Derrick Chua - Project Portfolio

Project: ABC Business Contacts

ABC Business Contact(ABC) is a free and open-source desktop Business Contact Management application that allows the user to store a local database of contacts, and track their appointments. There are a wide variety of commands to optimise organisation and management of a user's contact information. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java.

Code contributed: [Functional code] [Test code] [Reused code]

Enhancement Added: Sync

External behavior

Start of Extract [from: User Guide]

Logging in to Google Contacts: `login`

Command Name: `login`
Shorthand Alias: `li`
Function: Logs in to Google Contacts
Format: `login`

> **NOTE** | It is mandatory to execute this command before running `sync`

If you would like to login to Google Contacts:

1. Type in
   `>> login`
   (See Figure 4.17.1)

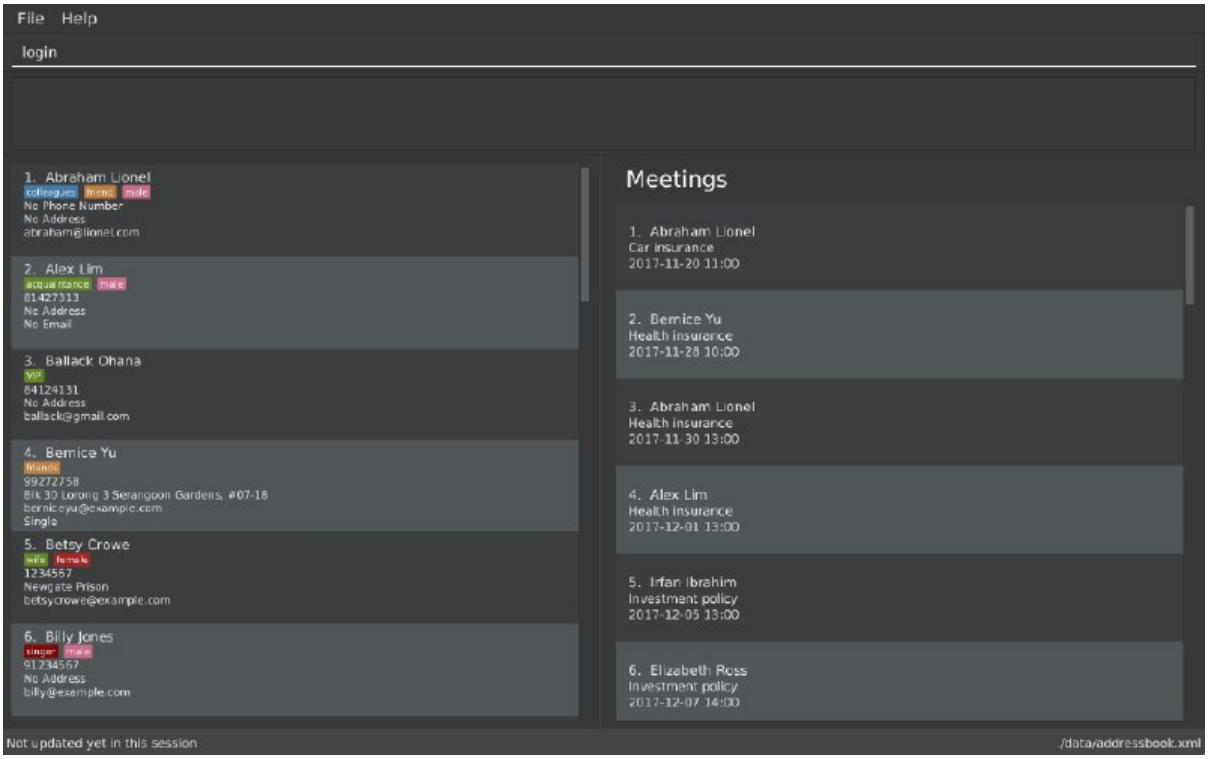


Figure 4.17.1

2. Press `Enter` and your default browser should open a login window
(See Figure 4.17.2)

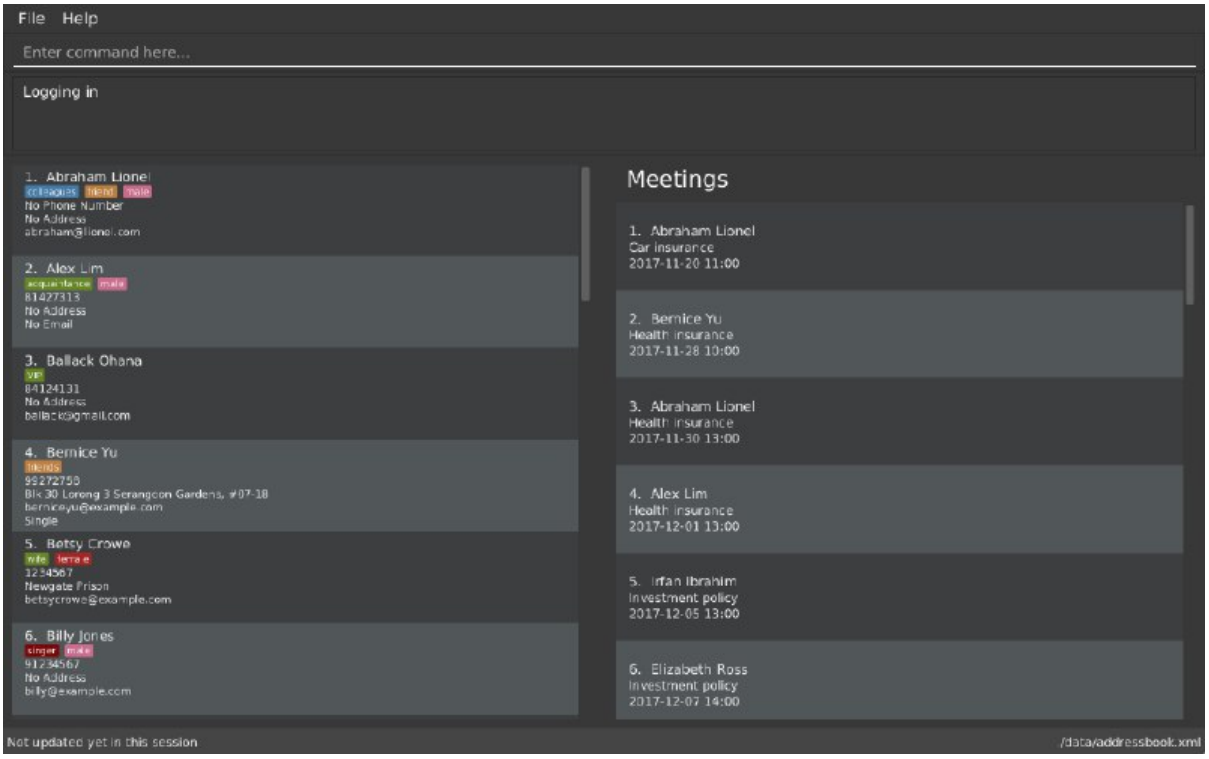


Figure 4.17.2

3. Enter your login details and press Next
(See Figure 4.17.3)

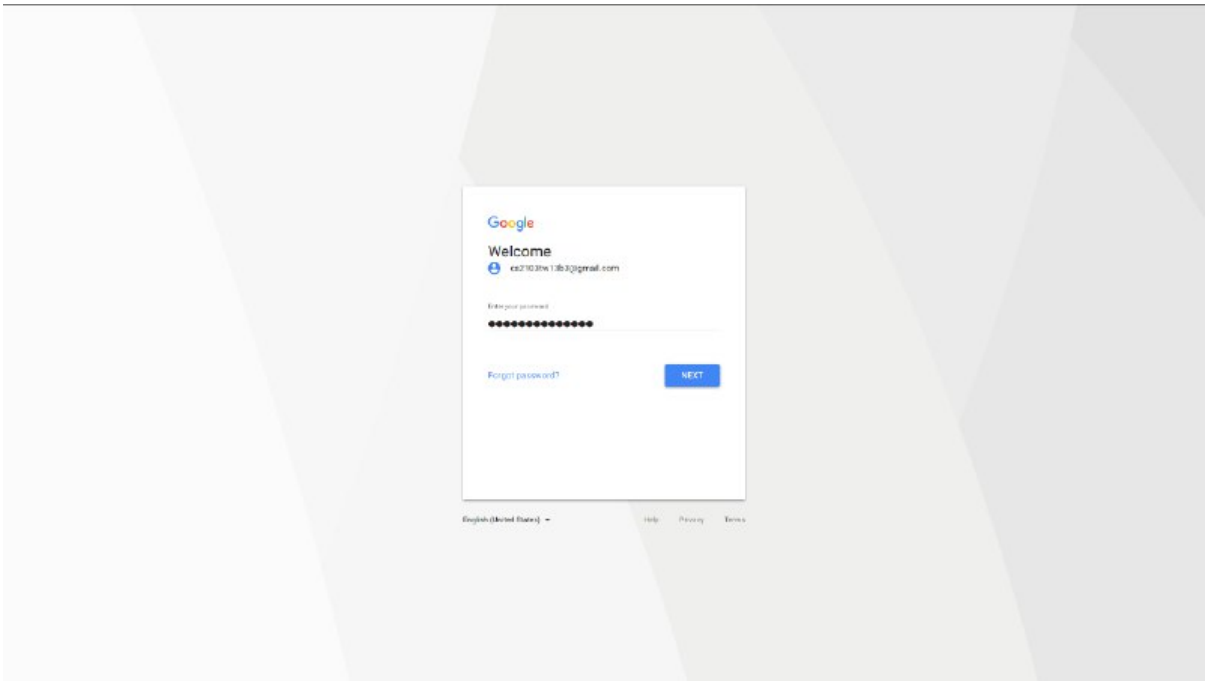


Figure 4.17.3

4. Allow **ABC** to access your Google Contacts information
(See Figure 4.17.4)

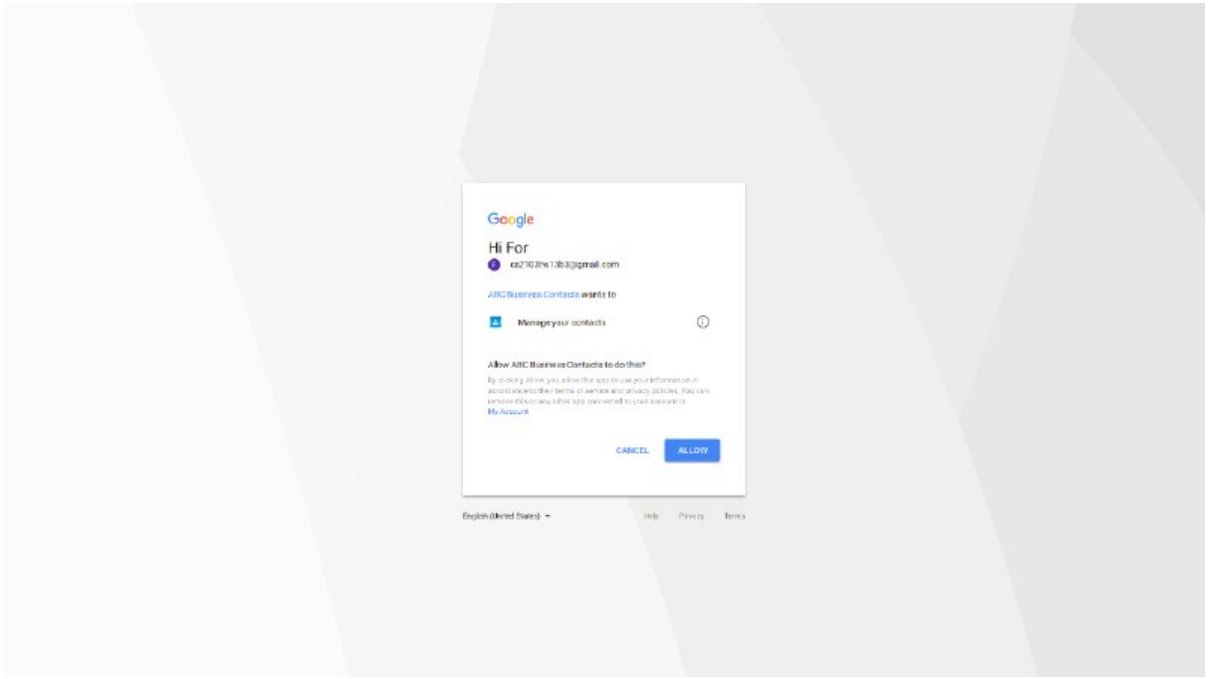


Figure 4.17.4

Synchronising with Google Contacts : `sync`

Command Name: `sync`
Shorthand Alias: `sy`
Function: Synchronises your contacts with Google Contacts after authentication
Format: `sync`

NOTE	A browser is necessary for logging in to Google
NOTE	You have to run the `login` command before you can run `sync`
NOTE	Synchronisation currently does not support Google Contacts with multiple email addresses, phone numbers, and/or addresses.

