DAT255
Software Engineering Project
Group: Flygande Gurkan
2014-10-17

Johansson Henrik
Källtén Magnus
Strömner David
Wijk Benjamin

Vision

Summary

An app that checks what legislations different hazardous materials have: flammable, explosive etc. Some cargo may not be allowed to be transported with others and there might be necessary precautions that the driver needs to be aware of, for example what signs they need to have on (explosive, corrosive etc.). When the loading is done the cargo information will be stored in a remote server so in case of emergency the paramedic and/or the fire department can check the license plate on the truck, and immediately be aware of any risks.

On mobile device:

Driver fills in cargo in the application and then checks out, in doing so the application connects to the internet and send the information to the database which in turn checks if the cargo can be transported together. The server sends a message back which will be displayed to the drive, the message is either empty, things to note or list of cargo that can't be transported together. If the message is empty or only things to note the client send another message to the server containing the cargo and license plate.

On server:

Waits for calls from clients that either send
1) Cargo lists to be checked .
2) Sends cargo and licensing plate.
3) Retrieve information.
If case one the server checks the list the list and send a message back containing information about the cargo. In case two the server stores the list together with the licensing plate in the server which can be access through case three, in which case the server send the cargo list and licensing plate back.

Database:

Contains all the elements that can be used inside the application but also got a separate table to store cargo that works together and a license number. The client application retrieves the complete database from the server once per version and stores it locally so the server won't be overloaded by endless requests about sending its list each time the application is ran.

Possible future additions:

While the main focus of this application is to make it easier for truck drivers to transport hazardous material it also got some aspects that could be lifted out into an application on its own. This would mainly be the features regarding fetching cargo lists and license number.
Of course of the limited time we can't cover all hazardous materials that exists since the list of those is well over 4000 so making this into a full fledged application would require that and taking all the rules

Johansson Henrik
Källtén Magnus
Strömner David
Wijk Benjamir

into consideration instead of the few limited one which was easy to work into a database. It would also most likely require some sophisticated GPS features which takes the cargo into consideration when planning the route it's going to travel, so the truck for example avoids driving through towns if it got unstable load on it.

User manual

When starting the app there are two tabs, one called Search and one called Current. In the Search Tab there is a searchfield where you can search for different elements that you would like to transport on a truck. You can search for either the elements UN number or the name of the element itself. When writing in the search field, elements will appear in a list below that match what has been written in the search field. Then just press *Add Element* at the same row as your desired element.

When you are done with choosing elements you can press the current tab to see what elements you have selected for the current trip. Here every chosen element will be displayed in separate panels.

Every panel contains the following:
A sign for the element (Such as *flammable*, *explosive* or *radioactive*)
The UN number for the element.
The full name of the element.
A text field where you write the volume of the element you want to transport. (In kilograms)
An information button that will show information about the element in question.

After you are done with your element selection you press the *Get Checklist* button on the bottom of the panels, and it will take you to a screen that shows you what might be wrong with the combination of elements you have selected, and if none exists it will tell you what signs you might need to use.

User stories

User Stories	Sprint #1
• As a developer I want to be able to remove elements in the database.	**David**
• As a user I want easy access to additional information about a material.	**Magnus**
• As a developer I want to access individual elements in the database.	**David**
• As a developer I want to be able to add elements in the database.	**Henrik**
• As a user I want to be able to switch in between the different views just by selecting a different tab.	**Magnus**
• As a user I want to be able to save and load to/from the database. Free in SQLite	**Henrik**

User Stories	Sprint #2
• As a developer I want to be able to clone an element so I can add it to a new list with a weight without destroying the 'master' list containing all the element	**David**
• As a developer I want to get a row from the database in a manageable way. So I can extract data from the row without any problems.	**David**
• As a developer I don't want any hassle checking if a list of elements can be transported together.	**Henrik**
• As a user I want to be able to easily filter elements in the database.	**Benjamin**
• As a user I want to know instantly and clearly if I'm able to ship a set of different elements together.	**Magnus**

