Appendix - OEADM16EIC Group D

Third Semester Exam Project

1. Group Contract

General
This is a team project, not an I project. If in doubt, ask. No stupid ideas. Be honest with each other.

Grammar
Project spelling/grammar is to be American based.

Conflict Handling
Should issues arise professionally in the group, e.g. disagreements over a procedure, a majority win vote should decide it.
Should someone break the contract, a penalty jar is set up. The penalty will be 5 kr.
Matthew will keep the money until the group decides to use it.
A log shall be kept over penalties. The group will vote on whether the penalty will be applied or not, a majority vote is needed to decide.
In case of disagreement, a die is rolled to decide what we do.

Attendance and Meetings.
You are expected to show up every project day. If unable to attend, inform the group via the common text chat.
If you cannot attend an event, please be honest about why not. Always respond to events ASAP. DO NOT RESPOND MAYBE.
We meet at 10:00 am, Monday-Friday; 14:00 pm Saturday and Sunday.
If late to the meeting, penalty applies.

Agile Usage
All group members are in charge of the task board, decisions and prioritizing of the task board is made together.
At the end of each sprint we will do a retrospective to reflect on our work
We will begin each day with a sit-down meeting.
Sprints will be 1 week in length and will begin and end on Monday.

Developer Tools
Organization of time, place, and specials meetings: Facebook and Events
Organization of project files, code, and version control: Git w/ GitHub and Google Drive.
Organization of tasks, project backlog etc: GitHub

Naming Conventions
Methods must describe what they do
Global variables are named with an underscore at the beginning, unless they are constant.
Constant variables are all upper case.
Variables always start with lowercase and methods always start with uppercase, and then camelcase is used after.
Parameters of methods are all lower case.
Variables should describe what they are.

Properties named like methods.

Architecture
User interface should be MVC and MVVM .
Main class library should be called Core.
Unit tests will be used.
File and Database management will be done in the Data Access Layer.
The code should follow the SOLID principles.

Quality Assurance
Completing tasks in the QA section of the task board is the top priority.
You cannot check tasks that you created (all grammar is checked by Matthew)
You must follow the given criteria when checking tasks
The scrum master will assign group members to check tasks

How you are expected to use Git / GitHub

Only change one method per commit.
Expected to make decent comments.
Sync whenever you leave your computer and whenever you feel it is necessary.

Logging
We will keep a decision log in our GitHub.
The decision log will be made of of issues, which have enough info on our decisions.

Voting
All votes are equal
You cannot abstain

Amending the contract
To amend the contract, a 75% vote is needed to approve of the amendment.

Team Firenado members:
Hedviga Arta Geriņa; hedv0149@edu.eal.dk
Matthew Peterson; matt2694@edu.eal.dk
Roxana Ion; roxa0188@edu.eal.dk
Jonas Laursen; jona8690@edu.eal.dk

2. Unified Process in Process Models

2.1. Predictive

Start up - need a lot of research
Requirements gathering - have many meetings with the customer in the initial phase

HLD - the general design will be very complex, working on different big systems (6 modules, 6 different things, ex: quality assurance)
LLD - if there are some mistakes in the HLD, it will be even harder to plan the LLD
Development -you need a good way to manage the tasks between the programmers, but after the plan is made it is easy to follow
Test - massive tests need to be done, no user review
Deployment - also hard, everything is released at once, needs a lot of user training
Maintenance - in relation to fixing bugs, also hard to fix all the bugs that you haven't found before

Planning - Planning a predictive SDM is quite easy, as the stages are clear cut, and have fixed time spans.
Making Budgets - Again, as planning is easy, the budget should be pretty straight forward. You know how long you need people for.
Quality Assurance - As the previous stages needs to be complete before moving on, you should have plenty of artifacts and documentation to guide you towards the quality required. This point is quite affected by the SDM used, and exactly how much traceability this includes.
Documentation - Is done in the beginning, but how much depends on the SDM.
Traceability - Depends on documentation/SDM that is being used.
Flexibility - Predictive SDMs tend to have little flexibility, since they are usually thoroughly planned out, so if something goes wrong i can be difficult to correct this.
Integrations - As the startup stages usually cover external and internal interfaces, this is usually not a problem, unless they change during the project. There can be quite a long time between startup and implementation in large projects, hence the interfaces can change too.
Need for refactoring - Again, flexibility, as you cannot usually to back to previous stages, refactoring is very restricted. Refactoring can occur within the stage, before moving on.
Requirements to the development team - Have a good overview of how the system is going to look.

Pros:

- Have proper sub-deadlines for features
- Good documentations & Traceability

Cons:

- LærDansk seems difficult to work with, cannot predict what they want, it might lead to many problems..

2.2. Iterative

Start up - A lot of initial research
Requirements gathering - Find all requirements, and then prioritize them
HLD - Make everything perfect about the general structure
LLD - Having in mind it will be specific, you need to redo many things
Development - In first iteration all system is made to work, but in low quality and fast, later prioritized requirements are improved

Test - A lot of user tests, user can use latest version and provide feedback
Deployment - A lot of time will still be needed to train the users, but it will be split in small pieces. The users need to be trained every time you deploy something (easier for the users to follow the process step by step), but it can be frustrating for the user to have to learn something that in the next few iterations will be replaced.
Maintenance - If you didn't find a bug during one iteration the user can find it while using the system in between iterations, and it can be fixed before it becomes bigger.

Planning - Usually difficult to plan, since major changes can happen quickly. Features can usually be removed or added on short notice.
Making Budgets - Linked to planning, as changes can happen, it can also be difficult to allocate the budgets. Difficult to predict it, as the length is unknown in most of the cases.
Quality Assurance - Can be obscure, as some iterative methods like XP have little documentation to check against.
Documentation - Usually minimal, just the basic documentation is needed.
Traceability - As documentation and QA, may be minimal due to lacking artifacts
Flexibility - Very flexible, changes can be done rapidly, without the need of changing artifacts at all.
Integrations - No problem, very supported.
Need of Refactoring - Yes. // todo - refactor
Requirements to the Development Team - Developers need to be experienced, and have imagine most of the documentation. Must be prepared to delete/remove lot of hard work, if a feature gets dropped.

Pros:

- As we deliver a working, but not completed, program, our product owner can see everything working and decide what they want to improve or drop.

Cons:

- Spend time working on features that may be dropped.
- Cannot easily add new features or ideas later.

2.3. Incremental

Start up - Have a general understanding of the whole system, have idea in the initial phase what you want to do
Requirements gathering - More research on a specific module
HLD - Working on only 1 module, so the design won't be very extensive, but you need to think about the integration with the other modules
LLD - Make the needed artefacts, for the module you are working on
Development - Focus on one module at the time
Test - Doing tests for every module, but at the end you need to test the how system to check if all the modules are integrating properly
Deployment - Users can be trained for each module, less time, no retraining, less money and only small part of users at time.

Maintenance - You can start maintenance on a module while you are working on a new one.

Planning - Bit worse than predictive, but not much. Each feature lifespan can be easily planned out. Difficult to plan everything beforehand.
Budgets - Difficult, as total project length is usually unknown.
Quality Assurance, Traceability, & Documentation - Like in waterfall, the documentation is completely done before a feature is implemented, hence you can easily see if it is as it needs to be. This makes the individual features easy to trace.
Flexibility & Integrations - Quite flexible and easy to integrate, as features can be added later, and the interfaces don't need to be fixed at the very beginning.
Refactoring - Can be difficult, as you cannot usually go back to a feature once it's done.
Requirements to the development team - Ability to switch focus, once a feature is done.

Pros:

- Add new features/modules after project startup

Cons:

- Lot of work can go into a feature/module to have it dropped
- End up with project missing features
- Cannot revise features once done

2.4. Agile

Start up - Start with finding out as much as you need and later find more information if needed
Requirements gathering - Get all requirements and add some more if needed
HLD - design it to work and improve it in next sprint
LLD - design it to work and improve it in next sprint
Development - make it work and improve it in next sprint
Test - Doing tests and user tests for every sprint
Deployment - The users need to be trained every time you deploy something (easier for the users to follow the process step by step), but it can be frustrating for the user to have to learn something that in the next few sprints will be replaced.
Maintenance - Bugs will probably be found earlier than in other process models. Every sprint you can fix bugs.

Planning & Budgets - As agile methods usually consists of cycles or sprints, you can easily plan & budget for the individual prints themselves, but planning for the entire project is often based on estimations and uncertain factors.
QA, Documentation & Traceability - Agile methods still often start with documentation, and builds up the artifacts over time. As you are constantly reworking and touching up the documents and code, you generally get a pretty high quality.
Flexibility & Integrations - Agile methods provide the maximum flexibility, as features can be added, dropped, or completely revised with little loss. This also means integration should be easy, as you just adapt the features when needed to integrate.

Refactoring - Easy to refactor, usually allows you to revise quickly.
Requirements of Dev Team - Members need to be flexible, and switch to different features on short notice.

Pros:

- Add new features/ideas after project start
- Decently easy to drop features if no longer wanted
- As we work on sprints, the quality of the previous parts will be assured
- Get to supply a working system
- High degree of collaboration between LærDansk and our team

Cons:

- Difficult to plan the overall project
- Might not have enough time to complete everything.
- Have to have meetings regularly

3. System Development Methodologies

Extreme Programming

- Pros
    - Allow client to change mind
    - Split up features
    - Used it a little
    - Know it's used in real world
- Cons
    - Few artifacts and tools to help visualize workflow

RAD

- Pro
    - Works well for prototypes
- Con
    - Drops everything in form of testing, and application performance
    - Little to no documentations
    - Very short development cycle
    - Not agile, more waterfally

SCRUM

- Pro
    - Lots of artifacts and tools used to help visualize workflow
- Con

- Used this before, should try to use another sdm for this exam

Waterfall

- Pro
  - Clear cut plan for when to finish
- Con
  - Not very flexible

ETHICS

- Pro
  - Users will be very happy with product
- Con
  - Old
  - Little to no development phase?
  - Requires many users to be involved, hassle

4. User-stories

US1 - As any User, I want to be able to interact with the lecture plan so that the class can be prepared.

