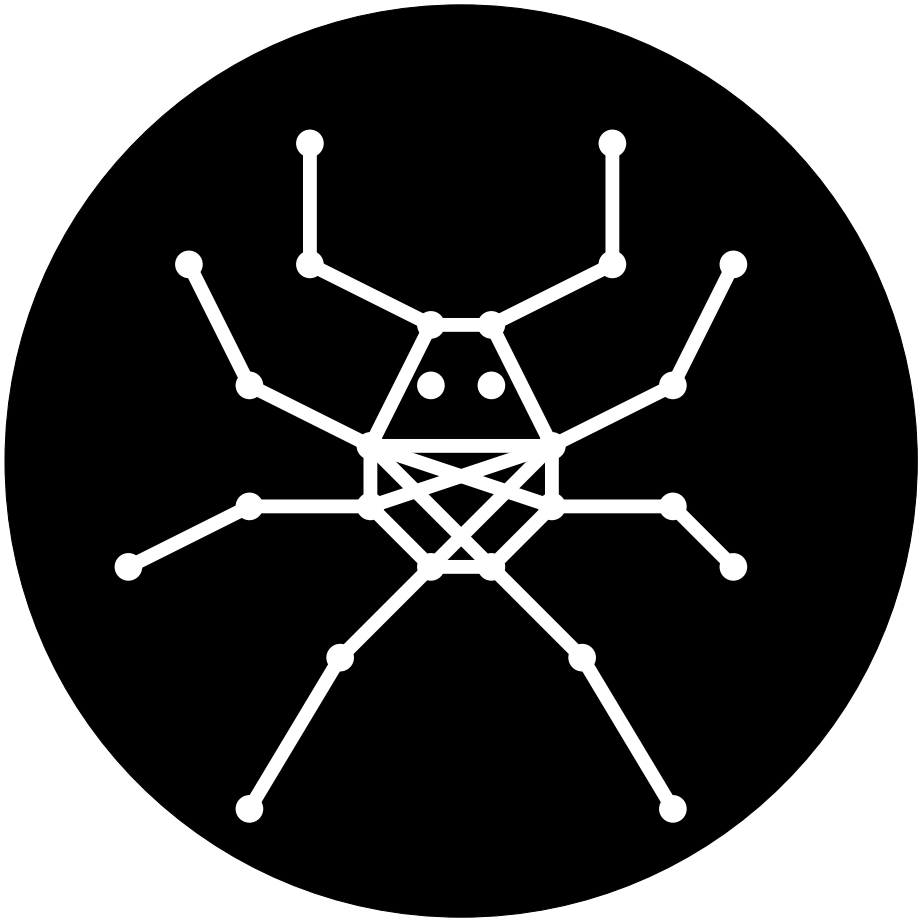
Group Project



Project Darknet – Specification

Johann Beleites, Ibtehaj Nadeem, Farah Kiran Patel,
Josh Treon, Augustin Zidek

University of Cambridge, Lent 2014, Version 2014-01-28 14:37:18

Contents

1. General Investigation of Problem and Background

Project Darknet is a new experimental theatre production currently under development. It is being designed to "explore the world of computer hacking and its impact on crime, warfare, sabotage and espionage"[1]. In fact, the ultimate aim is to raise awareness about today's problems with cybercrime. In order to do this effectively, the audience in this theatre production will be raised from a purely passive audience to actually being involved and emotionally invested in the show. This will be done with the audience members' digital data which will be collected before and/or during the show. During the show the audience will then be presented with the data in one way or another, causing emotional responses when audience members recognize it as being personal information, which was perhaps thought of as only being accessible to a small circle of people.

The key to this project is gathering, organising and presenting data from and to the audience. For the project to succeed, therefore, the data-collection phase is critical with the rest of the project building upon it. This is also one of the biggest issues in the digital world nowadays. Information about people is a million (if not billion) dollar business. In fact, large companies such as Google and Facebook depend on personal data for a large part of their profit. Just think of personalisation of ads or search results. Clearly, personal data is very valuable and any value of significance attracts both legitimate and criminal uses. As a result there is a huge set of available tools to harvest user data.

