PEPO

Project Document v0.2

Developed by
Team Ultimate!

Chua Wei Kuan (A0072749U)
Ju Chaoran (A0077858H)
Lim Gang Yi (A0077256X)
Wong Hong Wei (A0072341R)

PEPO

Developer Guide

Developed by
Team Ultimate!

Chua Wei Kuan (A0072749U)
Ju Chaoran (A0077858H)
Lim Gang Yi (A0077256X)
Wong Hong Wei (A0072341R)

Contents

1 Introducing PEPO

Welcome to PEPO's Developer Guide!

PEPO is the abbreviation for ***Personal Events Planning Organiser***. It is a piece of software designed to facilitate events planning such as in the aspects of budgeting, contacts management, scheduling of programmes and so on.

You will first be introduced to the **high level design** *(section 2)* of PEPO, followed by the **internal models** *(section 3)* of each component. Next, you will go into the **code level** *(section 4,5 & 6)* and be guided on the ways you can make use of the APIs. After which, you will be given instructions for **testing** *(section 7)*.

This guide assumes that you have some basic knowledge of Java, XML, HTML and CSS. The software also uses the JCalendar library - http://www.toedter.com/en/jcalendar/index.html.

---

2 Presenting the Architecture

This is a high-level view of PEPO's design. It consists of 3 separate **components**. In each component, a **controller** acts as an interface for the whole component. The general role for each component is as follows:

- **Graphical User Interface (GUI)** : Control the look of the software
- **Logic** : Process data for viewing
- **Database** : Store and retrieve data

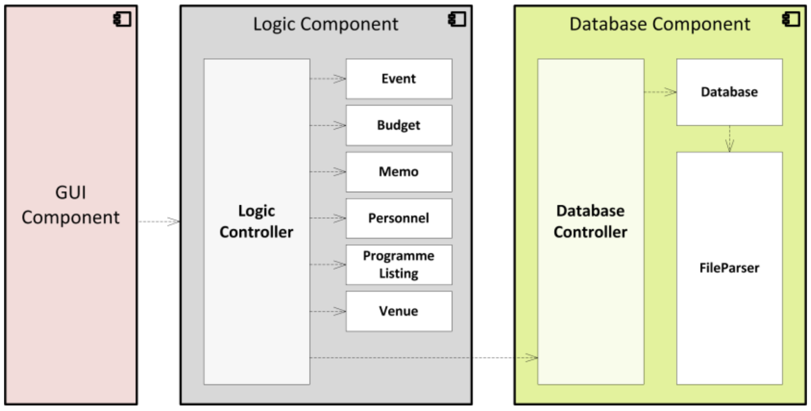


Figure 1 : PEPO Software Architecture

3 Understanding the Components

Let us now zoom in on the components and examine the behaviours in terms of the ways they interact with each other *(section 3.1)*, followed by the detailed structure of the components *(section 3.2)*.

3.1) Behaviour

When you first run PEPO, objects are loaded by the **Database** component. Following that, the objects (such as venues) are moved into **Logic** for interaction with **GUI**. The objects are returned for storage when PEPO exits. The models below show you the interaction between the components when PEPO is initialised and closed, followed by two other models which shows you how they are incorporated.

Note -

During termination, all objects must be saved or ID conflict might occur when loading.

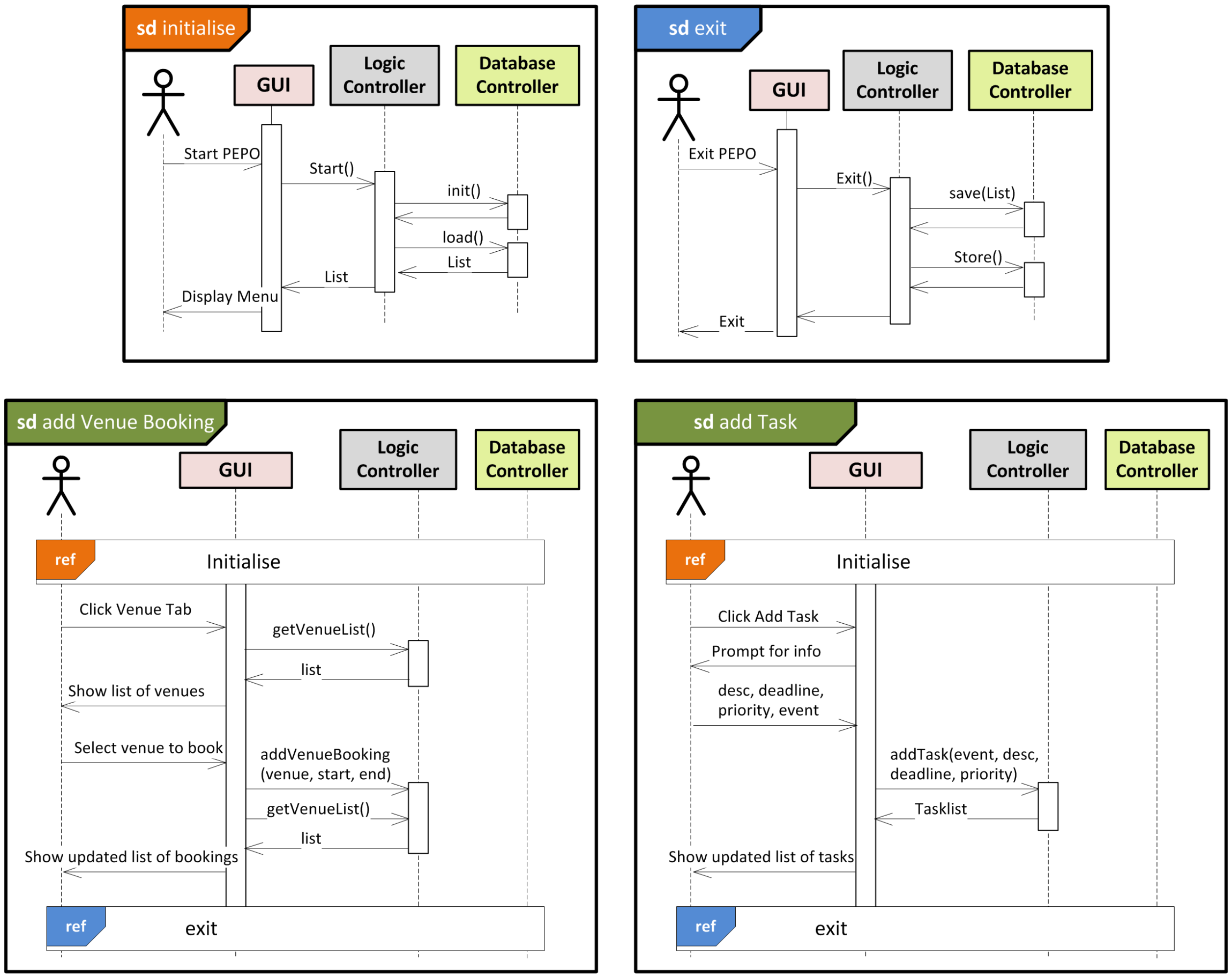


Figure 2 : PEPO Sequence Diagram

3.2) Structure

The following diagrams model the classes of the **Logic** component and **Database** component.

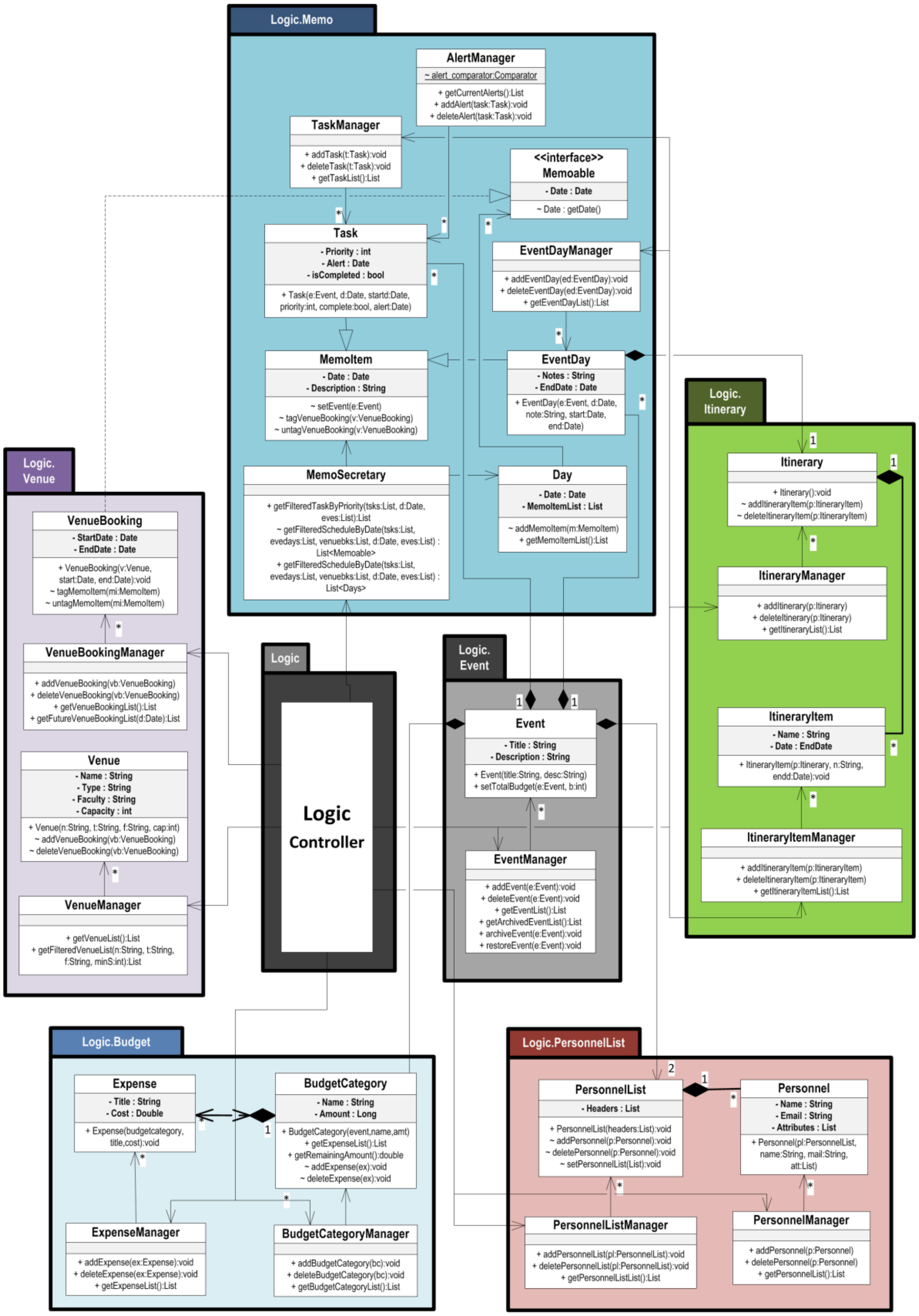

Logic.Memo
AlertManager
~ alert_comparator:Comparator
+ getCurrentAlerts():List
+ addAlert(task:Task):void
+ deleteAlert(task:Task):void
TaskManager
+ addTask(t:Task):void
+ deleteTask(t:Task):void
+ getTaskList():List
<<interface>> Memoable
- Date : Date
~ Date : getDate()
Task
- Priority : int
- Alert : Date
- isCompleted : bool
+ Task(e:Event, d:Date, startd:Date, priority:int, complete:bool, alert:Date)
EventDayManager
+ addEventDay(ed:EventDay):void
+ deleteEventDay(ed:EventDay):void
+ getEventDayList():List
MemoItem
- Date : Date
- Description : String
~ setEvent(e:Event)
~ tagVenueBooking(v:VenueBooking)
~ untagVenueBooking(v:VenueBooking)
EventDay
- Notes : String
- EndDate : Date
+ EventDay(e:Event, d:Date, note:String, start:Date, end:Date)
MemoSecretary
+ getFilteredTaskByPriority(tsks:List, d:Date, eves:List):List
~ getFilteredScheduleByDate(tsks:List, evedays:List, venuebks:List, d:Date, eves:List) : List<Memoable>
+ getFilteredScheduleByDate(tsks:List, evedays:List, venuebks:List, d:Date, eves:List) : List<Days>
Day
- Date : Date
- MemoItemList : List
~ addMemoItem(m:MemoItem)
+ getMemoItemList():List
Logic. Venue
VenueBooking
- StartDate : Date
- EndDate : Date
+ VenueBooking(v:Venue, start:Date, end:Date):void
~ tagMemoItem(mi:MemoItem)
~ untagMemoItem(mi:MemoItem)
VenueBookingManager
+ addVenueBooking(vb:VenueBooking)
+ deleteVenueBooking(vb:VenueBooking)
+ getVenueBookingList():List
+ getFutureVenueBookingList(d:Date):List
Venue
- Name : String
- Type : String
- Faculty : String
- Capacity : int
+ Venue(n:String, t:String, f:String, cap:int)
~ addVenueBooking(vb:VenueBooking)
~ deleteVenueBooking(vb:VenueBooking)
VenueManager
+ getVenueList():List
+ getFilteredVenueList(n:String, t:String, f:String, minS:int):List
Logic
Logic Controller
Logic. Event
Event
- Title : String
- Description : String
+ Event(title:String, desc:String)
+ setTotalBudget(e:Event, b:int)
EventManager
+ addEvent(e:Event):void
+ deleteEvent(e:Event):void
+ getEventList():List
+ getArchivedEventList():List
+ archiveEvent(e:Event):void
+ restoreEvent(e:Event):void
Logic. Itinerary
Itinerary
+ Itinerary():void
~ addItineraryItem(p:ItineraryItem)
~ deleteItineraryItem(p:ItineraryItem)
ItineraryManager
+ addItinerary(p:Itinerary)
+ deleteItinerary(p:Itinerary)
+ getItineraryList():List
ItineraryItem
- Name : String
- Date : EndDate
+ ItineraryItem(p:Itinerary, n:String, endd:Date):void
ItineraryItemManager
+ addItineraryItem(p:ItineraryItem)
+ deleteItineraryItem(p:ItineraryItem)
+ getItineraryItemList():List
Logic.Budget
Expense
- Title : String
- Cost : Double
+ Expense(budgetcategory, title,cost):void
BudgetCategory
- Name : String
- Amount : Long
+ BudgetCategory(event,name,amt)
+ getExpenseList():List
+ getRemainingAmount():double
~ addExpense(ex):void
~ deleteExpense(ex):void
ExpenseManager
+ addExpense(ex:Expense):void
+ deleteExpense(ex:Expense):void
+ getExpenseList():List
BudgetCategoryManager
+ addBudgetCategory(bc):void
+ deleteBudgetCategory(bc):void
+ getBudgetCategoryList():List
Logic.PersonnelList
PersonnelList
- Headers : List
+ PersonnelList(headers:List):void
~ addPersonnel(p:Personnel):void
~ deletePersonnel(p:Personnel):void
~ setPersonnelList(List):void
Personnel
- Name : String
- Email : String
- Attributes : List
+ Personnel(pl:PersonnelList, name:String, mail:String, att:List)
PersonnelListManager
+ addPersonnelList(pl:PersonnelList):void
+ deletePersonnelList(pl:PersonnelList):void
+ getPersonnelListList():List
PersonnelManager
+ addPersonnel(p:Personnel)
+ deletePersonnel(p:Personnel)
+ getPersonnelList():List
1
2
*