I can open the lecture plan.
US1.1 - As any User, I want to be able to open the lecture plan so that I can look at it.

I can read the lecture plan.
US1.2 - As any User, I want to be able to read the lecture plan so that I can gain the knowledge that it is trying to give me.

I can edit the lecture plan.
US1.3 -As Gustav or Lily, I want to be able to edit the lecture plan so that I can inform Kathleen.

I can't edit the lecture plan.
US1.4 - As Kathleen, I want to be able to view the lecture plan without being able to edit it so that I can't give false/no information to the class.

I can assign lecture plan to a class
US1.5 As Lily, I want to be able to assign lecture plan to a class so the class can interact with it.

US2 - As any User, I want to be able to authenticate with the service so that the service will know what type of user I am.

I can log in as a student
US2.1 As Kathleen, I want to be able to log in as a student so that I can interact with the student interface.

I can log in as a teacher
US2.2 As Gustav, I want to be able to log in as a teacher so that I can interact with the teacher interface.

I can log in as an admin
US2.3 As Lily, I want to be able to log in as an admin so that I can interact with the admin interface.

US3 - As any User, I want to be able to interact with the exercises so that I, and others, can gain more knowledge.

I can access the exercises
US3.1 - As a user, I want to access the exercises so that I can learn all the information being taught.

I can see my results
US3.2 - As a user, I want to see my results once I finished the exercise so that I can observe my improvement.

I can submit the exercise
US3.3 - As Kathleen, I want to be able to submit an exercise so that I can receive feedback on my work.

I can't edit the exercise
US3.4 - As Kathleen, I do not want to be able to edit the exercise so that I can't give false/no information to the class.

I can create an exercise
US3.5 - As Gustav or Lily, I want to be able to create and exercise so that Kathleen can practice what she has learned.

I can access submitted exercises
US3.6 - As Gustav or Lily, I want to be able to access submitted exercises so that I can check who has submitted.

I can check submitted exercises
US3.7 - As Gustav or Lily, I want to be able to check the exercises so that I can inform Kathleen of how she is doing.

5. FURPS+

Functionality – The system must be able to get information from a Web API and display it in the correct format. There must be different user types that have different privileges. Lecture plans must be editable and readable based on the logged in user. Exercises must be easy to add, complete and give feedback on.

Usability – The user interface is one of the priorities for the project because the product owner plans to use it as a phone app. The program should be intuitive and easy to use. Users should be able to do what they need to do in a very short amount of time wherever they are.

Reliability – The system should always be available on all devices at any time or place.

Performance – The program is not very complex, so it will not require much processing power. There should only be short delays while using the system.

Supportability – The system should be created using TDD. The phone app is created using xamarin tools that support Android and iOS development.

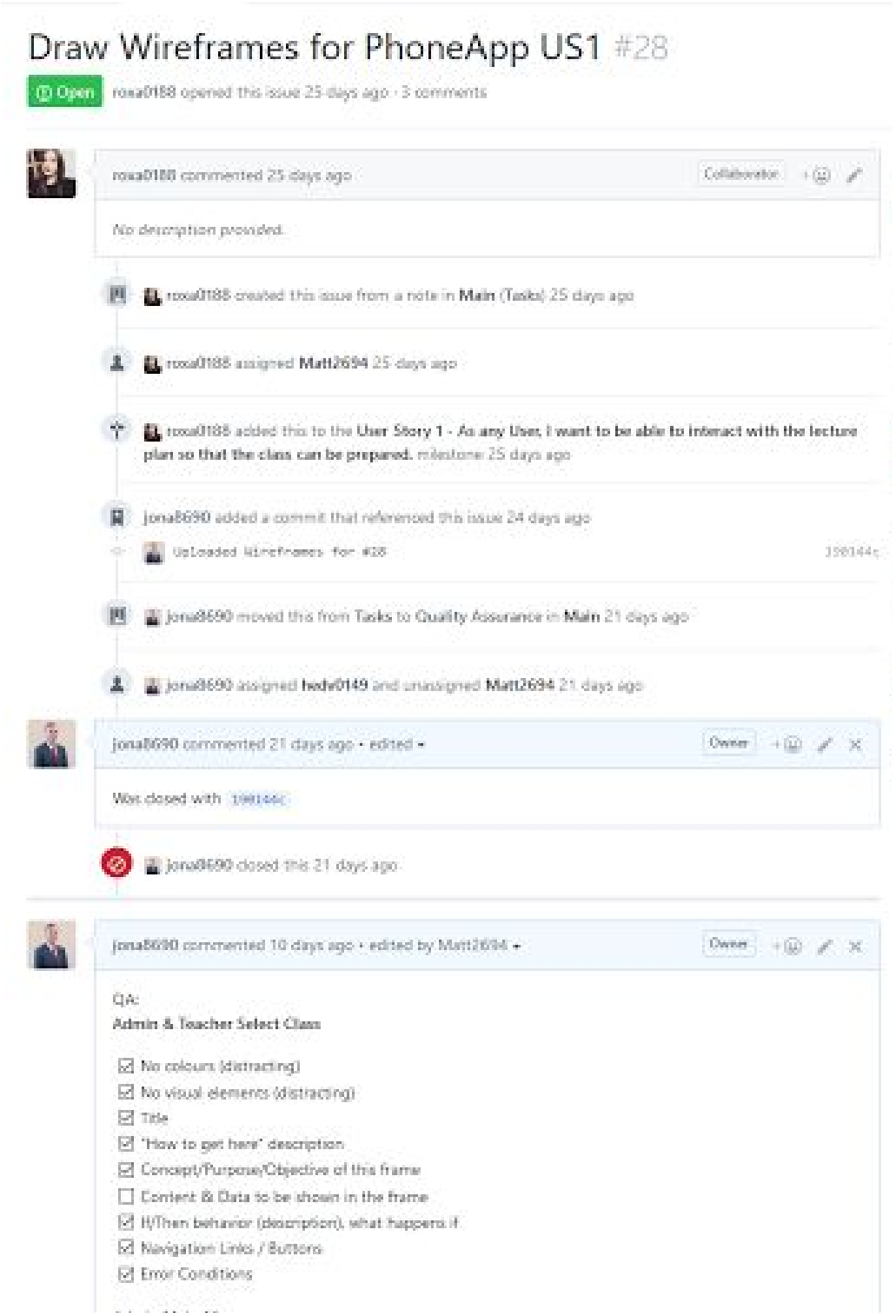
Design Constraints – The system must be able to integrate with other systems. The system must use a relational database, Xamarin app, C# and TDD.

Implementation Requirements – The programmers must adhere to the standards set out in the group contract. TDD must be used to guarantee that everything works correctly as well as premade criterias for artifacts.

Interface Constraints – It must interact with any system the product owner wants to use it with.

Physical Constraints – It should be able to run on both Android and iOS operating systems.

6. GitHub Issue Example



Student Main Menu

- [x] No colours (distracting)
- [x] No visual elements (distracting)
- [x] Title
- [x] "How to get here" description
- [x] Concept/Purpose/Objective of this frame
- [x] Content & Data to be shown in the frame
- [x] If/Then behavior (description), what happens if
- [x] Navigation Links / Buttons
- [x] Error Conditions