Data can exist in a variety of ways. Simply sending an HTTP-request to a web server can reveal information such as the browser used, browser language settings and the IP address (revealing an approximate geographic location). The use of Flash and JavaScript on web pages may increase the amount of information obtainable. Large companies like Google can use tracking cookies to collect statistics about which sites a user visits. Within minutes of browsing the web, the average user will have revealed a lot about themselves without even realising it. Many of these basic passive techniques can be used for this project. Additionally, for Project Darknet, a ticketing system is already in place, which can be used to obtain some basic data about the audience.

There are, of course, numerous more active approaches to obtaining data. Phishing, for example, or even simply asking the user to allow an app access to their phone, computer or Facebook can reveal a vast amount of personal data with the average user being unaware of it or simply not caring. In many cases this is not even necessary, as a large number of Internet users do not protect their online data sufficiently, perhaps due to lack of knowledge or awareness. Numerous people-search engines, in fact, specialize on scraping all data about people they can find in

[1] `http://www.junction.co.uk/node/6099`

public areas of the Internet and presenting them in a structured manner to anyone who might be searching. Again, one can see several data sources which could turn out to be very useful for this project.

All the methods of data collection discussed so far are relatively harmless and largely available to the general public. In order to create a convincing production, however, more may be necessary. Given the value of personal (or even secret) information, a range of more aggressive techniques has been developed, which is often generalised as "hacking." By penetrating users' devices, sniffing users' (Internet) connections and the like, nearly any piece of digital data in an average user's possession can be obtained, if done well. Popular concrete methods in this area include Man-in-the-Middle attacks, network sniffing, spyware and other forms of device and (especially) network penetration. Some of these may be interesting for Project Darknet as well.

Another key element is the organisation of the data collected. Again this is no trivial issue. Companies such as Palantir exist due to the need of data management and organization. How well data is organised and structured can be a deciding factor in problem solutions. Some sort of structured and organised database will thus be necessary for Project Darknet. Given already existing services such as the people-search engines mentioned earlier, there are structures already in existence which handle people's data. Although the back-end structures will most likely not be available for study there are certain design-decisions which can be made from the front-end presentation of such services, if one applies similar ideas to structuring the data for Project Darknet.

Lastly, presentation of the data will have a huge effect on the impact of this production. Given that the piece is currently still under development and does not have a clear plot or structure, a dynamic approach to this (and also some of the previous parts) is required. The piece will feature a stage and actors as traditional plays do. Also, equipment for audio and visual effects will be available. A large part of the audience will most likely be in possession of a smartphone or otherwise a basic mobile phone. All these factors can be included in the design of the presentation.

1.1 Notes on Legal and Ethical Issues

The project is about "hacking" and is intended to "scare" or emotionally involve the audience to some degree. This brings up a variety of ethical and especially legal issues. Although presenting the audience with their secret login information to online services, for instance, would certainly be quite impressive, it would also be illegal, especially when considering techniques of data collection which involve some sort of phishing, penetration or malware, legal implications have to be considered. Some things may be possible with the audience's consent; care needs to be taken in any case, though, as this project will involve handling peoples' personal data.

2. Facilities to be Provided

The main facility the system will provide is access to a database of personal data collected from various online sources for each audience member who has given their consent. The data will be collected using the information that each person from the audience provides in the ticket booking system. The collected data might not correspond with reality, however, so that uncertain/unverified data will be marked and a ranking based on the probability that the data is consistent will be provided.

Secondly, the system will provide a graphical interface for creating effects during the play. This might involve using the audio/video system in the theatre, calling or texting the audience, etc.

This system will help the client to raise awareness about the privacy and security on the internet among the public. It is not supposed to be used for any kind of hacking, stalking, or any other criminal activity. The whole system is a prototype and a rather experimental piece of software that will evolve together with the play itself. The system will be modular and therefore if the script of the play changes in the future, the system will be easily modifiable to the new needs of the play.

