Caesar Pang - Project Portfolio

Introduction

[github] [linkedin]

Hi! My name is Caesar Pang. I'm currently a Year 2 student studying Computer Science, in School of Computing (SOC) in NUS.

This portfolio page aims to document the contributions I have made in the development of TutorAid, which is a project that my team and I completed for the module CS2103T. This project has definitely taught me many skills that are extremely relevant in the real world and in the future. It has allowed me to be a more competent and knowledgeable software engineer.

PROJECT: TutorAid

Overview

TutorAid is an easy-to-use Command Line Interface (CLI) [i.e issue commands to the program by typing in text inputs] based application with a Graphical User Interface (GUI) [i.e. allows users to interact with the application through visual and graphical elements] that helps to makes the lives of tutors easier by organizing their workflow and making tutors more efficient. With a myriad of features such as student profile tracking, earnings, reminders, notes and tasks all in one application, tutors no longer have to search through their notes one-by-one excruciatingly.

The features of TutorAid has been well-thought out, keeping in mind the necessities and wants of tutors alike. To meet the exact needs of our target audience, we have conducted several surveys to ask current tutors what they would like in an application and thus, TutorAid was created. All of TutorAid's features and implementations are well documented with proper visualization in guides for users and developers respectively.

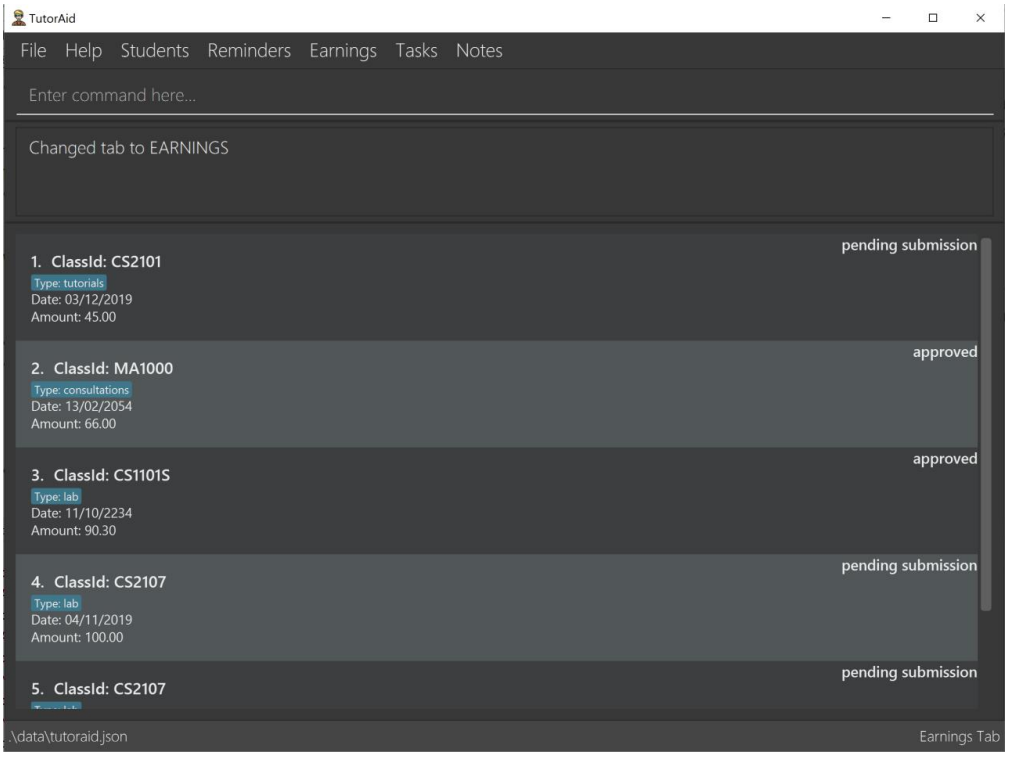

Figure 1. A view of TutorAid

Role

My main role was to prototype and develop the Login feature and the Earnings Tracker feature. The login command required me to think of how to put up a login window that only shows the main window when the user has logged in successfully. It also required me to think of how to encrypt the passwords stored and how to check whether they are valid as well. I also did the Earning feature where it tracks what earnings you have earned throughout the weeks. Developing it further, I decided to add a automated earnings addition to save the users time. It has a claim status that allows you to see if you have claimed your earnings from your respective faculties.