jona8690 commented 10 days ago — Owner

QA Done

jona8690 reopened this 10 days ago

Main automation moved this from **Quality Assurance** to **Tasks** 10 days ago

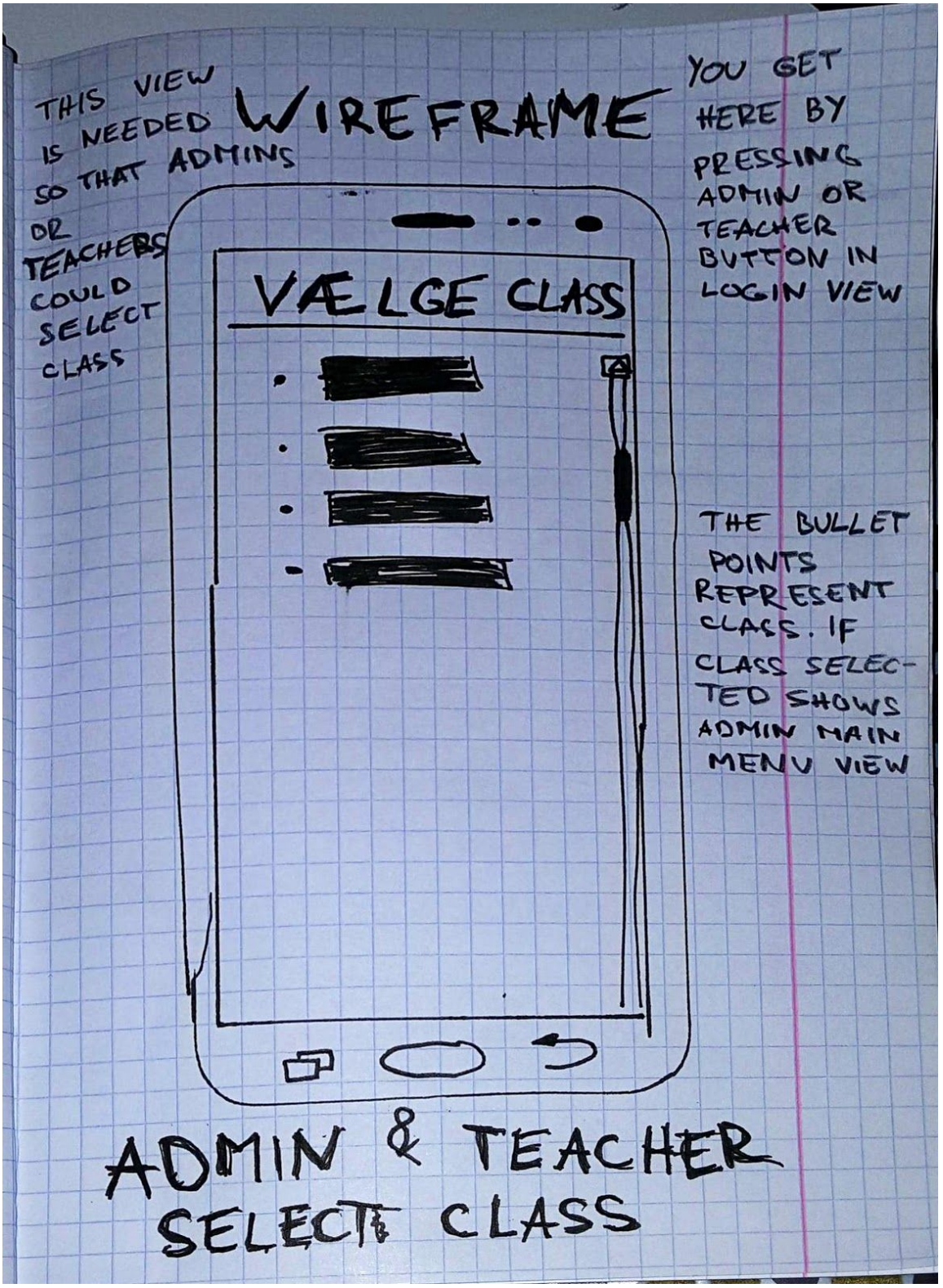
jona8690 added a commit that referenced this issue 9 days ago

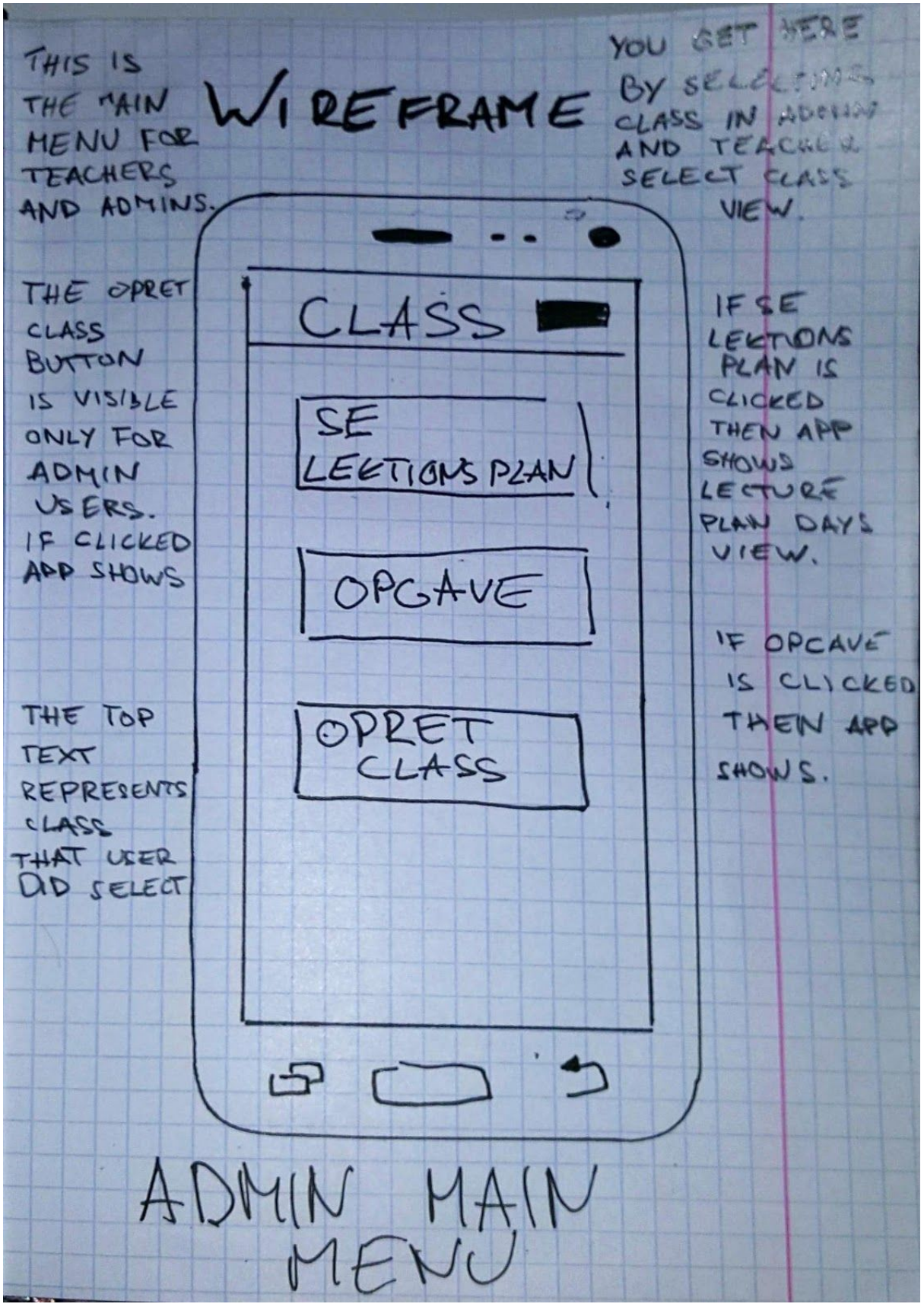
`Wireframes updated after QA, closing #28` daff702

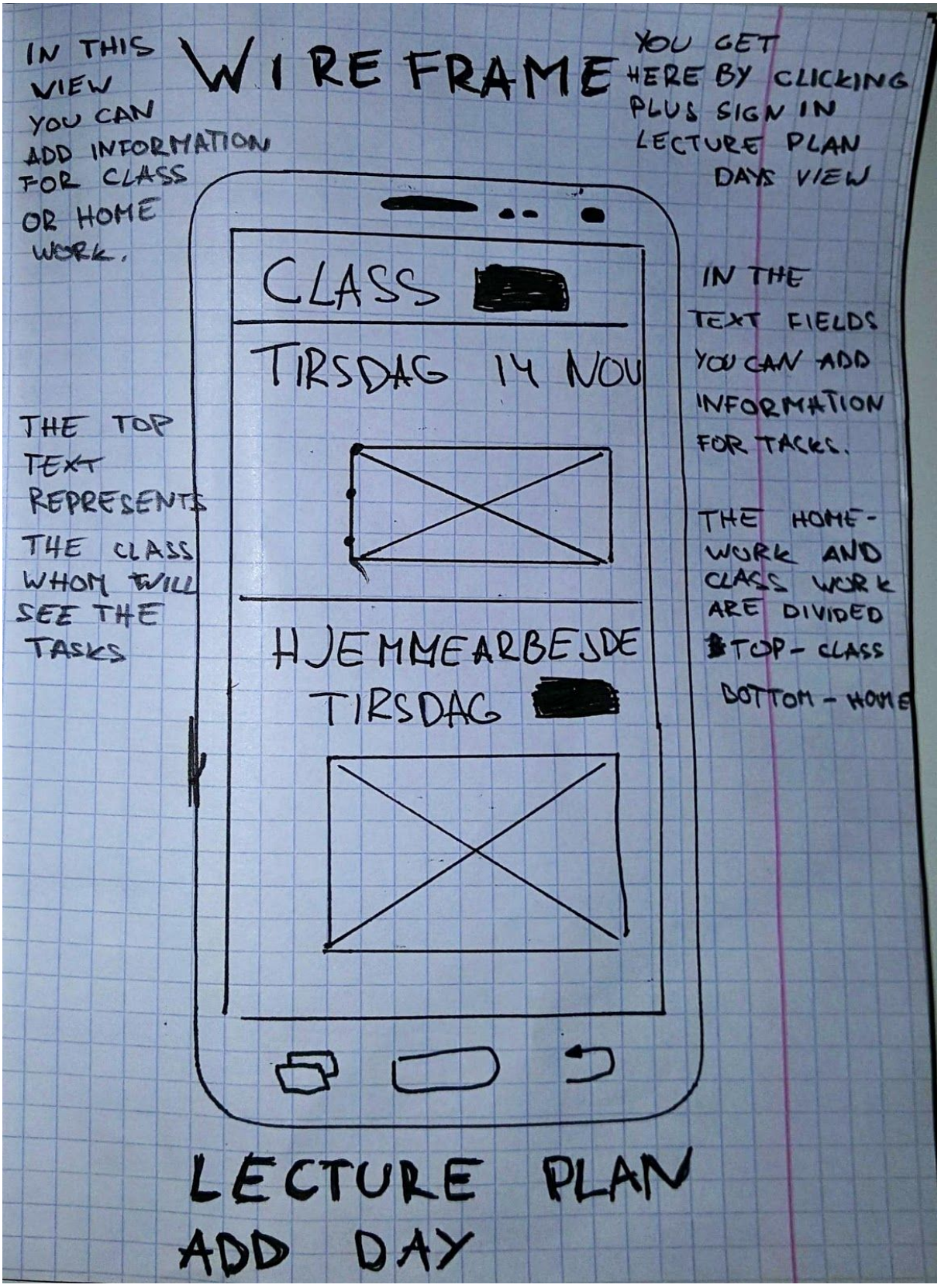
hedv0149 moved this from **Tasks** to **Quality Assurance** in **Main** 7 days ago