3. Major Components

Given the nature of the project, we suggest the major components of the system are implemented in Java. The rationale for this is that Java provides sufficiently high level abstractions capable of supporting the modularity requirements of this system.

Having analysed the functional requirements of the project, we propose a structural breakdown of our system into three distinct domains:

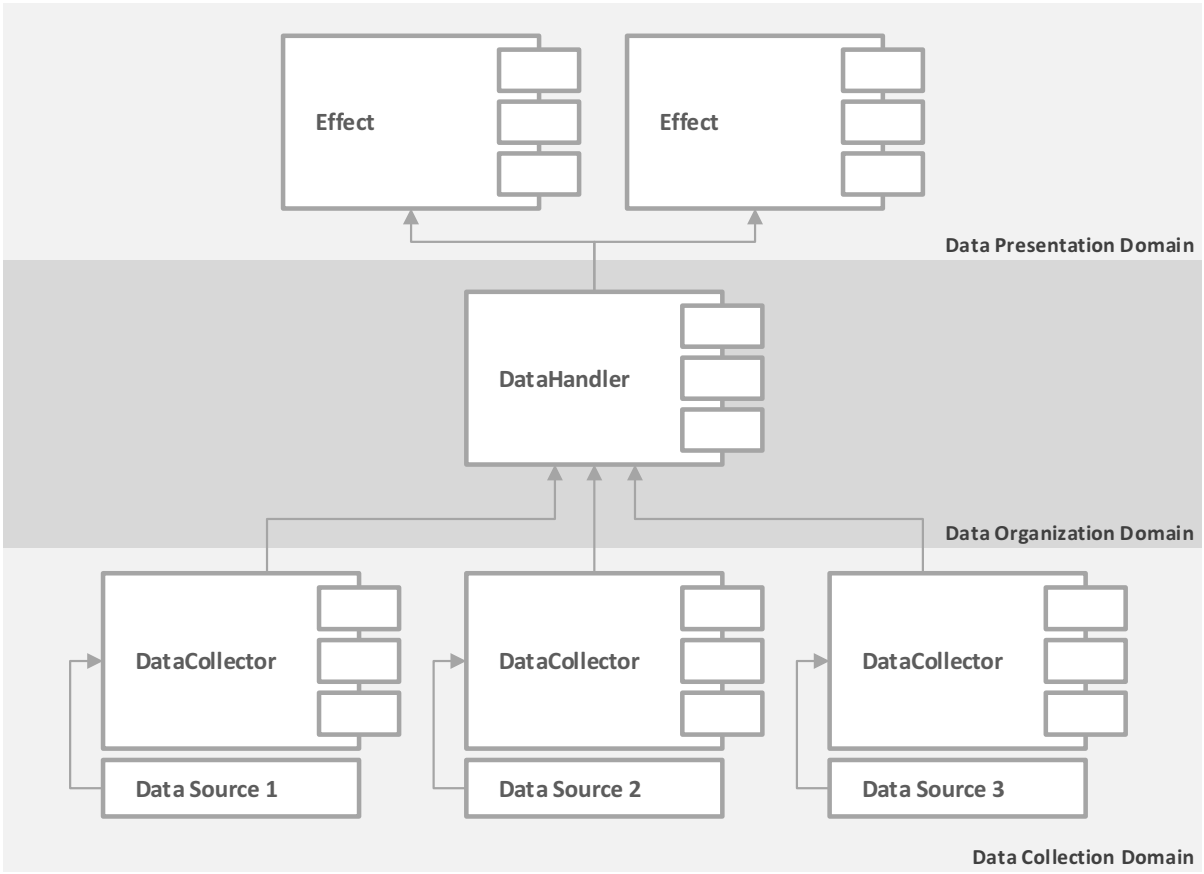


Figure 3.1: The structure of the system. The arrows represent data flow.

We can envisage multiple sources providing data for the execution of the desired effects. However, we face the challenge that the data collected by at least some of them may be

1. mutually contradictory;
2. mutually exclusive (or of different types);

3. and/or unreliable.

Furthermore, the volume of data being returned by the same source may vary from person to person.

