Lee Ee Jian - Project Portfolio

[github] Role: Developer, Analyst
Responsibilities: Code Quality

PROJECT: SecureIT

Overview

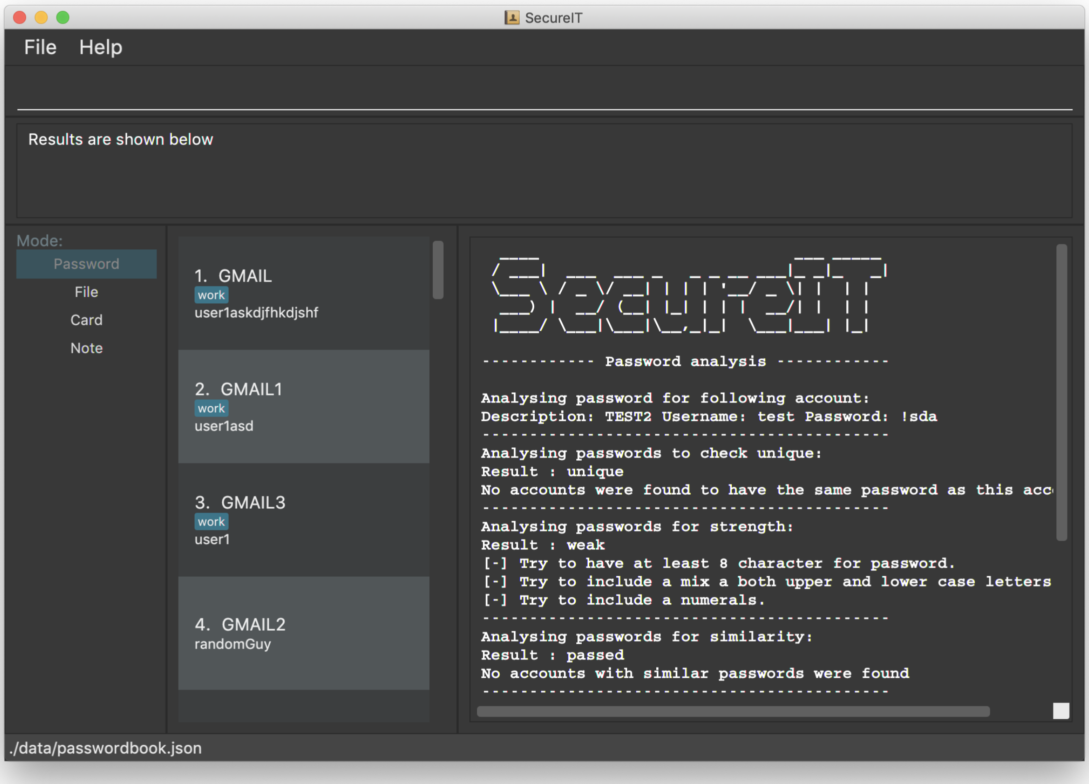
Nowadays, digital is just how business gets done. Employees are more connected, and on more apps and websites than ever before. However, this transformation has also made managing thier digital life that much more complex. Many times, workers end up practicing bad and unsafe security habits (e.g. reusing the same password for all their various accounts).

My group and I realised this problem, and hence set out to tackle this issue by creating **SecureIT**. SecureIT is a morphosis of the desktop addressbook application provided to us to enhance for our software engineering project.

SecureIT is designed to be your all-in-one management tool for all confidential information, and boasts management capabilities for:

- Passwords
- Secret Files
- Secret Notes
- Credit cards

SecureIT targets workers in small and medium enterprises, as well as those in startups, where security enforcement tends to be looser.



Results are shown below

1. GMAIL
work
user1askdjfhkdjshf

2. GMAIL1
work
user1asd

3. GMAIL3
work
user1

4. GMAIL2
randomGuy

```
 ____                           ___ _____
/ ___|  ___  ___ _   _ _ __ ___|_ _|_   _|
\___ \ / _ \/ __| | | | '__/ _ \| |  | |
 ___) |  __/ (__| |_| | | |  __/| |  | |
|____/ \___|\___|\__,_|_|  \___|___| |_|

----------- Password analysis -----------

Analysing password for following account:
Description: TEST2 Username: test Password: !sda
------------------------------------------
Analysing passwords to check unique:
Result : unique
No accounts were found to have the same password as this acc
------------------------------------------
Analysing passwords for strength:
Result : weak
[-] Try to have at least 8 character for password.
[-] Try to include a mix a both upper and lower case letters
[-] Try to include a numerals.
------------------------------------------
Analysing passwords for similarity:
Result : passed
No accounts with similar passwords were found
------------------------------------------
```

./data/passwordbook.json

Summary of contributions

This section acts as a summary of my contributions to SecureIT.

- **Major enhancement: the ability to analyse the level of security of the users' passwords.**
  - What it does: The password analysis feature provides the ability to evaluate the security level of each of the user’s passwords from six unique aspects. Following which, users are able to view a either (i) a **summary report** for **all** passwords, or (ii) a **detailed report** for a **specific** password. The six aspects are:
    - Unique Analysis: Checks that passwords are not the same.
    - Similarity Analysis: Checks that passwords are not at least 70% similar.
    - Strength Analysis: Checks that passwords are complex (length, character types.)
    - Dictionary Analysis: Checks that passwords do not contain known commonly-used passwords.(e.g."password")
    - Sequence Analysis: Checks that passwords do not contain commonly-used sequences. (e.g."ABC")
    - Keyboard Analysis: Checks that passwords do not contain commonly-used keyboard patterns. (e.g."ASD")
  - Justification: This feature greatly improves the utility of SecureIT because it can provide users with insight as to how secure their passwords really are in, and in a simple, hassle-free, human-readable way. Each analysis aspect allows the user to identify different weaknesses in their password and thereby make the necessary corrections to ensure the

security of each account.

  - Highlights: It was highly challenging to implement the analysis feature. Some of the key highlights include:
    - Design: The design of the entire analysis component follows that of a **Command Pattern**. This greatly increases the extensibility of the analysis feature to accommodate even more types of analysis aspects in the future. Also, careful consideration was needed to design and encapsulate the various objects used in the entire analysis feature. (e.g. Analyser, Result, Match, AnalysisReport)
    - Data structure and algorithms: The implementation of the various analysis aspects proved highly complex. Algorithms included the use of more complex concepts such as recursion and multiple pointers. The implementation also required knowledge on the use of more complex data structures such as TreeMaps and HashMaps in order to help boost the efficiency of the product.
  - Credits: [Levenshten-Distance algorithm]

- **Major enhancement: the ability to generate random passwords.**
  - What is does: The password generation feature provides the ability to generate truly random and complex passwords to use whenever they need a new password. Users are also allowed to customise the configuration of the password generation in order to better suit thier needs. (e.g. length, character sets required)
  - Justification: This feature greatly improves the utility of SecureIT because it alleviates the user of needing to think of a password, and guarantees that the password provided is random and thereby secure.
  - Highlights: It was challenging to implement truly random password generation. In-depth research was required on how best to generate truly random passwords. The generation of passwords leverages on the `java.security.SecureRandom` API for a cryptographically strong random number generator (RNG) to make the password generation as random as possible.