You can easily synchronise your **ABC** contacts with Google Contacts through the following steps:

1. Type in
   `>> sync`
   (See Figure 4.18.1)

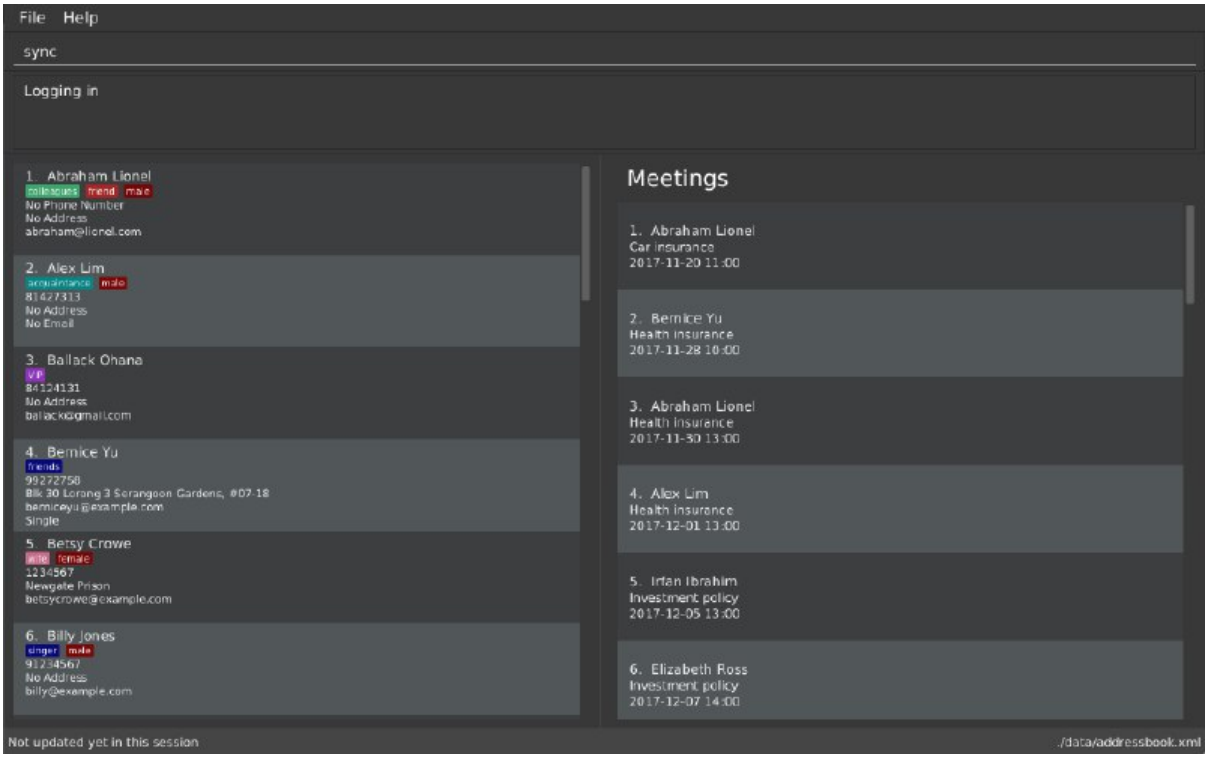


Figure 4.18.1

2. Your contacts are now synchronised