The proposed system structure depicted above presents our attempt at a solution to these problems. We allow different sources to communicate via one well defined interface to a data organising component in the middle, which in turn represents an abstraction to the sub-systems that produce effects. Data collectors will work in isolation of one another, storing their findings in individual tables, thereby solving the problem of exclusive data. The task of dealing with inconsistent data is delegated to the `DataHandler`.

To provide a heuristic facilitating this task, and to tackle the problem of unreliability, we require that each data collector associates a measure of confidence with all the data items it stores.[1] We refer to this measure as the *reliability* of the information. Some factors to consider when calculating the reliability may include the number of matching results returned by a particular search for an individual, the extent to which data agrees with the search parameters, and whether or not the information is realistic (for example, a UK address may be more realistic than a US address, or a profile picture with one face detected by face recognition software may be more realistic than one with no face or many faces). As a result of this discussion, we propose the following interface for `DataCollectors`:

```
1  /**
2   * The interface for classes responsible for data collection (mining) from
3   * various Internet services and APIs.
4   */
5  public interface DataCollector {
6      /**
7       * Perform a search for an individual and store the result. The 'hints'
8       * about the individual (passed in parameter <code>i</code>) should include
9       * some guaranteed basic information, such as name and email address, but
10      * will also include a unique identifier used within the system.
11      *
12      * @param i The known info ('hints') about the person.
13      */
14     public void lookup(final Individual i);
15
16     /**
17      * Perform a batch lookup. Some APIs support this. If not supported, can be
18      * implement by repeated calls to <code>lookup()</code>. Should spawn a new
19      * thread, and invoke a callback method on the <code>DataHandler</code>
20      * object when done.
21      *
22      * @param list The list of individuals which should be looked up.
23      * @param callback The handler that will process the obtained data further.
24      */
25     public void batchLookup(final List<Individual> list,
26             final DataHandler callback);
27
28     /**
29      * General methods to query common data items, such as name, date of birth
30      * and other personal data. If a specific data item is not supported by a
31      * <code>DataCollector</code>, it should throw a
32      * <code>DataItemNotSupported</code> exception, or return null if the data
33      * is not available for the particular individual. The returned DataItem
34      * includes the reliability. Batch equivalents may also be provided.
35      *
```

[1] In practice, this may be a derived value from an overall "confidence factor."

```
36      * @param i The individual.
37      * @return A piece of data wrapped inside <code>DataItem</code>.
38      */
39     public DataItem getName(final Individual i);
40     public DataItem getDoB(final Individual i);
41     // ...
42
43
44     /**
45      * Access the complete data available about an individual. Adds or
46      * overwrites information to <code>i</code> as indicated by the
47      * <code>Policy</code>.
48      *
49      * @param i The individual.
50      * @param p The policy specifying which data should be overwritten.
51      */
52     public void getAllData(final Individual i, final Policy p);
53
54     /**
55      * Information that is specific to a <code>DataCollector</code> may be made
56      * accessible using special annotated getters. The example shows how a
57      * Twitter <code>DataCollector</code> may return tweets.
58      *
59      * @param i The individual whose tweets should be obtained.
60      * @return A collection of individual's tweets.
61      */
62     @uniqueDataItem("tweets")
63     public DataItem getTweets(final Individual i);
64 }
```

This approach will not only allow the data handler to communicate with the collectors in a coherent manner, but also make the system extensible as this modular structure allows new collectors to be added in the future.

