Niu Yunpeng - Project Portfolio

Project: BoNUS

BoNUS is a desktop personal organizer application dedicated to NUS students to carry out various tasks such as storing contacts, scheduling for upcoming events, timetable planning as well as a calendar to better organise their campus life.

This project is the result of a semester-long software engineering module at NUS. It is built by my teammates and me, as listed here. Most of the backend codes are written in Java, while JavaFX is adapted as the frontend framework.

Code contributed:

- Functional code
- Test code

Enhancement Added: Generic Property

External behavior

Adding a customize property : `(cfg)config --add-property`

Adds a new customize property field available to all persons or events.
Format: `(cfg)config --add-property s/SHORT_NAME f/FULL_NAME [m/MESSAGE r/REGULAR_EXPRESSION]`

- This command does not add a new property to a specific person. Instead, it defines a property that will be available to all persons/events.
- The short name `s/` and full name `f/` of the new property are compulsory, while the constraint message `m/` and regular expression for validation `r/` are optional. However, `m/` and `r/` must come together, i.e. a regular expression must be accompanied by a constraint message, which will be shown when the validation fails.
- If constraint message and regular expression are not specified, the default would be `[^\s].*`, which mean the value of this property cannot be blank.
- Short name is the identity (primary key) of all properties. Thus, the short name must be unique. The command will fail if there is an existing property with the same short name.
- The given regular expression must use legal syntax. It will be checked by the Pattern.compile method.

Example:

- `config --add-property s/ag f/age`
- `config --add-property s/b f/birthday m/Birthday should be in the format of DD/MM/YYYY r/[^\s].*`

- Short name is used as the prefix for `add`/`addE` and `edit`/`editE` commands.
- Full name is used to display the name of each property on the right panel (to show details of the selected person, use `select` command).
- Constraint message is the string that will be shown in result display box when the

> input value for this property is invalid.
> - Regular expression is used to check whether the input value for this property is valid.

Justification

Our application **BoNUS** will not be able to stand out from all the similarproducts if we simply provide users with a few additional fields to store the information of their contacts. Instead, to suit the various needs from users, we have to provide the flexibility to allow users to define customize properties on their own.

Users can use the `config --add-property` command to define as many new properties as they want, after which they could set these properties when they use the `add` command to add a new contact.

In the current design and implementation, the `PropertyManager` class is used to store and manage the meta-data of all properties. Default properties like `Name`, `Phone`, `Email` and `Address` are pre-loaded into `PropertyManager`, while user-defined properties will be added into `PropertyManager` by `config --add-property` command.

Implementation

Generic property

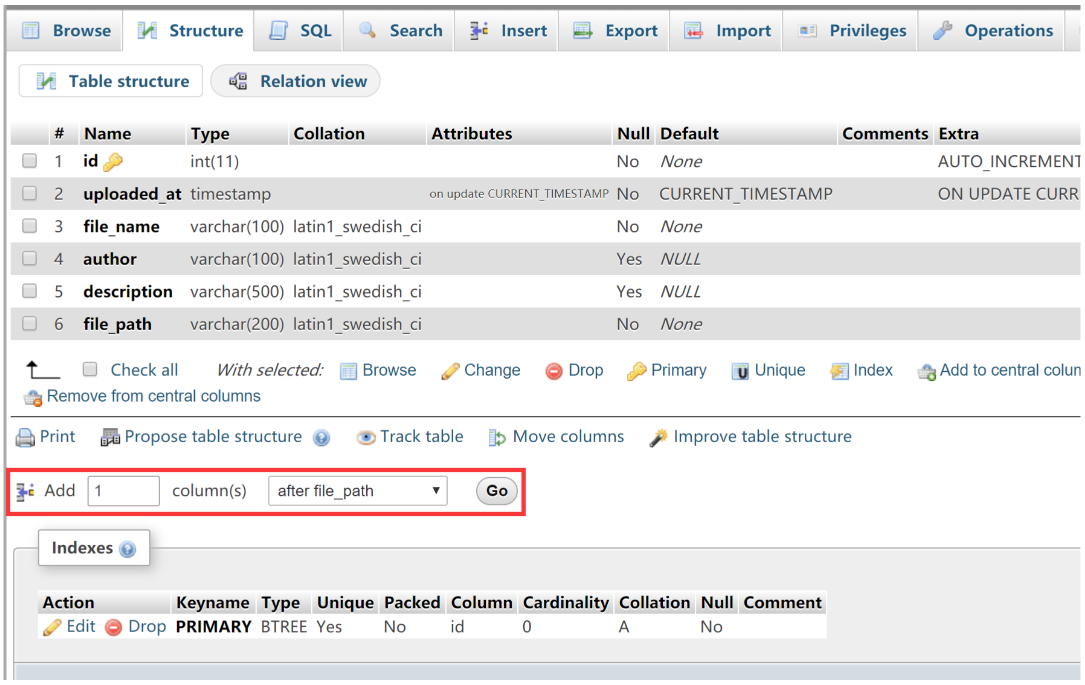
We are using a generic `Property` class to support **arbitrary field** feature.