Summary of contributions

- **Largest Enhancement**: Auto addition of earnings.
    - What it does: It allows the user to auto add all of the earnings of the day (provided that the earnings is a weekly routine), into the earnings list with only one short `auto` command.
    - Justification: This feature improves the product significantly because as a Tutor, it is normal to have tutorials or labs or consultations to give at the same day of the week, every week. If a tutor, for example, has many lessons in a day to teach, he/she has to add the earnings into the list one by one, which is extremely hard and tedious to do. If the tutor has added the earnings into the list once already, the tutor can repeat this earnings by invoking the `weekly_earnings` command and the number of times to add into the list. Afterwards, the tutor just has to invoke the `auto` command on the day itself to auto add all his routine lessons.
    - Implementation: This enhancement alters how TutorAid is used. Instead of just being a normal, "key in earnings" tracker, the addition of such a feature has helped to ease the use of earnings tracking for the users. This is facilitated by the use of a HashMap in the Earnings

list. This HashMap only has at most 7 keys as these keys are determined by the name of the days in a week. Inside the HashMap, there are ArrayList mapped as the values of the HashMap. This feature consists of 2 main commands. The first command is `weekly_earnings INDEX count/COUNT` and the second command is `auto`. Users have to add the earnings that they want to repeat over the weeks manually first. After they have added the earnings, they can invoke the `weekly_earnings` command to auto add the earnings for the next few weeks depending on the number of counts they have inputted. The `weekly_earnings` command add the key-value pair to a HashMap in `Earnings` class. Once the user invokes the `auto` command, the parser will check the current day of the week and get the value of the HashMap that is associated with the key value (that is the day of the week). If there exists an ArrayList, the parser will check through the entire list and check if the Earnings in the ArrayList has more than a `Count` of 0. If the `Count` is more than zero, then the parser adds the `Earnings` object into the current list. If the `Count` is zero, the `Earnings` object would be removed from the ArrayList.

- **Minor Enhancement**: Register and Login feature.
  - What it does: It allows the user to register and login into the application.
  - Justification: This feature improves the product significantly because as a Tutor, you would enter in critical and sensitive information of your students and of your earnings. By adding a register and login feature, it makes the app a lot more secure and tutors can confidently enter data into their application worry-free.
  - Implementation: This enhancement alters the way TutorAid is used. Normally, users could enter into the application once they launched it. But with the register and login feature, it only allows users to login when they have the correct credentials. The register and login feature is aided by a password encryption method. When users register, the passwords of the accounts created will be hashed with a salt so that there will be more security. I created a Login Window as well so that the users can have a visual cue when they login into their account.
- **Code contributed**: [All commits] [Project Code Dashboard]
- **Other contributions**:
  - Project management:
    - Managed and assigned the issues at the start of the project: #82, #94, #104
    - Managed release `First Draft`, `v1.2` and `v1.3.2` on Github
    - Ensured the travis build was always working and passed before milestone deadlines. #246, #173
    - Implemented Protected Branches on team repository so as to have teammates review each other and prevent accidental merges.
  - Enhancements to existing features:
    - Wrote multiple tests for existing features to increase code coverage incrementally (#386, #345, #337)
  - Documentation:
    - Added detailed implementation documentation for the account storage and earnings feature in Developer Guide, including diagrams (#201, #203)

        - Customized and updated ReadMe for TutorAid
    - Community:
        - Consistently reviewed and gave feedback to team members. PRs reviewed: #243, #214
    - Tools:
        - Set up Netlify
        - Set up Token Generator Account for team repository.

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.

