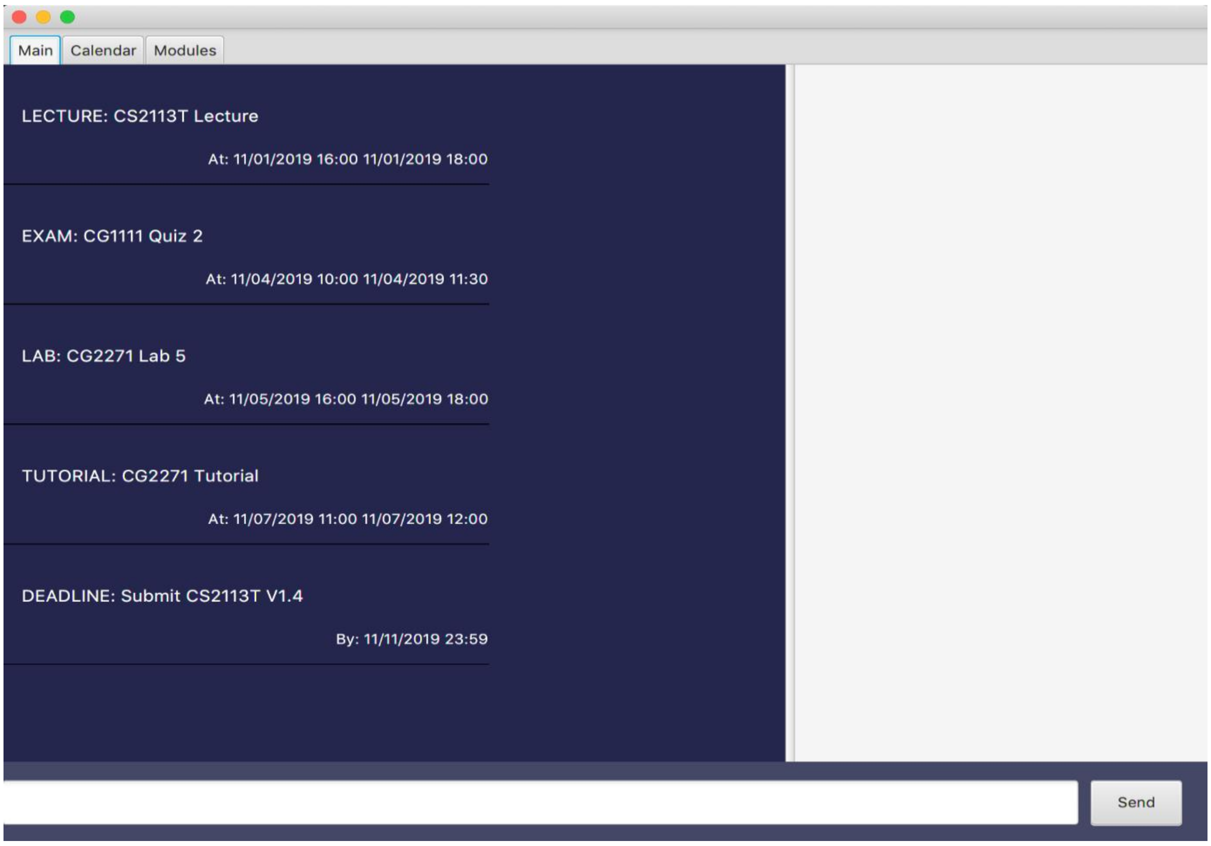
Project Portfolio Page for SpinBox

This project was built by 4 Software Engineering students tasked to build a Java application under specified constraints, to demonstrate understanding of Software Engineering & OOP.

We chose to transform our application from a command-line interface chatbot into a university life helper known as SpinBox. SpinBox allows NUS students to overcome the issue of context switching between modules and increase their productivity significantly by allowing them to focus on one thing at a time while still being able to view and appreciate the high-level idea.