Inspiration

Users should not be limited to the provided four fields, i.e. `Name`, `Email`, `Phone`, `Address` (we are talking about the contact component here, of course one more `DateTime` for event component). They should have the freedom to enter all kinds of information about their contacts apart from the pre-defined ones.

Through a brief product survey on other existing similar software in the market, we found that they usually ask users to type all other information in the *so-called* `Description` or `Details` field. This is not a good design because doing so will make the information stored messy. The application is used to organize personal information conveniently. It is a major drawback if the data are not stored (and thus presented to users) in a well-organized way.

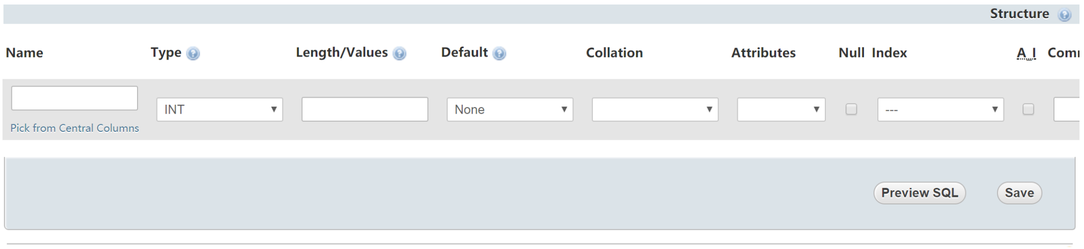
Learning from many modern database implementations, we should think of the data as two tables: one for **contacts** and the other one for **events**. Each table is composed of many rows and many columns. A single contact/event is one row, while all their different properties/fields are the columns (as can be seen from Figure 4.3.1.1 and 4.3.1.2).



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	id	int(11)			No	*None*		AUTO_INCREMENT
☐	2	uploaded_at	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP		ON UPDATE CURR
☐	3	file_name	varchar(100)	latin1_swedish_ci		No	*None*		
☐	4	author	varchar(100)	latin1_swedish_ci		Yes	*NULL*		
☐	5	description	varchar(500)	latin1_swedish_ci		Yes	*NULL*		
☐	6	file_path	varchar(200)	latin1_swedish_ci		No	*None*		

Action		Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit	Drop	PRIMARY	BTREE	Yes	No	id	0	A	No	

Figure 4.3.1.1 : Data Table View of phpMyAdmin (a MySQL visualization tool)



Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I	Comı
Pick from Central Columns	INT		None			☐	---	☐	

Figure 4.3.1.2 : Add New Column in phpMyAdmin

Design Consideration

Aspect: Where to store the "*metadata*" of different properties (short name, full name, regular expression, etc.)
Alternative 1 (current choice): Create a new class `PropertyManager` in Figure 4.3.2.1
Pros: Efficient (there is only one copy) and easy for future development since it is centralized.
Cons: Requires major change to `Model` component and `Storage` component.
Alternative 2: Store these data along with each specific property class, like `Name`, `Email`
Pros: Able to adapt the current implementation of `Model` component.
Cons: Hard to implement `AddPropertyCommand`, and difficult to manage as the project grows larger.