When an effect queries the data handler, it may specify what data it requires, what sources it would prefer and/or what kind of an individual would make for the most effective execution. It will be up to the data handler to rank the individuals according to those criteria and to return those individuals who best fit the description. A plausible interface for this interaction may be as follows:

```
1  /**
2   * Interface for classes that provide data to the front-end.
3   */
4  public interface DataProvider {
5      /**
6       * An effect can query the <code>DataHandler</code> for a suitable target
7       * for executing an effect. It can specify a set of strict requirements
8       * (e.g. requires a profile picture) and a set of non-strict requirements,
9       * or preferences (e.g. an older individual is preferable). The
10      * <code>DataHandler</code> should find the best match, considering
11      * requirements and reliabilities. If the requirements cannot be satisfied,
12      * an <code>UnsatisfiableRequirementsException</code> should be thrown.
13      *
14      * @param r The requirements for the target.
15      * @param p The preferences for the target.
16      * @return An individual that is the best match for the given requirements
17      *         and preferences.
18      */
19     Individual getBestTarget(final StrictRequirements r, final Preferences p);
20
```

```
21      /**
22       * An effect should be able to request multiple targets. The returned list
23       * should be sorted by suitability. This method will attempt to return as
24       * many individuals as are specified in the last argument, but may return
25       * fewer. The effect should check whether the length of the list is
26       * sufficiently long.
27       *
28       * @param r The requirements for the targets.
29       * @param p The preferences for the targets.
30       * @param howMany The number of individuals that are needed for the effect
31       * @return A list of individuals that are suitable for the given
32       *         requirements and preferences sorted by suitability.
33       */
34      List<Individual> getMultipleTargets(final StrictRequirements r,
35              final Preferences p, final int howMany);
36  }
```

4. Data Collection

4.1 Spektrix

Spektrix[1] is the box office booking system used by the venues which will host the production. It is a cloud based service which provides an API as well as convenient web tools to access customer data stored in their database. We will assume that the Spektrix API will be used as an entry point to gather prerequisite customer data needed to execute the remainder of the system.

4.2 Social Networking APIs

Social networks have become an ubiquitous and prevailing phenomenon in today's society. However, many users of social networks such as Facebook and Twitter do not realise how much personal information they are revealing via their online profiles. These two facts make social networks a very suitable source for collecting personal information about the spectators.

Most social networks provide general APIs that can be used to search for the profiles of arbitrary people. Nevertheless, such searches will only result in information that has been made publicly accessible by the profile holders. The obtained data will have to be used in effective ways to create the desired illusion of hacking the audience.

The following is an investigation into the applicability of the APIs of some common social networks to this project.

4.2.1 Facebook

The Facebook API allows an extensive search of users to be undertaken. The search parameters can involve names and email addresses, which may be obtained directly from the booking system. A successful search can return personal information, posts, comments, profile pictures and more. Another suitable feature that this API provides is batching, allowing for an efficient search for multiple subjects at once.

Technically the API works by sending a number of HTTP `GET` and `POST` requests between the client and server, but also features a structured query language called FQL. This offers a precise way to search the social network in batches. Data is returned in the JavaScript Object Notation (`JSON`) format. RestFB[2], a Java library facilitating the use of the Facebook API,

[1] `http://www.spektrix.com/`
[2] `http://restfb.com/`

provides a mechanism to map data formatted as JSON to Java objects. It also allows the use of unauthenticated Facebook clients, which are limited to accessing publicly visible data only.

4.2.2 Twitter

From a technical standpoint the Twitter API works in a manner very similar to the Facebook API. However, the type of data that can typically be obtained through its use is different. tweets are often publicly accessible and can be scanned for specific keywords. One application of this may be to raise the reliability of the data – we could for example search recent tweets for keywords such as "Darknet."

Just like RestFB, Twitter4J[3] is a third-party Java library that simplifies the use of the Twitter API from within Java applications.

4.2.3 LinkedIn

LinkedIn provides yet another perspective, although its integration seems less straight-forward. Whilst Java wrappers for the API are available, they have not been maintained and much of their functionality is outdated. Moreover, the LinkedIn API has slightly more rigid authentication requirements. Although incorporating LinkedIn as a data source may give rise to interesting information, it is not feasible to provide this support within the time frame of this project.

4.2.4 Conclusion

From the preceding discussion it is clear that the most popular social networks represent a realisable source for harvesting customer data. The APIs that we have presented make use of the same underlying techniques. Both JSON and XML have emerged as common standards to communicate user data in a structured way. This uniformity makes it feasible to invest in the integration of social networks into this system. Carefully selecting the most appropriate sources will allow us to fine-tune the trade-off between obtaining redundant yet more reliable data and increasing the breadth of information available to the system.