Auto Add Weekly Earnings: `weekly_earnings`

Adds an earnings from the list of existing earnings to be added in the on the same day every week.
Format: `weekly_earnings INDEX count/NUMBER_OF_WEEKS

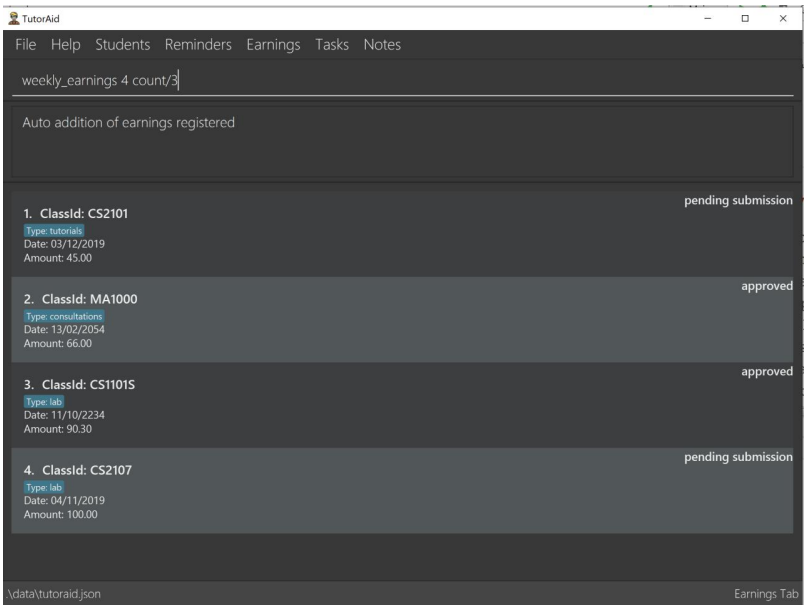

Figure 2. Weekly Earnings Example

Examples:

- `weekly_earnings 2 count/2
- `weekly_earnings 3 count/13

- Only numbers in the range of 0 - 13 (inclusive) are valid.
- This auto add will only occur on the day itself by invoking `auto` command.
- It is not allowed to add earnings 2 weeks prior and expect the application to add the earnings twice. It will only add on the day itself.
- Using this command assumes that all attributes of the indexed earnings are the same for future earnings other than the date.

Adds Weekly Earnings: `auto`

Adds all the earnings that were invoked by `weekly_earnings`.
Format: `auto`

Example:

- `auto`

- It must be invoked on the day itself for it to work.
- If user has missed a day, the earnings will not be added. For example, if an earnings has a date of 02/02/2019, and the `auto` command is invoked on the day of 10/02/2019 instead of 09/02/2019, the earnings will not be added.
- Suggested to invoked everyday.

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.

Add Earnings

The `add_earnings` command allows for tutors to add their earnings into TutorAid.

The format for the `add_earnings` command is as follows:

```
add_earnings d/<DATE> type/<TYPE> c/<CLASSID> amt/<AMOUNT>
```

Overview

The `add_earnings` mechanism is facilitated by `AddEarningsCommand` and `AddEarningsCommandParser`, taking in the following input from the user: `Date`, `Type`, `ClassId`, `Amount`, which will construct `Earnings` objects.