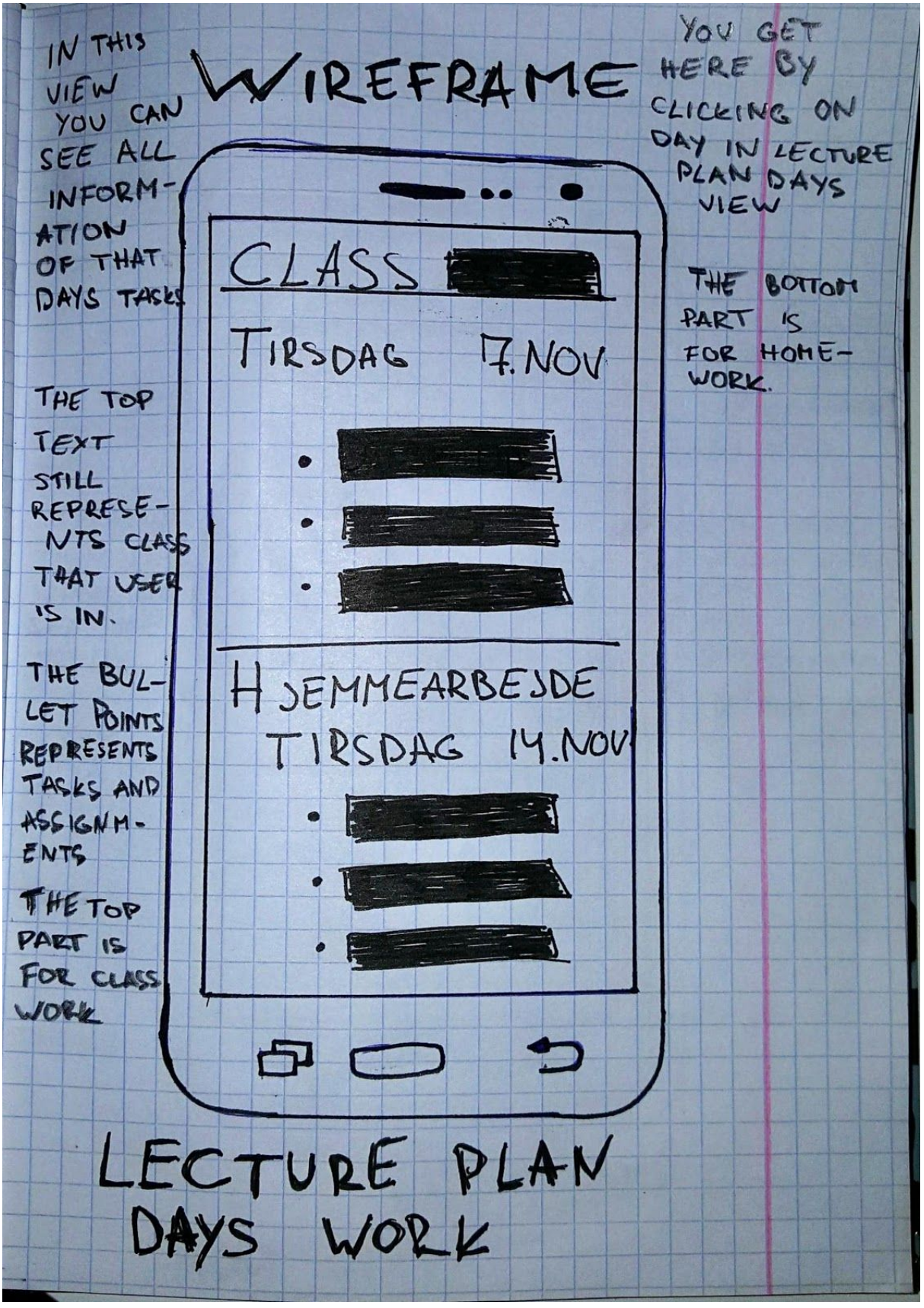
Matt2694 moved this from **Quality Assurance** to **Tasks** in **Main** 7 days ago