   (See Figure 4.18.2 and 4.18.3)

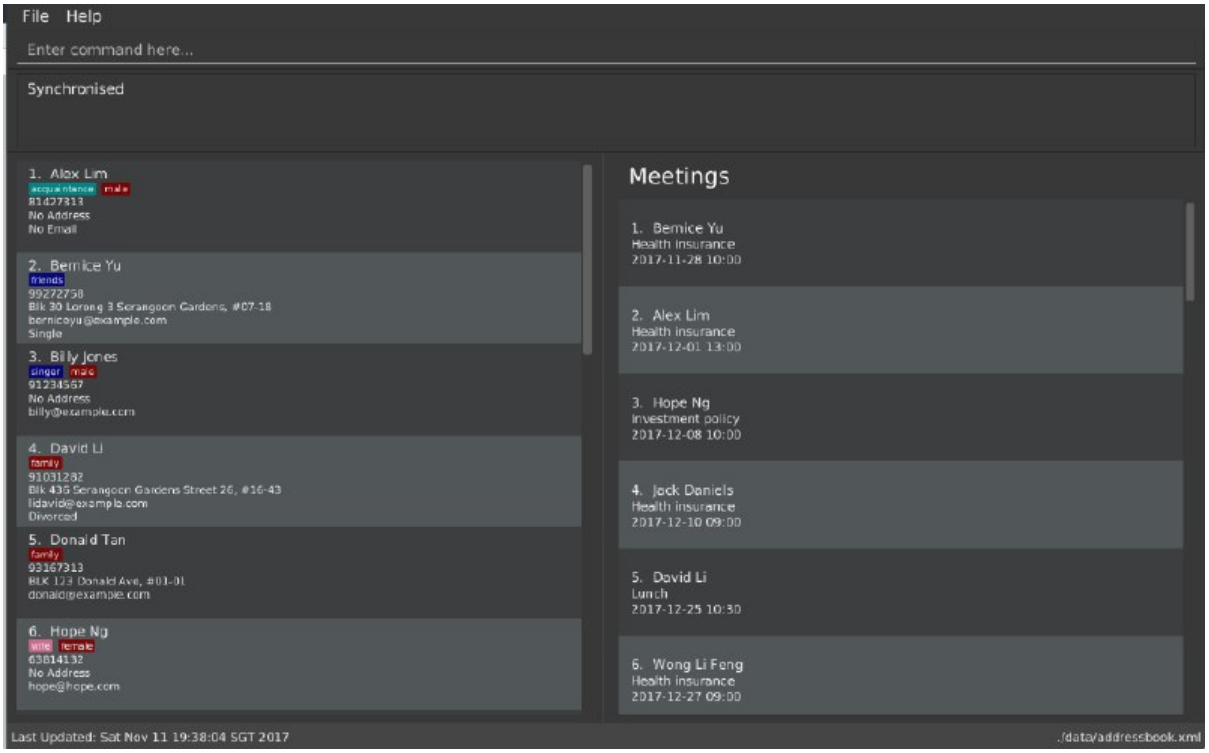


Figure 4.18.2

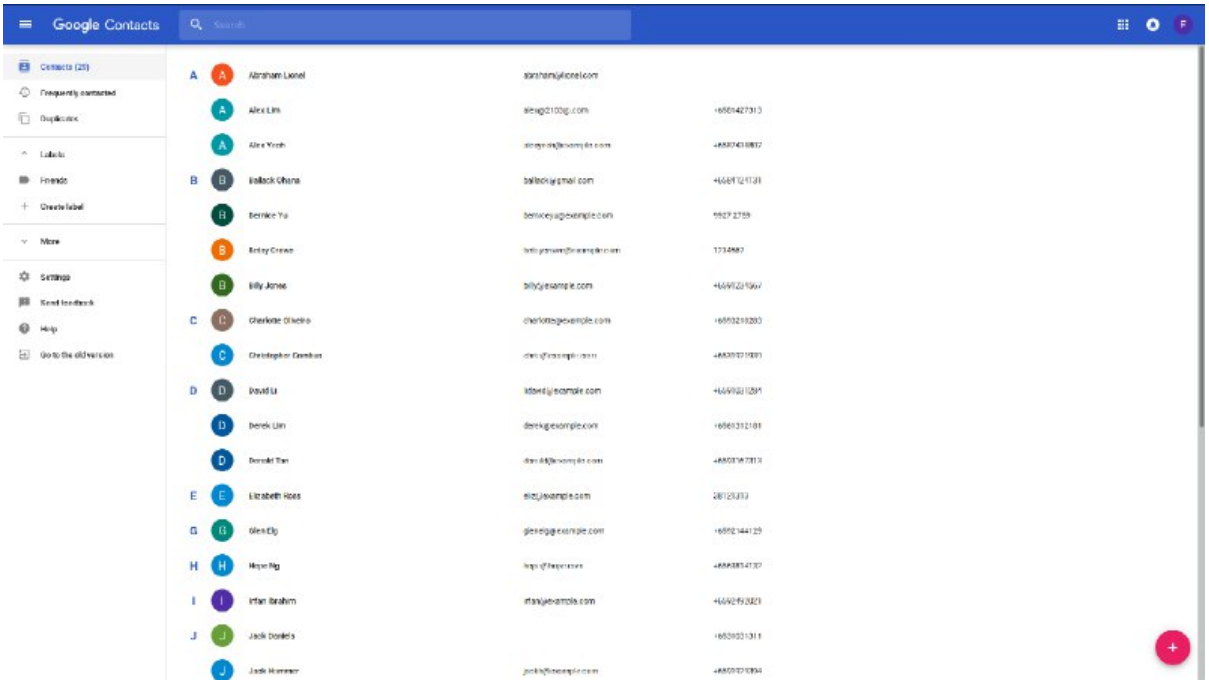


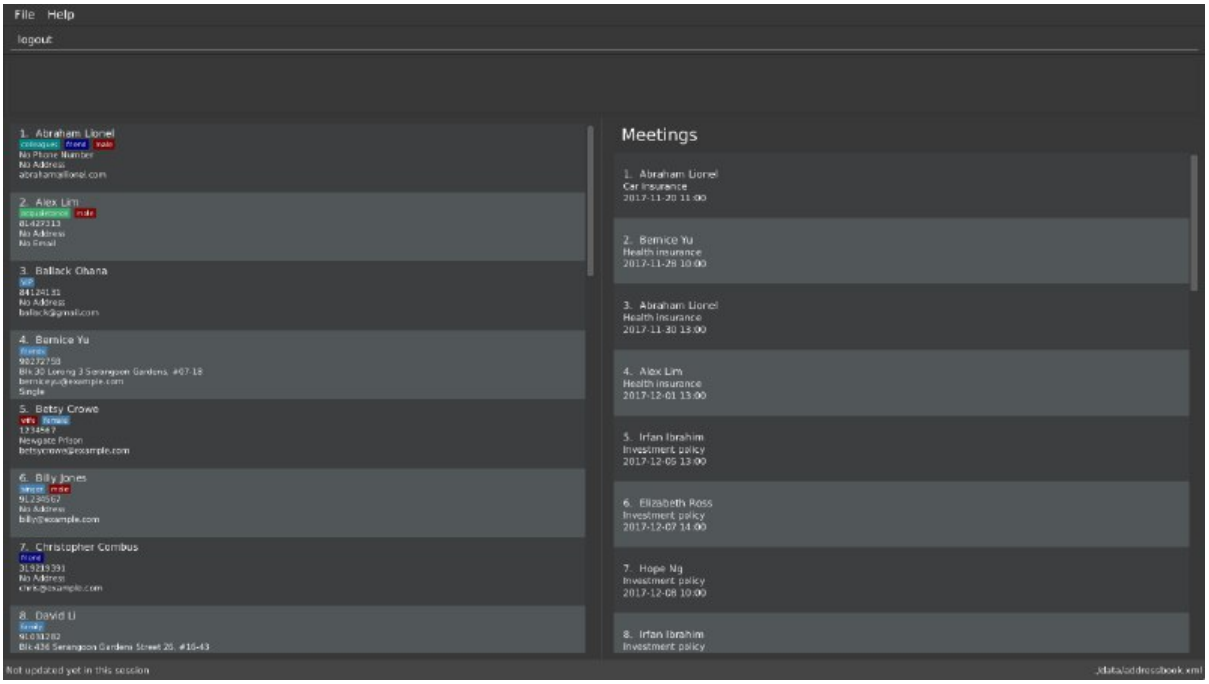
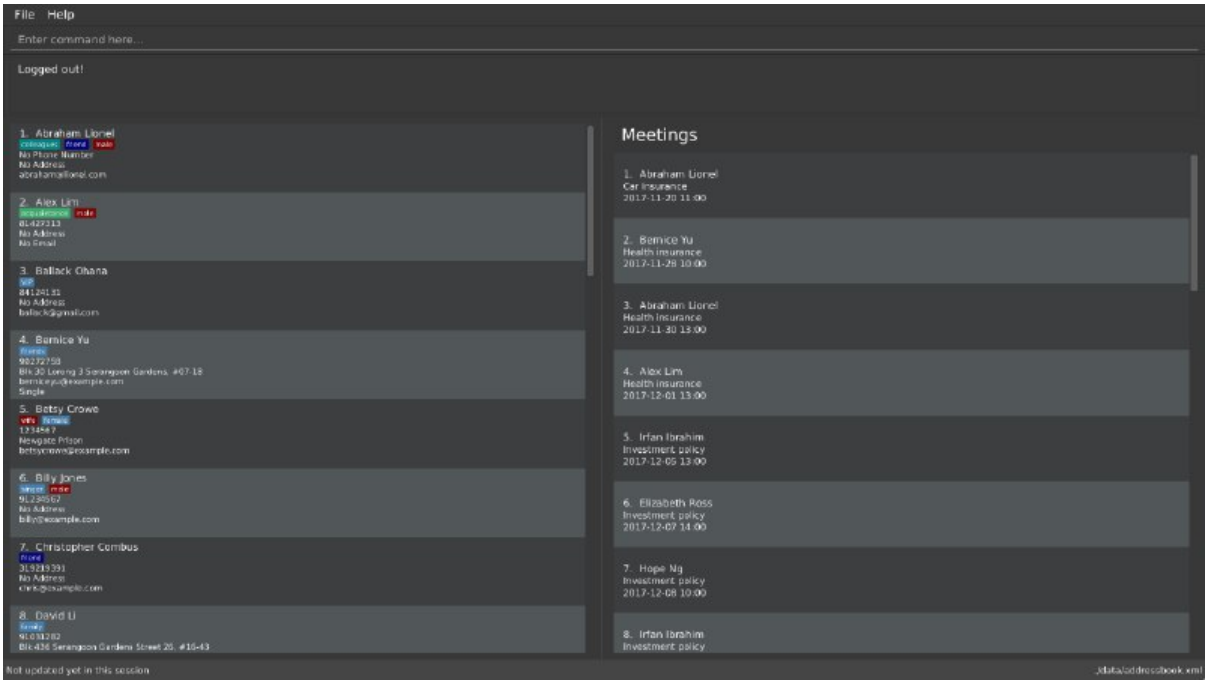
Figure 4.18.3