4.3 Man-in-the-Middle Attack (WiFi Hotspot)

One possible (and potentially very efficient) method of data collection is to perform what is essentially a MITM (Man-in-the-Middle) attack. On the night(s) of the show it would be no problem to set up a free hotspot for the audience to use if they wish. Any ongoing traffic through this hotspot can then be recorded to be presented in some way during the production. Tools to perform the actual sniffing are available in great variation already and could be incorporated into our application.

Given that very sensitive data (usernames, passwords, perhaps even payment information) might be transmitted over such a hotspot, though, legal implications need to be taken into account. The audience would at least need to be informed of – and agree to – the fact that their traffic is being recorded prior to connecting over the hotspot. Perhaps limiting the data recorded

[3] http://twitter4j.org/

to details such as visited domain names could help make this a legitimate method whilst still giving the audience the impression of being hacked.

4.4 WHOIS Domain Records

The WHIOIS protocol (as defined in RFC 3912[4]) is an internet protocol for querying database which stores information about registered internet users who own certain Internet resource, such as domain. In our case, we can use the domain owners database if someone from the audience owns their own domain. The WHOIS databases contain two useful pieces of information for us: physical address and a phone number of the domain owner. The WHOIS databases can be queried either using an online API (such as `https://www.robowhois.com/`), or using the standard UNIX tool `whois`.

4.5 Leaked Password Databases

There are databases of leaked passwords (and corresponding login names) available on the Internet. There is an Adobe leaked password database containing about 150 million leaked passwords and various others (SONY, Facebook). These can be used for a password lookup corresponding to the email addresses of the audience. The impact of this on the audience is huge, however there is low probability that the theatre will be visited by any user that is in any of these databases.

4.6 Generation of Fake Data

Various data, such as screenshots of Facebook statuses or deletion of emails in Gmail inbox, persons ID card or driver's licence, can be generated automatically if the system has sufficient data (e.g. profile name and picture for fake Facebook account, Gmail email address for fake Gmail email deletion, name, address and birthday and photo for a fake ID card or driver's licence). The system can generate these using Java libraries for image manipulation.

4.7 Summary

The following table summarises various data sources, their expected outcomes (best-case scenarios) and reliabilities.

[4] `http://www.ietf.org/rfc/rfc3912.txt`

Source	Obtainable data	Reliability	Note
Spektrix	Name, phone number, email	High	
Facebook	Photo(s), birthday, family, friends, public statuses	Medium	
Twitter	Photo, public tweets	Medium	
LinkedIn	Photo, birthday, occupation, address	Medium	Not feasible
MITM attack	Browsing history, emails, passwords, computer info	High	Legal issues
WHOIS	Address, phone number	High	Few people own a domain
Leaked passwords	Login details	Medium	Low probability of hit
Phishing	Login details	Low	Legal issues, low probability of hit

Figure 4.1: Summary of data sources

The second table summarises which data can be obtained.

Data	Probability
Name, email, phone no.	100%
Photo(s)	High
Facebook statuses, tweets	High
Birthday	Medium
Address	Medium
Occupation	Medium
Family, friends	Medium
Browsing history, computer info	Medium
Login details	Low

Figure 4.2: Summary of obtainable data with probabilities

5. Data Organisation

As this project is largely about collecting and presenting data, effective organisation methods are essential. We have briefly discussed the challenges that we face in coordinating data obtained from various sources. At this point we must consider the volumes of data that will have to be processed. A medium sized theatre could attract a few hundred visitors every night. Our system will need to process all of them ahead of the show. Since data collectors are required to work in isolation, we have scope to exploit concurrency, if only for the data collection phase. We have justified the need for the data returned by different collectors to be stored in private data structures. This greatly mitigates the potential for data corruption.

Bearing the above considerations in mind, we propose a simple relational database implementation. This approach would provide sufficient robustness, a coherent model of data storage and, above all, powerful tools to query the data in useful ways (cf. the `DataProvider` interface). In addition, it would allow us to use an existing system and thereby simplify the implementation of this component, which would otherwise be a lengthy task in its own right.