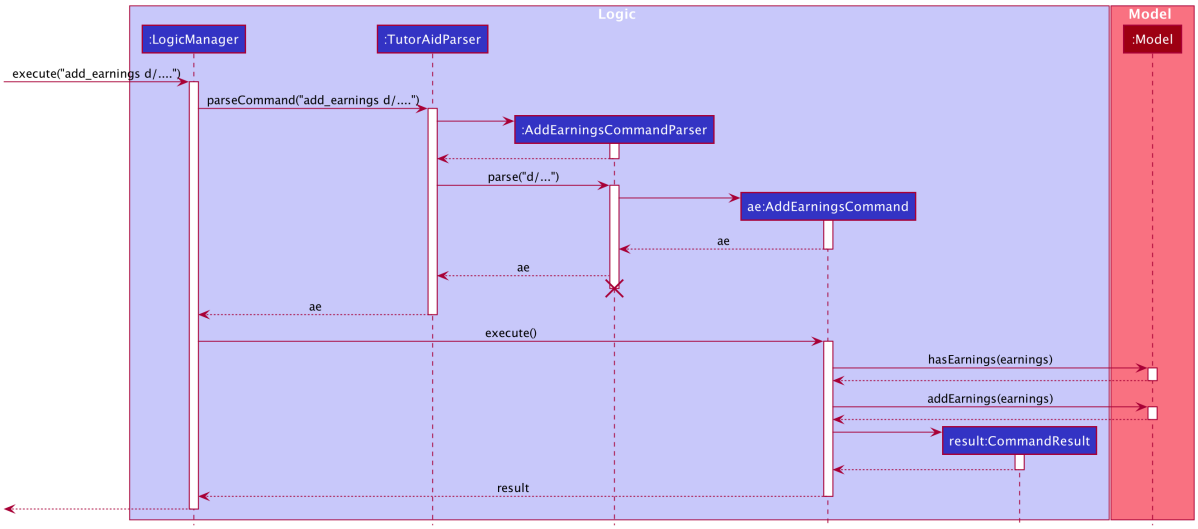

Figure 3. Add Earnings Command Sequence Diagram

The `AddEarningsCommand` implements `Parser` with the following operation:

- `AddEarningsCommandParser#parse()` - This operation will take in a `String` input from the user and create individual objects based on the prefixes `d/`, `c/`, `type/` and `amt/`. The `String` value after the respective prefixes will create the respective objects. A validation check will be done to ensure that the strings that are entered by the user is entered correctly. If any of the strings entered are invalid, an error will be shown to the user to enter the correct format of the respectives objects.
    - `date` would use `ParserUtil#parseDate()` to ensure that the date typed by the user is in the correct format of DD/MM/YYYY.
    - `type` would use `ParserUtil#parseType()` to ensure that the type entered by the user is valid.
    - `classId` would use `ParserUtil#parseClassId()` to ensure that the class id typed in by the user is in the correct format.

    - `amount` would use `ParserUtil#parseAmount()` to ensure that the amount entered by the user is in the correct format with 2 decimal places to represent the cents.

- After validation of the individual objects, an `Earnings` object would be created with the parameters `date`, `type`, `classId` and `amount`.
- `AddEarningsCommandParser` would then return a `AddEarningsCommand` object with the parameter, `Earnings` object.

Example Scenerio

- Step 1: The user enters `add_earnings d/04/08/2019 type/tut c/CS1231 amt/50.00` to add an earning for teaching classes. In this example, it adds an `Earnings` object that the user has earned $50.00 by teaching a CS1231 tutorial class on 4th August 2019.
- Step 2: `LogicManager` would use `TutorAidParser#parse()` to parse input from the user.
- Step 3: `TutorAidParser` would match the command word given by the user with the correct command. In this example, the given command is `add_earnings`, thus, `AddEarningsCommandParser` object would be created with the user's input.
- Step 4: `AddEarningsCommandParser` performs a validation check on each of the respective objects through `AddEarningsCommandParser#parse()`. In this case, it would use `ParserUtil#parseDate()`, `ParserUtil#parseType()`, `ParserUtil#parseClassId()` and `ParserUtil#parseAmount()`. It would then return a `AddEarningsCommand` object with an `Earnings` object.
- Step 5: `LogicManager` would execute `AddEarningsCommand#execute`. In this particular method, the `Earnings` object will be check with the rest of the prior `Earnings` object, to ensure that there is no duplicate `Earnings` object. If there are no similar `Earnings` object with the same parameters created previously, it would then be added into the earnings list.
- Step 6: `AddEarningsCommand` would then return a `CommandResult` to `LogicManager`, which would show the user that the new `Earnings` object have been successfully added.

Weekly Earnings

The `weekly_earnings` command allows users to add earnings into a list. This list adds earnings weekly by invoking the `auto` command automatically. This helps to lessen the workload on the user as the user does not need to add a new earnings every week.

The format for the `weekly_earnings` command is as follows:

```
weekly_earnings INDEX count/<NUM_OF_WEEKS>
```

Overview

The `weekly_earnings` mechanism is facilitated by `AutoAddEarningsCommand` and `AutoAddEarningsCommandParser`, taking in the following input from the user: `Index` and `Count`, which would be associated with the `Earnings` object that is referenced from the `Index`. The `Count` object represents the number of weeks the earnings are to be added to the list.