Figure 3 : PEPO Logic Class Diagram

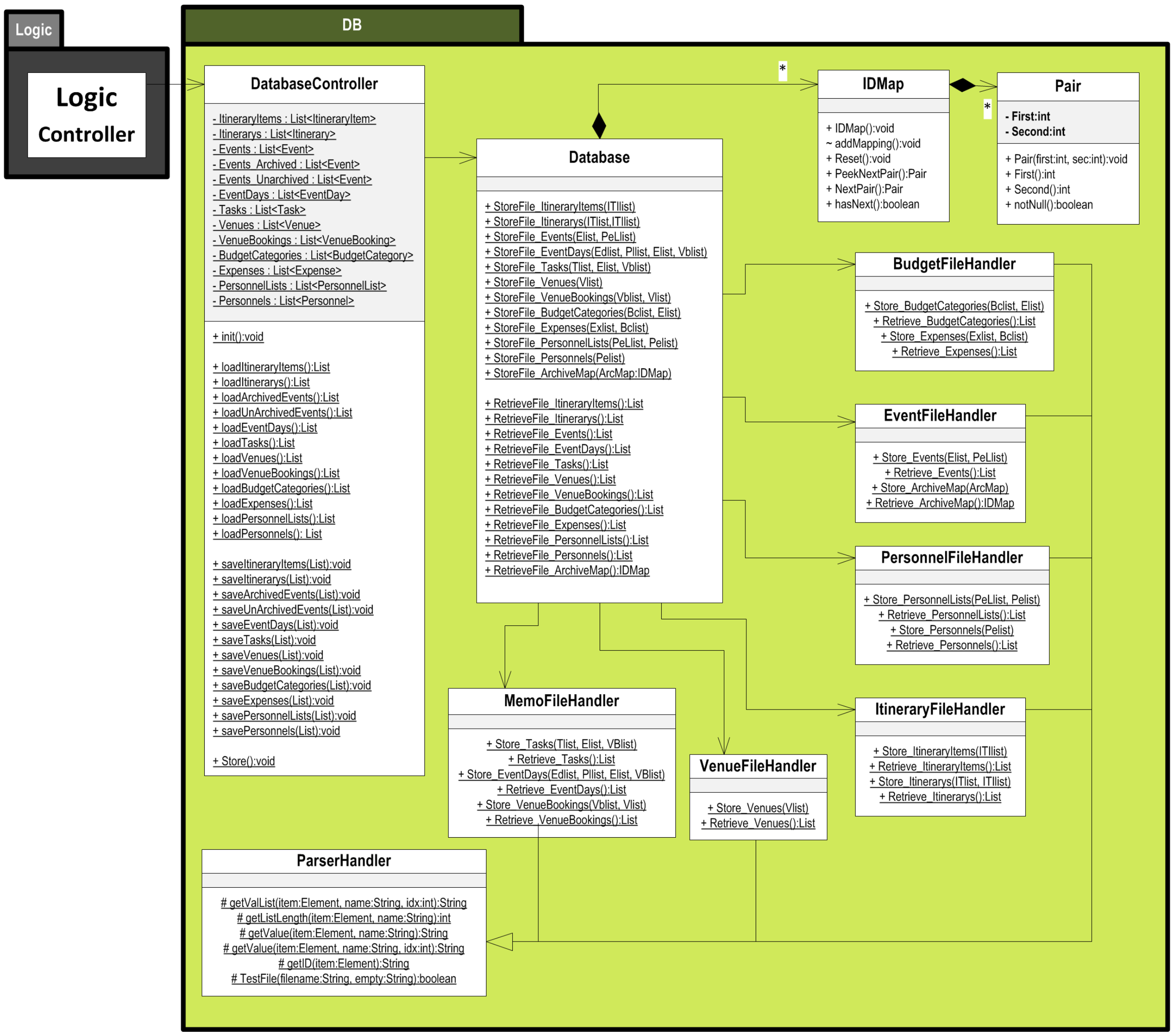

Logic
Logic Controller
DB
DatabaseController
- ItineraryItems : List<ItineraryItem>
- Itinerarys : List<Itinerary>
- Events : List<Event>
- Events_Archived : List<Event>
- Events_Unarchived : List<Event>
- EventDays : List<EventDay>
- Tasks : List<Task>
- Venues : List<Venue>
- VenueBookings : List<VenueBooking>
- BudgetCategories : List<BudgetCategory>
- Expenses : List<Expense>
- PersonnelLists : List<PersonnelList>
- Personnels : List<Personnel>
+ init():void
+ loadItineraryItems():List
+ loadItinerarys():List
+ loadArchivedEvents():List
+ loadUnArchivedEvents():List
+ loadEventDays():List
+ loadTasks():List
+ loadVenues():List
+ loadVenueBookings():List
+ loadBudgetCategories():List
+ loadExpenses():List
+ loadPersonnelLists():List
+ loadPersonnels(): List
+ saveItineraryItems(List):void
+ saveItinerarys(List):void
+ saveArchivedEvents(List):void
+ saveUnArchivedEvents(List):void
+ saveEventDays(List):void
+ saveTasks(List):void
+ saveVenues(List):void
+ saveVenueBookings(List):void
+ saveBudgetCategories(List):void
+ saveExpenses(List):void
+ savePersonnelLists(List):void
+ savePersonnels(List):void
+ Store():void
Database
+ StoreFile_ItineraryItems(ITlist)
+ StoreFile_Itinerarys(ITlist,ITIlist)
+ StoreFile_Events(Elist, PeLlist)
+ StoreFile_EventDays(Edlist, Pllist, Elist, Vblist)
+ StoreFile_Tasks(Tlist, Elist, Vblist)
+ StoreFile_Venues(Vlist)
+ StoreFile_VenueBookings(Vblist, Vlist)
+ StoreFile_BudgetCategories(Bclist, Elist)
+ StoreFile_Expenses(Exlist, Bclist)
+ StoreFile_PersonnelLists(PeLlist, Pelist)
+ StoreFile_Personnels(Pelist)
+ StoreFile_ArchiveMap(ArcMap:IDMap)
+ RetrieveFile_ItineraryItems():List
+ RetrieveFile_Itinerarys():List
+ RetrieveFile_Events():List
+ RetrieveFile_EventDays():List
+ RetrieveFile_Tasks():List
+ RetrieveFile_Venues():List
+ RetrieveFile_VenueBookings():List
+ RetrieveFile_BudgetCategories():List
+ RetrieveFile_Expenses():List
+ RetrieveFile_PersonnelLists():List
+ RetrieveFile_Personnels():List
+ RetrieveFile_ArchiveMap():IDMap
*
IDMap
+ IDMap():void
~ addMapping():void
+ Reset():void
+ PeekNextPair():Pair
+ NextPair():Pair
+ hasNext():boolean
*
Pair
- First:int
- Second:int
+ Pair(first:int, sec:int):void
+ First():int
+ Second():int
+ notNull():boolean
BudgetFileHandler
+ Store_BudgetCategories(Bclist, Elist)
+ Retrieve_BudgetCategories():List
+ Store_Expenses(Exlist, Bclist)
+ Retrieve_Expenses():List
EventFileHandler
+ Store_Events(Elist, PeLlist)
+ Retrieve_Events():List
+ Store_ArchiveMap(ArcMap)
+ Retrieve_ArchiveMap():IDMap
PersonnelFileHandler
+ Store_PersonnelLists(PeLlist, Pelist)
+ Retrieve_PersonnelLists():List
+ Store_Personnels(Pelist)
+ Retrieve_Personnels():List
ItineraryFileHandler
+ Store_ItineraryItems(ITlist)
+ Retrieve_ItineraryItems():List
+ Store_Itinerarys(ITlist, ITIlist)
+ Retrieve_Itinerarys():List
MemoFileHandler
+ Store_Tasks(Tlist, Elist, VBlist)
+ Retrieve_Tasks():List
+ Store_EventDays(Edlist, Pllist, Elist, VBlist)
+ Retrieve_EventDays():List
+ Store_VenueBookings(Vblist, Vlist)
+ Retrieve_VenueBookings():List
VenueFileHandler
+ Store_Venues(Vlist)
+ Retrieve_Venues():List
ParserHandler
# getValList(item:Element, name:String, idx:int):String
# getListLength(item:Element, name:String):int
# getValue(item:Element, name:String):String
# getValue(item:Element, name:String, idx:int):String
# getID(item:Element):String
# TestFile(filename:String, empty:String):boolean

Figure 4 : PEPO Database Class Diagram

4 Using the APIs

Now that you have an understanding of the architecture of PEPO, in the next few sections, you can now go to the codes themselves. The Logic and the Database component carry a set of **Application Programming Interfaces (APIs)** which can be used simply by calling the component **controller** class. Read further for more details on the use of APIs.

Note -

For the full list of Logic and Database APIs, please refer to the *Appendix C - API Library. (section 9)*

If you are a GUI Designer,
you have to make use of the APIs provided by the **Logic** component through its controller class, `LogicController`. The following is a partial list of APIs of the `LogicController`:

return type	identifier
static ArrayList<Event>	`getEventList()` Get a list of events.
static ArrayList<Task>	`getFilteredDayScheduleByDate(Date date, ArrayList<Event> eventlist, boolean venuebooking)` Get a list of days that contains tasks, days of event and venue bookings that belong to one of the events in `eventlist`.
static void	`addEvent(String name, String description)` Add an event.
static void	`deleteEvent(Event event)` Delete `event`.

For instance, to retrieve a list of events, you only need to call `LogicController.getEventList()` which returns an `Arraylist` of events which you can then display on the user interface. The same applies to the rest.

If you are working on the Logic component,
you have to communicate with the **Database** component by its controller class `DatabaseController`. The partial list of APIs of `DatabaseController` is as follows :

return type	identifier
static void	`init()` Load all objects from files into memory. Must be called once before any load/save functions can be used.
static ArrayList<Event>	`loadUnArchivedEvents()` Retrieve list of all unarchived events.
static void	`saveUnArchivedEvents(ArrayList<Event> list)` Save list of all unarchived events in memory.
static void	`store()` Stores all saved objects into files.

To load a list of events, you can simply write `DatabaseController.loadUnArchivedList()`. However, `init()` must be called before any of the save/load operations are functional.

Note -
It is mandatory to save all objects before `DatabaseController.store()` can be used.

5 Making sense of the Code

The following is an example of a complex algorithm that might be worth looking into.

The algorithm takes place in the **Logic** component. The objective is to create a list of `Day` objects which represents a single day in the schedule as follows :

Day	Event	Description	Time	Venue
8/3/2012	BootCamp	Camp	9am-11:59pm	LT27
	TechTalk	Talk#04	4pm-5pm	SR2
		[VenueBooked]	1pm-11.59pm	LT27
		[VenueBooked]	4pm-6pm	SR2
9/3/2012	BootCamp	Camp	12am-5pm	LT27
		[VenueBooked]	12am-6pm	LT27

Each `Day` object contains a unique `Date` and a list of `Memoable`[2] (can be a `Task`[1], `EventDay`[6] or `VenueBooking`) objects that falls on that date. These `Day` objects need to be sorted in order of their `Date`.

These are the steps to achieve the objectives

1. Call `getFilteredScheduleByDate()` in the `MemoSecretary` class to retrieve an `ArrayList` of `Memoable` objects that fall into the events and date that the users want to display in their schedule.

Event	Description	Time	Venue
BootCamp	Camp	8/3/2012 9am - 9/3/2012 5pm	LT27
TechTalk	Talk#04	8/3/2012 4pm-5pm	SR2
	[VenueBooking]	8/3/2012 1pm - 9/3/2012 6pm	LT27
	[VenueBooking]	8/3/2012 4pm-6pm	SR2

2. We hash the `Memoable` object to each `Day` that the object falls on. Note that a `Task` object only needs to be mapped to a single `Day` since it contains a deadline instead of a starting date and ending date. A `TreeMap` contains `<Date, Day>` is used to speed up the process and make sure that each `Day` is unique. Do take note that the hour, minute, seconds and milliseconds for the `Day` object's `Date` has to be set to 0 in order to ensure that the time of the day does not affect hashing to the correct `Day` object.

   A linear model of the result of hashing is as follows :

[1, 2, 6] *please refer to Appendix A - Glossary (Section 8) for a list of terms.*

Day	Memoable	Day object index
8/3/2012	Camp	0
9/3/2012	Camp	1
8/3/2012	Talk#04	0
8/3/2012	[VenueBooked] (LT27)	0
9/3/2012	[VenueBooked] (LT27)	1
8/3/2012	[VenueBooked] (SR2)	0

3. Lastly, the `Comparable` interface is also implemented in the `Day` class so that the `Collection.sort()` in the Java Collections Framework is used to sort the `Day` objects in chronological order.

Day	Memoable
8/3/2012	Camp
	Talk#04
	[VenueBooked] (LT27)
	[VenueBooked] (SR2)
9/3/2012	Camp
	[VenueBooked] (LT27)

4. The result of the sorted `Day` objects with its own list of `Memoable` objects would then be returned in the form of `ArrayList<Day>`.