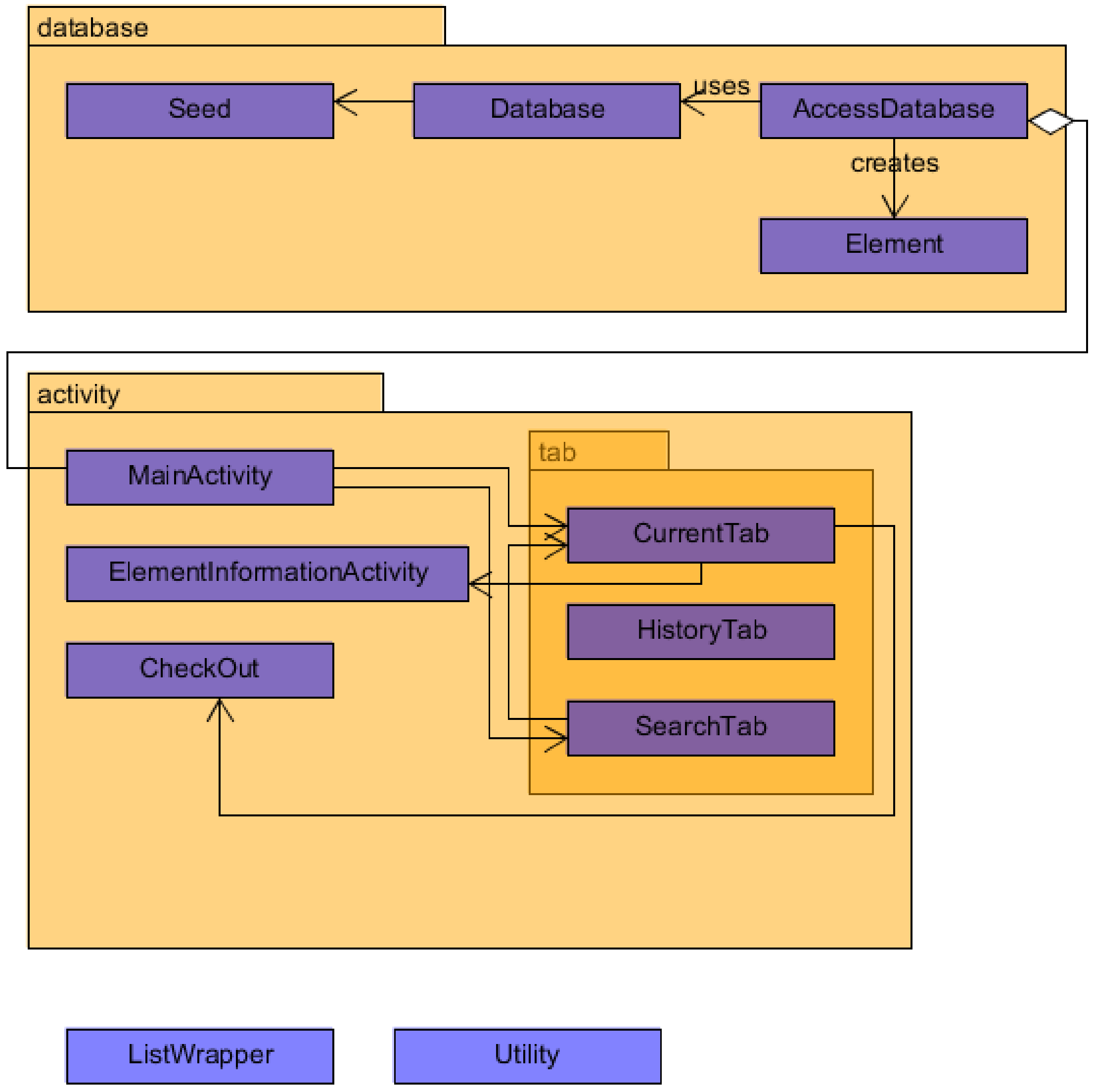
User stories	Sprint #3
• As a user I want to see maximum weight of certain elements.	**Henrik**
• As a developer I want to be able to make an easy startup seed for the database.	**Henrik**
• As a user I want to be able to add an item directly from the search list.	**Benjamin**
• As a developer I want the results of a check to come back in an array list. If problems exist they should be in the return otherwise specific things to think about(Signs) should be returned.	**David**
• Information tab, tab with text/information about selected element.	**Magnus**
• As a user I want an easy overview over what I'm transporting.	**Magnus**

User stories	Not achieved
• As a user I want to be able to see previously lists of cargo that I have transported.	
• As a user I should have limited access to the app if the truck is moving.	
• As a user I want to edit what I'm transporting without having to remove and re-add the material.	
• As a user I want to see a checklist of important information on the current cargo.	

Fuzzy	
As a power user I, and only I! can edit, create new and remove elements in the database.	
As a rescue worker I want to be able to track trucks that transport dangerous material and in case of an accident know what's inside instantly.	

Johansson Henrik
Källtén Magnus
Strömner David
Wijk Benjamir

UML



Major parts/components in the application

The application contains ten classes where four of them belong with the database and the remaining 7 belongs to the gui.

The four classes in the database are AccessDatabase, Database, Element and Seed.
The database class contains all the necessary raw data of the specific hazardous elements which are stored in a SQLite database. Because it is tedious for all developers to know SQLite specific commands, there is a factory class called AccessDatabase that has its own methods to interact with the database that are much easier to use.

When one uses getElement from AccessDatabase you are not returned a string from the sql database but instead an instantiated Element object with its own methods and values, which is as well easier to use than the raw data. Among others the Element class contains a UN number, which is a unique number for every element, a notCompatible list, which shows what other elements that cannot be shipped together with this specific element and a hazmat String that shows what type of sign is required for the element, flammable, explosive and etc. The fourth class, Seed, is used to fill the database with starting values so one does not have to manually add all the elements.

The GUI utilizes six different classes. Three of them are so-called activities that contains logic and information regarding different views. These four are SearchTab, CurrentTab, and CheckoutTab. The other three are MainActivity, and ElementInformationActivity. There were also plans for a fourth activity called HistoryTab that was never implemented.

MainActivity initializes the app by creating the database and the four activities by putting them into tabs for easy access. It also sends a reference of AccessDatabase to activities that need it. SearchTab is currently the only class that uses the database.

SearchTab is a view with a search bar that allows the user to search for a specific element in the database by either searching for its name or its UN-number. The element can then be directly added to CurrentTab. It utilizes AccessDatabase to access the list of elements.

CurrentTab contains the elements added from SearchTab. Here you can input the amount you will be carrying of each element (weight, in kilograms), see additional info, and remove elements that you don't want. ElementInformationActivity is used to display the additional info.

Checkout checks if there are any conflict between various elements (i.e. special regulations regarding transporting both in the same vehicle), or if an item exceeds the maximum allowed weight for that specific item.

HistoryTab was supposed to contain previous saved routes with elements and their weights that could be loaded into CurrentTab, in case a particular load was more common for the user. It is not implemented.

Design decisions

The intended API level from the start was 20, as this was required for the AGA SDK. Our project headed in a different direction, and AGA never got implemented. Thus our API level ended up at 14, due to utilizing methods that required it.

Aside from a few exceptions the code fulfills the low coupling, high cohesion used in object-oriented programming. A lack of previous knowledge (and therefore foresight) about android, combined with the deadline approaching created some of the more cluttered parts of the code, as doing it properly at this stage would have required massive changes.