- **Code contributed:** [Reposense]

- **Other contributions:**
  - Team contributions:
    - Implemented the method to copy text into the user's local clipboard, which was subsequently used by my teammates in the copy features they implemented. (examples: 1, 2, 3)
    - Reported bugs and offered suggestions for teammates. (examples: #189, #190)
    - Reviewed Pull Requests (with non-trivial review comments):
  - Project management:
    - Helped decide the timeline for the project what to acheive in various stages
    - Created and assigned issues to achieve for each milestone in the lead up to the project submission. (examples: #29, #28, #26)
  - Documentation
    - Made cosmetic improvements to the existing User Guide by creating the project logo.

Contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

Generating a new password: `generate`

Lazy to think of a strong password? Simply enter `generate` into the command box to get one. It's random, secure and totally hassle-free!

💡

Try it out!

- Want to customise your password? Don't worry, we've got that covered too. You can customise the following fields during generation:
  - Length of password (Between 4 and 25)
  - Exclusion of lower alphabets
  - Exclusion of upper alphabets
  - Exclusion of numbers
  - Exclusion of special characters
- Refer to usage format below for more details.

NOTE You are only required to indicate for fields you wish to **exclude**!

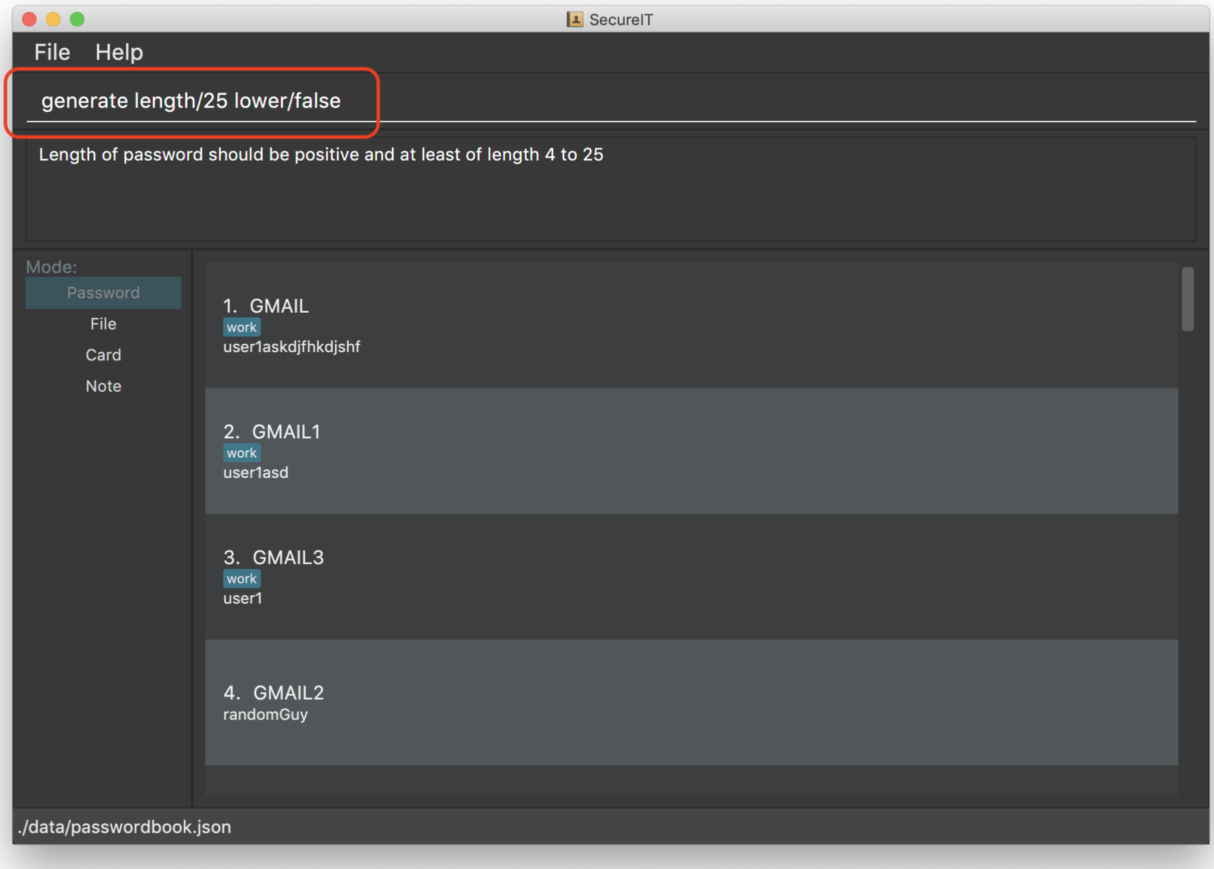
NOTE Please ensure **at least one character set is included**.

Format: `generate [length/LENGTH] [lower/FALSE] [upper/FALSE] [num/FALSE] [special/FALSE]`

Example: `generate`

Example: `generate length/10`

Example: `generate lower/false special/false`



```
generate length/25 lower/false
```

```
Length of password should be positive and at least of length 4 to 25
```