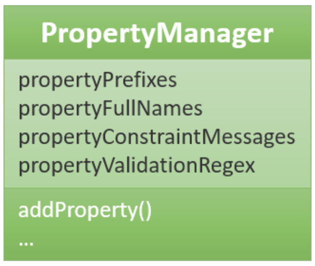


Figure 4.3.2.1 : Class Diagram for `PropertyManager`

Implementation Outline

1. Create a more general class to capture the common patterns among all columns (all different fields/properties): according to the basic OOP concept, a more generic class should become the superclass `Property`; then, other more specific classes (like `Name`, `Email`, `Phone`, etc.) can inherit

from it. This design reduces a lot of code duplicates.

2. Find a way to store the metadata of all columns (fields/properties): in popular SQL database implementation, they usually have a separate database reserved for the database server system itself. We must store similar information somewhere as well. Thus, we create a `PropertyManager` to store these "metadata", including short names, full names, constraint messages and regular expressions used for input validation. They are all `static` variables because there should only be one copy of these "metadata". It will waste a lot of resources if we store these "metadata" with each instance of the `Property` class.
3. *Pre-loaded properties*: Things like `Name`, `Email` and `Phone` are widely used. They should ship with the application and users do not need any additional setup steps to use them.
4. Add new customize properties: advanced users should be provided with a command (`config --add-property`) to add their own customize fields (as shown in Figure 4.3.3.1). They should have the freedom to arbitrarily choose things like short name, full name, etc. They can easily add/edit these properties of each contact stored in the application, just like the *pre-loaded* ones.

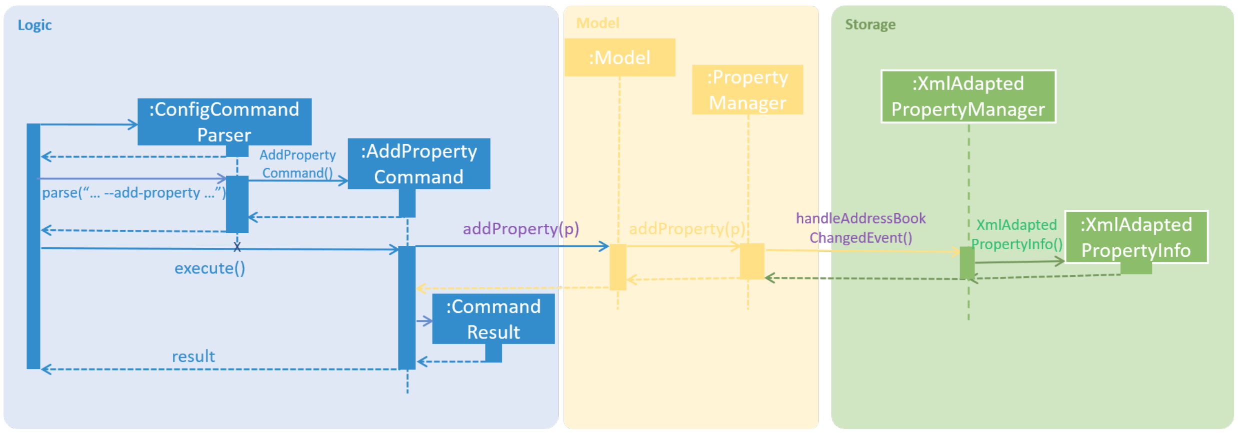


Figure 4.3.3.1 : Sequence Diagram for Adding a Customize Property

Enhancement Added: Import NUSMods Timetable

External behavior

From NUSMods URL

(Exclusive feature for NUS students)

The **BoNUS** team understands that NUSMods has become an indispensable school timetable builder for almost all students at NUS. Thus, you are definitely allowed to import your timetable from NUSMods to the **BoNUS** application.
Format: `(i)import --nusmods YOUR_NUSMODS_URL`

- The URL provided must be complete and should begin with `http(s)://nusmods.com/timetable/`.
- Directly copy from the address bar of your browser. Do **NOT** use the short URL generated by the *Sharing Timetable* feature provided by NUSMods.
- Final examinations for all modules in your NUSMods timetable will be automatically added as events to the application.