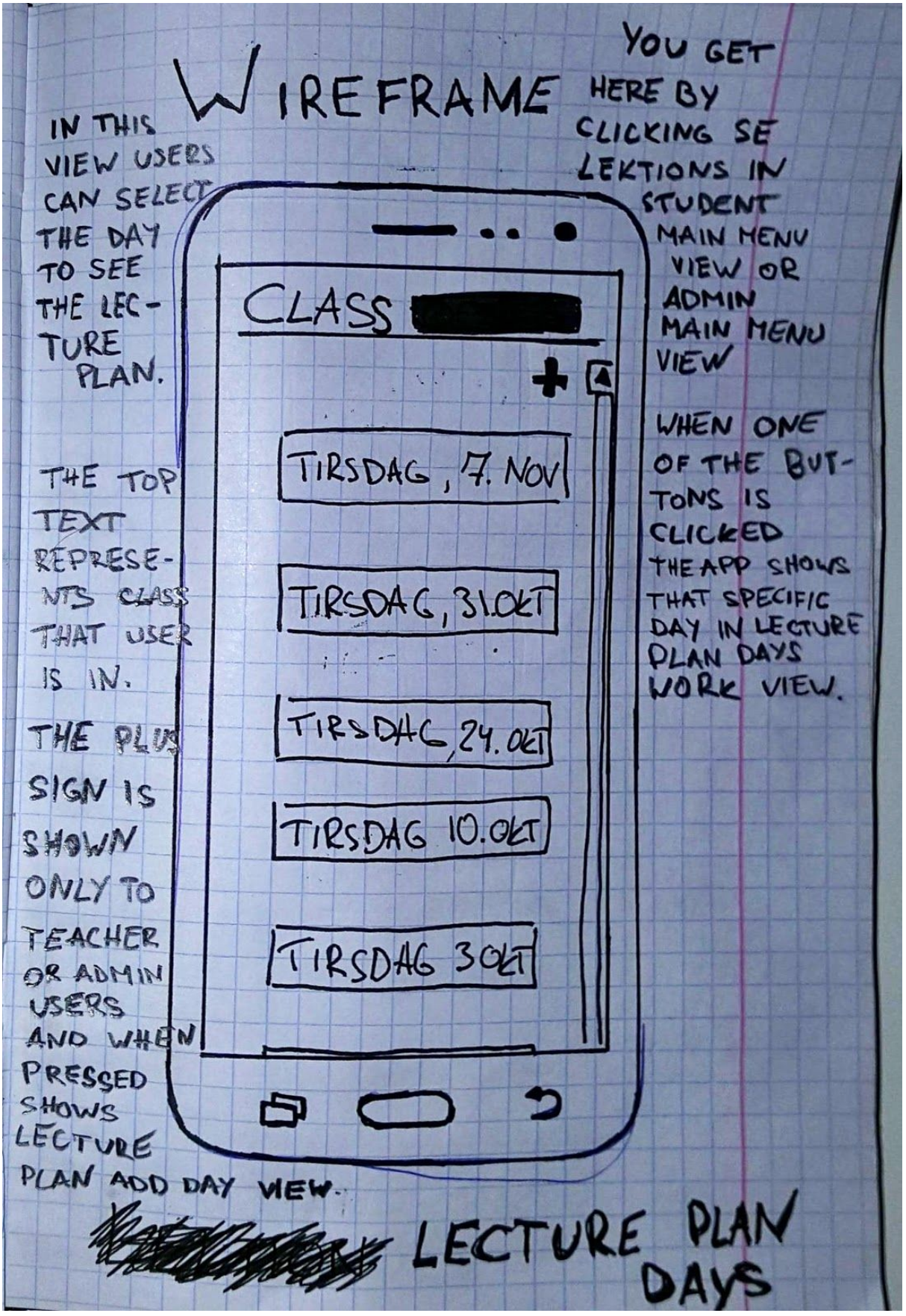
7. Wireframes

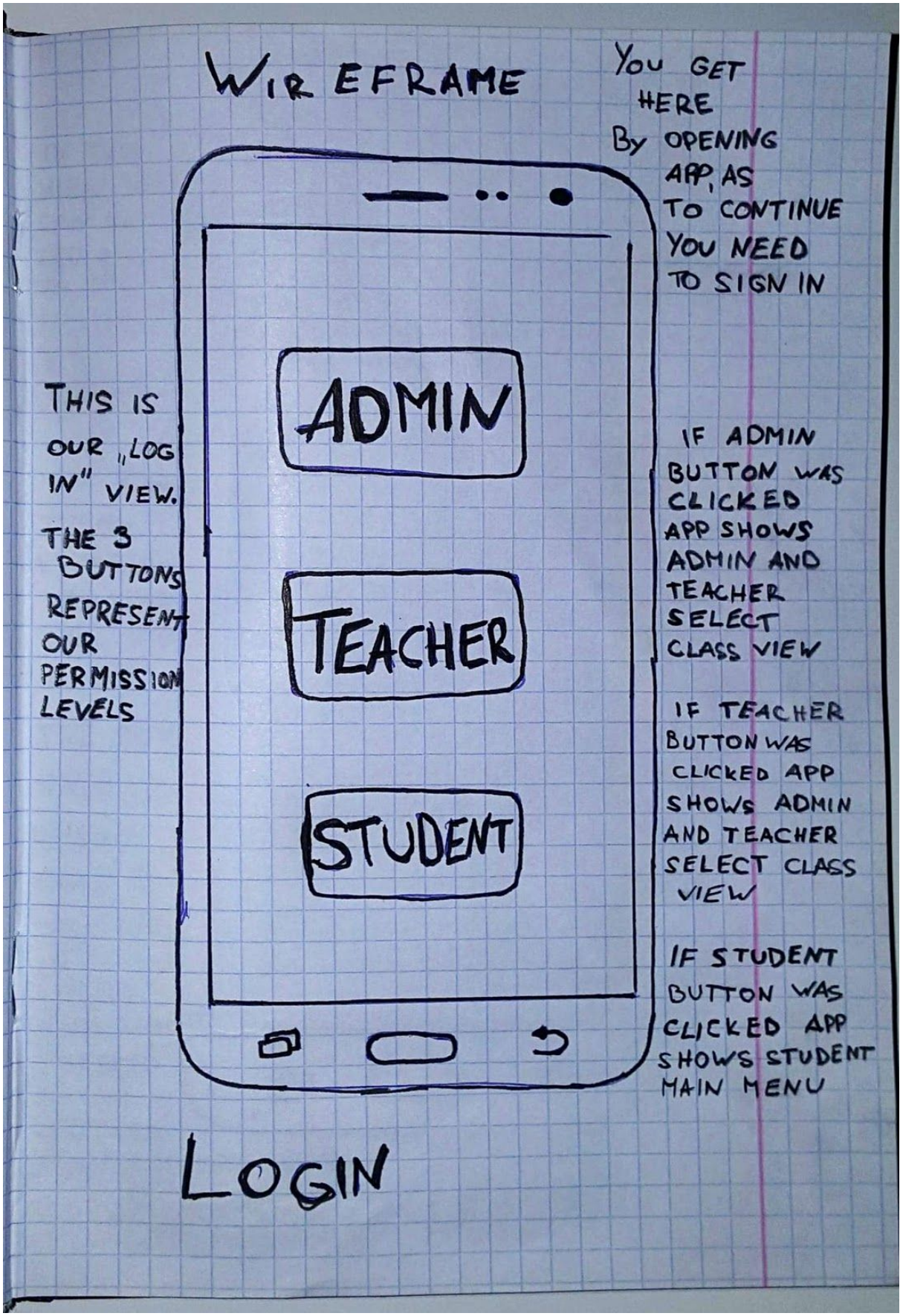


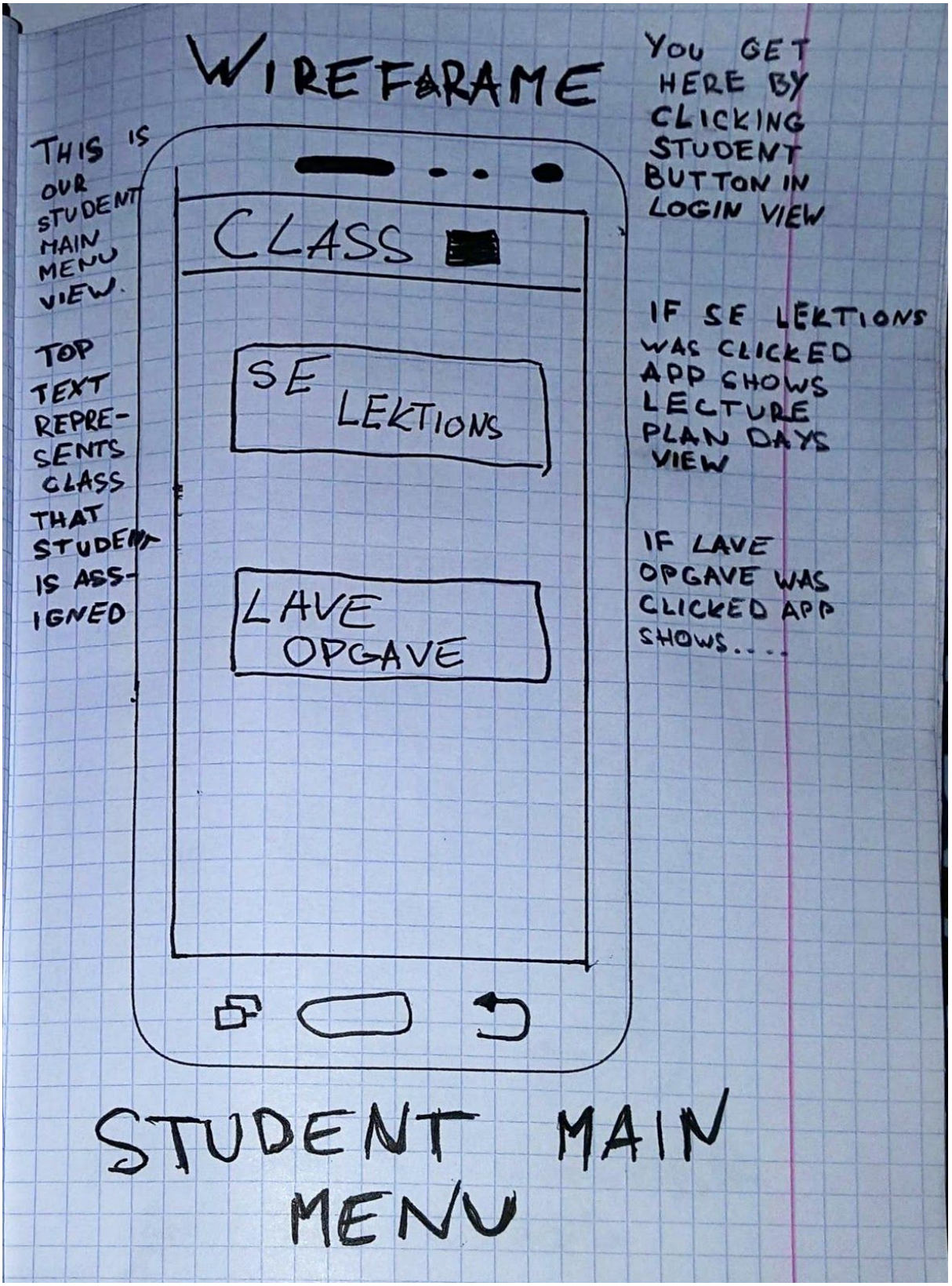

WIREFRAME
THIS IS THE MAIN MENU FOR TEACHERS AND ADMINS.
YOU GET HERE BY SELECTING CLASS IN ADMIN AND TEACHER SELECT CLASS VIEW.
THE OPRET CLASS BUTTON IS VISIBLE ONLY FOR ADMIN USERS. IF CLICKED APP SHOWS
IF SE LEKTIONS PLAN IS CLICKED THEN APP SHOWS LECTURE PLAN DAYS VIEW.
IF OPCAVE IS CLICKED THEN APP SHOWS.
THE TOP TEXT REPRESENTS CLASS THAT USER DID SELECT
CLASS
SE LEKTIONS PLAN
OPGAVE
OPRET CLASS
ADMIN MAIN MENU


IN THIS VIEW YOU CAN ADD INFORMATION FOR CLASS OR HOME WORK.
WIRE FRAME
YOU GET HERE BY CLICKING PLUS SIGN IN LECTURE PLAN DAYS VIEW
CLASS
TIRSDAG 14 NOV
IN THE TEXT FIELDS YOU CAN ADD INFORMATION FOR TACKS.
THE TOP TEXT REPRESENTS THE CLASS WHOM WILL SEE THE TASKS
THE HOME-WORK AND CLASS WORK ARE DIVIDED
TOP - CLASS
BOTTOM - HOME
HJEMMEARBEJDE
TIRSDAG
LECTURE PLAN
ADD DAY


WIREFRAME
IN THIS VIEW YOU CAN SEE ALL INFORM-ATION OF THAT DAYS TASKS
THE TOP TEXT STILL REPRESE-NTS CLASS THAT USER IS IN.
THE BUL-LET POINTS REPRESENTS TASKS AND ASSIGNM-ENTS
THE TOP PART IS FOR CLASS WORK
YOU GET HERE BY CLICKING ON DAY IN LECTURE PLAN DAYS VIEW
THE BOTTOM PART IS FOR HOME-WORK.
CLASS [redacted]
TIRSDAG 7. NOV
HJEMMEARBEJDE
TIRSDAG 14. NOV
LECTURE PLAN DAYS WORK

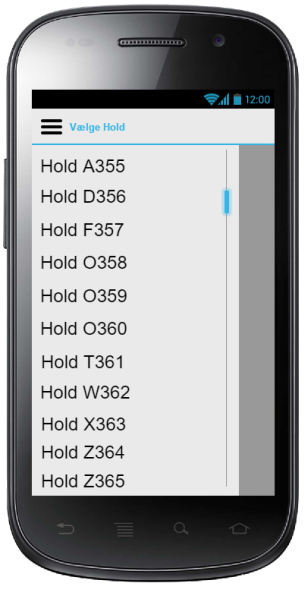

WIREFRAME
YOU GET HERE BY CLICKING SE LEKTIONS IN STUDENT MAIN MENU VIEW OR ADMIN MAIN MENU VIEW
IN THIS VIEW USERS CAN SELECT THE DAY TO SEE THE LEC-TURE PLAN.
CLASS [redacted]
TIRSDAG, 7. NOV
TIRSDAG, 31.OKT
TIRSDAG, 24. OKT
TIRSDAG 10.OKT
TIRSDAG 3 OKT
WHEN ONE OF THE BUT-TONS IS CLICKED THE APP SHOWS THAT SPECIFIC DAY IN LECTURE PLAN DAYS WORK VIEW.
THE TOP TEXT REPRESE-NTS CLASS THAT USER IS IN.
THE PLUS SIGN IS SHOWN ONLY TO TEACHER OR ADMIN USERS AND WHEN PRESSED SHOWS LECTURE PLAN ADD DAY VIEW.
[redacted] LECTURE PLAN DAYS