LECTURE: CS2113T Lecture
At: 11/01/2019 16:00 11/01/2019 18:00

EXAM: CG1111 Quiz 2
At: 11/04/2019 10:00 11/04/2019 11:30

LAB: CG2271 Lab 5
At: 11/05/2019 16:00 11/05/2019 18:00

TUTORIAL: CG2271 Tutorial
At: 11/07/2019 11:00 11/07/2019 12:00

DEADLINE: Submit CS2113T V1.4
By: 11/11/2019 23:59

Figure 1: SpinBox upon launch, with the main tab open. Within the main tab, an aggregated list of the top five most urgent tasks can be seen

This portfolio makes use of the following conventions:

1) Code components are written using the Consolas font. E.g. `Storage`
2) 💡: Refers to important notes for the reader, also used within the user guide.
3) Commands are in bolded Calibri font, with user specified parameter(s) capitalized and square-bracketed. E.g. **add [MODULE_CODE] / grade [DESCRIPTION] [WEIGHTAGE]%**

Summary of contributions (Reposense)

As team leader, I was actively involved in maintaining code quality as well as monitoring the progress of the team over the semester. I ensured that proper Software Engineering workflows and practices were adhered to, such as branching, imperative commit messages, code reviews and the use of GitHub's "Squash and merge" feature to prevent pollution of the repository's commit history. My own contributions to the project are documented below.

Hierarchical Data Persistence & Snapshot exports (#8, #87, #105, #136, #140, #202, #215)

This is intended to be a plug-and-play solution to establish data permanence and exporting.

Justification: Users of the application would want their data saved across SpinBox runs. Advanced users will want to be able to access/modify the data directly in an efficient manner, supported by the implemented hierarchy of file-folders. Additionally, users would want to export their data into English, so that they can naturally use their preferred reminder system in combination with SpinBox to increase their productivity.

Potential use cases and scalability: This has widespread applications across SpinBox – as an example, it can be used in the GUI to save state in between runs of the program and is fully extensible for future classes of items and their containers. Currently, a **`Storage`** instance is used within **`ModulesContainer`**, within which each **`Module`** contains four components that make use of their own **`Storage`** instance – **`NotePad`**, **`GradesList`**, **`TaskList`** and **`FilesList`**.

How: Data-persistence is facilitated by independent instances of the **`Storage`** and **`Exporter`** classes, which are designed to be context-agnostic and play the role of functional elements within a larger, highly cohesive component. Through this, greater separation of concerns can be achieved by abstraction of the process of storage and optionally, retrieval of data from files.

Project management:

- Contributed 100% of my Duke repository as the base for phase 2 (#1)
- Managed milestones and releases v1.0 – v1.4 (Milestones, #5, #7, #104, #229)
- Reviewed and merged most of the PRs to the repository (merge commits)
- Introduced Travis CI, Coverall and Logging to the repository (#4, #178, #213)

Other code contributions / Enhancement(s): (Full history (GitHub))

- Responsive GUI styling and functionality: (#129, #142, #177, #203, #210, #211)
- Module-specific notepad (#88, #95)
- Module-specific graded components (#68)
- Items abstract class, interfaces, bug fixing, architecture, exceptions (#11, #63, #68 …)

Contributions to the User Guide

In a typical module, the assessments are spread out through the semester and have varying weightages. For a student, these can quickly get overwhelming to manually keep track of, and SpinBox makes this process pain-free through the implementation of graded components.

The following excerpts from the user guide introduce the user to graded components and their associated commands. Additionally, the export feature is also featured afterwards. You may refer to the full reference for these components within the user guide, if required.