Example:

- `import --nusmods https://nusmods.com/timetable/2017-2018/sem1?CS2103T[TUT]=C01`

- Make sure you have stable Internet connection when using this command.
- You may need to wait for a while as the application is retrieving information from NUSMods.

Justification

This feature addresses a specific need of NUS students, since they are our target users.

For university students, the lessons and exams reflected on their school timetable account for a large part of their schedule. When they are using **BoNUS**, it is very likely they want to put all these activities into the application as upcoming events. However, it becomes very tedious to do so manually.

Most of the NUS students are currently using NUSMods as their school timetable builder. Their NUSMods timetable contains information about all their lessons and exams. Thus, it would be very convenient if users could directly import their NUSMods timetable into **BoNUS** in one step.

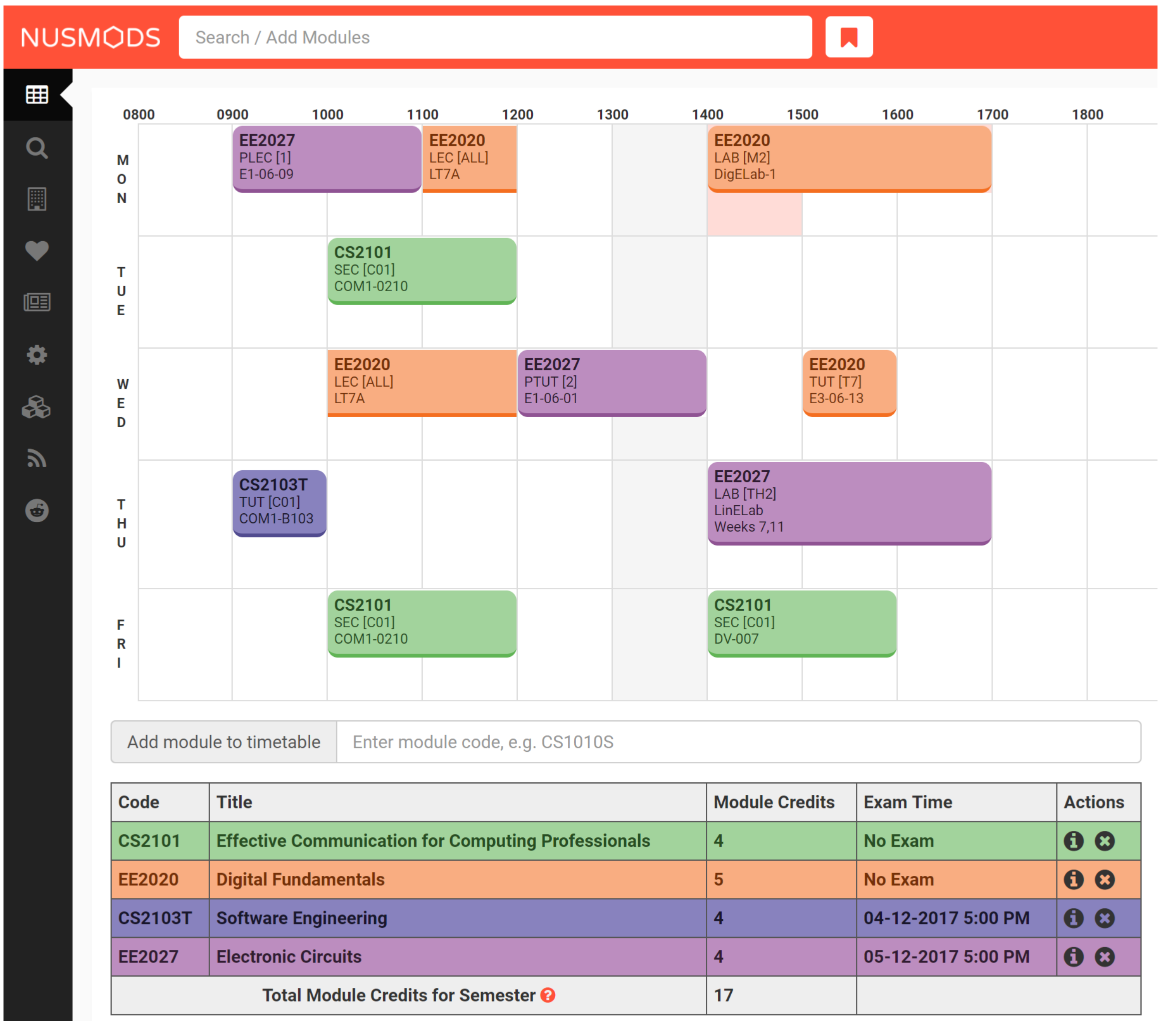
Implementation

Import timetable from NUSMods

We implement an `ImportNusmodsCommand` to help users directly import their NUSMods timetable to **BoNUS** by simply copy-paste the URL.

Inspiration

As stated in User Guide, **BoNUS** helps you *better(B) organize(o) your NUS life*. Thus, we want to make the application an integrated personal manager for NUS students. The main activities for most students are study-related and most NUS students are currently using NUSMods to build their school timetable (as shown in Figure 4.7.1.1).



Code	Title	Module Credits	Exam Time	Actions
CS2101	Effective Communication for Computing Professionals	4	No Exam	ℹ ✖
EE2020	Digital Fundamentals	5	No Exam	ℹ ✖
CS2103T	Software Engineering	4	04-12-2017 5:00 PM	ℹ ✖
EE2027	Electronic Circuits	4	05-12-2017 5:00 PM	ℹ ✖
Total Module Credits for Semester ?		17		

Figure 4.7.1.1 : NUSMods Website Interface

Let’s imagine some of our users want to use the event feature in**BoNUS**. They want to input final examinations for all the modules they are taking as upcoming events. It would be very inconvenient and tedious if they have to do this manually. Even worse, it is very likely they are already using NUSMods and thus are not willing to add these events again. They may end up not using the event feature at all.