The partial code for hashing a single `EventDay` :

```
if (cur instanceof EventDay ){
        Date ending = ((EventDay)cur).getEndDate();
        // While have not hashed until the end date
        while (curDate.getTime().before(ending)){  // curDate’s time already set to 12am
                if (schedule.containsKey(curDate.getTime())) {       // If Day already exist
                        // Simply append it to the ArrayList
                        Day today = schedule.get(curDate.getTime());
                        today.addMemoItem(cur);
                } else { // Else, create a new entry in the TreeMap
                        Day newday = new Day(curDate.getTime());
                        newday.addMemoItem(cur);
                        schedule.put(curDate.getTime(),newday);
                }
                // Increase date by 1 day (24 hours)
                curDate.add(GregorianCalendar.DATE, 1);
        }
}
```

6 Writing your Code

Now that you know the overall design of PEPO, and understand the use of APIs and complex algorithms in PEPO, you are now ready to write your code! Writing of code varies across different components. Refer to section 6.1 for writing to **Logic** and 6.2 for writing to **Database**.

In addition, PEPO consists of a separate `Exporter` class which interfaces with the Logic component to generate a report in `.html` format. To learn how to use the Exporter class, refer to section 6.3.

6.1) Logic

Your job is to process the data from either the **GUI** or **Database** and convert them to meaningful information to display on screen. What you write depend largely on what you want to convey, and thus, there is no fixed format for coding. However, there are rules that need to be followed in order to preserve the integrity of the design.

Let us look at this example where an expense is added in the class `LogicController`.

```
public void addExpense(BudgetCategory b, String t, double c)
{
        Expense ex = new Expense(b,t,c);             // Construct Expense object
        expenseM.addExpense(ex);                     // Add into ExpenseManager

        // Checks if expense is tagged under any budget category
        // If not, add it to the 'uncategorized' category
        if (b != null) budgetcategoryM.addExpense(b,ex);
        else budgetcategoryM.addExpense(b.getEvent().getUncategorizedBudget(), ex);
}
```

The job of the controller class `LogicController` is to relay the function to an object of `BudgetManager` class. This is similar for all of its functions. The primary goal is is to call the correct manager to handle the operation. The controller class is not allowed to store any object or manipulate the objects directly.

For a detailed look at the flow of data in **Logic**, please refer to *Appendix C - Sample Code (Section 10.1).*

6.2) Database

Your primary role would be the loading and saving of objects. Currently, these operations use the **DOM Parser** library to store and load from XML files. Let us look at how a list of objects belonging to the class `venue` is stored.

1. Initialise the document to store the object and declare the root element. In this case, the root element is `<Venue>`. The next line binds the root element to the document.

```
Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
Element root = doc.createElement("Venue");
doc.appendChild(root);
```

2. You can now create a loop to iterate through each `venue` object that is stored inside the list `Vlist`. The attributes of the object are first grouped under the element `<Item>`. Each `<Item>` element will be appended to the root element defined in step 1.

```
for(int i=0; i<Vlist.size(); i++) {
       Element item = doc.createElement("Item");
        root.appendChild(item); }
```

3. So far you have stored a list of `venue` objects but none of their attributes. For identification purpose, each object comes with an `ID` which is generated based on its position in `Vlist`. You can now store the object's ID under its `<Item>` element.

```
for(int i=0; i<Vlist.size(); i++) {
       ...
        Attr id = doc.createAttribute("ID");
        id.setValue(Integer.toString(i));
        item.setAttributeNode(id);
        … }
```

4. You can now add an element `<name>` which stores the name of the `Venue`. The same method can be used to store anything of value. In this example, the Type, Faculty and Capacity of a `Venue` are also stored but the code is not shown.

```
for(int i=0; i<Vlist.size(); i++) {
       ...
        Element name = doc.createElement("Name");
        name.appendChild(doc.createTextNode(Vlist.get(i).getName()));
        item.appendChild(name);
        … }
```

5. After every `<Item>` element is stored, you have to transform them into a XML file.

```
DOMSource src = new DOMSource(doc);
StreamResult result = new StreamResult(new File(Vfile));
TransformerFactory.newInstance().newTransformer().transform(src, result);
```

The resulting XML file should be similar to the following :

```
<Venue>
  <Item ID="0">
    <Name>Lee Kong Chian Art Museum</Name>
    <Type>Museum</Type>
    <Faculty>Others</Faculty>
    <Capacity>100</Capacity>
  </Item>
</Venue>
```

For the full code on storing and loading a venue complete with annotations, please refer to *Appendix C - Sample Code (Section 10.2 & 10.3).*

6.3) Exporter

The `Exporter` class is used primarily for exporting data into `.html` format. To understand how it works, some basic knowledge of HTML and CSS is needed. Some functions are already written to ease the process of translation which you will be introduced to in the steps below. Let us look at how a table of tasks are generated in the file.

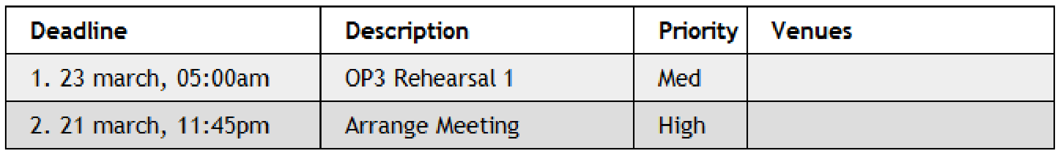
1. Write the heading using `writeCSS(String class_selector, String value)`. 3 different CSS classes are predefined - `text1, text2, text3`, in ascending order of their font size. It is not advisable to introduce any other CSS classes for consistency purposes. You are required to use `text3` for the title. The next statement simply writes writes a blank line.

```
writeCSS("text3", "Tasks to Do");
write("<br>");
```

2. You can now create a table to display the data. To do so, you must first call the function `openTable()`, followed by `closeTable()` after all the data are added into it.

```
openTable();
...
closeTable();
```

3. In the table, you can add a row of data. This works similarly to the opening and closing of tables. There are two ways to add a row depending on your purpose.



Deadline	Description	Priority	Venues
1. 23 march, 05:00am	OP3 Rehearsal 1	Med	
2. 21 march, 11:45pm	Arrange Meeting	High	

`openRowHeader()` creates a row similar to the topmost row in the screenshot above.
`openRow(int rownum)` creates a row similar to the other 2 rows.

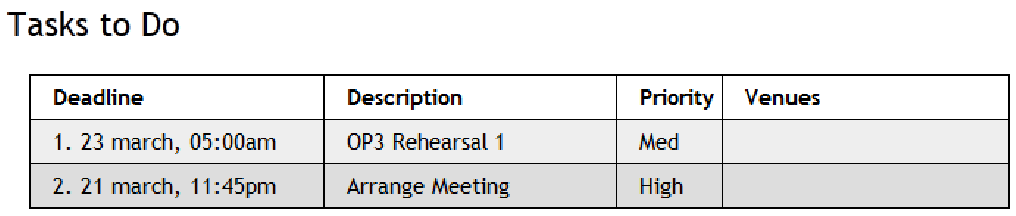
```
openTable();

       openRowHeader();
       ...
       closeRow();

      for(int i=0; i<Tasks.size(); i++) {
              openRow(i);
             ...
              closeRow();
       }

closeTable();
```

4. After opening the table and the row, you can start writing data to each cell in the row. To do so, you can call `writeCell(String class_selector, String value)`. You can also specify the width of the cell by calling `writeCell(String class_selector, int cellwidth, String value)`.

```
openTable();

       openRowHeader();
              writeCell("text1", 30, "<b>Deadline</b>");
              writeCell("text1", 30, "<b>Description</b>");
              writeCell("text1", 8, "<b>Priority</b>");
              writeCell("text1", "<b>Venues</b>");
       closeRow();

      for(int i=0; i<Tasks.size(); i++) {
              openRow(i);
             ...
              closeRow();
       }

closeTable();
```

5. After closing the table, you can call `write("<br/></br>")` to print 2 blank lines for formatting purposes.

The content of the `.html` file will be similar to the following :



Tasks to Do

Deadline	**Description**	**Priority**	**Venues**
1. 23 march, 05:00am	OP3 Rehearsal 1	Med	
2. 21 march, 11:45pm	Arrange Meeting	High	

For the full code on exporting tasks, please refer to *Appendix C - Sample Code (Section 10.4)*.

7 Testing

7.1) Using TextUI

To facilitate testing, a command-line user interface (**TextUI**) was implemented which can perform any function in `LogicController` by writing a line of command. To use it, simply double click on `PEPO TextUI.exe` located in the PEPO folder.

There are 3 types of commands that can be entered into **TextUI** :

1. **Manual Test** –
   Call a specific function in `LogicController` and print its output.

2. **Validation Unit Test** –
   Test and verify the output of a list of 'Manual Test' commands against an expected output.

3. **Validation System Test** –
   Test and verify the output of a list of 'Manual Test' commands against an expected output and storage data.

Caution! -
For command type 2 and 3, all existing data will be removed and replaced with a dummy data in order for test output to be consistent.

The format of the **Manual Test** is as follows:

```
[Function] [Parameter1, Parameter2, ..]
```

Different parameters are separated by a comma. An example of which is :

```
AddEvent Camp,Orientation Camp for Freshmen
```

The command adds an event which consists of the title "`Camp`" and description "`Orientation Camp for Freshmen`" into PEPO. The example is valid and therefore no result is displayed. In situations where the data are not permissible, the *expected output* should be an error message.

You can also execute multiple command lines at once by writing them in a `.txt` file and directing **TextUI** to it with the following format:

```
BatchCommand [Filename]
```

Note -
For instructions to use **Validation Unit Test** commands, refer to Unit Testing *(Section 7.2)*.
For instructions to use **Validation System Test** commands, refer to System Testing *(Section 7.3)*.

7.2) Unit Testing (White Box)

Your goal is to ensure that each subsystem of PEPO is correct and outputs the correct error message if it fails. You can use a **Validation Unit Test** in **TextUI** to automate the unit testing process. The format of such a test is as follows:

```
PepoTest Unit;[Test Directory name]
```

The test directory must be stored inside the folder `PEPOTest` and contains the following:

1. `testcommands.txt` –
   the list of 'Manual Test' commands directed to the target unit and ends with "ENDOFTEST".

2. `expectedoutput.txt` –
   the correct output to be generated when the commands in `textcommands.txt` are executed.

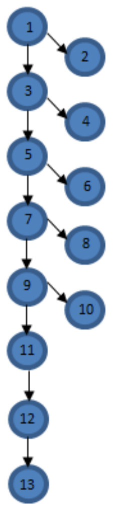
3. `initialData directory` –
   contains data (in .XML files) that is loaded before execution of the commands in `textcommands.txt.`

Upon executing the test, an output will be generated and compared with the `expectedoutput.txt` where the correctness is determined.

The following test cases will cover the more essential functions of PEPO, which includes functions related to memo and events. *Basis Path Testing* is used to ensure that every branch is tested with a suitable test case. The path diagram will be serving as a guide to the flow of the code.

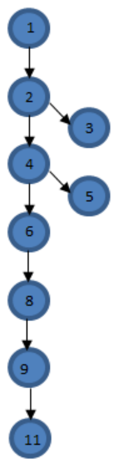
`addBudgetCategory`

Case#	`event`	`name`	`budget`	**Basis Path**	**Output**
1		food	20	1-2	BudgetCategory is null
2	0	food	-20	1-3-4	Amount cannot be negative
3	0	food	999999901	1-3-4-6	Amount too large
4	0		20	1-3-5-7-8	Name is invalid
5	0	~>200 chars		1-3-5-7-9	Name too long
6	0	food	20	1-3-5-7-9-11-12-13	Success!



`addEvent`

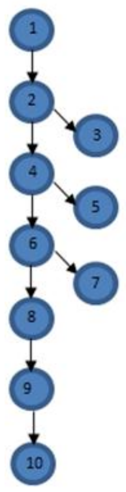
Case#	`name`	`description`	**Basis Path**	**Output**
1		sports	1-2-3	Title is invalid
2	~>200 chars	sports	1-2-4-5	Title too long



3	facultygames	sports	1-2-4-6-7-8-9-10-11-12	Success!

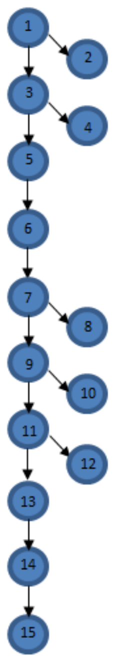
addEventDay

Case#	event	descri-pttion	notes	start -date	end -date	Basis Path	Output
1	0		meeting	201103051700	201103052100	1-2-3	Description is invalid
2	0	~>200 chars	meeting	201103051700	201103052100	1-2-4-5	Description too long
3	0	monthly review	meeting	201103052100	201103051700	1-2-4-6-7	StartDate is after Enddate
4	0	monthly review	meeting	201103051700	201103052100	1-2-4-6-8-9-10	Success!



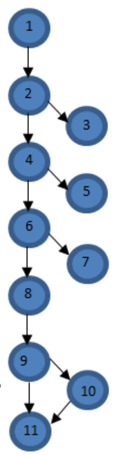
addExpense

Case#	budgetcategory	title	cost	Basis Path	Output
1		milo	26	1-2	BudgetCategory is null
2	0;0		26	1-3-4	Title is invalid
3	0;0	milo	-26	1-3-5-6-7-8	Cost cannot be negative
4	0;0	milo	999999901	1-3-5-6-7-9-10	Cost too large
5	0;0	~>200 chars	26	1-3-5-6-7-9-11-12	Title too long
6	0;0	milo	26	1-3-5-6-7-9-11-13-15-16	Success!



addTask

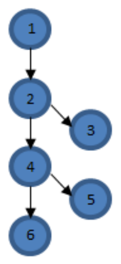
Case#	event	descripti-on	Date	prior-ity	Basis Path	Output
1	0		201103052100		1-2-3	Description is invalid
2	0	~>200 chars	201103052100		1-2-4-5	Description too long
3	0	meeting	201103052100	-1	1-2-4-6-7	Priority out of range
4	0	meeting	201103052100	3	1-2-4-6-7	Priority out of range



5	0	Meeting	201103052100	1	1-2-4-6-8-9-10-11	Success!
6		Meeting	201103052100	1	1-2-4-6-8-9-11	Success!

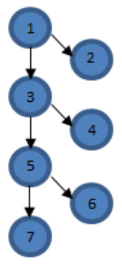
`setBudgetCategoryAmount`

Case#	`budgetcategory`	`amount`	Basis Path	Output
1	0;0	-3	1-2-3	Amount cannot be negative
2	0;0	999999901	1-2-4-5	Amount too large
3	0;0	40	1-2-4-6	Success!