1. GMAIL
work
user1askdjfhkdjshf

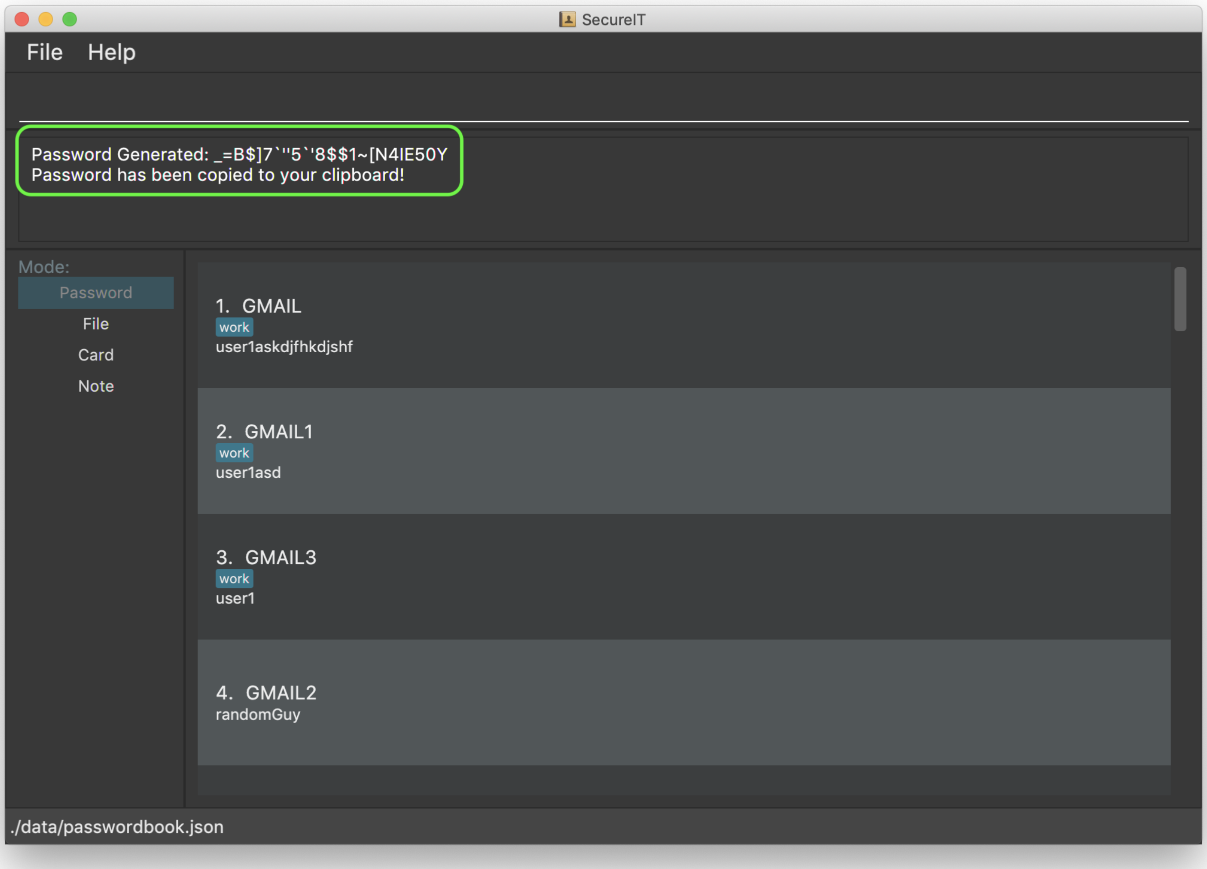
2. GMAIL1
work
user1asd

3. GMAIL3
work
user1

4. GMAIL2
randomGuy

./data/passwordbook.json

Figure 1. Example user input for generate password



```
Password Generated: _=B$]7`"5`'8$$1~[N4IE50Y
Password has been copied to your clipboard!
```

1. GMAIL
work
user1askdjfhkdjshf

2. GMAIL1
work
user1asd

3. GMAIL3
work
user1

4. GMAIL2
randomGuy

./data/passwordbook.json

Figure 2. Example outcome for generate password

Analysing passwords : `analyse`

Curious to know how secure your passwords really are? Type `analyse` into the command box to find out. You might be surprised…

Try it out!

- You can opt to view:
  - A general, summary security report for **all** passwords, or
  - A detailed security report for a **specific** password.
- To view the detailed analysis of a particular password, simply add in the `strong` prefix with the `INDEX`

NOTE `INDEX` used should be that of an existing password.

- Refer to usage format below for more details.

Format: `analyse [strong/INDEX]`

Example: `analyse`

Example: `analyse strong/8`

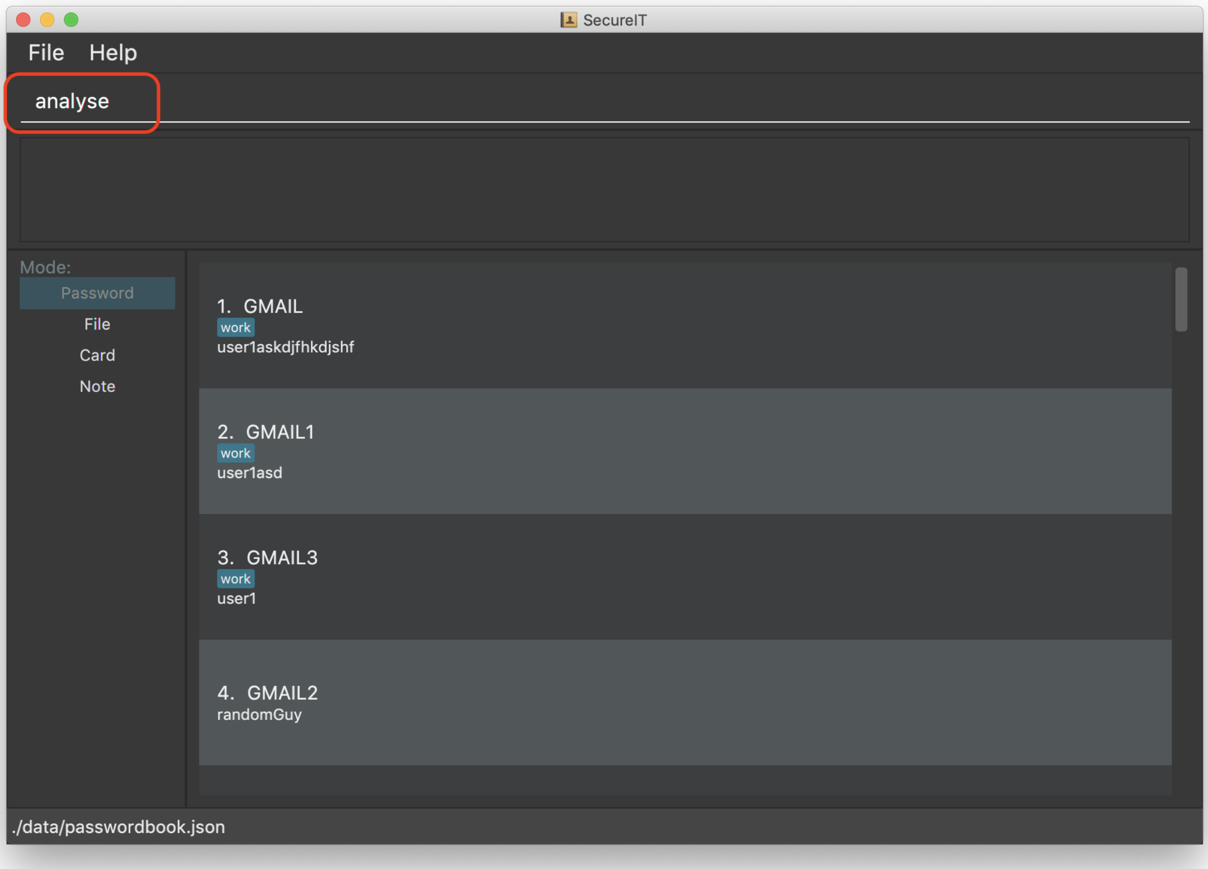
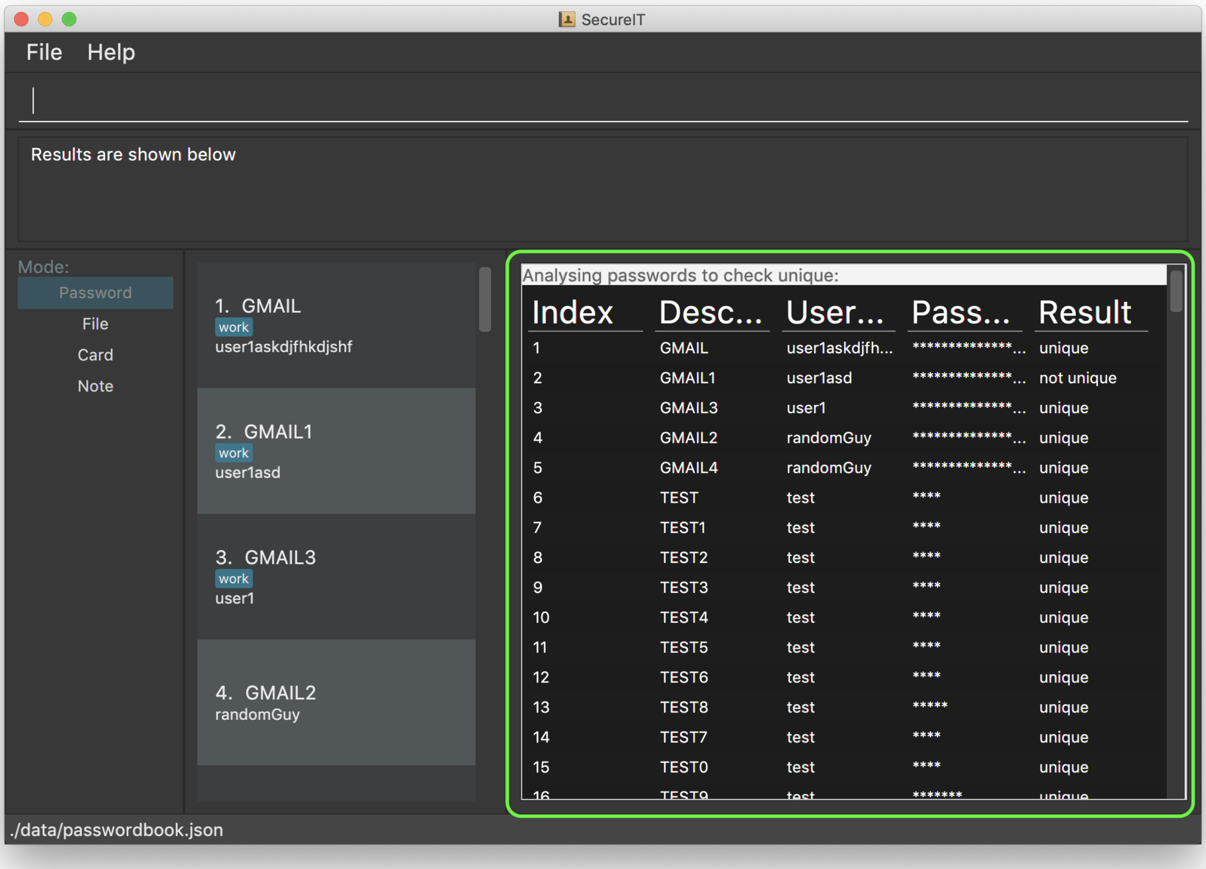