However, users may find it very useful if they can import their NUSMods timetable using a simple command. Eventually, they would choose **BoNUS** because they can manage both contacts and events in one application conveniently and the migration from NUSMods to **BoNUS** is not troublesome.

Design Considerations

Aspect: Relationship between `ImportXmlCommand` and `ImportNusmodsCommand`
Alternative 1 (current choice): Add a new abstract `ImportCommand` class and let both of them become its sub-commands (inherit from it).
Pros: This is inspired by many popular command-line tools (like Git). `import` is called the actual command, while `--xml` and `--script` is called the options. Most Unix/Linux users would be used to this approach. This is important for us because we assume our users are typists and they are very likely to frequently use these command-line tools.
Cons: Need to write extra codes and parsing also becomes more complicated.
Alternative 2: Implement these two commands separately.
Pros: Easy to implement and similar to other commands.
Cons: Our users may not be used to it. The command word will become longer. It is not a good OOP practice as well because common details are not abstracted into a parent class and this produces duplicate codes.

(Similar strategy has been adopted in `ConfigCommand`)

Aspect: How to obtain user's NUSMods timetable
Alternative 1 (current choice): Let users copy-paste the URL as a parameter of `ImportNusmodsCommand`.
Pros: Simple to use and easy to implement as well
Cons: Need to check whether the URL is valid and from NUSMods (currently using regular expression).
Alternative 2: Implement a built-in browser and render the NUSMods page
Pros: Users are more used to this interface.
Cons: Need much extra work to implement the built-in browser. The page may not be rendered well since the built-in browser is typically smaller than OS browser and NUSMods does not fully adopt responsive UI framework and may not work well on a small browser window.