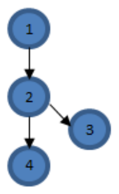
setBudgetCategoryName

Case#	`budgetcategory`	`name`	Basis Path	Output
1		logistics	1-2	BudgetCategory is null
2	0;0		1-3-4	Name is invalid
3	0;0	>~200chars	1-3-5-6	Name too long
4	0;0	logistics	1-3-5-7	Success!



setEventDayDate

Case#	`eventday`	`startdate`	`enddate`	Basis Path	Output
1	0;0	201103052100	201103051700	1-2-3	StartDate is after Enddate
2	0;0	201103051700	201103052100	1-2-4	Success!



7.3) System Testing (Black Box)

7.3.1) Regression Testing

Regression Testing involves finding glitches in the entire software whenever any changes are made to it, which is very commonplace in the final phase. The **TextUI** can be used to automate this process. The format of a **Validation System Test** is as follows:

```
PepoTest System;[Test Directory name]
```

The test directory must be stored inside the folder `PEPOTest` and contains the following:

1. `testcommands.txt` –
   the list of 'Manual Test' commands directed to PEPO and ends with "ENDOFTEST".

2. `expectedoutput.txt` –
   the correct output to be generated when the commands in `textcommands.txt` are executed.

3. `initialData directory` –
   contains data (in .XML files) that is loaded before execution of the commands in `textcommands.txt.`

4. `ExpectedData directory` –
   contains data (in .XML files) that is expected to be stored after execution of the commands in `textcommands.txt.`

The test is very similar to **Validation Unit Test**. The key difference lies in the fact that the commands are targeted towards the whole system instead of a single unit. Both the **Logic** component and the **Database** component are tested and verified by matching the output with `expectedoutput.txt` and the resulting `.XML` files with the ones in the `ExpectedData` directory.

There is an existing system test data that you can use by simply writing :

```
PepoTest System;System-v0.2
```

7.3.2) Graphical User Interface Testing

Your job here is to locate product level defects and to ensure that it fulfils customer requirements. A well tested product can increase confidence in your clients.

Memo Tab

	What to test	How to test	Input data	Expected Result
1	Open Addtask	Click on '+' icon		Pop-out Addtask
2	Funtionality of Addtask	Input all boxes	Des: meeting Priority: Med Event: Deadline: Mar 30, 2012 2pm 00	Under memo, Meeting will be reflected both under your task and right column when date selected is not after Mar 30, 2012
3	Error Handling test: Addtask	Input all boxes	Des: <input more than 200 words> Priority: Med Event: Deadline: Mar 30, 2012 2pm 00	Warning pop-up: Description too long…
4	Error Handling test: Addtask	Do not write anything under description	Des: Priority: Med Event:	Warning pop-up: Description is invalid…

			Deadline: Mar 30, 2012 2pm 00	
5	Open edit Task	Click on the task to edit, new icons will appear along the box. Click on the 'pencil' icon		Pop-up with task to be edited
6	To edit Task	Make changes to description	Des:meeting2	Changes reflected under memo, Meeting2 will be reflected both under your task and right column when date selected is not after Mar 30, 2012
7	To setAlert	In pop-up addTask/edit task, click on no alert to enable alert	Alert: : Mar 30, 2012 2pm 30	At 230pm, a pop-up will be displayed at the top left hand corner of the monitor
8	Multiple tasks reflected under memo	Add another task (follow testcase #2 changing only the des)	Des: family time Priority: Med Event: Deadline: Mar 30, 2012 2pm 00	Under Your Tasks and 30 Mar(Fri), family time will be under meeting2
9	Piority check unders memo	Edit family time with priority set to High	Des: family time Priority: high Event: Deadline: Mar 30, 2012 2pm 00	Under Your Task, Family time will be placed above meeting2

Event Tab

	What to test	How to test	Input data	Expected Result
1	Open AddEvent	Click on '+' icon		Pop-out Addtask
2	Create Event	Open Addevent and input all boxes	Title: Nike 3 on 3 Des: 3 on 3 basketball match	Both inputs will be displayed on the left
3	Error Handling test: Add event	Open Addevent and do not input title	Title: Des: 3 on 3 basketball match	Pop-up: Oops, title needed…
4	No description test	Open Addevent and do not input des	Title: Nike 3 on 3 Des:	Only title will be reflected
5	Adding multiple Events	Open Addevent and input all boxes	Title: CloudAsia Des: Cloud computing conference	Title will be reflect under events. Click on it will reflect description below
6	Edit event	Click on nike 3 on 3 and click 'pencil' sign above the box	Des: annual basketball 3 on 3	New description will be displayed on the left
7	Delete event	Click on CloudAsia and click 'bin' sign above the box. Pop-up will prompt deletion.		CloudAisa will not be reflected on the left

		click yes to delete		
8	Add day of event	Click '+' under day of event	Title: team1 vs team2 Notes: first 5 to win	Title will be displayed on the centre screen and notes will be reflected at the bottom of the program
9	Adding/editing total budget	Under budget tab, click on the icon beside total budget	Total Budget: $2000	Total budget updated to $2000
10	Insert Budget category	Under edit budget, click on the '+' icon	Name: logistics Amount($): 200	Logistics will be reflected in the box and unallocated will be adjusted
11	Add Description	Under budget tab, input all boxes	Category: Logistics Desc: Milo Cost($):10	Milo will be reflected under logistics on the left box
12	Edit items	Under budget list, click on desired item and click 'pencil' icon. Make necessary changes	Desc:Horlicks Cost($):20	Horlicks will be reflected instead of Milo. Cost will also be amended
13	Error Handling test: Edit items	When editing item, leave description blank		Pop-up: Title is invalid…
14	Delete Items	Under budget list, click on the desired item and click on 'bin' icon. deletion will be prompted		Milo will be deleted from the list
15	Input attributes in contacts	Click on 'pencil' icon, 'gear' icon. pop-up 'set participant attributes'. click '+' to add	Rename 'new attributes' to 'email add.'	Attribute 'email add.' is added to the right column
16	To delete attribute	Click on desired attribute and click on 'X' icon		Attribute will be deleted
17	Insert information	Double click on desired box to be filled in.		Color change from yellow to white
18	Editing Information	Double click on desired box edit	Lim	Lim will be replaced in the box
19	Import excel	Click on 'document' icon next to 'participant'. select excel to be imported		Duplication of the excel information will be reflected on screen

Venue Tab

	What to test	How to test	Input data	Expected Result
1	Filter type	Under type, Choose computer lab		12 venues will be filtered out with type: computer lab
2	Filter faculty	Under faculty, choose computing		28 venues will be filtered out from faculty: computing
3	Filter type and faculty	Type: discussion room Faculty: computing		8 venues that are from computing with type discussion room
4	Filter by name	Under name box	L	All venues with 'L' in the name of venue will be filtered

5	Filter by name	Under name box	I	All venues with 'I' in the name of venue will be filtered
6	Filter name and type		Name: L Type: Computer lab	All venues with 'L' in the name of venue and with type: Computer lab will be filtered (12 results)
7	Filter minimum capacity		Minimum capacity:500	9 results with minimum capacity>500
8	Filter by name, type, faculty, minimum capacity	Input the following data	Name: L Type: Computer lab Faculty: Arts and Social Sciences Minimum capacity: 20	4 venues will be reflected with the given criteria
9	Book a venue	Click on the CL1 and pop-up will prompt for date and time	Start: Apr 8 2012, 8am End: Apr 8 2012, 9am	Venue confirmed will be reflected under 'Your Venue Bookings'
10	Edit booked venue	Click on venue to change (line will change colour). Click on 'pin' icon to select eventday with the corresponding time		Go to event tab and under eventday selected, venue will be shown as previously tagged venue. Under memo tab, eventday will be updated with venue included

8 *Appendix A* - Glossary

1. **Memo**
   A system that resembles a personal notebook that is used to record tasks to be done and what is going on in the days to come.

2. **Memoable**
   An item in the memo schedule. This can refer to a venue booking, a task or a day of an event.

3. **Task**
   An activity that needs to be done in the future. This can be used refer to meeting, contacting someone, managing people and so on.

4. **Budget Category**
   A group of expenses. Common examples of budget categories are Logistics, Transport and so

on. A budget can be allocated for any budget category.

5. **Expense**
   An object or person that needs to be purchased or hired for the purpose of the event. This may include props for a play, hiring a trainer, food and so on.

6. **EventDay**
   A day of an event. This can include “Preliminary Round”, “Finals” of a sports event or simply “Day 1”, “Day 2” and “Day 3” for a camp.

---

9 *Appendix B* - API Library

9.1) `LogicController` API

return	identifier
`static void`	`addBudgetCategory(Event event, String name, long budget)` Create a new BudgetCategory with the specified inputs and adds it to the specified Event.
`static Event`	`addEvent(String name, String description)` Create a new Event with the specified inputs.
`static EventDay`	`addEventDayWithItinerary(Event event, String title, String notes, Date startdate, Date enddate, Itinerary pl)` Create and add a new EventDay with the specified inputs to the specified event with itinerary.
`static void`	`addExpense(BudgetCategory budgetcategory, String title, long cost)` Create a new Expense with the specified inputs and add it to the specified BudgetCategory.
`static ItineraryItem`	`addItineraryItem(Itinerary itinerary, String name, Date startdate, Date enddate)` Create and adds an ItineraryItem to the specified Itinerary
`static void`	`addPersonnel(PersonnelList personnellist, String name, String email, ArrayList<String> attributes)` Create and adds a Personnel to the specified PersonnelList.
`static Task`	`addTask(Event event, String description, Date date, int priority)` Create a new Task with the specified inputs and add it to the specified Event.
`static void`	`addTaskAlert(Task task, Date alert)` Add an Alert to a specified Task with the specified Date.
`static void`	`addVenueBooking(Venue venue, Date startdate, Date enddate)` Create and add a new VenueBooking with the specified inputs.

static void	`archiveEvent(Event event)` Archive an Event.
static void	`deleteBudgetCategory(BudgetCategory budgetcategory)` Delete the specified BudgetCategory.
static void	`deleteEvent(Event event)` Delete the specified Event.
static void	`deleteEventDay(EventDay eventday)` Delete the specified EventDay.
static void	`deleteExpense(Expense expense)` Delete the specified Expense.
static void	`deleteHelperList(Event event)` Delete the helper list of a specified Event.
static void	`deleteItineraryItem(ItineraryItem itineraryitem)` Delete the specified ItineraryItem.
static void	`deleteParticipantList(Event event)` Delete the participant list of a specified Event.
static void	`deletePersonnel(Personnel personnel)` Delete the specified Personnel.
static void	`deleteSponsorList(Event event)` Delete the sponsor list of a specified Event.
static void	`deleteTask(Task task)` Delete the specified Task.
static void	`deleteTaskAlert(Task task)` Delete the Alert of a specified Task.
static void	`deleteVendorList(Event event)` Delete the vendor list of a specified Event.
static void	`deleteVenueBooking(VenueBooking venuebooking)` Delete the specified VenueBooking.
static void	`exit()` Exit the LogicController.
static ArrayList<Event>	`getArchivedEventList()` Get a list of archived events.
static ArrayList<Task>	`getCurrentAlerts()` Get tasks which alarm is activated.
static ArrayList<Event>	`getEventList()` Get a list of events.
static ArrayList<Day>	`getFilteredDayScheduleByDate(Date date, ArrayList<Event> eventlist, boolean venuebooking)` Get a list of days that contains tasks, days of event and venue bookings that belong to one of the events in `eventlist`.
static ArrayList	`getFilteredTaskByPriority(Date date, ArrayList<Event> eventlist)` Get a list of tasks sorted according to their priority.

`<Task>`	
`static ArrayList <Venue>`	`getFilteredVenueList(String name, String type, String faculty, int minCapacity)` Get a list of venues which are filtered by `name`, `type`, `faculty` and a capacity that is more than `minCapacity`.
`static ArrayList <VenueBoo king>`	`getFutureVenueBookingList(Date date)` Get a list of venue bookings that happen after `date`.
`static ArrayList <VenueBoo king>`	`getVenueBookingList()` Get a list of venue bookings in all of the venues.
`static ArrayList <String>`	`getVenueFacultyList()` Get a list of faculties in all of the venues.
`static ArrayList <Venue>`	`getVenueList()` Get a list of venues.
`static ArrayList <String>`	`getVenueTypeList()` Get a list of types in all of the venues.
`static void`	`sendAnnouncement(String username, String password, String subject, String body, ArrayList<PersonnelList> pl)` Send an email announcement to every personnel in the personnel lists.
`static void`	`setBudgetCategoryAmount(BudgetCategory budgetcategory, long amount)` Replace the amount of a specified BudgetCategory to the specified amount.
`static void`	`setBudgetCategoryName(BudgetCategory budgetcategory, String name)` Replace the name of a specified BudgetCategory to the specified name.
`static void`	`setCompleted(Task task, boolean completed)` Replace the completion status of a specified Task to the specified status.
`static void`	`setEventDayDescription(EventDay eventday, String description)` Replace the description of a specified EventDay to the specified description.
`static void`	`setEventDayDate(EventDay eventday, Date enddate)` Replace the start date and end date of a specified EventDay to the specified dates.
`static void`	`setEventDayNotes(EventDay eventday, String notes)` Replace the notes of a specified EventDay to the specified notes.
`static void`	`setEventDescription(Event event, String description)` Replace the description of the specified Event to the specified description.
`static void`	`setEventTitle(Event event, String title)` Replace the title of the specified Event to the specified title.
`static void`	`setEventTotalBudget(Event event, long amount)` Replace the total budget of the specified Event to the specified amount.
`static void`	`setExpenseBudgetCategory(Expense expense, BudgetCategory budgetcategory)` Replace the BudgetCategory of a specified Expense to the specified BudgetCategory.
`static`	`setExpenseCost(Expense expense, long cost)`

void	Replace the cost of a specified Expense to the specified cost.
static void	`setExpenseTitle(Expense expense, String title)` Replace the title of a specified Expense to the specified title.
static void	`setHelperList(Event event, ArrayList<String> headers)` Set the helper list headers of a with the specified headers.
static void	`setParticipantList(Event event, ArrayList<String> headers)` Set the participant list headers of a with the specified headers.
static void	`setPersonnelAttribute(Personnel personnel, String name, String email, ArrayList<String> attributes)` Set the name, email and attributes of the specified Personnel.
static void	`setSponsorList(Event event, ArrayList<String> headers)` Set the sponsor list headers of a with the specified headers.
static void	`setTaskDate(Task task, Date date)` Replace the date of a specified Task to the specified date.
static void	`setTaskDescription(Task task, String description)` Replace the description of a specified Task to the specified description.
static void	`setTaskEvent(Task t, Event event)` Replace the event of a specified Task to the specified event.
static void	`setTaskPriority(Task task, int priority)` Replace the priority of a specified Task to the specified priority.
static void	`setVendorList(Event event, ArrayList<String> headers)` Set the vendor list headers of a with the specified headers.
static void	`start()` Start the LogicController.
static void	`tagEventDayVenueBooking(EventDay eventday, VenueBooking venuebooking)` Tag a specified VenueBooking from a specified EventDay.
static void	`tagTaskVenueBooking(Task task, VenueBooking venuebooking)` Tag a specified VenueBooking to a specified Task.
static void	`unArchiveEvent(Event event)` Unarchive an Event.
static void	`untagEventDayVenueBooking(EventDay eventday, VenueBooking venuebooking)` Un-tag a specified VenueBooking from a specified EventDay.
static void	`untagTaskVenueBooking(Task task, VenueBooking venuebooking)` Un-tag a specified VenueBooking from a specified Task.

