

Fredrik Einarsson - Niklas Johansson - René Niendorf   
Anders Nordin - Sofie Peters

For ChalmersOnTheGo 1.0, Jelly Bean 4.0 and API 16

Chalmers On The Go – the Complete Chalmers Experience

This document describes the different user stories, their associated acceptance tests and the unit tests made when developing the ChalmersOnTheGo Androis application.

Test Report

Software Engineering Project – DAT255 - lp4, 2013Chalmers University of Technology, 23.05.2013

Test Report

Table of Contents

1 Testing 2

1.1 User stories 2

1.2 Acceptance tests 4

1.2.1 Verification 4

1.2.2 Validation 4

1.3 Unit tests 4

1.3.1 Verification 7

1.3.2 Validation 7

# Testing

In the texts below you will find a complete list of user stories, their associated acceptance tests, as well as unit tests made during the course of developing the application ChalmersOnTheGo.

## User stories

Please note that some user stories are generally formulated, for an example “I want to see all rooms”. These are tested only with a limited amount of data, not the data for *all* the rooms there are in the Chalmers, due to time limitations regarding collection of data. As long as more data is added in the correct way described in the [Developer Manual](Developer%20manual.docx), those user stories are probable to be correct.

**General**

* As a user, I should be able to exit the application at any time from inside the application1
* As a user, I want to be able to reverse my actions with a back-button2

**Map**

* As a user, I should only be able to see and navigate inside the Chalmers area3
* As a user opening the application, I want to see a fixed view of the Chalmers area, which always has the same starting coordinates4
* As a user, I want to be able to click on a marked location, generating a popup window informing me about the name and floor of the location5

**Navigation**

* As a user having gotten a location marked on the map, I want to be able to get the shortest path from my current location to the wanted location, by clicking the locations information window6
* As a user, I want to be able to get the map centered at my current location7

**Searching**

* As a user typing in a search for some item, I want a dropdown menu to appear with word-completed suggestions8
* As a user searching and getting suggestions, I want to be able to click any suggestion and get the location I clicked marked on the map9
* As a user, I want to be able to search for a room, mark it and get the closest entry to the room marked on the map10
* As a user, I want to be able to search for a building and get all the rooms in the building marked on the map11
* As a user, I should be able to search for different room types, and get all the rooms of the specific type as suggestions12

**Layer function**

* As a user, I want to be able to have a checkbox-regulated layer function where I can choose between location types13
* As a user checking any layer, I want to be able to see all the rooms concerned by the specific layer14
* As a user, I want to have layers with computer rooms, lecture halls, group rooms and pubs15

**Different application modes**

* As a user, I should be able to switch between night and day mode at any point or time in the application
* As a user currently in night mode, I want to see the buildings coloured according to their associated section’s colour,
* As a user currently in day mode, I want to see the buildings coloured neutrally

**Non-navigational features**

* As a user, I want features not only concerned with the map and navigation
* As a user... step counter
* As a user… drinking progress
* As a user… iCal sync

**Design, appearance and user support**

* As a user, I want a nice looking application icon16
* As a user, I want to be able to reach all the application’s functions from a menu system similar to Google maps17
* As a user searching for items, I want to see an icon symbolising the type of room, building or pub for each suggestion I get18
* As a user checking the “pub layer”, I want to be able to see the buildings where the pubs are painted with the colour of the respective pub’s sections19
* As a user opening the application, I want to be prompted to activate the GPS if it is not already enabled20
* As a user, I want the GUI to have an appealing appearance21
* As a user, I want the application to perform wanted actions reasonably fast22

**Current user stories (20 may-current):**

Get distance and duration between two points

The user should be able to see where his next Calendar event is located on the map when he presses a button

As a user, I want to know how long it will take me to get from one point to another on the map. Also if i am estimated to show up late for class(if synced with calendar) , the app should tell me to run

A user should be able to switch orientation without loosing al his choices

A user should be able to empty the map from markers

As a user located inside the Chalmers area and opening the application, I want to see a map over my current area

Drinking progress bars

Step counter

As a user, i want to be able to search for two separate locations, and get the path between them. – future implementation

As a user, I want to be able to mark which floor I’m interested in when checking a specific layer

As a user, I want to be able to see where I am on campus with just one click (should we have this? It’s a good function to have, but perhaps difficult?)

LOOK THROUGH THE APP FOR MORE USER STORIES/FUNCTIONS

## Acceptance tests

What is tested

How is it tested

Expected result

Actual result

Potential fix

New result

Connected user story

Acceptance tests – white box testing – customer testing – tests user stories

Coverage: stuff that the unit tests might miss

Unit tests – black box testing – programmer testing – tests

Coverage: either statements or branches/paths in the code

Acceptance test:

“Writing cash reegister software”

Acceptance test: Shopping cart for generating a receipt

Create a shopping cart with:

1 lollipop, 3 bags of cough drops, 1 litre milk

Prices: 1 crown/lollipop, 3 crowns/cough drops, 10 crowns/litre milk

Verify total is 20 crowns

### Verification

Verification/Validation

Verification, check according to specification

Validation, check according to initial requirements

Verification:You implemented it correctly Validation:You implemented the correct thing

### Validation

## Unit tests

All database methods in the DAO (Data Access Object) class have been unit tested, using the public Assert class. The methods are tested both to pass and to fail and the tests have extensive statement coverage.