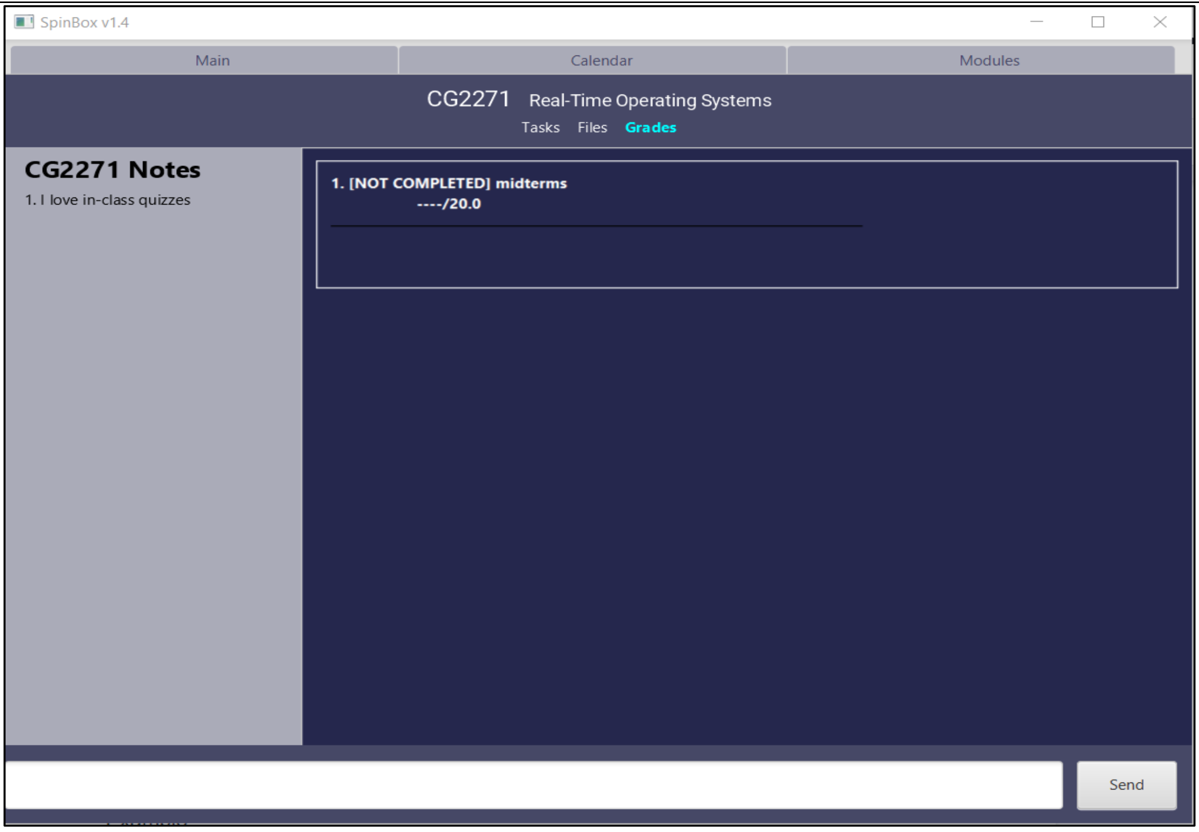
User Guide Excerpt 1: Viewing the Grades section

Assessments form the basis for grading student performance in most modules, and SpinBox offers a natural way of keeping track of your progress in these high-stakes assessments - using graded components found under the ‘grades’ subtab of a module.

Condensed command, if currently viewing a module: **view / grades**

Full command, to specify a module explicitly: **view / modules [MODULE_CODE] grades**

An example of the expected graphical output is shown below in Figure 2.



1. [NOT COMPLETED] midterms
----/20.0

Figure 2: Module-specific view, with the Grades subtab selected

User Guide Excerpt 2: Adding a graded component

Add a new graded component under the selected module.

Condensed command, if currently viewing a module:

add / grade [COMPONENT] weightage: [WEIGHTAGE]%

Full command, to specify a module explicitly:

add [MODULE_CODE] / grade [COMPONENT] weightage: [WEIGHTAGE]%

💡: **No restriction on numerical input** for **[WEIGHTAGE]** - to accommodate for special cases such as negative grading, bonus points etc.

Examples:

- **add / grade Written Report weightage: 12.5%**
  (adds a graded component to the currently selected module)

- **add CG1111 / grade Written Report weightage: 12.5%**
  (adds a graded component to the module CG1111)

User Guide Excerpt 3: Updating graded component scores with relative values

You can update a graded component for the currently selected module with the actual scores received. This will be converted using the weightage provided when adding the component. **Additionally, this will mark the graded component as done**.

💡: The marks entered will be automatically be converted to an absolute percentage, using the graded component's weightage as the baseline. E.g. for a Written Report worth 10%, entering 27/36 will be converted into 7.5%.