Figure 3. Example user input for analyse



Results are shown below

1. GMAIL
work
user1askdjfhkdjshf

2. GMAIL1
work
user1asd

3. GMAIL3
work
user1

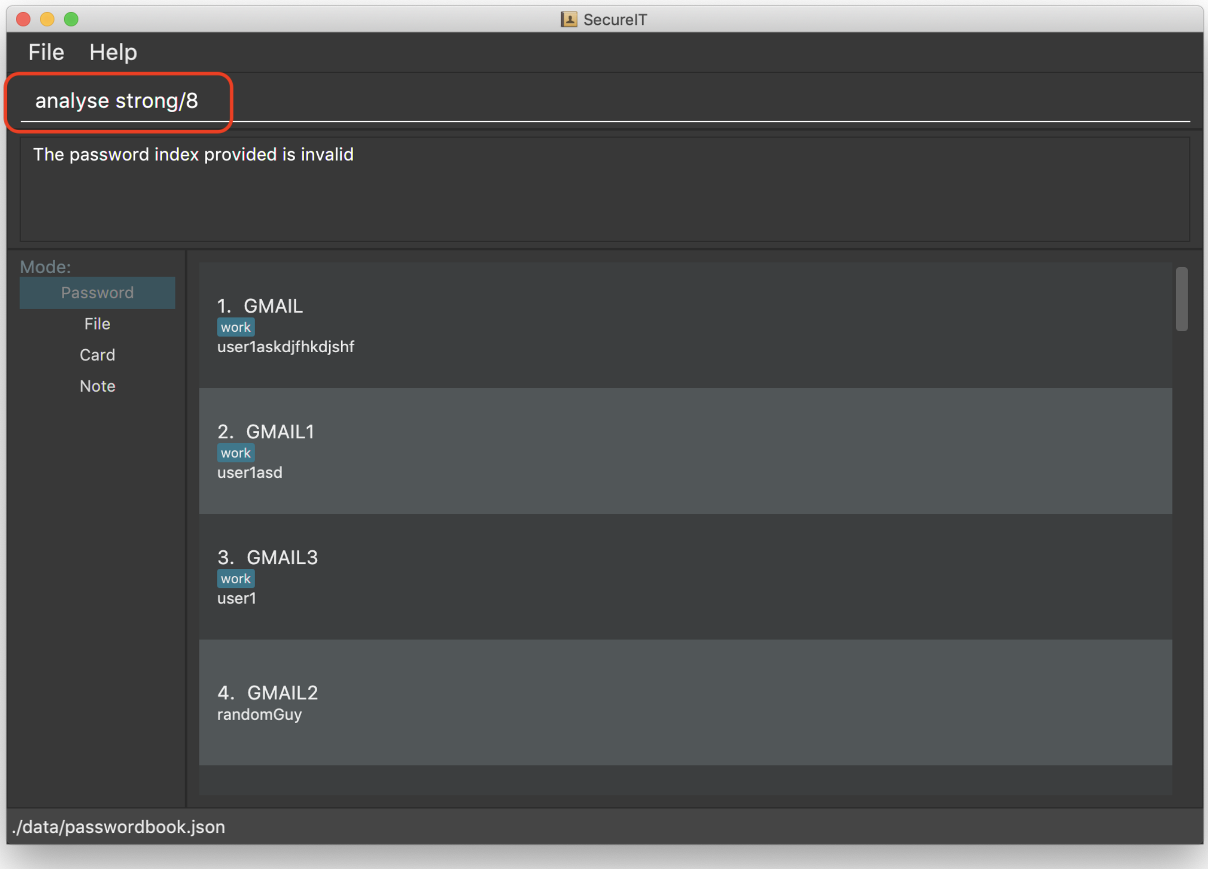
4. GMAIL2
randomGuy

Analysing passwords to check unique:

Index	Desc...	User...	Pass...	Result
1	GMAIL	user1askdjfh...	**************...	unique
2	GMAIL1	user1asd	**************...	not unique
3	GMAIL3	user1	**************...	unique
4	GMAIL2	randomGuy	**************...	unique
5	GMAIL4	randomGuy	**************...	unique
6	TEST	test	****	unique
7	TEST1	test	****	unique
8	TEST2	test	****	unique
9	TEST3	test	****	unique
10	TEST4	test	****	unique
11	TEST5	test	****	unique
12	TEST6	test	****	unique
13	TEST8	test	*****	unique
14	TEST7	test	****	unique
15	TEST0	test	****	unique
16	TEST9	test	*******	unique

./data/passwordbook.json

Figure 4. Example outcome for analyse



```
analyse strong/8
```

```
The password index provided is invalid
```