* **Insertion and getting in table 4 (buildings table), test case**
  + insertIntoTable4 and getAllFromTable4 were tested together:
    - A building name (String) was inserted into table 4 via insertIntoTable4 and fetched with getAllFromTable4
* **Insertion and getting in table 2 (room types table), test case**
  + insertIntoTable2 and getAllFromTable2 were tested together:
    - Three room types (String) were inserted into table 2 via insertIntoTable2 and fetched with getAllFromTable2
* **Insertion and getting in table 1 (coordinates and buildings table), test suite**
  + insertIntoTable1 and getClosestEntry were tested together:
    - A pair of coordinates (Double) and a building name (String) were inserted into table 1 via insertIntoTable1.
    - The coordinates (Double) were used to create an object (LatLng) containing latitude and longitude.
    - The object (LatLng) and the building name (String) served as input in getClosestEntry.
    - The result of getClosestEntry (LatLng) and the object (LatLng) containing the coordinates were compared and found to be equal.
    - Calculating the closest entry
  + insertIntoTable1 and getClosestEntry were tested together:
    - An object (LatLng) containing zero coordinates, the current coordinates, were created.
    - Five different coordinate pairs (Double) and a building name (String) were inserted into table 1 via insertIntoTable1.
    - The pair of coordinates (Double) closest to the zero coordinates, were in addition used to create an object (LatLng) containing latitude and longitude.
    - The zero coordinates object (LatLng) and the building name (String) served as input in getClosestEntry.
    - The result of getClosestEntry (LatLng) and the closest coordinate pair object (LatLng) were compared and found to be equal.
    - Insertion and getting in table 3 (room name, coordinates, room type, building and floor table)
  + insertIntoTable3 and getRoomCoordinates were tested together:
    - A pair of coordinates (Double) were used to create an object (LatLng) containing latitude and longitude.
    - A room type (String) was inserted into table 2 via insertIntoTable2.
    - A building name (String) was inserted into table 4 via insertIntoTable4.
    - The room name (String), the coordinates (Double, the room type (String), the building name (String) and a floor (String) were inserted into table 3 via insertIntoTable3.
    - The room name (String) served as input in getRoomCoordinates.
    - The result of getRoomCoordinates (LatLng) and the object (LatLng) containing the coordinates were compared and found to be equal.
    - Getting all rooms in a specific building
  + insertIntoTable2, insertIntoTable3, insertIntoTable4 and getAllRoomsInBuilding were tested together:
    - A room type (String) was inserted into table 2 via insertIntoTable2.
    - A real building name (String) was inserted into table 4 via insertIntoTable4
    - A false building name (String) was inserted the same way.
    - Room name1 (String), coordinate pair1 (Double), the room type (String), the true building name (String) and floor1 (String) were inserted into table 3 via insertIntoTable3.
    - Room name2 (String), coordinate pair1 (Double), the room type (String), the true building name (String) and floor2 (String) were inserted into table 3 via insertIntoTable3.
    - A false room name (String), coordinate pair1 (Double), the room type (String), the false building name (String) and floor1 (String) were inserted into table 3 via insertIntoTable3.
    - The true building name (String) served as input in getAllRoomsInBuilding.
    - The result of getAllRoomsInBuilding (ArrayList<String>) was tested using methods size and contains, and found to be satisfactory.
    - Getting all rooms with a specific type
  + insertIntoTable2, insertIntoTable3, insertIntoTable4 and getAllRoomsInBuilding were tested together:
    - A room type (String) was inserted into table 2 via insertIntoTable2.
    - A false room type (String) was inserted the same way.
    - A building name (String) was inserted into table 4 via insertIntoTable4
    - Room name1 (String), a coordinate pair1 (Double), the true room type (String), the building name (String) and floor1 (String) were inserted into table 3 via insertIntoTable3.
    - Room name2 (String), coordinate pair1 (Double), the true room type (String), the building name (String) and floor2 (String) were inserted into table 3 via insertIntoTable3.
    - A false room name (String), coordinate pair1 (Double), the false room type (String), the building name (String) and floor1 (String) were inserted into table 3 via insertIntoTable3.
    - The building name (String) served as input in getAllRoomsInBuilding.
    - The result of getAllRoomsInBuilding (ArrayList<String>) was tested using methods size and contains, and found to be satisfactory.
    - Getting suggestions
  + insertIntoTable2, insertIntoTable3, insertIntoTable4 and suggestions were tested together:
    - A room type (String) was inserted into table 2 via insertIntoTable2.
    - A building name (String) was inserted into table 4 via insertIntoTable4
    - A room name (String), a coordinate pair (Double), the room type (String), the building name (String) and a floor (String) were inserted into table 3 via insertIntoTable3.
    - Different strings of letters matching the strings in table 3 served as input in suggestions.
    - The result of suggestions (ArrayList<String>) was tested using methods for size and null, and found to be satisfactory.
    - Getting room names
  + insertIntoTable3 and getName were tested together:
    - A room name (String), a coordinate pair (Double), a room type (String), a building name (String) and a floor (String) were inserted into table 3 via insertIntoTable3.
    - The room name (String) served as input in getType.
    - The result of getName (String) and the room name were compared and found to be equal.
    - Getting room types
  + insertIntoTable3 and getType were tested together:
    - A room name (String), a coordinate pair (Double), a room type (String), a building name (String) and a floor (String) were inserted into table 3 via insertIntoTable3.
    - The room name (String) served as input in getType.
    - The result of getType (String) and the room type were compared and found to be equal.
    - Getting floor
  + insertIntoTable3 and getFloor were tested together:
    - A room name (String), a coordinate pair (Double), a room type (String), a building name (String) and a floor (String) were inserted into table 3 via insertIntoTable3.
    - The room name (String) served as input in getFloor.

The result of getFloor (String) and the floor were compared and found to be equal.

### Verification

Verification/Validation

Verification, check according to specification

Validation, check according to initial requirements

Verification:You implemented it correctly Validation:You implemented the correct thing

### Validation