Condensed command, if currently viewing a module: **score / [INDEX] marks: [XX/YY]**

Full command, to specify a module explicitly: **score [MODULE_CODE] / [INDEX] marks: [XX/YY]**

Examples:

- **score / 1 marks: 27/36**
  (scores the 1st graded component under the currently selected module)
- **score CG1111 / 2 marks: 27/36**
  (scores the 2nd graded component under the module CG1111)

User Guide Excerpt 4: Exporting data

SpinBox items can be exported at any time for inclusion into a dedicated ecosystem, i.e. an Excel planner. This is achieved via the export command. You can export all tasks, files or grades of a specific module, or deadlines across all modules to a human-readable text file. The exported file will be stored in the 'SpinBoxData/exports' folder.

Possible values for **[ITEM_TYPE]**: One of **files**, **grades**, **tasks**

Format when not within the desired module: **export [MODULE_CODE] / [ITEM_TYPE]**

Format when currently viewing desired module: **export / [ITEM_TYPE]**

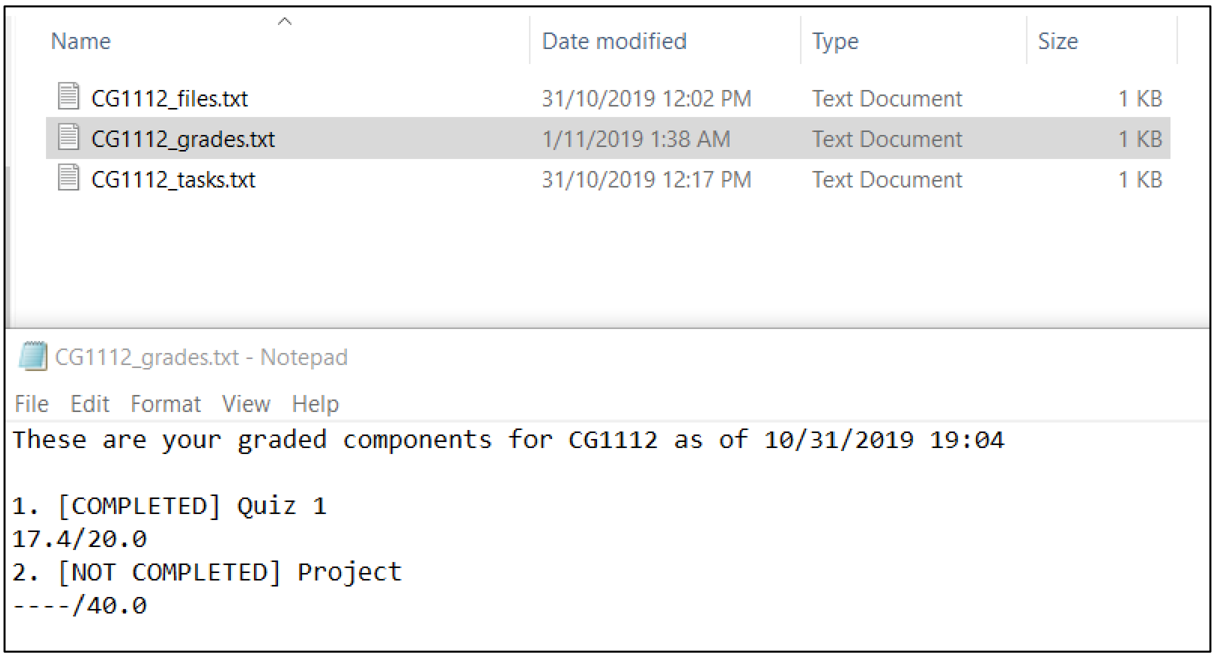
Format to export deadlines across all modules: **export / deadlines**

Examples:

- **export CG1112 / grades**

  (Exports the graded components within the CG1112 module)

An example of the sample output is shown in Figure 3 below.



Name	Date modified	Type	Size
CG1112_files.txt	31/10/2019 12:02 PM	Text Document	1 KB
CG1112_grades.txt	1/11/2019 1:38 AM	Text Document	1 KB
CG1112_tasks.txt	31/10/2019 12:17 PM	Text Document	1 KB

```
These are your graded components for CG1112 as of 10/31/2019 19:04

1. [COMPLETED] Quiz 1
17.4/20.0
2. [NOT COMPLETED] Project
----/40.0
```

Figure 3: Module-specific view, with the Grades subtab selected

Contributions to the Developer Guide

As for the developer guide, I discuss data persistence and exports, GradedComponents and how these components work together within a **`Module`** to satisfy the user's requirements.