Logging out of Google Contacts : `logout`

Command Name: `logout`
Shorthand Alias: `lo`
Function: Logs out of your linked Google Account after you have logged in
Format: `logout`

NOTE	You should only use this command if you would like to log out of your linked Google account

You can log out of your linked Google Account by doing the following:

1. Type in
   `>> logout`
   (See Figure 4.19.1)
   
   Figure 4.19.1
2. You are now logged out
   (See Figure 4.19.2)
   
   Figure 4.19.2

End of Extract

Justification

People use their smartphones mainly to access their contacts, and adding synchronisation with Google Contacts provides a way for the modern user to use the ABC Business Contacts application without the hassle of importing and exporting his contacts constantly.

Implementation

Start of Extract [from: Developer Guide]

Synchronisation with Google Contacts

Mechanism

Authentication and bi-directional synchronisation of data with a user's Google Contacts is done via the `sync` command, which is a subclass of `Command`. This command works in conjunction with the Google Client and People API.

A `PeopleService` instance is required and obtained via the `LoginCommand` before synchronisation is possible. `PeopleService` is then used to perform Create, Read, Update, and Delete (CRUD) operations on the user's Google Contacts, which is used in `SyncCommand`. The four primary methods in `SyncCommand` are `checkContacts`, `updateContacts`, `importContacts` and `exportContacts`.

The sequence diagram for the command can be seen below, in Figure 3.5.1.1:

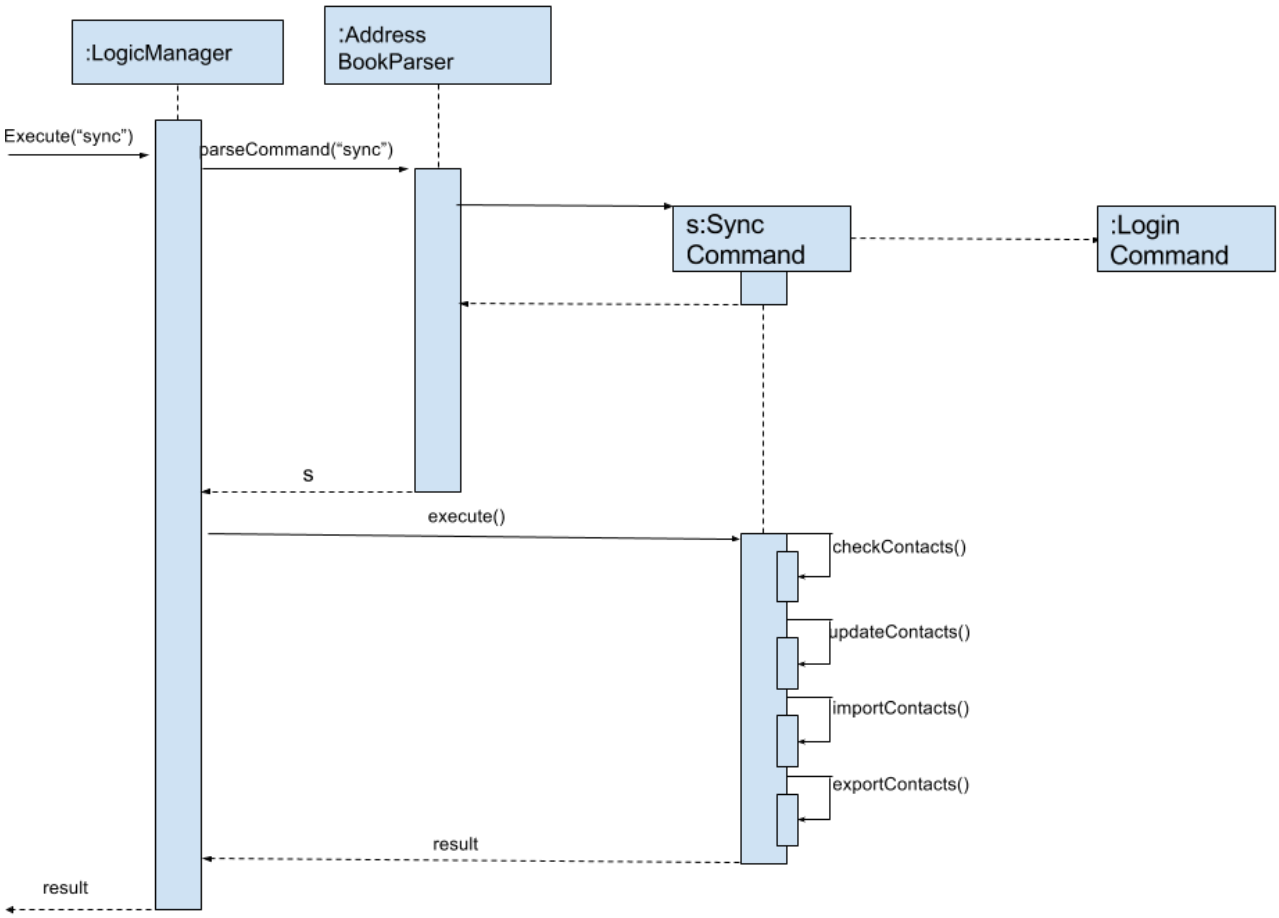


Figure 3.5.1.1: Sync Command Sequence Diagram

Methods

SyncCommand

Below is the implementation of SyncCommand. Upon execution, the command checks if a PeopleService instance has been instantiated, and throws a `CommandException` if it has not. It then runs the `initialise` method, which preprocesses the **ABC** and Google Contacts data, before performing the 4 main functions, `checkContacts`, `updateContacts`, `importContacts` and `exportContacts`.

```
public class SyncCommand extends Command {
  //...variables, constructor, other methods

    @Override
    public CommandResult execute() throws CommandException {

        if (clientFuture == null || !clientFuture.isDone()) {
            throw new CommandException(MESSAGE_FAILURE);
        } else {

            syncedIDs =  (loadStatus() == null) ? new HashSet<String>() : (HashSet)
loadStatus();

            try {
                List<ReadOnlyPerson> personList = model.getFilteredPersonList();
                initialise();
                checkContacts();
                updateContacts();
                exportContacts(personList);
                if (connections != null) {
                    importContacts();
                }

                saveStatus(syncedIDs);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return new CommandResult(String.format(MESSAGE_SUCCESS));
    }
}
```

Checking and updating of contacts

A `syncedIDs.dat` file is maintained to keep track of the contacts that have been synchronised currently, as each link between an **ABC** and a Google contact consists of an unique ID. `checkContacts` ensures that contacts on both ends still exist, and removes the link if either of them no longer exist. `updateContacts`, on the other hand, compares linked contacts, and if there is a difference, it updates the contact that has an older timestamp. Below is the implementation of the 2 functions respectively.

```
public class SyncCommand extends Command {
  // variables, constructor and other methods
  private void checkContacts() throws Exception {
        List<ReadOnlyPerson> personList = model.getFilteredPersonList();
        for (ReadOnlyPerson person : personList) {
            String id = person.getId().getValue();

            if (!hashGoogleId.containsKey(id)) {
                logger.info("Deleting local contact");
                model.deletePerson(person);
                syncedIDs.remove(id);
                continue;
            }

        }
  }

      private void updateContacts() throws Exception {
        List<String> toRemove = new ArrayList<String>();
        for (String id : syncedIDs) {
            seedu.address.model.person.ReadOnlyPerson aPerson;
            Person person;
            // Checks whether person and aPerson still exists, mainly for defensive
programming
            // We check the last updated times for both contacts
            if (compare < 0) {
                // We update the remote contact
            } else if (compare > 0) {
                // We update the local contact
            }
        }
    }
}
```

Importing and exporting of contacts

We then move on to importing of new Google Contacts, and exporting of new **ABC** contacts to Google servers. To achieve this, we iterate through all Google Contacts and **ABC** contacts respectively, and import or export them accordingly if they are not linked with an ID yet.

```
public class SyncCommand extends Command {
  // variables, constructor and other methods
    private void importContacts () throws IOException {

        for (Person person : connections) {
                String id = person.getResourceName();
                String gName = retrieveFullGName(person);
                if (!syncedIDs.contains(id)) {
                    if (!hashName.containsKey(gName)) {
                        // We import the contact if there is no contact of a similar
name
                    } else if (hashName.containsKey((gName))) {
                        seedu.address.model.person.ReadOnlyPerson aPerson =
hashName.get(gName);
                        if (equalPerson(aPerson, person)) {
                            //We link the 2 contacts if they have the same details
                        } else {
                            // We can safely import the contacts as they have
different details
                        }
                    }
                }
        }
    }

    private void exportContacts (List<ReadOnlyPerson> personList) throws Exception {
        for (ReadOnlyPerson person : personList) {
            if (person.getId().getValue().equals("")) {
                if (hashGoogleName.containsKey(person.getName().fullName)) {
                    // We can safely export the contact as there is no one with a
similar name
                } else if (hashGoogleName.containsKey(person.getName().fullName)) {
                    // We check if the person is identical, and link them if they are
                    Person gPerson = hashGoogleName.get(person.getName().fullName);
                    if (equalPerson(person, gPerson)) {
                        // We link the similar contacts
                    } else {
                        // We can safely export the contact as their other details are
not similar
                    }
                }
            }
        }
    }
}
```

Design Considerations

Aspect: It is difficult to keep track of which contacts have been synchronised
Alternative 1 (current choice): An unique ID is assigned to each contact, and a global syncedIDS HashTable is stored
Pros: It is easy to link/unlink synchronised contacts and keep track of the IDs that are in use.
Cons: Requires a persistent data file to be stored for synchronised IDs.

Alternative 2: Use an ID field for each contact, but keep no global data file
Pros: Less resources and room for error
Cons: Requires more time for synchronisation, and it will be difficult to remove Google Contacts that have been deleted locally.

Aspect: Authorisation cannot be performed synchronously due to the Google People library
Alternative 1 (current choice): Introduce a new command `login` which is asynchronous, while `sync` remains synchronous
Pros: `Model` can be updated as we remain on the application thread
Cons: It is difficult to control the number of open threads, which can impact system resources, and we have to run `login` before `sync`

Alternative 2: We implement synchronous usage of the Google Peopl library
Pros: No threading and hence complication is required.
Cons: It is extremely difficult to achieve this.

End of Extract

Enhancement Added: Add command `note`

External behavior

Start of Extract [from: User Guide]

Making a note: `note`

Command Name: `note`
Shorthand Alias: `n`
Function: Inserts a NOTE for the contact specified by INDEX in the **ABC**
Format: `note INDEX [NOTE]`

NOTE	Each contact can have at most 1 note
TIP	NOTE can be blank to delete existing note, i.e. `note 1`

If you want to add a note for a contact:

1. Locate the contact and take note of its index
2. Type in your desired INDEX and NOTE
   `>> note 1 This is an important note`
   (See Figure 4.13.1)