1. GMAIL
work
user1askdjfhkdjshf

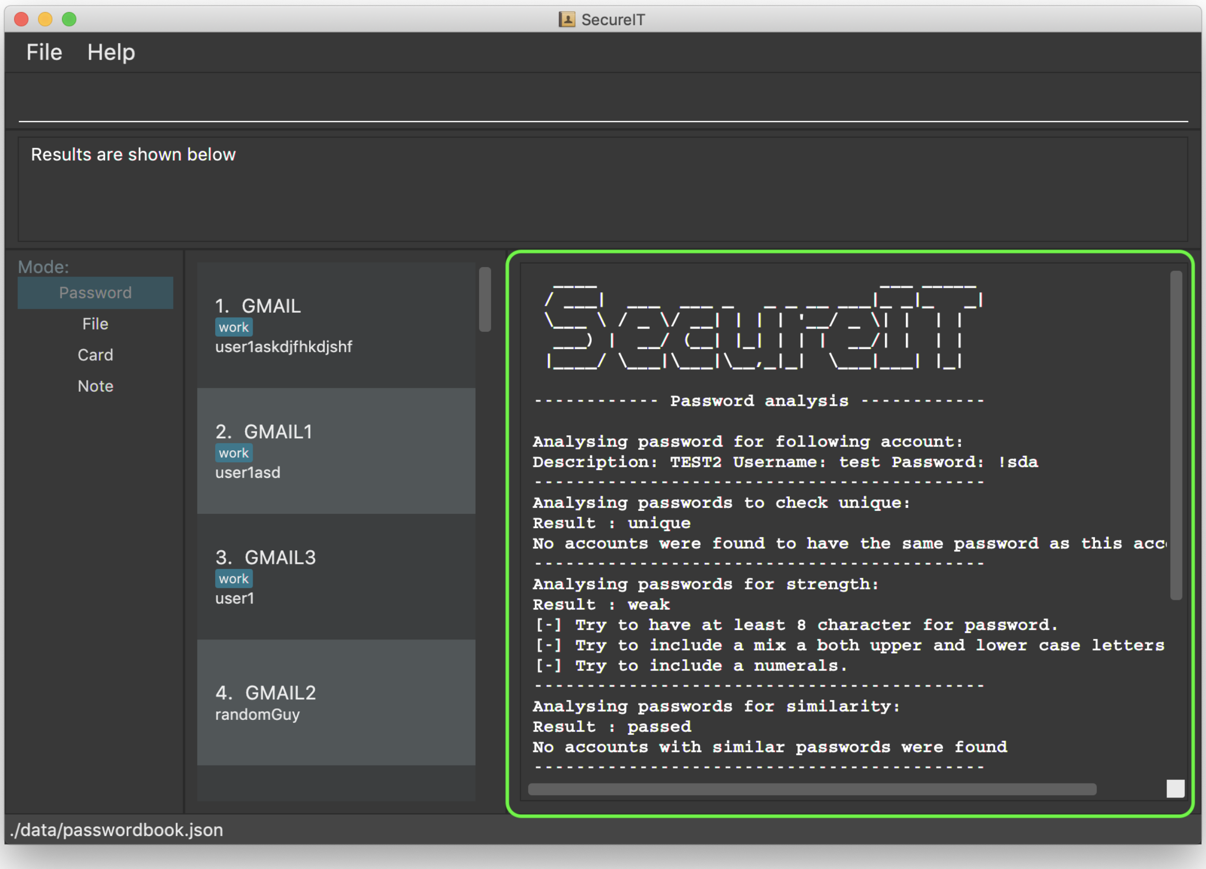
2. GMAIL1
work
user1asd

3. GMAIL3
work
user1

4. GMAIL2
randomGuy

./data/passwordbook.json

Figure 5. Example user input for analyse strong



Results are shown below

```
------------ Password analysis ------------

Analysing password for following account:
Description: TEST2 Username: test Password: !sda
------------------------------------------
Analysing passwords to check unique:
Result : unique
No accounts were found to have the same password as this acc
------------------------------------------
Analysing passwords for strength:
Result : weak
[-] Try to have at least 8 character for password.
[-] Try to include a mix a both upper and lower case letters
[-] Try to include a numerals.
------------------------------------------
Analysing passwords for similarity:
Result : passed
No accounts with similar passwords were found
------------------------------------------
```

Figure 6. Example outcome for analyse strong

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Generate Password

This section provides implementation details on the password generation feature.

Overview

The purpose of password generation is to provide users with a hassle-free way to generate random, secure passwords.

By default, the password generation feature will generate a random password that:

- is 10 letters long
- includes lower case alphabets
- includes upper case alphabets
- includes numerals

- includes Special characters

However, users can alternatively choose to customise any of the aforementioned fields as per their needs through their input.

The generation of truly random passwords is made possible through the use of the `java.security.SecureRandom` API. It provides a cryptographically strong random number generator (RNG) that will be used in the password generation.

The intricacies will be further reviewed in the **Implementation** section below.

Implementation

The following activity diagram summarises the general steps taken during password generation:

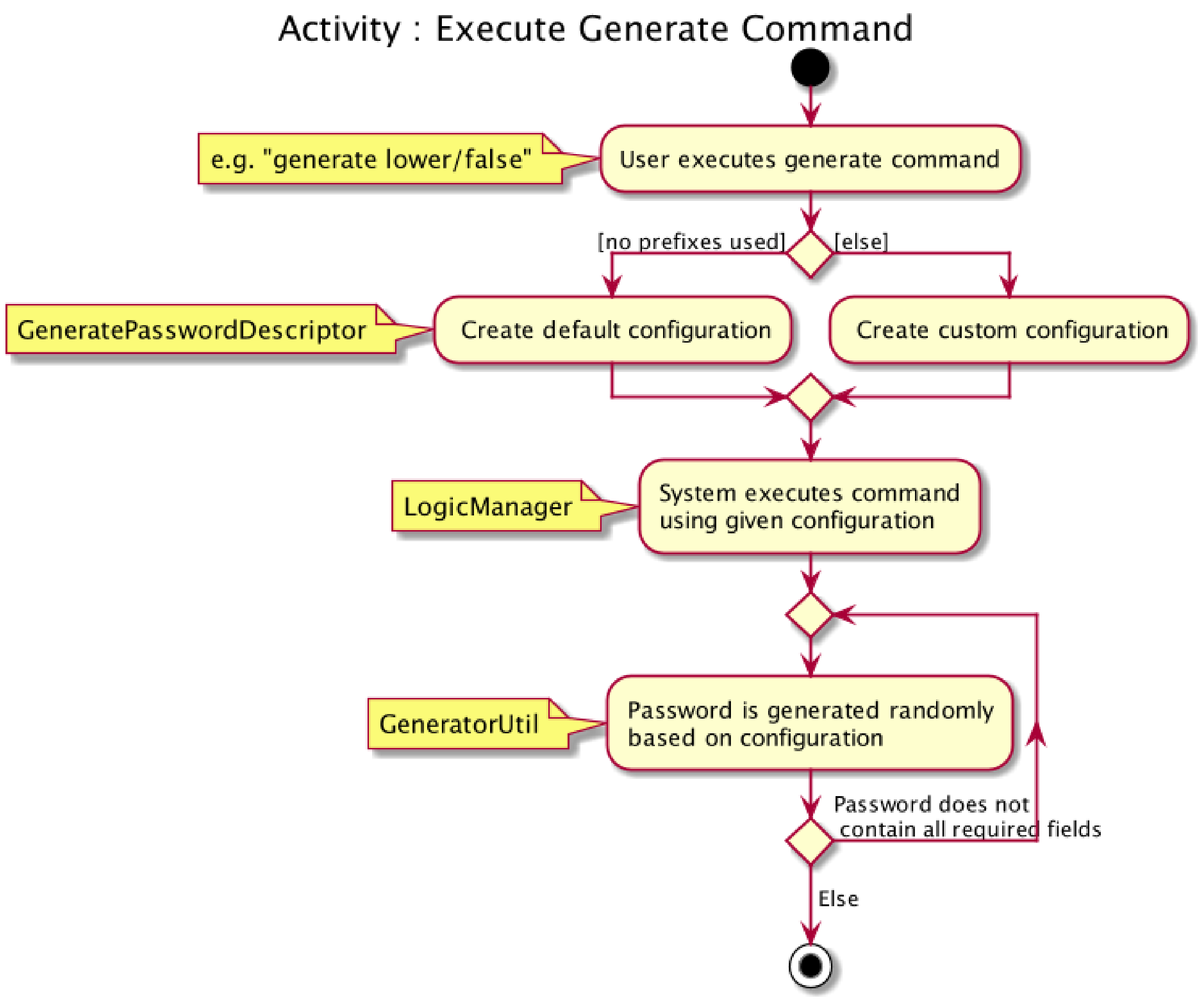

Activity : Execute Generate Command
e.g. "generate lower/false"
User executes generate command
[no prefixes used]
[else]
GeneratePasswordDescriptor
Create default configuration
Create custom configuration
LogicManager
System executes command using given configuration
GeneratorUtil
Password is generated randomly based on configuration
Password does not contain all required fields
Else

Figure 7. Summary flow of actions during password generation.