The `AutoAddEarningsCommand` implements `Parser` with the following operation:

- `AutoAddEarningsCommandParser#parse()` - This operation will take in an `int` input from the user, followed by a `String` input from the user and create individual objects based on the prefix `count/`. The `String` value after the prefix will create a `Count` object. A validation check will be done to ensure that the string that is entered by the user is entered correctly. If the string entered is invalid, an error will be shown to the user to enter the correct format of the `Count` object.
    - `index` would use `ParserUtil#parseIndex()` to ensure that the index typed in by the user is in the correct format and is valid.
    - `count` would use `ParserUtil#parseCount()` to ensure that the count typed by the user is in the correct format and between 0 - 13 (inclusive).
- After validation of the individual objects, the particular `Earnings` object would have a `Count` object and be added to a list that is ready to be added automatically.
- `AutoAddEarningsCommandParser` would then return a `AutoAddEarningsCommand` object with the parameters, `Index` and `Count` object.

Example Scenerio

- Step 1: The user enters `weekly_earnings 2 count/3` to add that indexed earnings to a list of earnings to be added on the same day of the week, every week. For example, if the referenced earnings has a date of `01/11/2019`, which is a Friday, that particular earnings will be added on every Friday for a total of 3 weeks.
- Step 2: `LogicManager` would use `TutorAidParser#parse()` to parse input from the user.
- Step 3: `TutorAidParser` would match the command word given by the user with the correct command. In this example, the given command is `weekly_earnings`, thus, `AutoAddEarningsCommandParser` object would be created with the user's input.
- Step 4: `AutoAddEarningsCommandParser` performs a validation check on each of the respective objects through `AutoAddEarningsCommandParser#parse()`. In this case, it would use `ParserUtil#parseIndex()` and `ParserUtil#parseCount()`. It would then return a `AutoAddEarningsCommand` object with an `Index` and `Count` objects.
- Step 5: `LogicManager` would execute `AutoAddEarningsCommand#execute`. In this particular method, the `Earnings` object will be check with the rest of the prior `Earnings` object that has been added to the auto addition of earnings list, to ensure that there is no duplicate `Earnings` object in the list. If there are no similar `Earnings` object with the same parameters created previously, it would then be added into the auto addition earnings list.
- Step 6: `AutoAddEarningsCommand` would then return a `CommandResult` to `LogicManager`, which would show the user that the new `Earnings` object have been successfully added to the list.

Auto Add

This command, `auto`, allows the user to add all the earnings that has been previously added before and the command `weekly_earnings` has been used on the particular earnings. You can refer to the activity diagram below to have a clearer understanding.