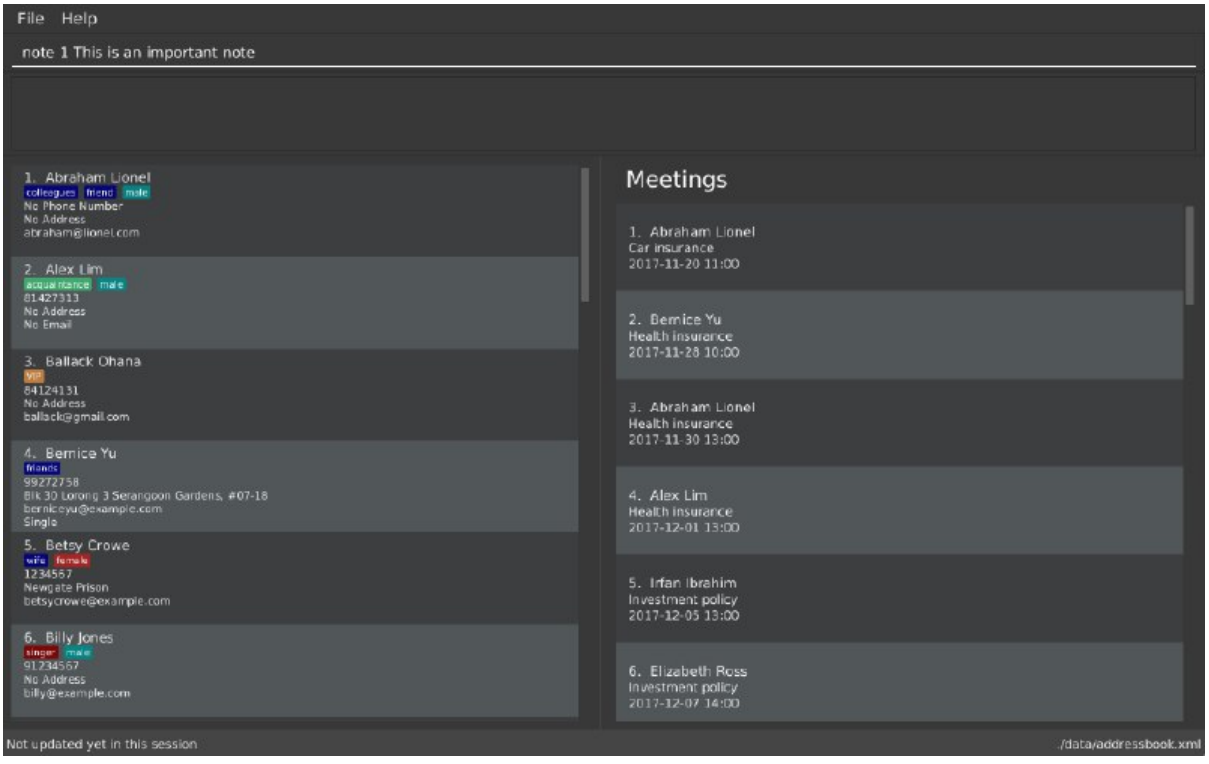


Figure 4.13.1

3. Press `Enter` and your note should appear as the last row in your contact's details

(See Figure 4.13.2)

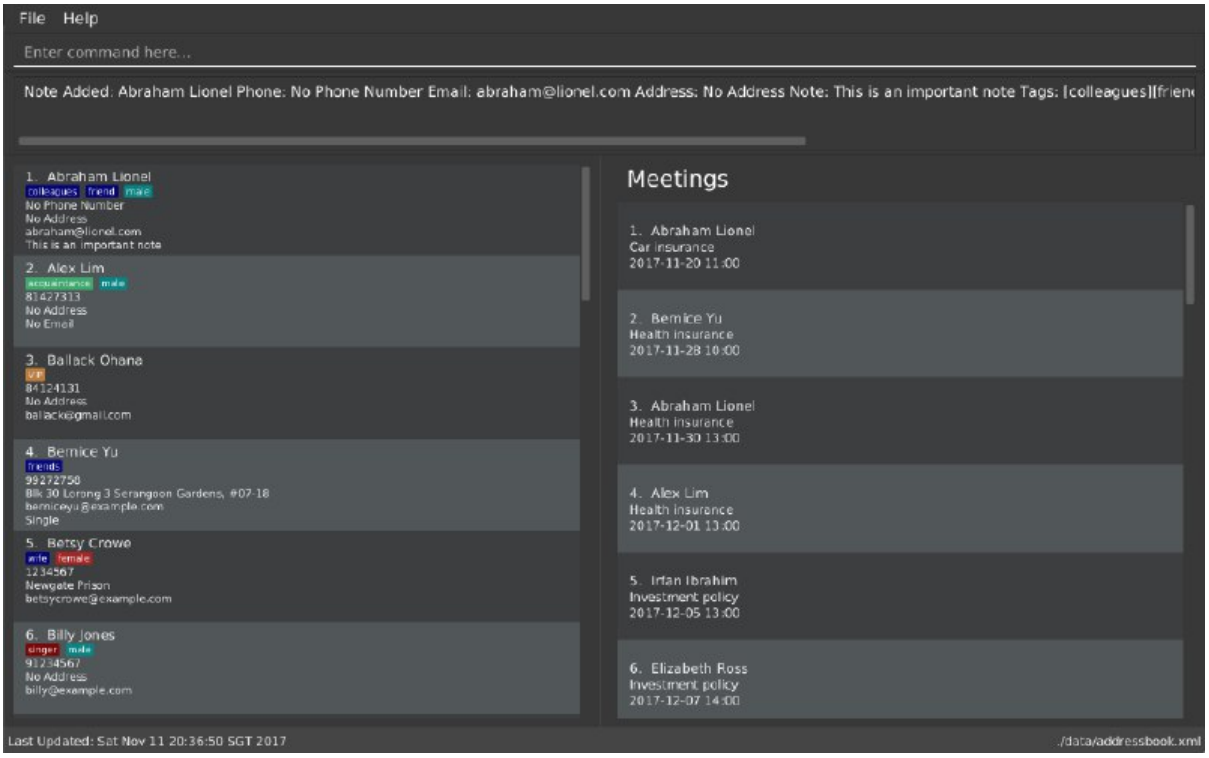


Figure 4.13.2

Here are some other ways to change your ABC contact's note:

- `>> note 2`
  Removes the existing note from the 2nd person
- `>> n 3 This is a note`
  Changes the 3rd contact's note to "This is a note"
- `>> n 3`
  Removes the existing note from the 3rd person

End of Extract

Justification

Adding a `note` command enables the user to add notes for his contacts quickly should he need to jot something down.