Implementation Outline

`ImportCommand` abstract class

Create an `ImportCommand` abstract class and let `ImportXmlCommand` and `ImportNusmodsCommand` inherit from it (as in Figure 4.7.3.1). It is also a good practice to use an enumeration `ImportType` because the possible types of the import are within a fixed set of values. This leads to better modularity in `ImportCommandParser` as well.

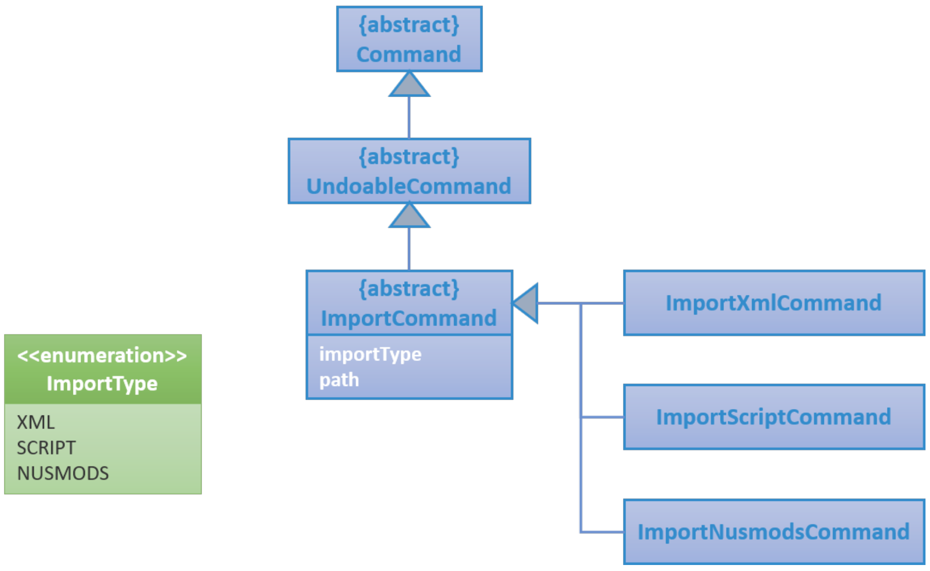


Figure 4.7.3.1 : Class Diagram for Related Import Commands

Parsing of NUSMods Timetable URL

Implement utility method to validate a given URL and parse the `GET` parameters. Although it is possible to utilize external library like Apache HttpComponents, we decide to implement on our own because it is relative simple to do so and using external library comes with extra expenses (such as licence, etc).

```
public static Map<String, String> fetchUrlParameters(URL url) throws UnsupportedEncodingException {
    String query = urlDecode(url.getQuery());

    if (Strings.isNullOrEmpty(query)) {
        return Collections.emptyMap();
    }

    Map<String, String> pairs = new HashMap<>();
    for (String pair: query.split("&")) {
        int index = pair.indexOf("=");
        pairs.put(pair.substring(0, index), pair.substring(index + 1));
    }

    return pairs;
}
```

Fetch information from NUSMods API

The URL parsed just now only contains the module codes and grouping for each module. In order to add upcoming events, we need more information such as module names, examination dates, etc. We decide to use API provided by NUSMods to fetch the information we need. NUSMods API is in JSON format, which would be very simple for us as we already use Jackson library in our project.

```
/**
 * Read JSON data from a given URL and convert the data to an instance of the given class.
 * @param url is the URL to the remote JSON data.
 */
public static <T> T fromJsonUrl(URL url, Class<T> instanceClass) throws IOException {
    return objectMapper.readValue(url, instanceClass);
}
```

Add upcoming events

After obtaining all the information we need, we can simply use the `addEvent` method in `ModelManager` class to add the final examinations as upcoming events into **BoNUS**. This should be a similar process as `AddEventCommand`.

Enhancement Proposed: Import From BoNUS-specified Script file

External behavior

From `.bo` format

(Coming in v2.0)

Imports the data in an external BoNUS script file (which ends with `.bo`), including data from all three components: **Contacts**, **Events** and **Calendar**, into the application.
Format: `(i)import --script FILEPATH`

- You must explicitly provide the `--script` parameter.
- `FILEPATH` must end with an extension of `.bo`.
- The file name in `FILEPATH` should not be empty, nor should it contain any prohibited characters `?!%*+:|"<>`.
- If a relative path is provided, the data will be imported from a location relative to the **BoNUS** installation directory.
- The provided script file should include one or multiple lines of valid **BoNUS** command(s). Each line can only have **at most one** command.
- The **command** here refers to any command mentioned in this guide.