The proposed layout of the data is shown in the diagram below:

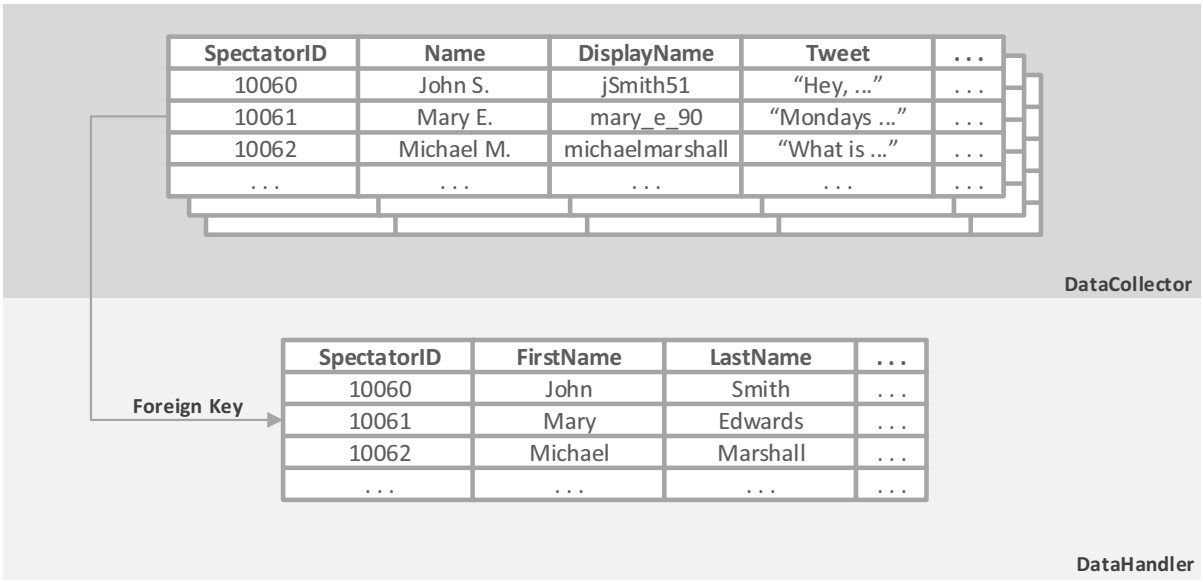


Figure 5.1: The data layout

The data handler maintains a table with the most basic personal data about each customer. This data is directly obtained through the booking system and will serve as the search parameters for the data collectors. Each data collector will maintain an individual table, in which it will

store the results of its own searches. This approach is chosen in foresight of the varying nature of data expected from the various data providers. The data handler will interact with this information via the `DataCollector` interface.

6. Execution of Effects

The execution of effects will be broken down into a series of modular units, each of which is associated with an effect and the data required to implement that effect, if any. This design approach avoids creating dependencies between effects and is crucial in facilitating the dynamic nature of the project; it is very likely that new ideas for effects will be proposed throughout the project's timeframe, and so our system must be adaptable to changes and modifications. A module that uses audience members' personal information will have access to relevant parts of the database described in the above section. Thus once implemented, a module should not have to be explicitly fed arguments upon every instance of execution.

In order for the effects to be executed during the production, we will produce a GUI that will contain all of the effects we are producing for the client. During a show, a crew member can select the desired effect in the GUI and then select to run it at the correct moment. When the effect is targeting a specific audience member, the GUI will also ideally display a ranking of people the effect can be used on that will indicate how impactful the information we have will be so that it will be easy to select whom the effect will target. This interface should be easy to use during a performance and should clearly indicate if an effect will have a time lapse between the start of the run and the effect taking place (or a delay required for the code to compile or run) so that it can be timed well in the performance.

Because the data we are collecting is not necessarily reliably available or suitable for our purposes, we cannot guarantee to acquire the data required for any given effect from an audience member. To combat this risk, we will include a few fallback personas in the database that can be accessed via the execution GUI. These personas will have enough information in the database so all of the effects can be executed on them. They could also be modelled with cast or crew members' information so that it will be possible for the people the personas are based on to act the part of audience members during the performance if needed. These fallback personas will always be available to the crew member running the effects so that if there is no audience member with the information required for a specific effect, they can be used instead so there is no disruption to the flow of the performance.