9.2) Database API

return	identifier
static void	`init()` Load all objects from files into memory. Must be called once before any load/save functions can be used.
static ArrayList<Event>	`loadArchivedEvents()` Retrieve list of all archived events.

static ArrayList <BudgetCategory>	loadBudgetCategories() Retrieve list of all budget categories.
static ArrayList<EventDay>	loadEventDays() Retrieve list of all days of event.
static ArrayList<Expense>	loadExpenses() Retrieve list of all expenses.
static ArrayList <ItineraryItem>	loadItineraryItems() Retrieve list of all itinerary items.
static ArrayList <Itinerary>	loadItinerarys() Retrieve list of all itineraries.
static ArrayList <PersonnelList>	loadPersonnelLists() Retrieve list of all personnel lists.
static ArrayList <Personnel>	loadPersonnels() Retrieve list of all personnels.
static ArrayList<Task>	loadTasks() Retrieve list of all tasks.
static ArrayList<Event>	loadUnArchivedEvents() Retrieve list of all unarchived events.
static ArrayList <VenueBooking>	loadVenueBookings() Retrieve list of all venue bookings.
static ArrayList<Venue>	loadVenues() Retrieve list of all venues.
static void	saveArchivedEvents(ArrayList<Event> list) Save list of all archived events in memory.
static void	saveBudgetCategories(ArrayList<BudgetCategory> list) Save list of all budget categories in memory.
static void	saveEventDays(ArrayList<EventDay> list) Save list of all days of event in memory.
static void	saveExpenses(ArrayList<Expense> list) Save list of all expenses in memory.
static void	saveItineraryItems(ArrayList<ItineraryItem> list) Save list of all itinerary items in memory.
static void	saveItinerarys(ArrayList<Itinerary> list) Save list of all itineraries in memory.
static void	savePersonnelLists(ArrayList<PersonnelList> list) Save list of all personnel lists in memory.
static void	savePersonnels(ArrayList<Personnel> list) Save list of all personnels in memory.
static void	saveTasks(ArrayList<Task> list) Save list of all tasks in memory.
static void	saveUnArchivedEvents(ArrayList<Event> list) Save list of all unarchived events in memory.
static void	saveVenueBookings(ArrayList<VenueBooking> list)

	Save list of all venue bookings in memory.
static void	Store() Stores all saved objects into files.

9.2) GUIUpdater API

return	identifier
ArrayList<Event>	getEventFilterTableSelectedEvent() Get the unfiltered events for the memo calendar
static void	importHelpers() Import helpers from a CSV file and update helper table.
static void	importParticipants() Import participants from a CSV file and update participant table.
static void	importSponsors() Import sponsors from a CSV file and update sponsor table.
static void	importVendors() Import vendors from a CSV file and update vendor table.
static void	submitContactsTable() Log the changes made to the contacts table.
static void	updateAddExpense() Update the combo box for selection of budget categories.
static void	updateAnnouncementContactsListTable() Update the list of contacts for announcement.
static void	updateAnnouncementSubject() Update the subject of the announcement email.
static void	updateArchivedTable() Update the table for archived events.
static void	updateContactsCount() Update the number of contacts.
static void	updateContactsTable() Update the contacts table.
static void	updateEventDayItineraryTextArea() Update the text area for Itinerary of an event day.
static void	updateEventDayNotesTextArea() Update the text area for the notes of an event day.
static void	updateEventDayTable() Update the table of event days.
Static void	updateEventDescriptionTextArea() Update the text area for the description of an event day.
static void	updateEventFilterTable() Update the table for filtering of events to display for memo calendar.
static void	updateEventList()

	Update the table of events.
static void	`updateExpenseTable()` Update the table of expenses.
static void	`updateExportEventDayTable()` Update the table of event days in export.
static void	`updateItineraryTable()` Update the table for displaying the itinerary.
static void	`updateRemainingBudget()` Update the remaining budget.
static void	`updateScheduleByDateTable()` Update the memo Calendar.
static void	`updateTaskByPriorityTable()` Update the table of tasks sorted by their priority.
static void	`updateTotalBudget()` Update the total budget.
static void	`updateTotalExpense()` Update the sum of expenses.
static void	`updateUnarchivedTable()` Update the table of unarchived events.
static void	`updateVenueBookingTable()` Update the table of venue bookings.
static void	`updateVenueFilter()` Update the search filters for venues.
static void	`updateVenueTable()` Update the tables of venues filtered by the search filters.

10 *Appendix C* - Sample Code

10.1) Add Expense

- **GUI** - When "**Add Expense**" button is clicked, **GUI** calls the function **addExpense** in **Logic**.
- **Logic**

```
public void addExpense(BudgetCategory b, String t, double c)
{
        Expense ex = new Expense(b,t,c);              // Construct Expense object
        expenseM.addExpense(ex);                      // Add into ExpenseManager

        // Checks if expense is tagged under any budget category
        // If not, add it to the 'uncategorized' category
        if (b != null) budgetcategoryM.addExpense(b,ex);
        else budgetcategoryM.addExpense(b.getEvent().getUncategorizedBudget(), ex);
}
```

- **Logic (Expense Manager)**

```
// add to global list of expenses
public void addExpense(Expense ex){ expenses.add(ex); }
```

- **Logic (BudgetCategory Manager)**

```
// call addExpense(ex:Expense) for the correct budget category
public void addExpense(BudgetCategory b, Expense ex){ b.addExpense(ex); }
```

- **BudgetCategory**

```
// Add expenses to the List of Expenses tagged to this BudgetCategory
void addExpense(Expense ex){ expenseList.add(ex); }
```

10.2) Storing Venue

```
// Initialise document for writing
Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();

// Define root element
Element root = doc.createElement("Venue");
doc.appendChild(root);

// Scan each venue in Vlist
for(int i=0; i<Vlist.size(); i++) {
       Element item = doc.createElement("Item");

       // Add ID
       Attr id = doc.createAttribute("ID");
       id.setValue(Integer.toString(i));
       item.setAttributeNode(id);

       // Add Name
        Element name = doc.createElement("Name");
        name.appendChild(doc.createTextNode(Vlist.get(i).getName()));
        item.appendChild(name);

       // Add Type
        Element type = doc.createElement("Type");
        type.appendChild(doc.createTextNode(Vlist.get(i).getType()));
        item.appendChild(type);
        // Add Faculty
        Element fac = doc.createElement("Faculty");
        fac.appendChild(doc.createTextNode(Vlist.get(i).getFaculty()));
        item.appendChild(fac);

        // Add Capcity
        Element cap = doc.createElement("Capacity");
        cap.appendChild(doc.createTextNode(Integer.toString(
                                           Vlist.get(i).getCapacity())));
        item.appendChild(cap);

        // Append item to the root element
        root.appendChild(item);
}

// Transform to XML
DOMSource src = new DOMSource(doc);
StreamResult result = new StreamResult(new File(Vfile));
TransformerFactory.newInstance().newTransformer().transform(src, result);
```

10.3) Loading Venue

```
// Initialise document for reading
Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new
                 File(Vfile));
doc.getDocumentElement().normalize();

// Get a NodeList of items
NodeList items = doc.getElementsByTagName("Item");

// Scan the list of items
for(int i=0; i<items.getLength(); i++) {
        // Retrieve the ith item
        Element item = (Element)items.item(i);

        // Create a new venue with the values stored in the ith item
        Venue newitem = new Venue(getValue(item, "Name"),
                                  getValue(item, "Type"),
                                  getValue(item, "Faculty"),
                                  Integer.parseInt(getValue(item, "Capacity")));

        // Add the new venue to the list
        Vlist.add(newitem);
}
```

10.4) Export Task

```
// Write heading
writeCSS("text3", "Tasks to Do");
write("<br>");
```

```
// Build a table
openTable();

       // Create a row header
       openRowHeader();
              // Write column names
               writeCell("text1", 30, "<b>Deadline</b>");
               writeCell("text1", 30, "<b>Description</b>");
               writeCell("text1", 8, "<b>Priority</b>");
               writeCell("text1", "<b>Venues</b>");
       closeRow();

       SimpleDateFormat formatter = new SimpleDateFormat ("d MMMM, hh:mma");
       for(int i=0; i<Tasks.size(); i++) {
              // Create new row for every task
               openRow(i);
                       Task task = Tasks.get(i);

                     // Write task's date
                       writeCell("text1", 30, (i+1) + ". " +
                               formatter.format(task.getDate()).toLowerCase());

                     // Write task's description
                       writeCell("text1", 30, task.getDescription());

                     // Write task's priority
                       switch(task.getPriority()) {
                               case 0: writeCell("text1", 8, "High"); break;
                               case 1: writeCell("text1", 8, "Med"); break;
                               case 2: writeCell("text1", 8, "Low");
                       }

                     // Write task's venue(s)
                       String venues = "";
                       for(int j=0; j<task.getTaggedVenueBookingList().size(); j++) {
                               venues = venues + task.getTaggedVenueBookingList().
                                            get(j).getVenue().getName();
                               if(j != task.getTaggedVenueBookingList().size()-1)
                                       venues = venues + ", ";
                       }
                       writeCell("text1", venues);
               closeRow();
       }

closeTable();

write("<br/></br>");
```

PEPO

User Guide

Developed by
Team Ultimate!

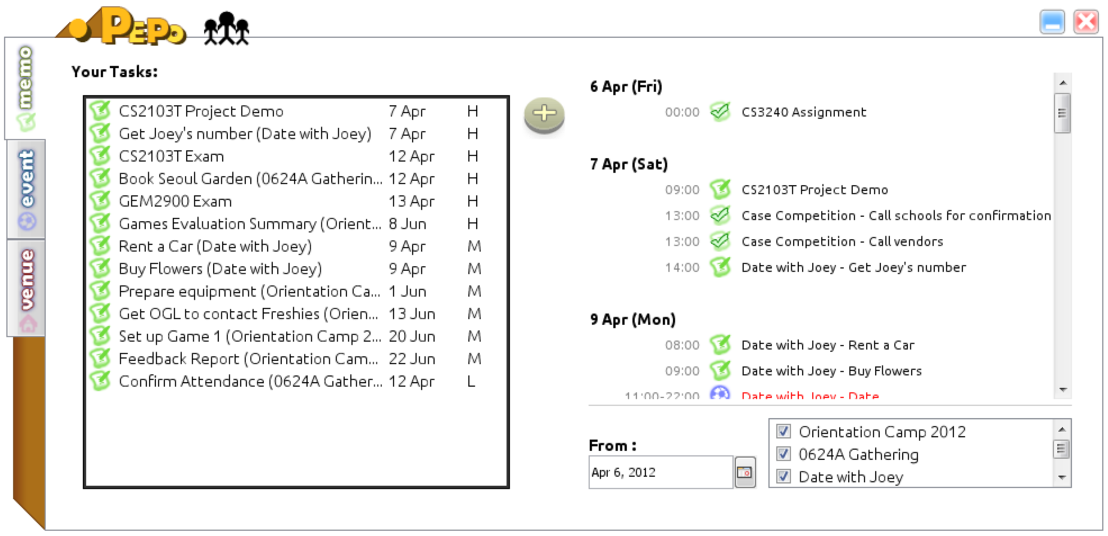
Chua Wei Kuan (A0072749U)
Ju Chaoran (A0077858H)
Lim Gang Yi (A0077256X)
Wong Hong Wei (A0072341R)

Contents

1 Getting Started

Welcome! Events management has always been a tedious task for event organisers. It does not simply consist of a name and a date, but also requires the handling of budget, advertising, venue bookings, registrations, and many other responsibilities. Worry not, because PEPO is here to help!

So, let's get started! Open the program and you should see the following screen –



Your Tasks:

CS2103T Project Demo	7 Apr	H
Get Joey's number (Date with Joey)	7 Apr	H
CS2103T Exam	12 Apr	H
Book Seoul Garden (0624A Gatherin...	12 Apr	H
GEM2900 Exam	13 Apr	H
Games Evaluation Summary (Orient...	8 Jun	H
Rent a Car (Date with Joey)	9 Apr	M
Buy Flowers (Date with Joey)	9 Apr	M
Prepare equipment (Orientation Ca...	1 Jun	M
Get OGL to contact Freshies (Orien...	13 Jun	M
Set up Game 1 (Orientation Camp 2...	20 Jun	M
Feedback Report (Orientation Cam...	22 Jun	M
Confirm Attendance (0624A Gather...	12 Apr	L

6 Apr (Fri)
00:00 CS3240 Assignment

7 Apr (Sat)
09:00 CS2103T Project Demo
13:00 Case Competition - Call schools for confirmation
13:00 Case Competition - Call vendors
14:00 Date with Joey - Get Joey's number

9 Apr (Mon)
08:00 Date with Joey - Rent a Car
09:00 Date with Joey - Buy Flowers
11:00-22:00 Date with Joey - Date

From : Apr 6, 2012

☑ Orientation Camp 2012
☑ 0624A Gathering
☑ Date with Joey

Figure 1.1: PEPO

If you are unable to open PEPO or see the transparent frame, you will need jdk7 which is available for download at http://jdk7.java.net/.