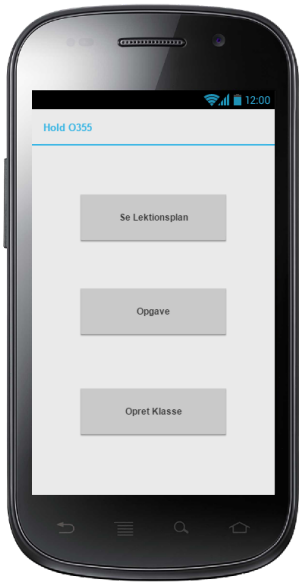

WIREFRAME
You get here By opening app, as to continue you need to sign in
ADMIN
TEACHER
STUDENT
This is our „Log in" view.
The 3 buttons represent our permission levels
If admin button was clicked app shows admin and teacher select class view
If teacher button was clicked app shows admin and teacher select class view
If student button was clicked app shows student main menu
LOGIN

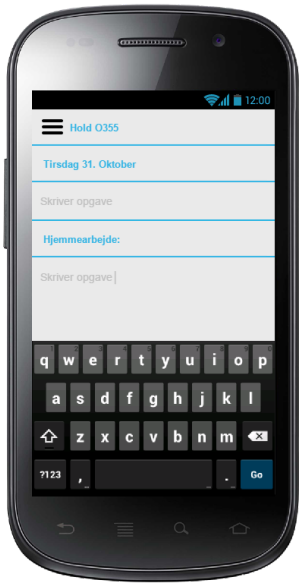

WIREFRAME
THIS IS OUR STUDENT MAIN MENU VIEW.
TOP TEXT REPRE-SENTS CLASS THAT STUDENT IS ASS-IGNED
CLASS
SE LEKTIONS
LAVE OPGAVE
YOU GET HERE BY CLICKING STUDENT BUTTON IN LOGIN VIEW
IF SE LEKTIONS WAS CLICKED APP SHOWS LECTURE PLAN DAYS VIEW
IF LAVE OPGAVE WAS CLICKED APP SHOWS....
STUDENT MAIN MENU

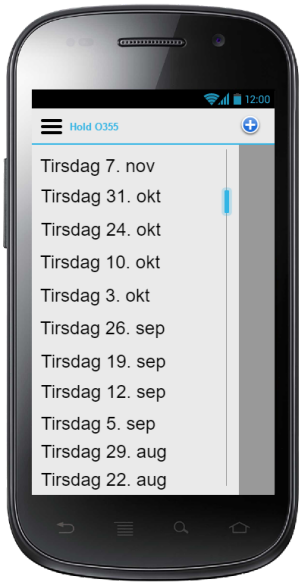
8. Mockups

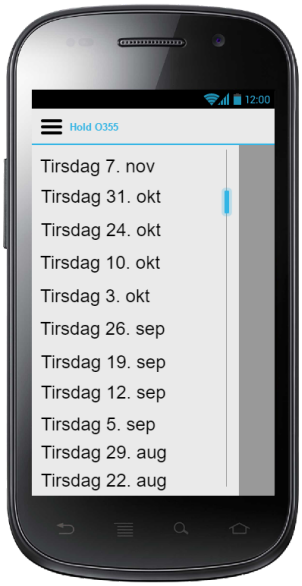

12:00
Hold O355
Se Lektionsplan
Lave Opgave


12:00
Vælge Hold
Hold A355
Hold D356
Hold F357
Hold O358
Hold O359
Hold O360
Hold T361
Hold W362
Hold X363
Hold Z364
Hold Z365


12:00
Hold O355
Se Lektionsplan
Opgave
Opret Klasse


12:00
Hold O355
Tirsdag 31. Oktober
Skriver opgave
Hjemmearbejde:
Skriver opgave
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
Go

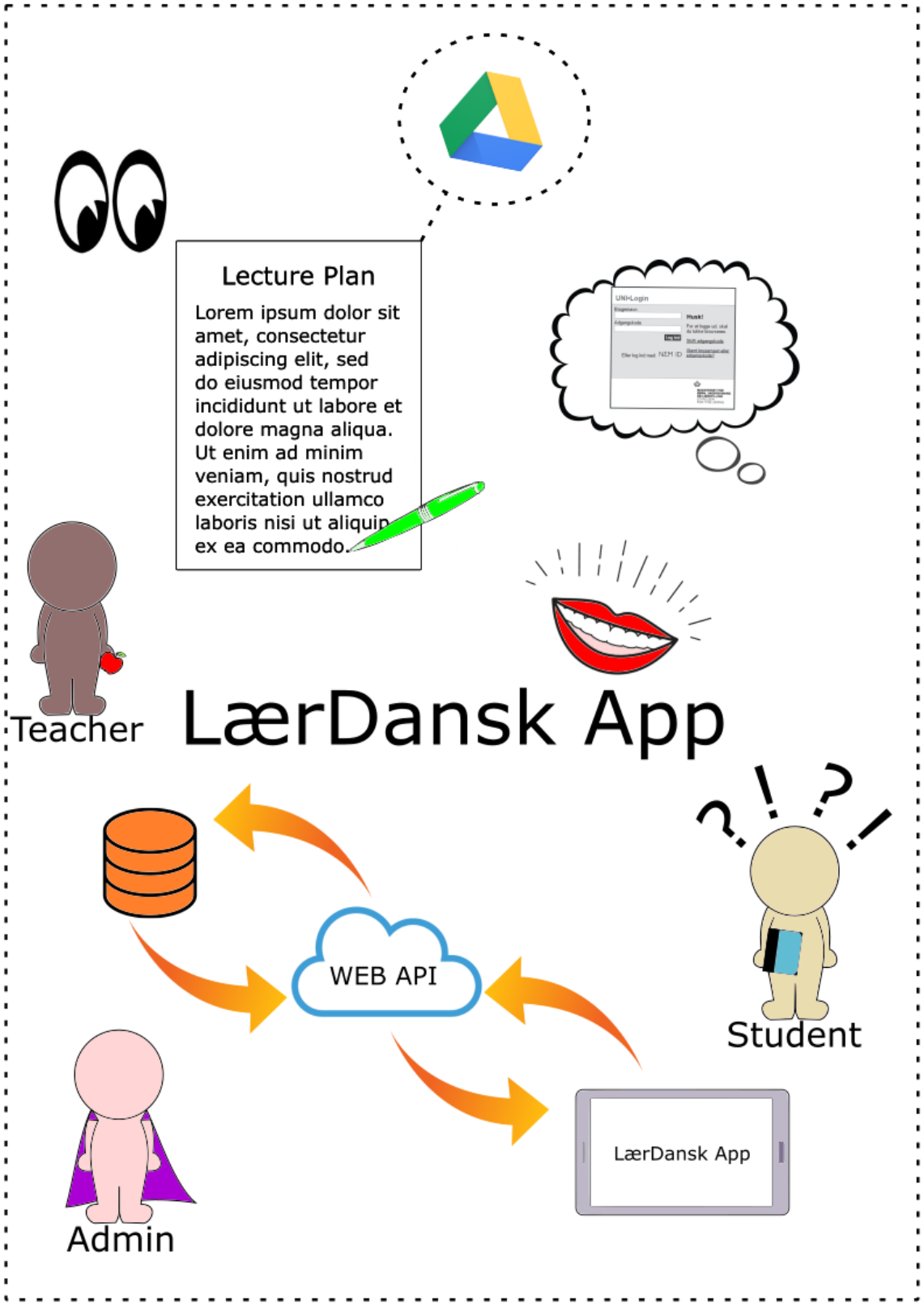

12:00
Hold O355
Tirsdag 7. nov
Tirsdag 31. okt
Tirsdag 24. okt
Tirsdag 10. okt
Tirsdag 3. okt
Tirsdag 26. sep
Tirsdag 19. sep
Tirsdag 12. sep
Tirsdag 5. sep
Tirsdag 29. aug
Tirsdag 22. aug


12:00
Hold O355
Tirsdag 7. nov
Tirsdag 31. okt
Tirsdag 24. okt
Tirsdag 10. okt
Tirsdag 3. okt
Tirsdag 26. sep
Tirsdag 19. sep
Tirsdag 12. sep
Tirsdag 5. sep
Tirsdag 29. aug
Tirsdag 22. aug


12:00
Hold O355
Tirsdag 31. Oktober
Dansk Udtale for begyndere - Kapitel 1
Vi gennemgår hjemmearbejdet
Hvordan skriver man en jobansøgning
VMD: øvelse 2, s. 102
VMD: s. 113. Adjektiver (grund-t og e-form)
VMD: s.25 + øvelse 18, s. 27
Hjemmearbejde:
Skriv en ansøgning
VMD: øvelse 17 og 18, s. 113