Enhancement Added: Optional fields for `add`

External behavior

Start of Extract [from: User Guide]

Adding a person: `add`

Command Name: `add`
Shorthand Alias: `a`
Function: Adds a person to **ABC**
Format: `add n/NAME [p/PHONE_NUMBER] [e/EMAIL] [a/ADDRESS] [t/TAG]…`

TIP	A person can have any number of tags (including 0)
TIP	Parameters can be in any order e.g. `n/NAME p/PHONE_NUMBER`, `p/PHONE_NUMBER n/NAME` are equivalent

If you want to add a new contact to your **ABC**:

1. Type in
   `>> add n/Betsy Crowe t/friend e/betsycrowe@example.com a/Newgate Prison p/1234567 t/criminal`
   (See Figure 4.2.1)

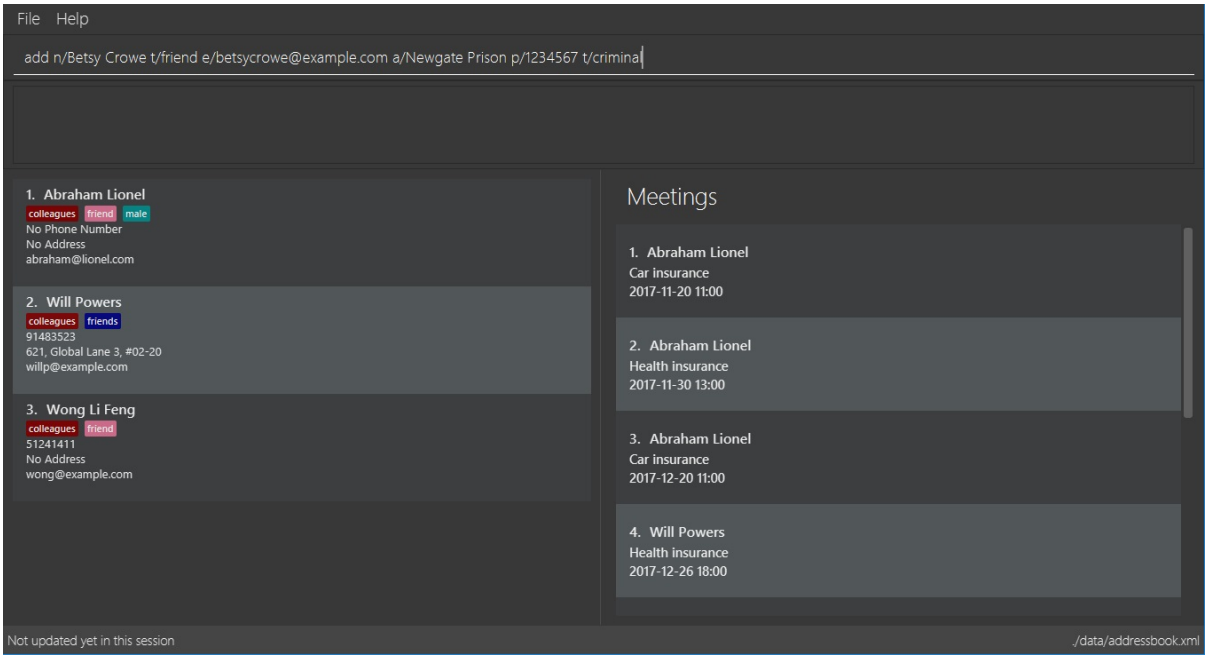


Figure 4.2.1

2. Press `Enter` and you should see that a new contact has been added
   (See Figure 4.2.2)

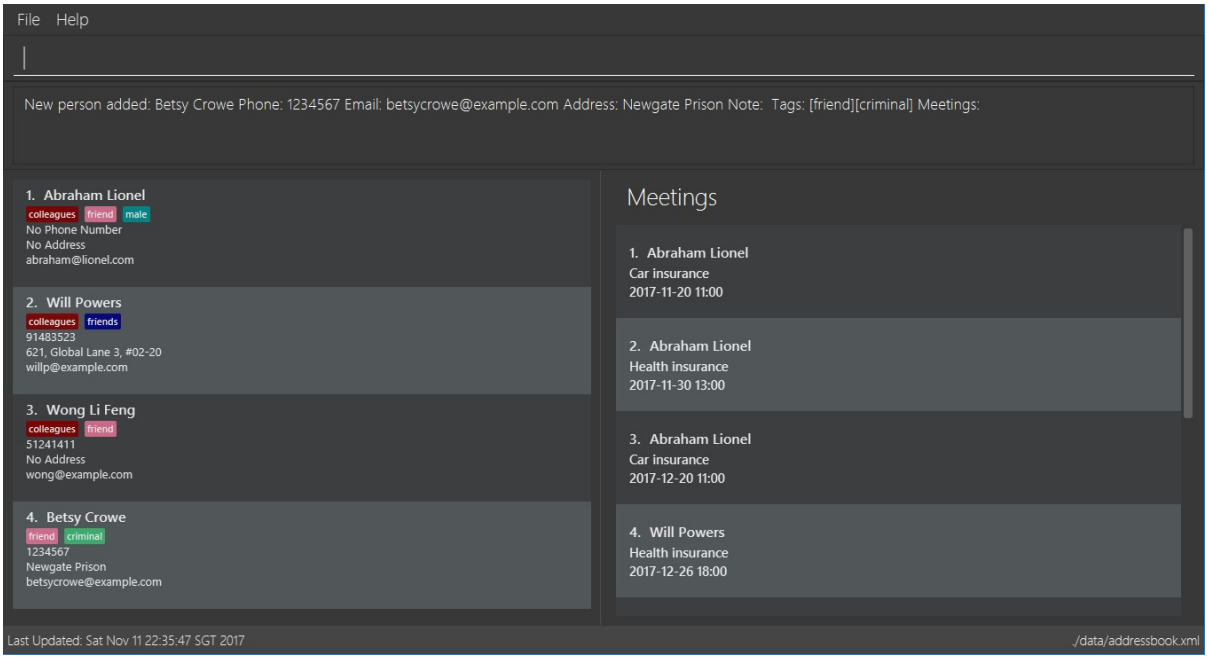


Figure 4.2.2

Here are some other ways you can add contacts:

- `>> add n/John Doe p/98765432 e/johnd@example.com a/John street, block 123, #01-01`
- `>> add n/Betsy Crowe t/friend e/betsycrowe@example.com a/Newgate Prison p/1234567 t/criminal`
- `>> add n/Jack Daniels`
- `>> a n/John Watson p/83331122 e/johnw@example.com a/John Avenue, block 2, #01-01`
- `>> a n/Dave`

End of Extract

Justification

Not all details are always known about a contact, but it should not prevent a user from adding him/her.

Enhancement Proposed: Add different views for Meetings

A command `view` is added, where the user can specify how he wants to view his meetings, i.e. by week, month or year

Justification

Users should be able to categorise their contacts by period, i.e. week, month or year

Other contributions

- Implement tag colours (Pull request #1)
- Fix white PersonListPanel CSS (Pull request #75)
- Discovered bugs during acceptance testing (Issue #125)