Developer Guide Excerpt 1: Relation between Storage & GradedComponents

This section details how the **`Storage`** is related to a **`Module`**'s **`GradeList`** instance, which is essentially a container for **`GradedComponents`**. The relevant parts of the structure of the program is shown below, and the behaviour will be detailed at a later stage.

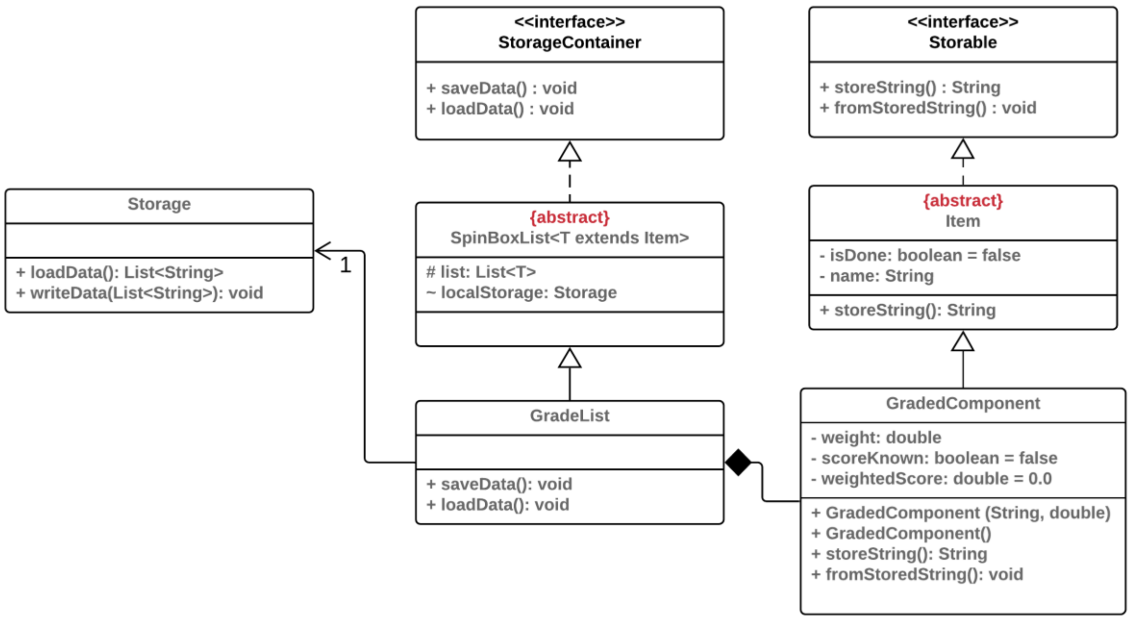


Figure 4: UML class diagram for Storage and GradeList

As can be seen above, the **`Storage`** instance is embedded within a **`GradeList`**. The **`GradeList`** implements the methods required by the **`StorageContainer`** interface, such that it can convert the data to/from a **`List<String>`** for storage. To achieve this conversion, it can call each **`GradedComponent`**'s methods implemented from the **`Storable`** interface.

Developer Guide Excerpt 2: Scoring a GradedComponent with relative scores

Given below is a typical use case for the GradeList module, and the sequence of events when marking a GradedComponent as complete, after being made aware of the grade obtained.

Step 1: The user starts up SpinBox to score a written report worth 20% under CS2999. SpinBox scans through the SpinBoxData folder and populates its variables in memory.

Step 2: The user views a module's grades through **view CS2999 / grades**.

Step 3: The user executes **score / 2 marks: 87/100** to update their score into SpinBox.

The technical workflow can be described through the activity diagram below, which starts at the GUI layer, after which the command is parsed to identify the module and component.