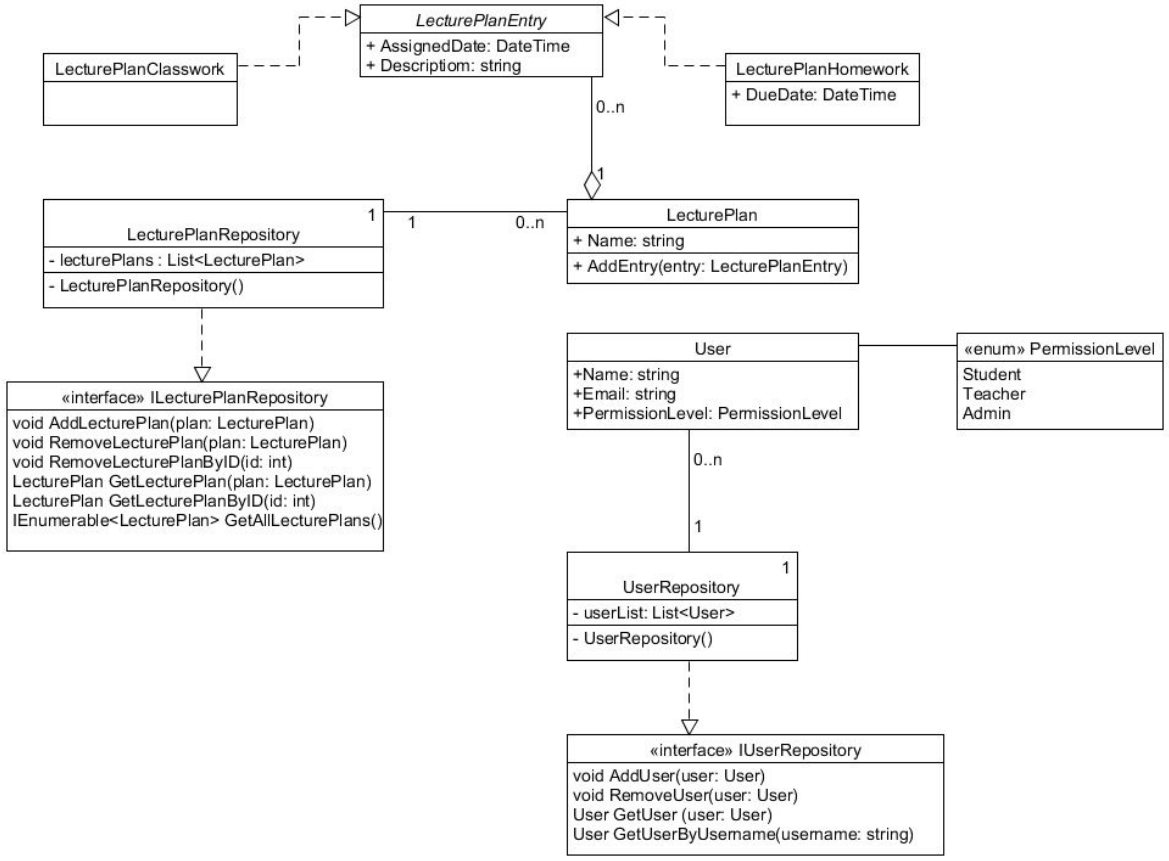

12:00
Admin
Teacher
Student

9. Rich Picture

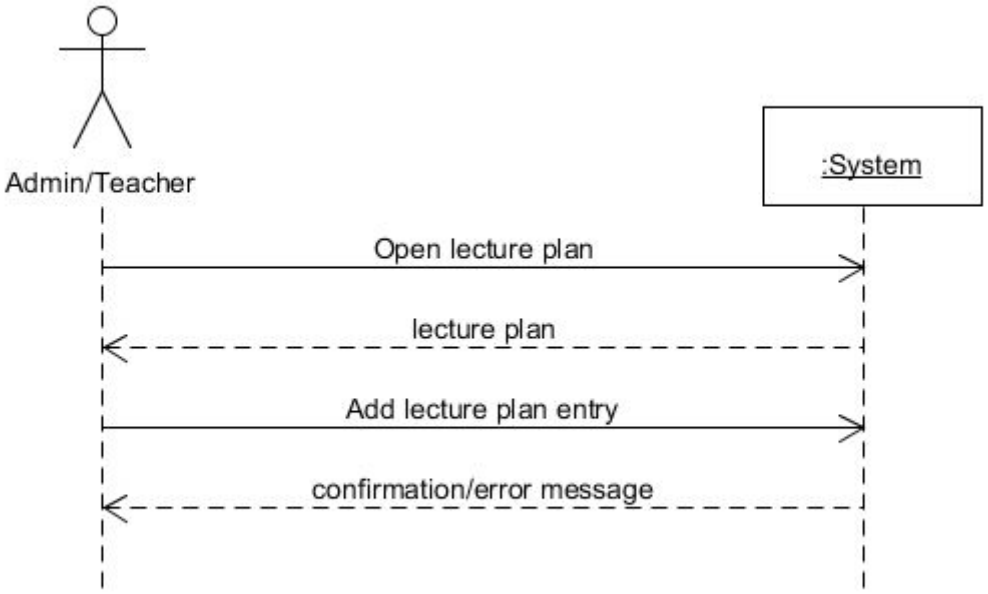


10. SSDs

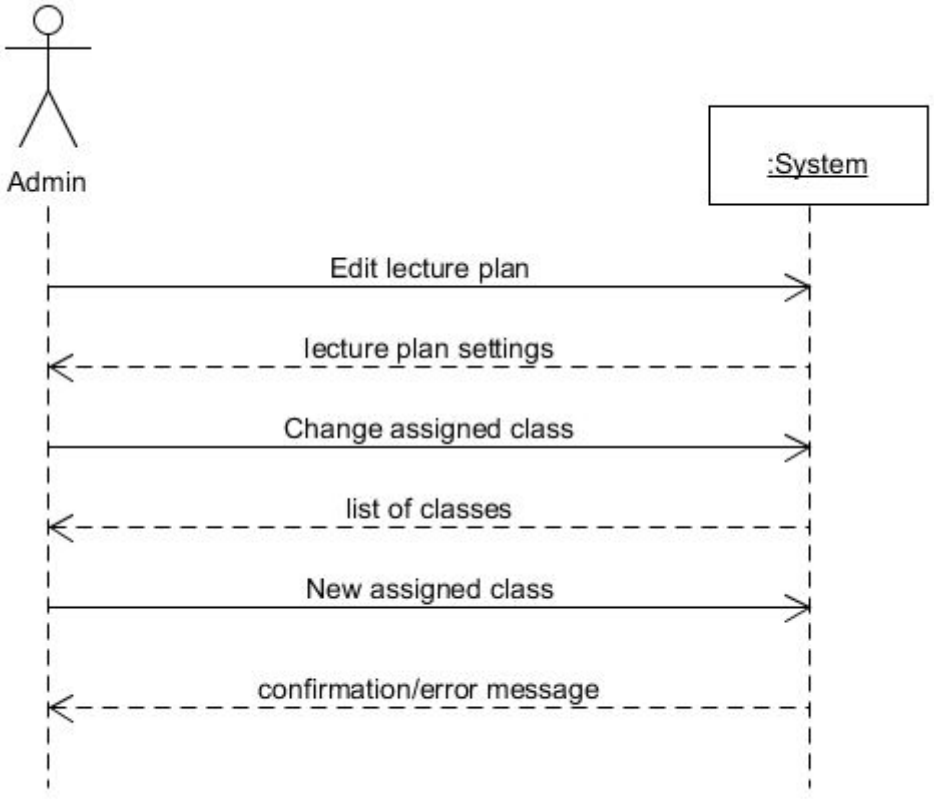
10.1. DCD



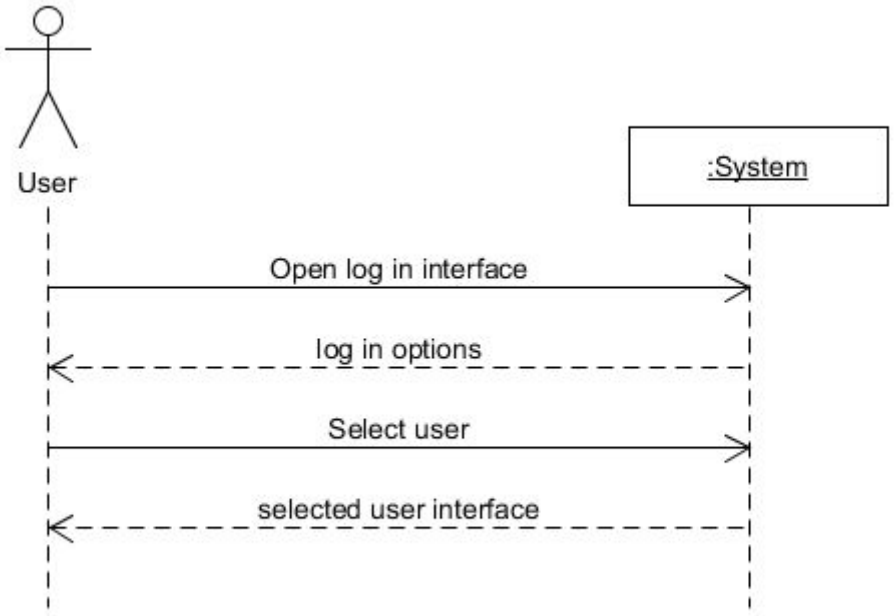
10.2. US 1.3


Admin/Teacher
:System
Open lecture plan
lecture plan
Add lecture plan entry
confirmation/error message