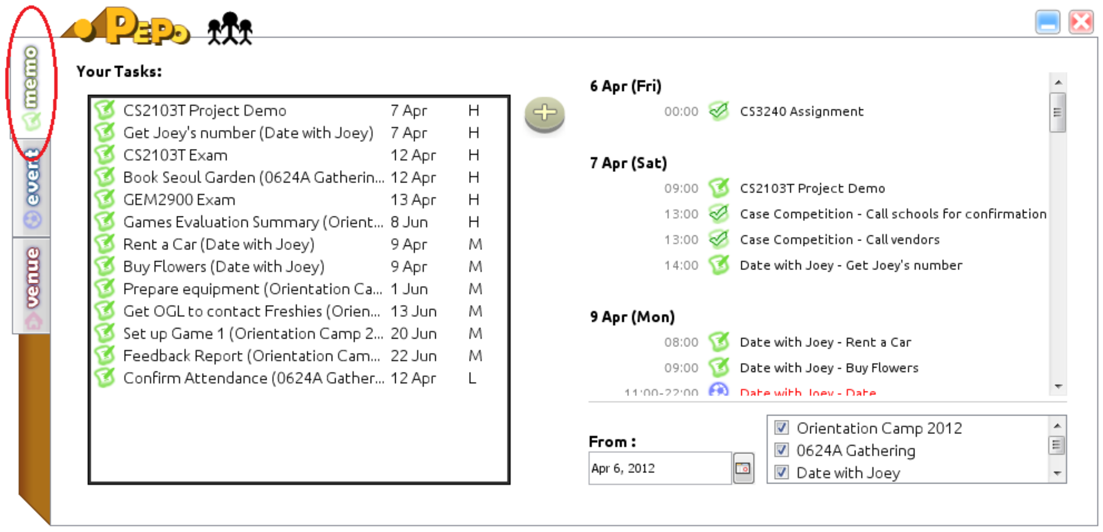
You can close PEPO at any time with the red button.
Or minimize PEPO with the blue button will send it to the dock for ease of access.

---

2 Handling Memo

2.1) Introduction

The **Memo** feature helps you in coordinating your own activities with your events and venue bookings. You can access this feature by simply clicking on the memo tab at the left side of PEPO.



Your Tasks:

CS2103T Project Demo	7 Apr	H
Get Joey's number (Date with Joey)	7 Apr	H
CS2103T Exam	12 Apr	H
Book Seoul Garden (0624A Gatherin...	12 Apr	H
GEM2900 Exam	13 Apr	H
Games Evaluation Summary (Orient...	8 Jun	H
Rent a Car (Date with Joey)	9 Apr	M
Buy Flowers (Date with Joey)	9 Apr	M
Prepare equipment (Orientation Ca...	1 Jun	M
Get OGL to contact Freshies (Orien...	13 Jun	M
Set up Game 1 (Orientation Camp 2...	20 Jun	M
Feedback Report (Orientation Cam...	22 Jun	M
Confirm Attendance (0624A Gather...	12 Apr	L

6 Apr (Fri)
00:00 CS3240 Assignment

7 Apr (Sat)
09:00 CS2103T Project Demo
13:00 Case Competition - Call schools for confirmation
13:00 Case Competition - Call vendors
14:00 Date with Joey - Get Joey's number

9 Apr (Mon)
08:00 Date with Joey - Rent a Car
09:00 Date with Joey - Buy Flowers
11:00-22:00 Date with Joey - Date

From : Apr 6, 2012

☑ Orientation Camp 2012
☑ 0624A Gathering
☑ Date with Joey

Figure 2.1: Memo Screen

You are given a calendar on the right side of the screen. The calendar shows all the existing ***tasks***, ***days of events*** and ***venue bookings*** sorted by their date. You can choose to show only part of the calendar starting from a desired date by simply selecting the date at the bottom. You can also choose to filter out some of the events from the calendar at the bottom right.

You are also given a to-do list at the left side of the screen. This is where you will manage your *tasks*.

2.2) Task

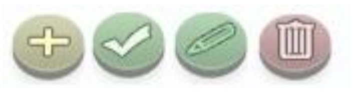
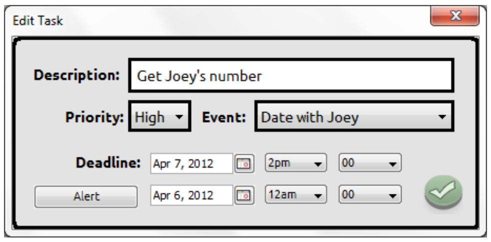


Figure 2.2.1: Add/Mark/Edit/Delete icons



Edit Task

Description: Get Joey's number

Priority: High Event: Date with Joey

Deadline: Apr 7, 2012 2pm 00

Alert Apr 6, 2012 12am 00

Figure 2.2.2: Edit Task

Figure 2.2.1 shows the buttons (by the side) of Add/Mark/Edit/Delete functions.

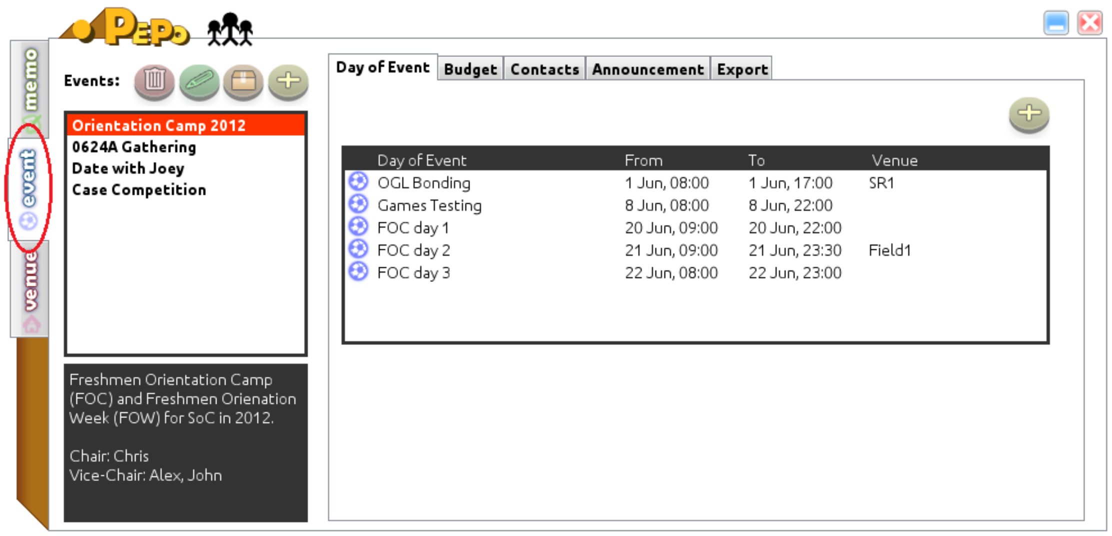
To add or edit anew *task*, click on the '**add**' icon or '**edit**' icon respectively. Next, you can fill in the *descriptio*n, set the *priority* (H, M, L), *deadline* and *alert* (if any). Alerts are in the form of popups once the pre-set date and time is reached. The *task* can be chosen to be tagged under an event or none.

To remove a *task*, click on the '**delete**' icon.
To mark a *task* as completed, simply click on the '**mark**' icon. Once this action is taken, the *task* is removed from the to-do list. The *task* will, however, still show up in the calendar with a 'completed' symbol.

3 Managing Event

3.1) Introduction

The **Event** feature helps you in managing the days of ***events***, ***budget***, ***contacts***, ***announcements*** and the ***report*** of your events. You can use this feature by clicking on the event tab at the left side of PEPO.



Orientation Camp 2012
0624A Gathering
Date with Joey
Case Competition

Freshmen Orientation Camp (FOC) and Freshmen Orienation Week (FOW) for SoC in 2012.

Chair: Chris
Vice-Chair: Alex, John

Day of Event	From	To	Venue
OGL Bonding	1 Jun, 08:00	1 Jun, 17:00	SR1
Games Testing	8 Jun, 08:00	8 Jun, 22:00	
FOC day 1	20 Jun, 09:00	20 Jun, 22:00	
FOC day 2	21 Jun, 09:00	21 Jun, 23:30	Field1
FOC day 3	22 Jun, 08:00	22 Jun, 23:00	

Figure 3.1.1: Event Screen

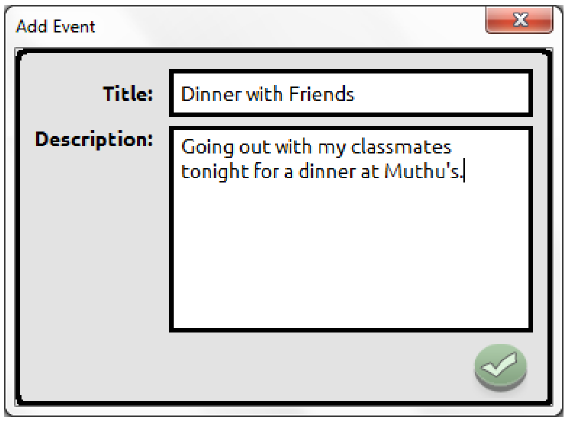
On the left side of PEPO, you can see a list of your current ***events***. To view the details or manage a certain event, simply click on the event.

The details of the selected event will be reflected on the right side of the screen. You can choose which item you want to see by clicking on the respective tabs at the top.

Figure 3.1.2: Event Tabs

3.2) Event

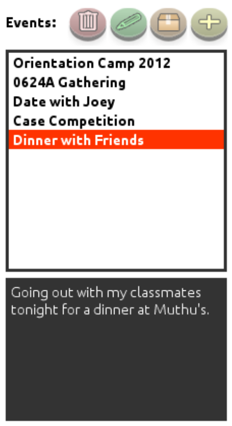
Click on the '**add**' or '**edit**' icon at the top of the list of events to add a new event or edit an existing event respectively. Insert the *title* and *description* of event and click on the '**tick**' icon. PEPO will now display the event with its description as follows:


Add Event

Title: Dinner with Friends

Description: Going out with my classmates tonight for a dinner at Muthu's.

Figure 3.2.1: Create Event


Events:

Orientation Camp 2012
0624A Gathering
Date with Joey
Case Competition
Dinner with Friends

Going out with my classmates tonight for a dinner at Muthu's.

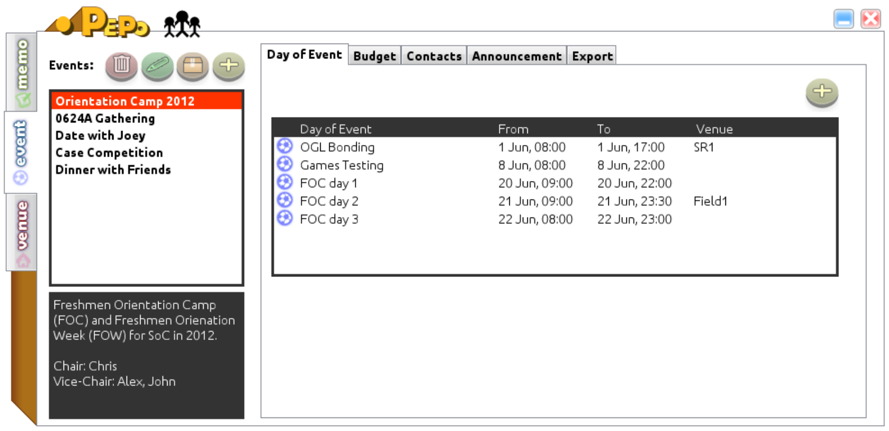
Figure3.2.2: Event List

You can also hide or unhide an event by clicking on the ‘**archive’** button.

To remove an existing *event*, click on the ‘**delete**’ icon.

3.3) Day of Event

The **Day of Event** feature enables you to plan the schedule(s) or important days of your event. Examples of *Day of Event* can include “Preliminary Round” and “Finals” of a soccer match.



Events:

Orientation Camp 2012
0624A Gathering
Date with Joey
Case Competition
Dinner with Friends

Freshmen Orientation Camp (FOC) and Freshmen Orienation Week (FOW) for SoC in 2012.

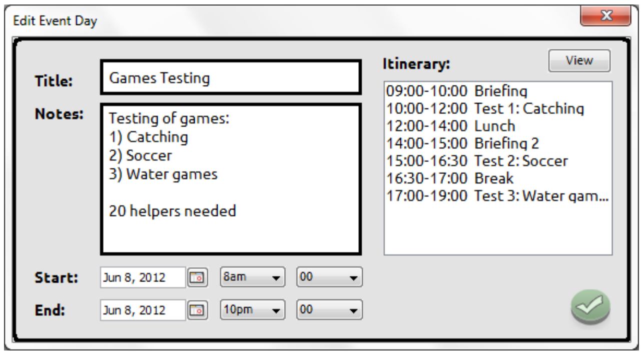
Chair: Chris
Vice-Chair: Alex, John

Day of Event	From	To	Venue
OGL Bonding	1 Jun, 08:00	1 Jun, 17:00	SR1
Games Testing	8 Jun, 08:00	8 Jun, 22:00	
FOC day 1	20 Jun, 09:00	20 Jun, 22:00	
FOC day 2	21 Jun, 09:00	21 Jun, 23:30	Field1
FOC day 3	22 Jun, 08:00	22 Jun, 23:00	

Figure 3.3.1: Day of Event Screen

To add or edit a *day of event*, click on the ‘**add**’ or ‘**edit**’ icon in the ‘Days of Event’ tab. Fill in the *title*, *notes* and then click on the calendar icon to determine the *start* and *end time* of the *day of event*.

You can click on the ‘**tick**’ icon to save the *day of event*.