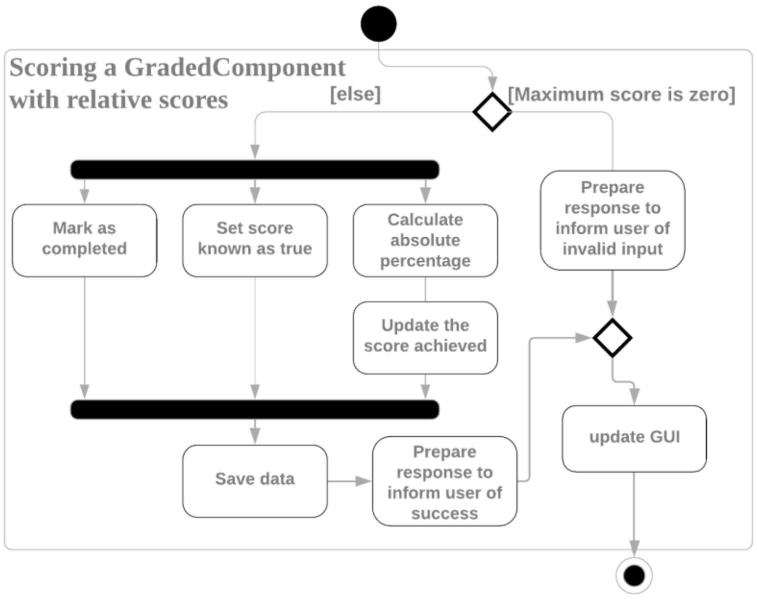

Scoring a GradedComponent with relative scores
[else]
[Maximum score is zero]
Mark as completed
Set score known as true
Calculate absolute percentage
Prepare response to inform user of invalid input
Update the score achieved
update GUI
Save data
Prepare response to inform user of success

Figure 5: Activity diagram for scoring of a graded component using relative scores

Developer Guide Excerpt 3: Exporting all tasks

The export feature can export all tasks of a module, as shown in the diagram below.

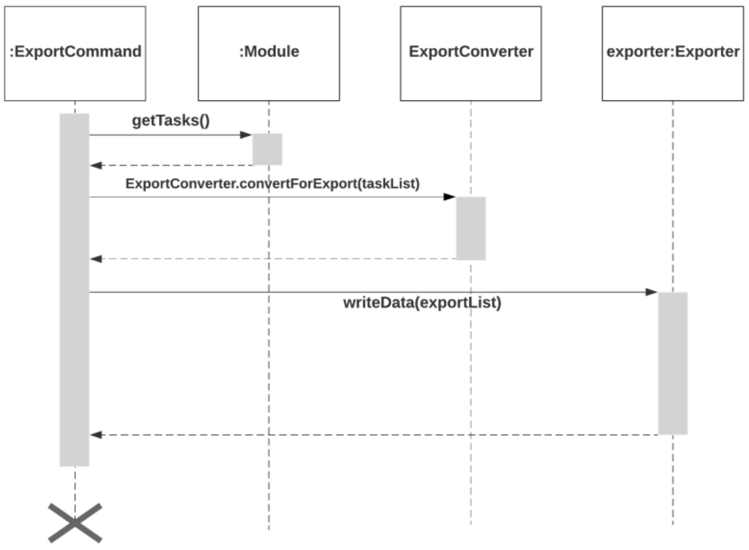


Figure 6: Sequence diagram for exporting task data of a specific module

Design considerations shaped the implementation of the storage system, as detailed below.

Aspect	**Alternative 1 (Chosen)**	**Alternative 2**
Distribution of data	One separate instance/file per container. Pros: • No concurrent I/O issues (simultaneous reads + writes) if multi-threaded • Uniform data type - no requirement to filter through irrelevant entries • Enables hierarchical storage hierarchy for intuitive editing experience for advanced users • Single file corruption will not affect all data • Higher cohesion • Simple to implement and reusable in a customised manner within any Container implementing StorageContainer Cons: • Storage must be generalized to a certain extent, currently a requirement of List<String> • Requires external conversion to adhere to the Law of Demeter • Time/space overhead due to multiple files	One instance that stores into one file. Pros: • Space efficient Cons: • High coupling • Filtering must be done on entries • More difficult to users to edit and remain corruption-free
Function set for storage	Set/Reset + Load (Current choice) Pros: • Easy to implement • Appropriate as each list is scoped small-enough that it is not expected to store too many items • Can extract out & reuse for efficient export Cons: • Processing overhead	CRUD Pros: • Mirrors actions performed on Containers • Performance-friendly Cons: • Requires additional state or stored information (e.g. indexes) • Slightly harder implementation