The steps below outline explicity the generation of passwords :

- The user input is `#parse()` by the `GeneratePasswordCommandParser`.
- Based on the user input, a `GeneratePasswordDescriptor` is instantiated.
  - If the user input includes optional prefixes to customise configuration, we modify the

attributes of the `GeneratePasswordDescriptor` through various setter methods.

  - Else, the default `GeneratePasswordDescriptor` instantiated through the static method `GeneratePasswordDescriptor#getDefaultConfiguration()`.

NOTE The `GeneratePasswordDescriptor` is an object that encapsulates the settings of the **configuration** used for password generation.

- Upon `GeneratePasswordCommand` is instantiated with the given configuration.
- Upon `#execute()`, the static method `GeneratorUtil#generateRandomPassword()` is invoked:
  - Based on the configuration, the relevant character sets (lower-case alphabets, upper case alphabets, numerals, special characters) are added into a list.
  - The `GeneratorUtil` class uses the method `java.security.SecureRandom.nextInt()` to choose a random characters set, followed by a random character within the set to be used in the new password.
  - This process of choosing a character to include in the password is repeated for the length of the password.
- The generated password is then checked to see if it includes all the user requirements (ie. whether it includes all the character sets specified by the user.)
- This process of generating a password is repeated until the user requirements are met.

The activity diagram below depicts the way in which a password is generated by the method `GeneratorUtil#generateRandomPassword()`:

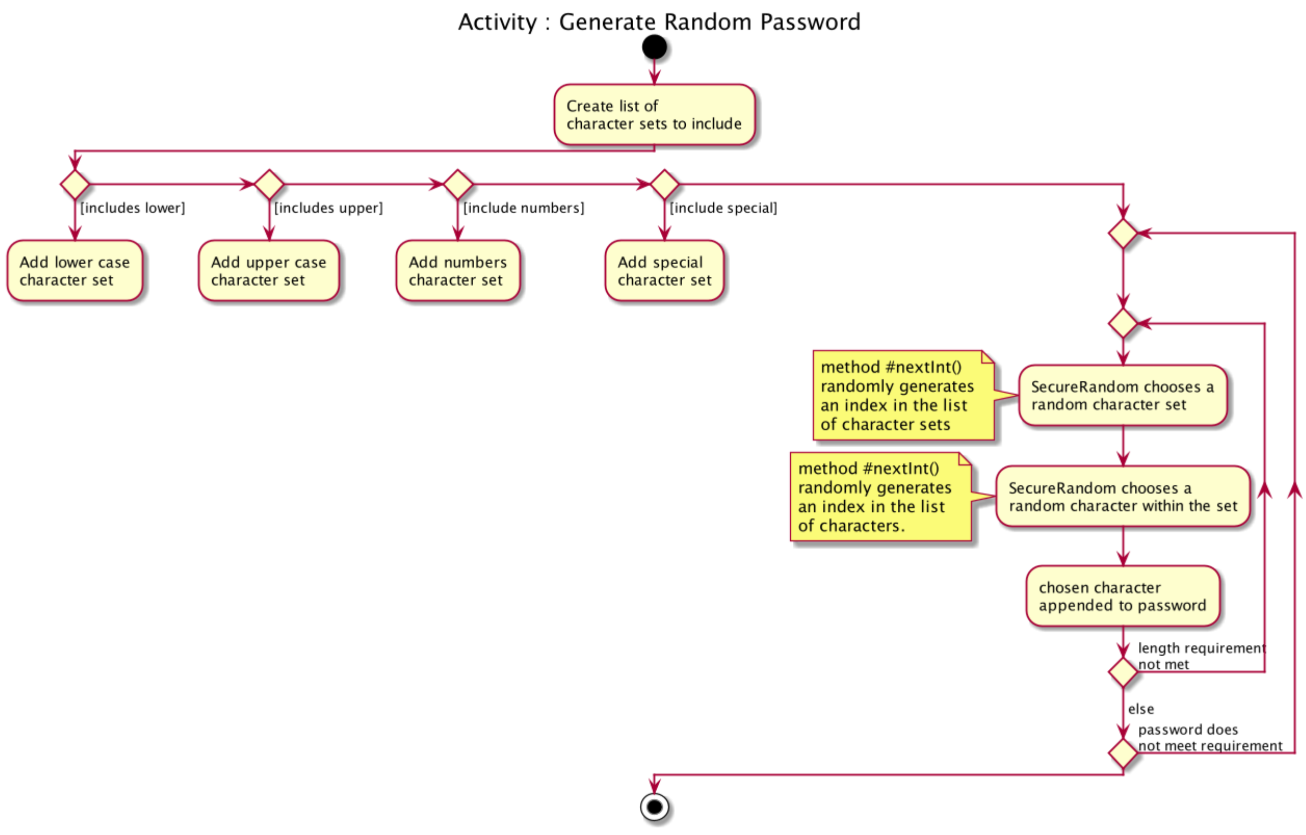

Activity : Generate Random Password
Create list of character sets to include
[includes lower]
[includes upper]
[include numbers]
[include special]
Add lower case character set
Add upper case character set
Add numbers character set
Add special character set
method #nextInt() randomly generates an index in the list of character sets
SecureRandom chooses a random character set
method #nextInt() randomly generates an index in the list of characters.
SecureRandom chooses a random character within the set
chosen character appended to password
length requirement not met
else
password does not meet requirement

Figure 8. Summary flow of actions during the invokation of #generateRandomassword().

Design considerations

These are the considerations we came across when implementing the generate feature:

Aspect: How to generate a random password

Alternative 1 (Current choice)	**Alternative 2** How to generate a truly random password
Randomly choose a character set, followed by a character within the set to include in password: *Pros:* Randomness is achieved because there is no predictability in the way a character set, or character is chosen. *Cons:* Efficiency is compromised. Generating a password this way may produce a password that does not meet user configuration (e.g. random password does not include special characters, although it was supposed to). As such, program needs to keep producing new random password until all the user configuration are met.	**Ensuring all user configurations are met by hard coding a pattern in the way character sets are included** *Pros:* Efficiency is achieved. We are guaranteed to generate a password that matches users' configuration everytime (e.g. lower case → upper case → numeral → special case patter) *Cons:* Randomness is compromised. Generating passwords in such a pattern makes the password style very predictable, hence compromising randomness and security.
Why did we choose Alternative 1: Although less time efficient, the password generated is more random, and thus more secure. Generating a secure password is inline with our use case, and so is a more important factor.	

Analyse Password

This section provides implementation details on the analyse password feature.

Overview

The purpose of analysing passwords is to provide users with information on the level of security of their passwords.