Title: Games Testing

Notes:
Testing of games:
1) Catching
2) Soccer
3) Water games

20 helpers needed

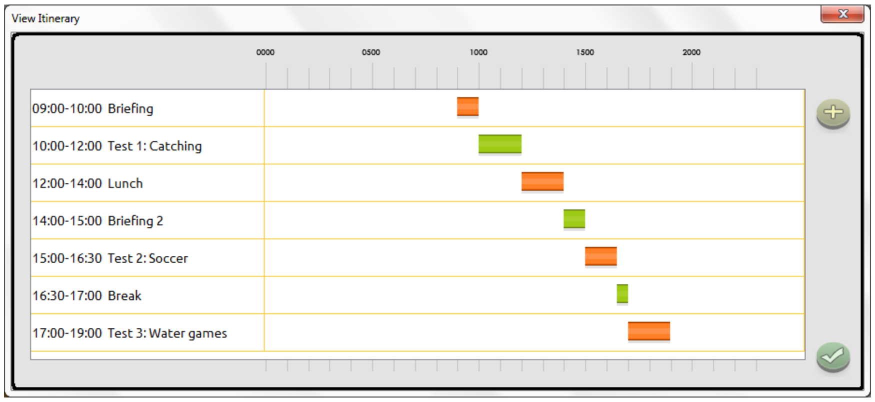
Itinerary:
09:00-10:00 Briefing
10:00-12:00 Test 1: Catching
12:00-14:00 Lunch
14:00-15:00 Briefing 2
15:00-16:30 Test 2: Soccer
16:30-17:00 Break
17:00-19:00 Test 3: Water gam...

Start: Jun 8, 2012 8am 00

End: Jun 8, 2012 10pm 00

Figure 3.3.2: Add/Edit Day of Event

Click the '**View**' button to manage the itinerary of the *day of event*. A graphical display of the time slot for each item will be displayed to aid you in scheduling your activities.



09:00-10:00 Briefing
10:00-12:00 Test 1: Catching
12:00-14:00 Lunch
14:00-15:00 Briefing 2
15:00-16:30 Test 2: Soccer
16:30-17:00 Break
17:00-19:00 Test 3: Water games

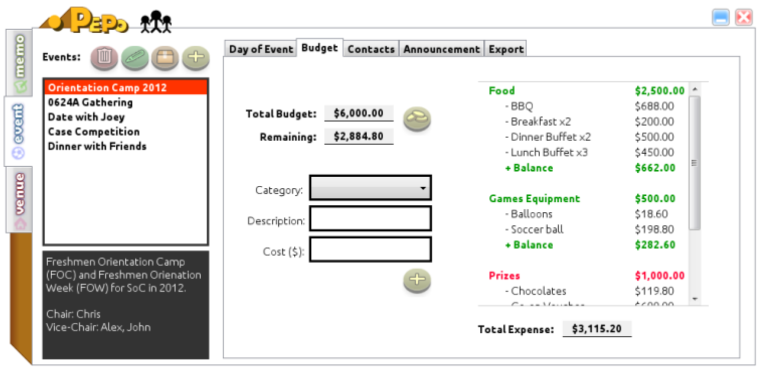
Figure 3.3.3: Itinerary

When the itinerary is filled in, click on the '**tick**' icon to save.

To remove an existing *day of event*, click on the '**delete**' icon.

3.4) Budget

The **Budget** feature allows you to handle the expenses involved in the event.



Orientation Camp 2012
0624A Gathering
Date with Joey
Case Competition
Dinner with Friends

Freshmen Orientation Camp (FOC) and Freshmen Orienation Week (FOW) for SoC in 2012.

Chair: Chris
Vice-Chair: Alex, John

Total Budget: $6,000.00
Remaining: $2,884.80

Category:
Description:
Cost ($):

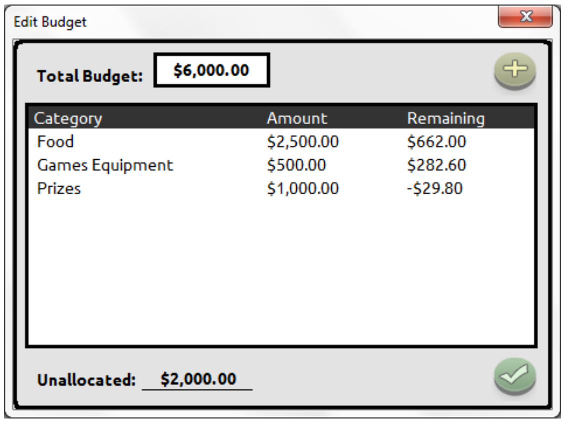
Food	$2,500.00
- BBQ	$688.00
- Breakfast x2	$200.00
- Dinner Buffet x2	$500.00
- Lunch Buffet x3	$450.00
+ Balance	$662.00
Games Equipment	$500.00
- Balloons	$18.60
- Soccer ball	$198.80
+ Balance	$282.60
Prizes	$1,000.00
- Chocolates	$119.80

Total Expense: $3,115.20

Figure3.4.1: Money Icon

Figure3.4.2: Budget Screen

Click on the "**coin**" icon as shown in *Figure 3.4.1*in the 'Budget' tab to initiate a pop-up as shown in *Figure3.6*. User can set the *total budget* of the event and manage new *categories* with their own budget. Off course, every budget category name should be unique.



Edit Budget

Total Budget: $6,000.00

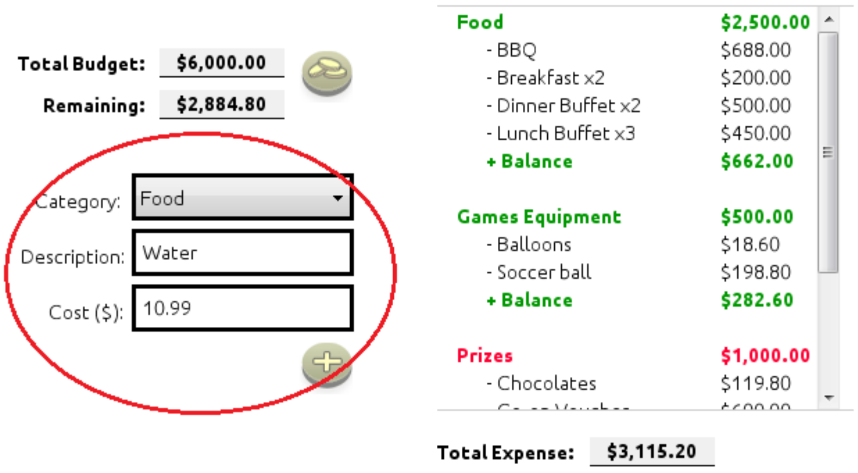
Category	Amount	Remaining
Food	$2,500.00	$662.00
Games Equipment	$500.00	$282.60
Prizes	$1,000.00	-$29.80

Unallocated: $2,000.00

Figure3.4.3: Edit Budget

The amount of money that is left after allocating the budget to the categories is shown under 'Unallocated'.

After the budgeting is completed, you can add an expense item under it. Note that you need not choose a category for the expense item and these expenses go under the 'uncategorized' category.



Total Budget: $6,000.00

Remaining: $2,884.80

Category: Food

Description: Water

Cost ($): 10.99

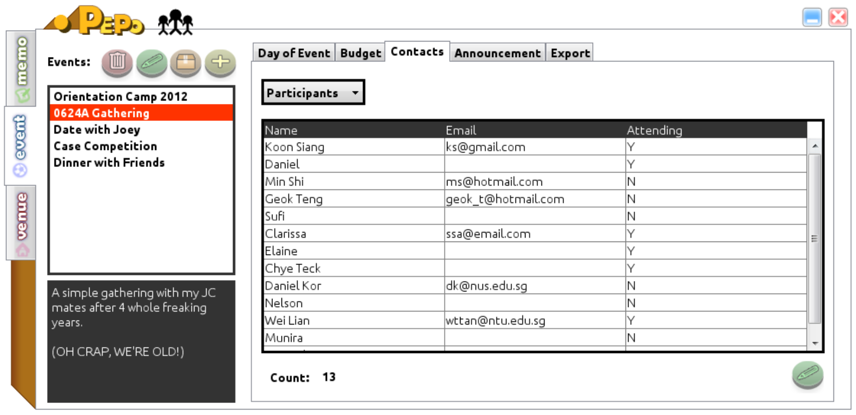
Food	**$2,500.00**
- BBQ	$688.00
- Breakfast x2	$200.00
- Dinner Buffet x2	$500.00
- Lunch Buffet x3	$450.00
+ Balance	**$662.00**
Games Equipment	**$500.00**
- Balloons	$18.60
- Soccer ball	$198.80
+ Balance	**$282.60**
Prizes	**$1,000.00**
- Chocolates	$119.80

Total Expense: $3,115.20

Figure3.4.4: Add Expense

3.5) Contacts

The **Contacts** feature gives you the ability to manage the contact list of participants, helpers, sponsors and vendors.



Orientation Camp 2012
0624A Gathering
Date with Joey
Case Competition
Dinner with Friends

A simple gathering with my JC mates after 4 whole freaking years.

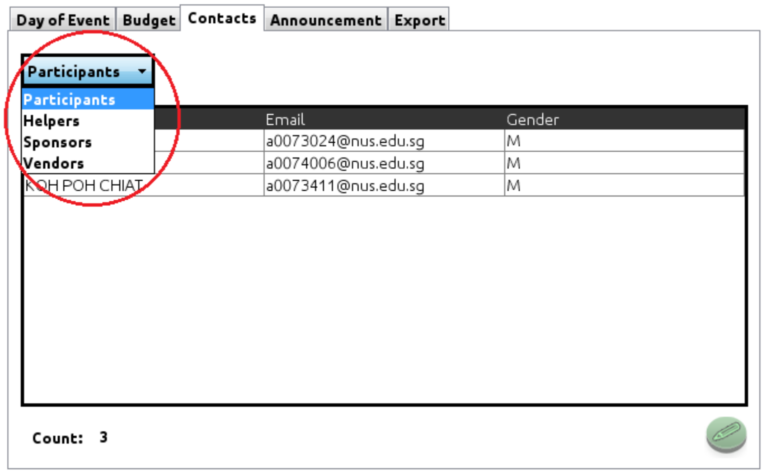
(OH CRAP, WE'RE OLD!)

Name	Email	Attending
Koon Siang	ks@gmail.com	Y
Daniel		Y
Min Shi	ms@hotmail.com	N
Geok Teng	geok_t@hotmail.com	N
Sufi		N
Clarissa	ssa@email.com	Y
Elaine		Y
Chye Teck		Y
Daniel Kor	dk@nus.edu.sg	N
Nelson		N
Wei Lian	wttan@ntu.edu.sg	Y
Munira		N

Count: 13

Figure 3.5.1: Contacts Screen

To view the contact list you want, you can choose it from the drop-down menu in the 'Contacts' tab.



	Email	Gender
	a0073024@nus.edu.sg	M
	a0074006@nus.edu.sg	M
KOH POH CHIAT	a0073411@nus.edu.sg	M

Count: 3

Figure3.5.2: Types of Contact Lists

3.5.1) Attribute Column

To make any changes to the contact list, first click on the '**Edit**' icon at the lower right.

Next, to change or add a new field for the contact list, click on the '**Gear**' icon at the top left hand corner of the information box.

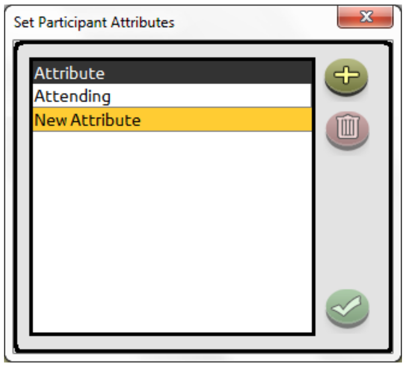


Figure3.5.1.1: Set Attributes

Click on the '**add**' icon to add a new attribute. Double click on any of the attributes to edit the name of that attribute. To delete an attribute, click on the '**delete**' icon.

Click on the '**tick**' icon to save the changes. Note that the attributes "Name" and "Email" cannot be changed or removed.

3.5.2) Contact Information

To make any changes to the contact list, first click on the '**Edit**' icon at the lower right.

Next, to add a new person, click on the '**Add**' icon. A new row will appear in the contacts list table. Alternatively, you can click on the '**Import**' icon to import a `.csv` file. If the number of columns in the file is greater than the number of existing attributes, the number of attributes will increase to hold all of the data.

Figure 3.5.2.1: Import Icon

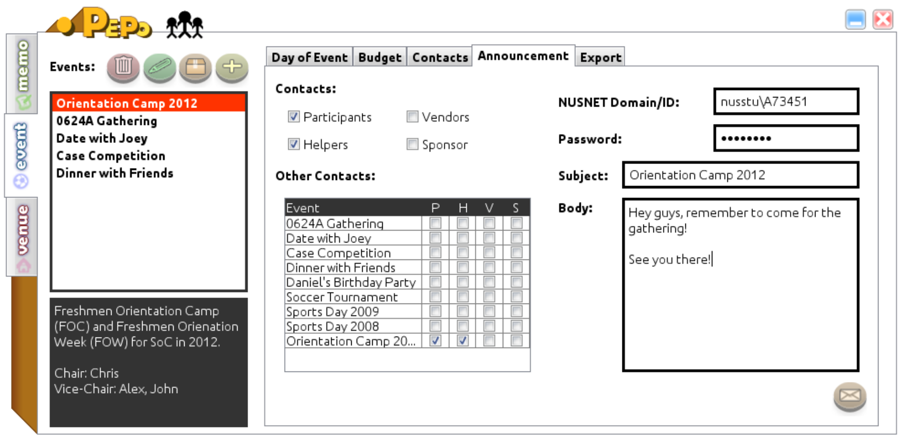
To edit, double click any cell in the table.

Click on the '**tick**' icon to save the current contact list. If there are any unwanted changes, click on the '**cross**' icon to undo them.

3.6) Announcement

The **Announcement** feature gives you the ability to send an email to the current event's participants, helpers, sponsors or vendors' list or any list from other events.

This feature is only valid for students or staff of NUS.



Events:

Orientation Camp 2012
0624A Gathering
Date with Joey
Case Competition
Dinner with Friends

Freshmen Orientation Camp (FOC) and Freshmen Orienation Week (FOW) for SoC in 2012.

Chair: Chris
Vice-Chair: Alex, John

Contacts:

☑ Participants ☐ Vendors
☑ Helpers ☐ Sponsor

Other Contacts:

Event	P	H	V	S
0624A Gathering	☐	☐	☐	☐
Date with Joey	☐	☐	☐	☐
Case Competition	☐	☐	☐	☐
Dinner with Friends	☐	☐	☐	☐
Daniel's Birthday Party	☐	☐	☐	☐
Soccer Tournament	☐	☐	☐	☐
Sports Day 2009	☐	☐	☐	☐
Sports Day 2008	☐	☐	☐	☐
Orientation Camp 20...	☑	☑	☐	☐

NUSNET Domain/ID: nusstu\A73451
Password: ••••••••
Subject: Orientation Camp 2012
Body: Hey guys, remember to come for the gathering!