Examples:

- For `Windows` users:

`import --script C:\Users\John Doe\Documents\bonus.bo`

- For `macOS` and `Linux` users:
  `import --script /Users/John Doe/Documents/bonus.bo`

> For `Windows` users, use `\` as the name-separator in your `FILEPATH`.
> For `macOS` and `Linux` users, use `/` instead.

Justification

This feature is meant for advanced users. To build an **epic** application, we need to essentially build an *ecosystem* for this software. According to the *Unix* philosophy, providing a shell scripting and command language is the minimum requirement for a complete system.

With the support for scripting file, advance users (or system administers if **BoNUS** is used as an enterprise application) can generate a scripting file and import it into **BoNUS**. Data management and frequent operations can be done easily in one step. Otherwise, they have to type one command each time in the GUI interface, which becomes a very tedious work.

Enhancement Proposed: Export Data to Microsoft Excel^{TM} Worksheet

External behavior

To Microsoft Excel^{TM} Worksheet

(Coming in v2.0)

Exports the current data in the application to an external file of Microsoft Excel^{TM} format.
Format: `(p)export --excel FILEPATH`

- The file name should follow similar rules to the section above.
- However, it must end with an extension of `.xls` (`.xlsx` is currently not supported).

Examples:

- For `Windows` users:
  `export --excel C:\Users\John Doe\Documents\bonus.xls`
- For `macOS` and `Linux` users:
  `export --excel /Users/John Doe/Documents/bonus.xls`

> For `Windows` users, use `\` as the name-separator in your `FILEPATH`.
> For `macOS` and `Linux` users, use `/` instead.

Justification

We already support `ExportCommand` so that users can save data to a separate location as a backup. However, `.xml` files are not considered to be user-friendly (except for programmers and geeks).

Thus, it becomes essential for us to natively support exporting data to a format with better visuals. As stated before, data stored in **BoNUS** can be considered as *tables*. Naturally, it comes to my mind that Microsoft Excel^{TM} would be a perfect format to present data to users.

Other contributions

- Repository setup, CI setup and Slack automatic notification using WebHook (Pull requests #1, #6, #7)
- Design and refine multiple parts of UI (Pull requests #41, #43, #44, #82, #106)
- Support setting customize colors for tags (Pull requests #83, #101, #147)
- Support natural language parsing (Pull request #148)
- Support adding avatar to contacts (Pull request #162)
- Write unit tests for various classes
- Update various sections in `AboutUs`, `ContactUs`, `UserGuide` and `DeveloperGuide`, etc.

Reuse offer

- Generic property: Issue #180
- Set customize color for tags: Issue #199

Helping others

- On the forum: Issue #195, #197, #200
- Report bug for UniBook: Issue #72
- Report bug for module website: Issue #18, #28, #29, #34, #39, #41, #43

Project: Lions Befrienders Management System

I worked at CVWO (Computing for Voluntary Welfare Organisations) as a web developer during the Year 1 summer vacation.

My team at CVWO rewrote, upgraded and deployed the management system for Lions Befrienders (Singapore), which was formed in 1995 and served thousands of senior citizens. This system saves a lot of time for staff at Lions Befrienders to manage their growing enormous clients across the country, while ensuring high service quality.

Project: A Multi-User 3D Game Environment for CS1101S

This project aims at building a 3D game environment to assist CS1101S students in learning programming. The game is built using Unity3D game engine. It provides vivid visualization to help students have a deeper understanding of the related data structure and algorithms used.

CS1101S is a module taught in the Department of Computer Science at the NUS School of Computing. It serves as a rigorous and thorough introduction to programming methodology.

Last updated 2017-11-13 14:42:52 +00:00