The user can either :

1. View a summary table of results of all password by inputting `"analyse"` OR
2. View a detailed review of a specific password by inputting `"analyse strong/INDEX"`

SecureIT analyses 6 core components of every `Password`, using various `Analysers`. The following table summarises the functionality of each `Analyser`:

	Purpose / Checks for

UniqueAnalyser :	Checks that every password is unique.
StrengthAnalyser :	Checks the complexity of every password.
SimilarityAnalyser :	Checks that no two accounts share a password that is at least 70% similar.
DictionaryAnalyser :	Checks that password does not contain any commonly-used passwords. (e.g. "password")
SequenceAnalyser :	Checks that password does not contain any commonly-used sequences. (e.g. "ABC")
KeyboardAnalyser :	Checks that password does not contain any commonly-used keyboard patterns. (e.g. "qwerty")

The following class diagram is the current structure of the `Analyser` component.

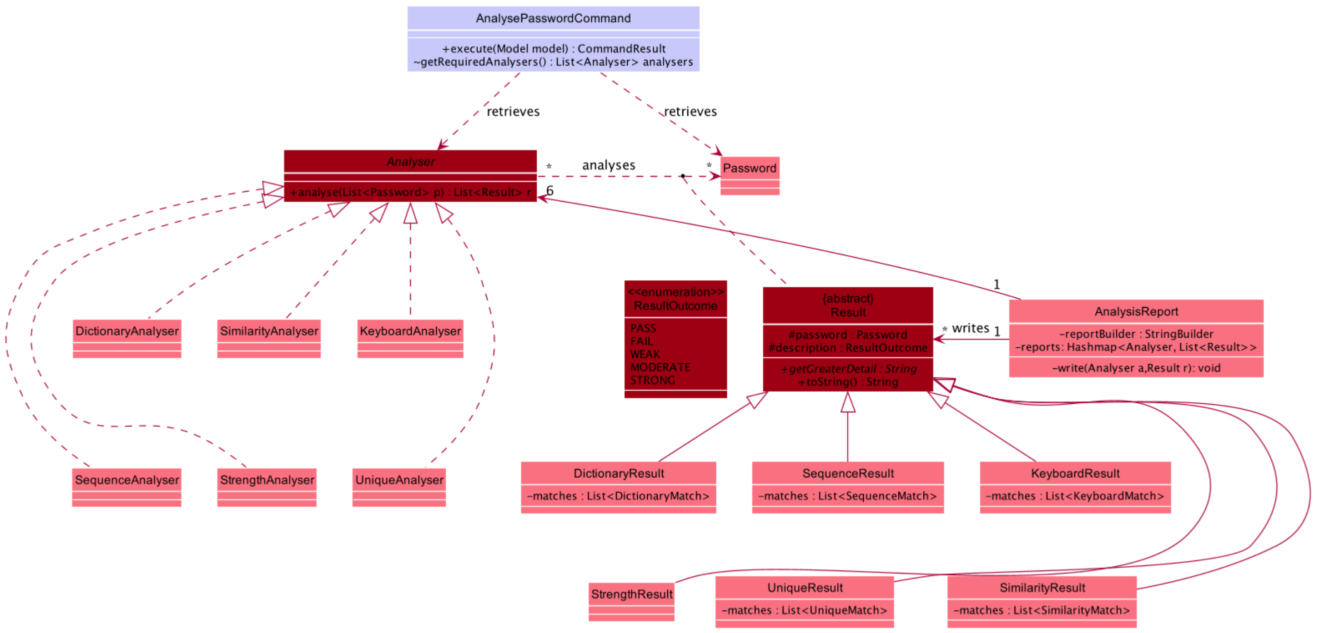

AnalysePasswordCommand
+execute(Model model) : CommandResult
~getRequiredAnalysers() : List<Analyser> analysers
retrieves
retrieves
Analyser
+analyse(List<Password> p) : List<Result> r
*
analyses
*
Password
6
1
DictionaryAnalyser
SimilarityAnalyser
KeyboardAnalyser
<<enumeration>>
ResultOutcome
PASS
FAIL
WEAK
MODERATE
STRONG
{abstract}
Result
#password : Password
#description : ResultOutcome
+getGreaterDetail : String
+toString() : String
* writes 1
AnalysisReport
-reportBuilder : StringBuilder
-reports: Hashmap<Analyser, List<Result>>
-write(Analyser a,Result r): void
SequenceAnalyser
StrengthAnalyser
UniqueAnalyser
DictionaryResult
-matches : List<DictionaryMatch>
SequenceResult
-matches : List<SequenceMatch>
KeyboardResult
-matches : List<KeyboardMatch>
StrengthResult
UniqueResult
-matches : List<UniqueMatch>
SimilarityResult
-matches : List<SimilarityMatch>

Figure 9. Class diagram depicting the structure of the Analyser component.

In summary,

1. The analysis of passwords is initiated upon `#execute()` of the `AnalysePasswordCommand`.
2. Each `Analyser` analyses every password in the password book.
3. The analysis of each password produces the respective `Result` for that password.
4. These `Results` are compiled and written into a `AnalysisReport`, which is then shown to the user in the form of a summary table.
5. Users can opt to view a detailed report for a specific password to get more information on the respective `Result`.

Implementation

On `#execute()`, AnalysePasswordCommand retrieves the current list of passwords via

Model#getFilteredPasswordList(). It also retrieves the required analysers via #getRequiredAnalysers().

Each Analyser has it’s own implementation of #analyse(), which will subsequently be invoked by the AnalysePasswordCommand given the current list of passwords.

The following sequence diagram breaks down the general flows of events stated above, and in the context of a DictionaryAnalyser:

NOTE The other Analysers are instantiated in a similar fashion in #getRequiredAnalsyers() and hence omitted to the make the diagram less congested.