See you there!

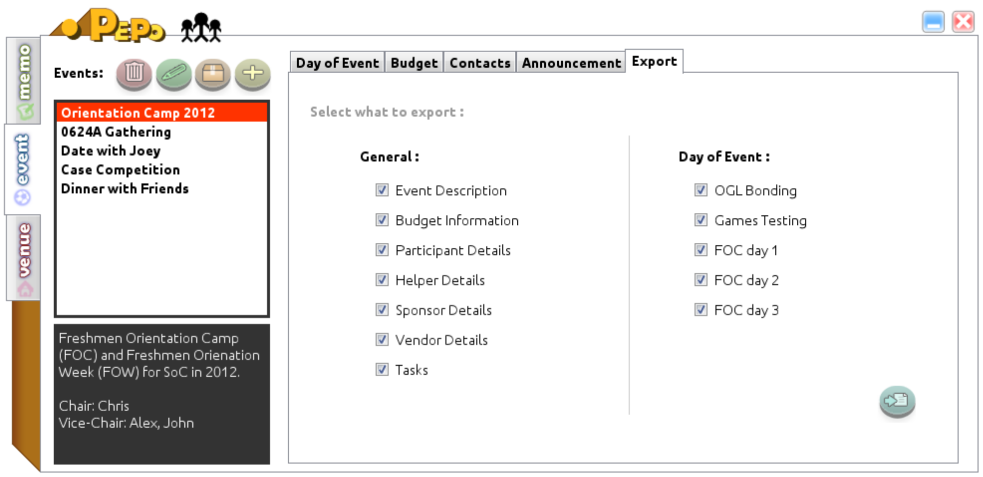
Figure3.6.1: Announcement Screen

You can select the mailing lists to send to at the left of the 'Announcement' tab. The options under "*Contacts*" refer to the contact lists for the current event. The options under "*Other Contacts*" refer to contact lists for other events (P: Participants, H: Helpers, V: Vendors, S: Sponsors).

After selecting the mailing lists, fill in the *NUSNET Domain/ID*, *Password*, *Subject* and *Body*. After finishing these, click the "**Email**" icon at the bottom right to send the announcement.

3.7) Export

The Export feature allows you to generate a report on the event in the `.html` format.



Events:

Orientation Camp 2012
0624A Gathering
Date with Joey
Case Competition
Dinner with Friends

Freshmen Orientation Camp (FOC) and Freshmen Orienation Week (FOW) for SoC in 2012.

Chair: Chris
Vice-Chair: Alex, John

Select what to export :

General :

☑ Event Description
☑ Budget Information
☑ Participant Details
☑ Helper Details
☑ Sponsor Details
☑ Vendor Details
☑ Tasks

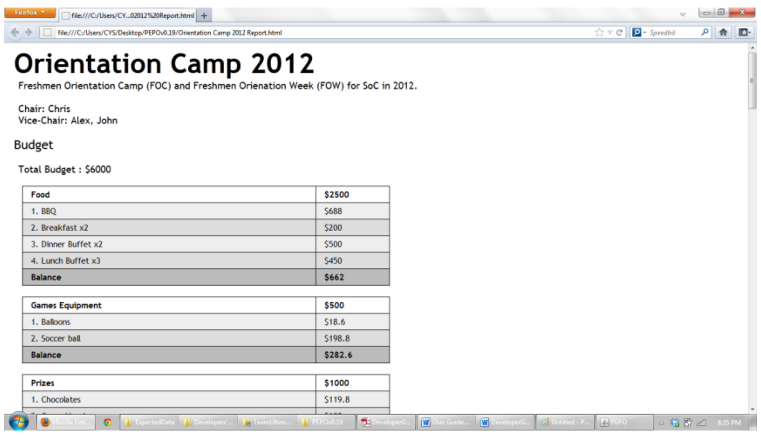
Day of Event :

☑ OGL Bonding
☑ Games Testing
☑ FOC day 1
☑ FOC day 2
☑ FOC day 3

Figure3.7.1: Export Screen

You will be presented with a list of options. You can select which item you want to export and which to exclude. After selecting the options, click on the ‘**export**’ button to generate the report in the same folder as PEPO.

The report generated will look similarly to this –



file:///C:/Users/CYS/Desktop/PEPOv0.19/Orientation Camp 2012 Report.html

Orientation Camp 2012

Freshmen Orientation Camp (FOC) and Freshmen Orienation Week (FOW) for SoC in 2012.

Chair: Chris
Vice-Chair: Alex, John

Budget

Total Budget : $6000

Food	$2500
1. BBQ	$688
2. Breakfast x2	$200
3. Dinner Buffet x2	$500
4. Lunch Buffet x3	$450
Balance	**$662**

Games Equipment	$500
1. Balloons	$18.6
2. Soccer ball	$198.8
Balance	**$282.6**

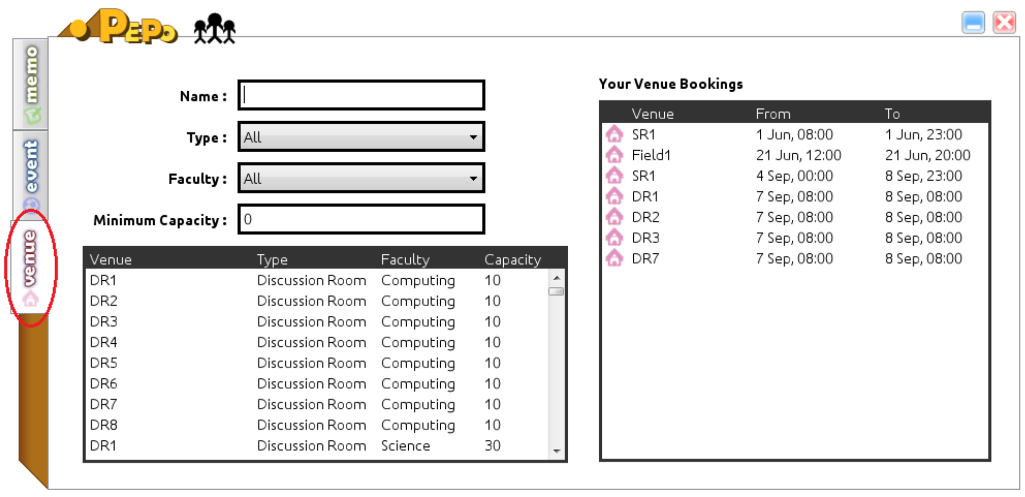
Prizes	$1000
1. Chocolates	$119.8

Figure3.7.2: Generated Report

4 Booking Venue

4.1) Introduction

The **Venue** feature lets you select venues to book and then use for your tasks and events. You can select this feature by clicking on the venue tab at the left side of PEPO.



Name :
Type : All
Faculty : All
Minimum Capacity : 0

Venue	Type	Faculty	Capacity
DR1	Discussion Room	Computing	10
DR2	Discussion Room	Computing	10
DR3	Discussion Room	Computing	10
DR4	Discussion Room	Computing	10
DR5	Discussion Room	Computing	10
DR6	Discussion Room	Computing	10
DR7	Discussion Room	Computing	10
DR8	Discussion Room	Computing	10
DR1	Discussion Room	Science	30

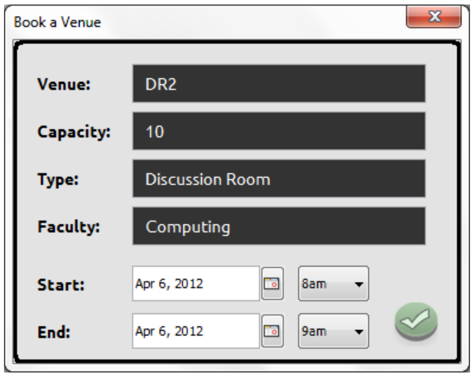
Your Venue Bookings

Venue	From	To
SR1	1 Jun, 08:00	1 Jun, 23:00
Field1	21 Jun, 12:00	21 Jun, 20:00
SR1	4 Sep, 00:00	8 Sep, 23:00
DR1	7 Sep, 08:00	8 Sep, 08:00
DR2	7 Sep, 08:00	8 Sep, 08:00
DR3	7 Sep, 08:00	8 Sep, 08:00
DR7	7 Sep, 08:00	8 Sep, 08:00

Figure 4.1.1: Generated Report

In order to book a venue, you must first find the venue you want to book. The list of venue displayed at the left side of PEPO can be filtered by choosing from the *types*, *facultie*s and *minimum capacities*. You can also search by the *name* of the venue.

To book a venue, select that venue and press the '**add**' button.

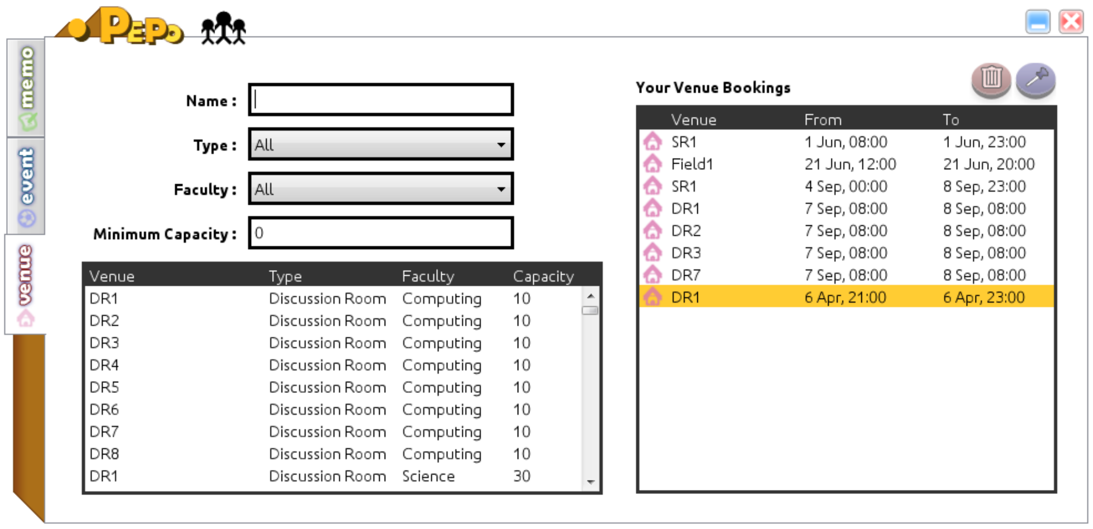


Book a Venue

Venue:	DR2
Capacity:	10
Type:	Discussion Room
Faculty:	Computing
Start:	Apr 6, 2012 8am
End:	Apr 6, 2012 9am

Figure 4.1.1: Add Venue Booking

PEPO will ask you for the *start time* and *end time* of the booking. After selecting the time, click on the '**tick**' button to complete the booking of the venue selected. The venue booking will now appear at the right side of PEPO.



Name :
Type : All
Faculty : All
Minimum Capacity : 0

Venue	Type	Faculty	Capacity
DR1	Discussion Room	Computing	10
DR2	Discussion Room	Computing	10
DR3	Discussion Room	Computing	10
DR4	Discussion Room	Computing	10
DR5	Discussion Room	Computing	10
DR6	Discussion Room	Computing	10
DR7	Discussion Room	Computing	10
DR8	Discussion Room	Computing	10
DR1	Discussion Room	Science	30

Your Venue Bookings

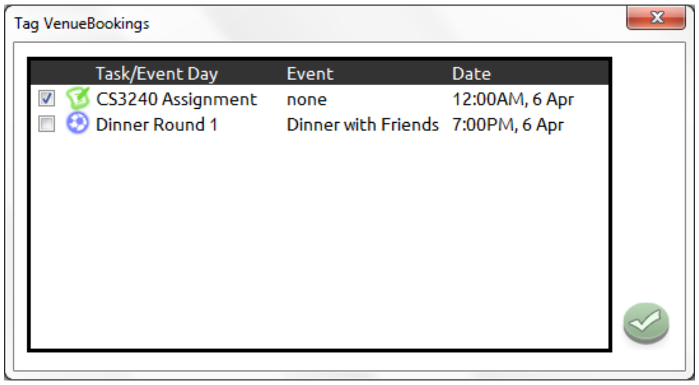
Venue	From	To
SR1	1 Jun, 08:00	1 Jun, 23:00
Field1	21 Jun, 12:00	21 Jun, 20:00
SR1	4 Sep, 00:00	8 Sep, 23:00
DR1	7 Sep, 08:00	8 Sep, 08:00
DR2	7 Sep, 08:00	8 Sep, 08:00
DR3	7 Sep, 08:00	8 Sep, 08:00
DR7	7 Sep, 08:00	8 Sep, 08:00
DR1	6 Apr, 21:00	6 Apr, 23:00

Figure 4.1.2: Venue Booked

You can now select what you are using the venue for by clicking on the '**pin**' icon.

Figure 4.1.3: Pin Icon

You will be shown a list of ***days of event*** and ***tasks***. Only the ones that take place at the same time as the venue booking will be shown.



	Task/Event Day	Event	Date
✓	CS3240 Assignment	none	12:00AM, 6 Apr
☐	Dinner Round 1	Dinner with Friends	7:00PM, 6 Apr

Figure 4.1.4: Tag Venue Booking

After selecting what you are using the venue for, click on the '**tick**' icon to save the information. To make any amendment, click on the '**pin**' icon again. Note that the venue booking timing cannot be changed.

To remove an existing *venue booking*, click on the '**delete**' icon.

---

5 Bonus

By clicking on the coloured area on the left side of PEPO, the colour will be changed. There are a total 4 colours that you can choose from.

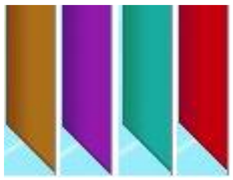


Figure 5.1.1: Colour Selection

6 Troubleshooting

For any issues on troubleshooting, free feel to contact us at the following addresses:

Chua Wei Kuan - a0072749u@nus.edu.sg
Ju Chaoran - a0077858h@nus.edu.sg
Lim Gang Yi - a0077256x@nus.edu.sg
Wong Hong Wei - a0072341r@nus.edu.sg

With that we wish you all the best in creating plenty of accomplished and wondrous events of your own. Thank you for choosing PEPO!