Although there are many ways in which we could target an individual – such as projecting their personal information – the effects that we deliver are likely to have a much greater impact on the audience if multiple members are involved. One such idea is to have a moment where all the mobile telephones in the theatre are simultaneously rung or texted. Others include convincing the audience that their card details have been obtained and used to defraud them and that their emails have been deleted. We have not yet accurately estimated the time cost of implementing a module, and so have not decided which particular effects we should like to implement. Additionally, we are constrained by facilities provided by the client in the theatre. The set of feasible effects will become much clearer during our first client meeting.

7. Acceptance Criteria

We hope to produce a working prototype system that can deliver effects in a theatre that succeed in "freaking out" the audience, making them feel like the victim of cybercrime.

The system should consist of a series of effect modules, each of which should be executable with a single command or action. All modules acting on audience members' personal information should be able to access the relevant parts of the database and retrieve the data upon which they act. This means that a user of the system should only have to specify which effect module to use when executing effects, and not the corresponding data. Because the project is very dynamic, the independence of modules from each other is very important – it is not known how many we will produce, nor is it certain which effects we should like to implement.

It is highly probable that ideas for new effects will be conceived of after the time frame of the project. Although the implementation for such effects will be the responsibility of the client, our system should provide a simple way for effects and the types of data required to be documented for easy organisation of the modules. This functionality should be provided in the form of a graphical interface which allows a user to view and edit a list of existing effects, as well as add new ones. Types of data will be treated in the same fashion. Finally, effects and types of data can be linked together to form modules.

Knowing whether the effects we produce will have the desired impact on the audience is left to our judgement, and is not a concrete criterion to which we can refer. We will have an opportunity to test our system during the project presentation, which is realistically the only gauge we will have on the effectiveness of the modules we implement.

To summarise, although there is a lot of flexibility regarding the scope of the project, we have identified the firm criteria our system should satisfy: modularity, extensibility and ease of use.

8. Management Strategy

Organisationally, our group has decided to take a more decentralised approach to the project. Instead of having a single group leader, we have spread the responsibilities to three different chair positions. The first of these positions is the communication chair, who will be the primary liaison with the client and also facilitate inter-group communication through utilisation of email and a Facebook group. The second role is a documentation chair who is responsible for ensuring that all of our required documentation is produced on time and presented in a coherent fashion using LaTeX. The last position is the technical chair who is responsible for maintaining our code in GitHub and ensuring we deliver a functional prototype system on time. By group consensus, Ibtehaj will serve as communications chair, Augustin as documentation chair and Johann as technical chair.

To allocate technical responsibilities, we have divided the system into a rough front- and back-end. The front-end will consist of the major component concerning the execution of effects described above and the back-end will consist of the data collection and organisation modules. Josh and Farah will be responsible for the front-end and Johann, Augustin and Ibtehaj will take responsibility for the back-end. Because our project depends heavily on the clients concerns with how we will be able to integrate with a performance, we anticipate a need for adjustments to our plans during the project.

We also are concerned with the timeline of the project and will only commit to pursuing data sources and effect options we think are feasible and will add more should we have time. To ensure we are able to cope with evolving nature of our project due to the client needs and timeline, we have decided to take a dynamic approach and still require weekly communication between the people working on separate modules during the Tuesday/Thursday slots allocated for group project work to allow for any changes to be agreed upon.

9. Documentation

Since one of the aims of this project is extensibility, it will be important to document the final interface to our system in the form of a comprehensive Javadoc. Also, the database of the gathered data will be documented, so that it can be reused or used for future performances.

The system will require human interaction. Therefore, a user manual will be written and delivered with the system. The user manual will be non-technical, explaining how to use the system.

Because of the dynamic nature of the project, a project plan will be devised after the first client meeting. We will strive to document our plan of action and periodically review our progress. Approximately halfway through the project, a progress report will be written.