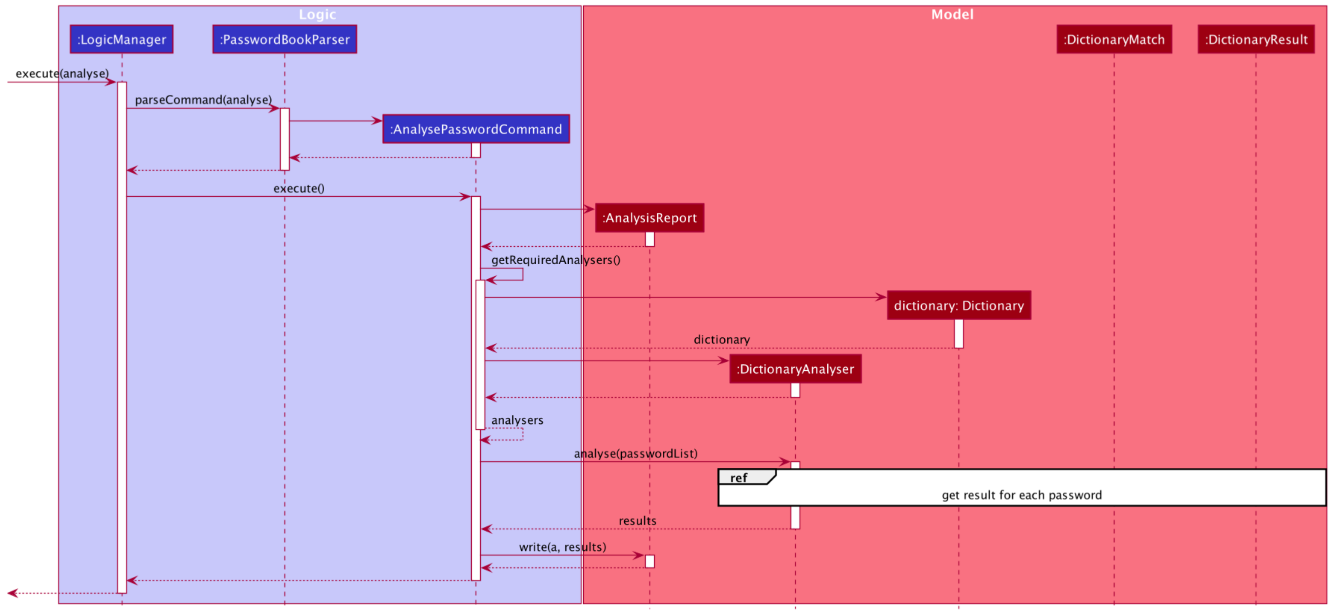


Figure 10. Sequence diagram depicting the flow of analysing the list of passwords, in the context of DictionaryAnalyser.

In the following discussion, we will be reviewing how Results are produced for each password, in the context of the DictionaryAnalyser.

In the case of the DictionaryAnalyser#analyse(), every password is checked to see if it contains any subsequence that is listed as a commonly used password in the instantiated Dictionary. This is done in the internal method #getAllMatches(password) within #analyse().

NOTE The Dictionary is an object that maps commonly used passwords to their ranking in terms of how commonly used they are. (e.g. "123456" is mapped to rank 1 in the Dictionary because it is the most commonly used password)

If there exists a subsequence that tests positive when looked up against the Dictionary , a DictionaryMatch is created to note down the details of the subsequence.

NOTE A Match is an object that is created to encapsulate the details of any matches found when analysing a given password. Each Analyser will create it’s respective Match when needed.

Thus, we see that the method #getAllMatches(password) will return a list of all DictionaryMatch found for a given password.

As long as a DictionaryMatch is found, then the password has failed the analysis. A DictionaryResult with the attribute ResultOutcome.FAIL will be created for that password. This process is repeated for every password. Following, the list of DictionaryResult are returned to be compiled and written by the AnalysisReport.

The following sequence diagram depicts the flow of events mentioned above:

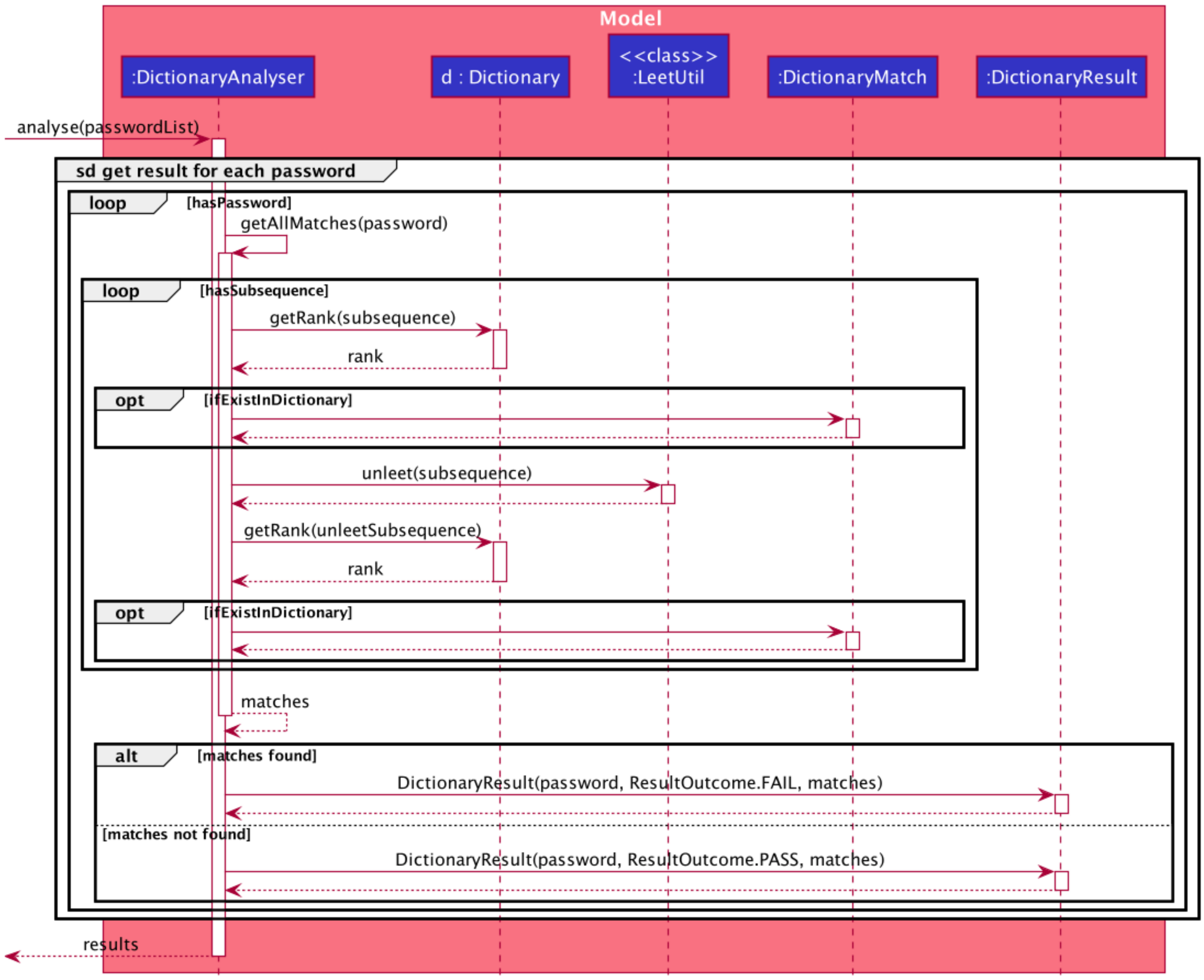


Figure 11. Sequence diagram depicting the flow of getting all Match objects for a given password.

It is also worth mentioning that DictionaryAnalyser is capable of identifying commonly used passwords even in leet-ed variations (e.g. p@5sw0rd).

This is made possible because of the method #LeetUtil.translateLeet(), which is a recursive algorithm designed to return all possible un-leet variations, given a leet-ed password. The details of the algorithm can be found here:

Design Considerations

Aspect: How analyse/analyse strong executes

Alternative 1 (Current choice)	Alternative 2
Always analyse the entire list of Password objects for every "analyse" command, even if the list of Passwords was unchanged. *Pros:* Easy to implement, not required to check state if the current list of Passwords has been modified. *Cons:* May have performance issues in terms of speed of the programme.	**Save in memory the result produced by the Analyser objects, and update result upon modification of list of Passwords (e.g. new passwords added/ passwords deleted.)** *Pros:* Performance of programme will be a lot faster and efficient. *Cons:* Hard to implement. Have to keep track of state of the list of Password objects and check if the list has been modified from the last time they were analysed.
Why did we choose Alternative 1: From a more practical point of view, users are not expected to constantly want to keep analysing their passwords, so it may be a waste of memory to constantly save the results, as per Alternative 2. Also, considering the fact that each password is capped at a length of 25, time performance will not be affected significantly when analysing each password.	