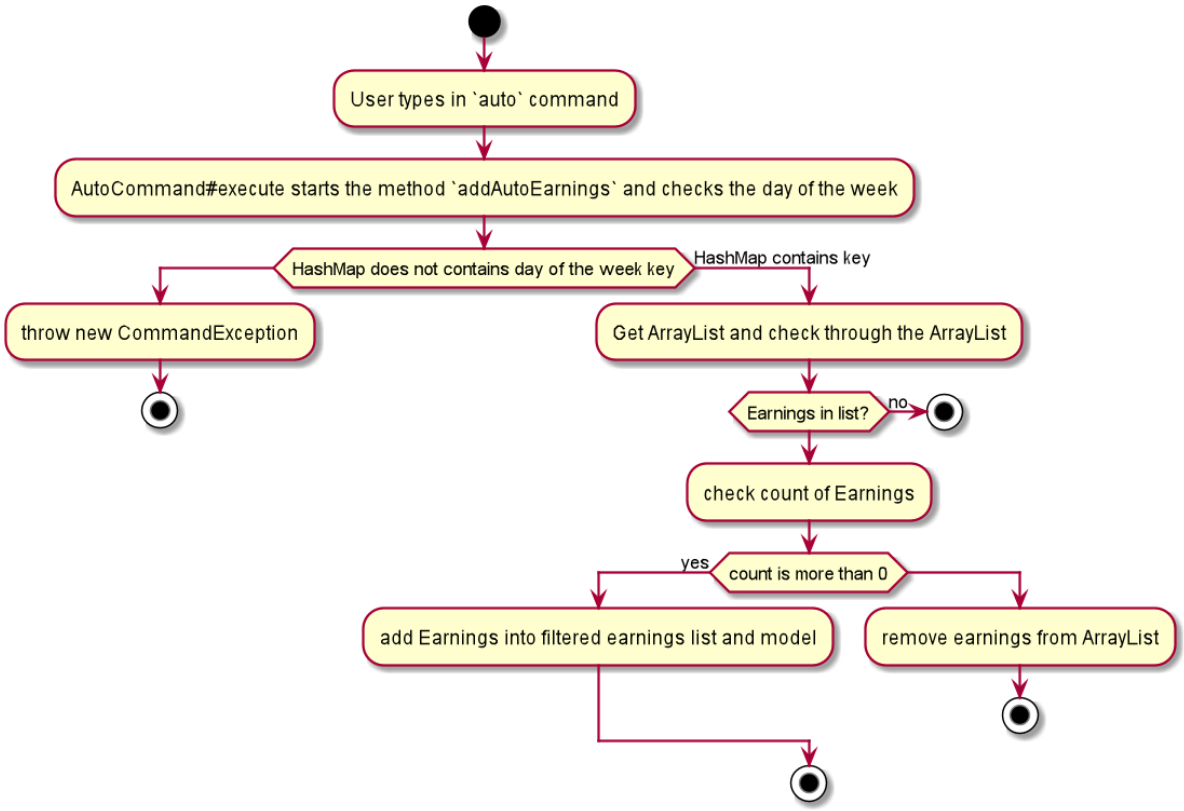

Figure 4. Auto Command Activity Diagram

Overview The auto command is facilitated by `AutoCommand`.

- `AutoCommand#execute` - This operation checks the current day of the week and checks against the `HashMap` of the earnings list. If there are `Earnings` object associated with the current day of the week and the `Earnings` object has a `Count` object associated with it that is more than 1. The earnings inside the `ArrayList` inside the `HashMap` would be added into the earnings list.

Example Scenario

- Step 1: The user enters `auto` to add all his days earnings into the earnings list.
- Step 2: `LogicManager` would use `TutorAidParser#parse()` to parse input from the user.
- Step 3: `TutorAidParser` would match the command word given by the user with the correct command. In this example, the given command is `auto`, thus, `AutoCommand` object would be created with the user's input.
- Step 4: `LogicManager` would execute `AutoCommand#execute`. In this particular method, the `execute` command would check the day of the week. It would then check with the `HashMap` to check if there is any `ArrayList` of `Earnings` object to be added in. If there is no `ArrayList` available, no earnings would be added. If there is an `ArrayList` mapped to the day of the week, the method would then check through every `Earnings` object in the `ArrayList`. In each `Earnings` object, there is a `Count` object associated to it. If the `Count` object has a more than 1, the `Earnings` object would be added into the current earnings list. If the `Count` object has a value of 0, the `Earnings` object would be removed from the `ArrayList`.
- Step 6: `AutoCommand` would then return a `CommandResult` to `LogicManager`, which would show the user that the new `Earnings` object have been successfully added to the list.

Account Storage

All accounts are stored in a JSON file called accountslist.json. This file is generated once you register an account. The username and password is stored inside the the JSON file with salt hashing thus, the account details cannot be seen by everyone.

Design Considerations We realised that storing the accounts by itself would allow anyone to see the username and passwords. Hence, we decided to use a password authentication method and salt hashing to cover the passwords of the accounts.

In addition, we decided to go with using a JSON file instead of XML file due to these considerations:

Table 1. Account File Storage function alternatives

Design Consideration	Pros and Cons
JSON File (Current Choice)	**Pros** : More familiarity with JSON files. More compact and can be easily loaded. Flexible **Cons** : Bad Schema support and namespace support
XML File	**Pros** : Easy transfer of data between seperate systems. Good at storing data that will be readby 3rd parties. **Cons** : Not familiar with it and we would require more time to learn how to use it.

Security Considerations As of now, only password hashing is done to protect the accounts from being seen by unwanted eyes. We have planned ahead of time and decided to implement better security options for v2.0.

- **Store accounts on a backend server:** The project restricts us and as of now, it is not implemented. The advantages of storing the accounts in database is that it can be a lot more secure.
- **Encrypt the Account JSON file:** This will prevent other users from easily clicking into the data file and make it more secure.