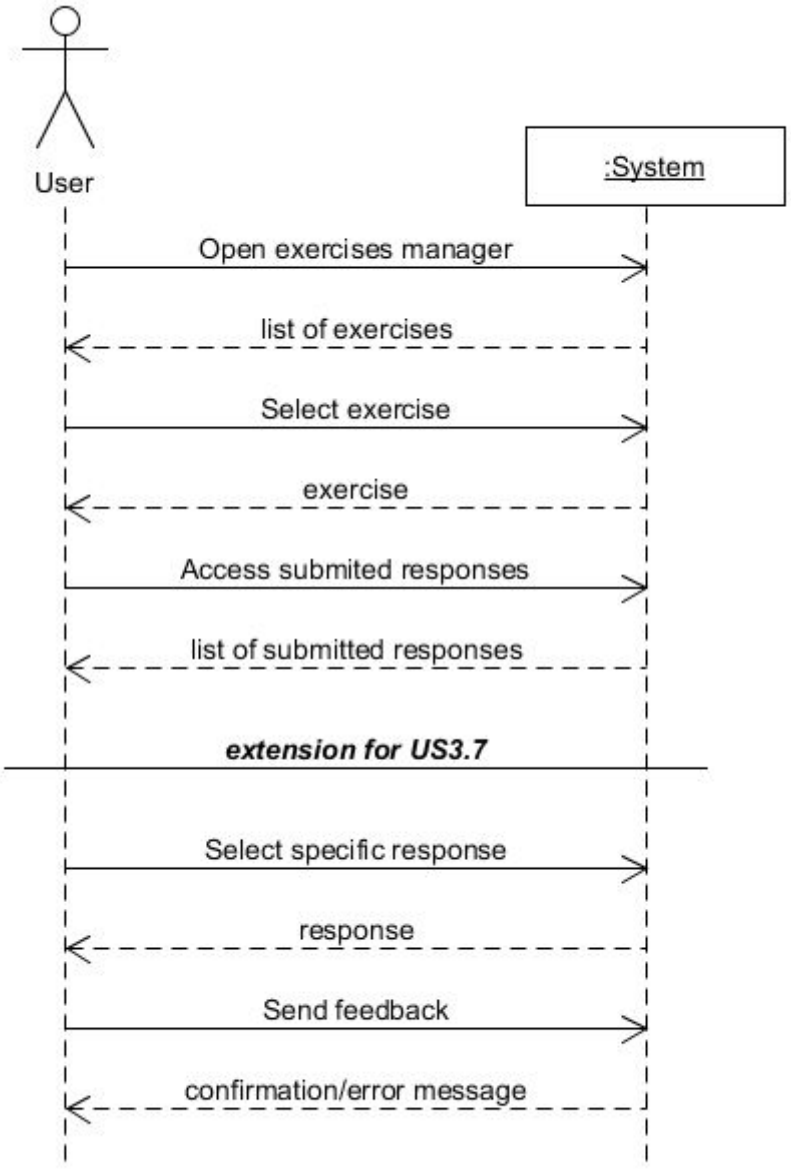
10.3. US 1.5


Admin
:System
Edit lecture plan
lecture plan settings
Change assigned class
list of classes
New assigned class
confirmation/error message

10.4. US 2


User
:System
Open log in interface
log in options
Select user
selected user interface

10.5. US 3.6


User
:System
Open exercises manager
list of exercises
Select exercise
exercise
Access submited responses
list of submitted responses
extension for US3.7
Select specific response
response
Send feedback
confirmation/error message

11. Criterias

11.1. Wireframes / Mockups

11.1.1. Mockups

The most common use of mockups in software development is to create user interfaces that show the end user what the software will look like without having to build the software or the underlying functionality.

- Mockups provide visual details, such as colors and typography.
- Mockups are built to give the viewer a more realistic impression of how the end product will look(superior visuals).
- Mockup adds visual richness to the foundation laid out by the wireframe.
- https://www.youtube.com/watch?v=GrV2SZuRPv0

11.1.2. Wireframes[1]

A wireframe should be a basic visual guide to suggest the structure of an interface, and the relationships between the pages. Serves as sketch/blueprint that defines the product's structure, content and functionality.

- No colours (distracting)
- No visual elements (distracting)
- Title at the top
- "How to get here" description
- Concept/Purpose/Objective of this frame
- Content & Data to be shown in the frame
- If/Then behavior (description), what happens if
- Navigation Links / Buttons
- Error Conditions, as a result of user interaction, and the messages displayed

11.2. User Stories

In few words **user stories** are:

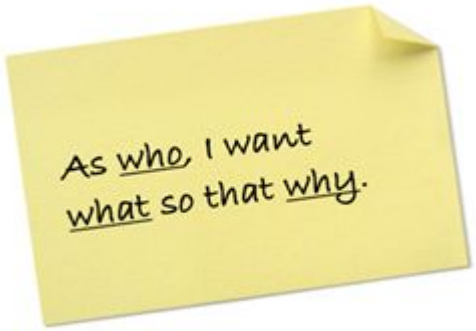
- short;
- simple descriptions;
- told from the perspective of the person who desires the new capability;
- easy to understand;
- not confusing or using ambiguous terms;

[1] https://www.usability.gov/sites/default/files/creating-wireframes.pdf

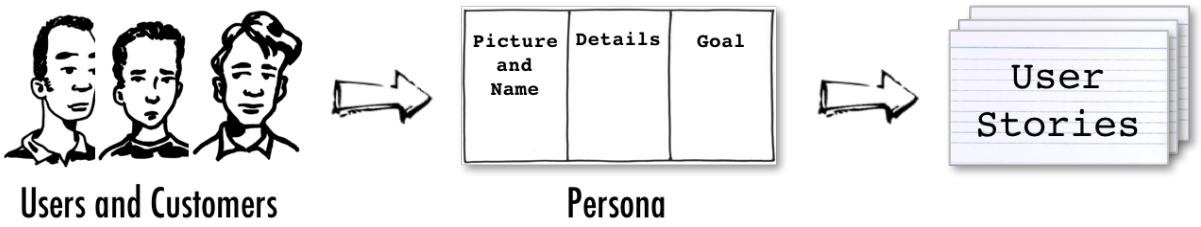
- follows template.

TEMPLATE:

As a < type of user >, I want < some goal > so that < some reason >.

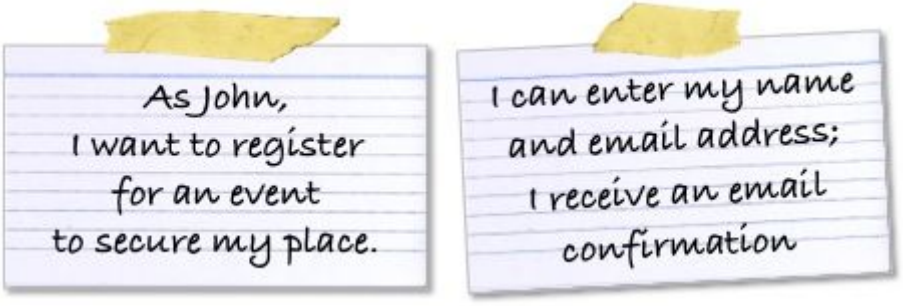


By communicating with your product owner and making personas it is easy to see their needs and make Epics. User stories are divided into **Epics** and **normal user stories**.

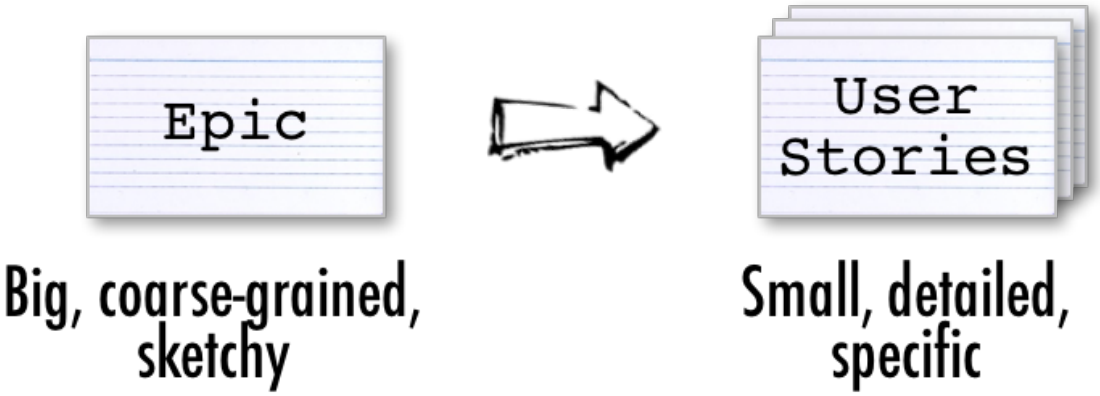


Start with Epics

An epic is a big, sketchy, coarse-grained story. It is typically broken into several user stories over time. Starting with epics allows you to sketch the product functionality without committing to the details. This is particularly helpful for describing new products and features: It allows you to capture the rough scope, and it buys you time to learn more about how to best address the needs of the users. After making Epic – left card – you need the acceptance criteria – captured on the right card – to make smaller user stories. They are used as checklist.



Break your epics
Break your epics into smaller, detailed stories until they are ready: clear, feasible, and testable. Ready stories are small, detailed stories that can be implemented.



11.3. System Sequence Diagrams

11.3.1. Why Draw an SSD?

Basically, a software system reacts to three things: 1) external events from actors (humans or computers), 2) timer events and 3) faults or exceptions (which are often from external resources). Therefore, it is useful to know what, *precisely*, are the external input events, the system events.

System behavior is a description of what a system does, without explaining how it does it. One part of that description is a system sequence diagram.

Draw an SSD for a main success scenario of each use case, and frequent or complex alternative scenarios.

11.3.2. Criteria

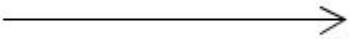
- System events should be expressed at the abstract level of intention rather than in terms of the physical input device. (i.e. “enterItem” is better than “scan” (that is, laser scan) because it captures the *intent* of the operation while remaining abstract and noncommittal.)
- Everything else is the same as for SDs.

11.4. Sequence Diagram

Criteria:

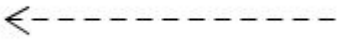
- illustrate interactions in kind of fence format, in which each new object is added to the right;

- clearly shows sequence or time ordering of messages;
- singleton instance in UML is marked with ”1” in the upper right corner of the lifeline box;
- lifeline boxes include a vertical dashed line extending below that represent actual lifespan of objects;
- SD starts with an opening solid ball with sender;
- Messages (arrows) can be shown in the following ways:
    - An asynchronous Message has an open arrow head.

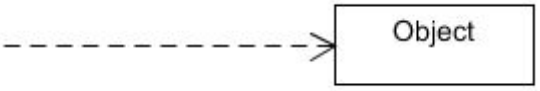


    - A synchronous Message has a filled arrow head.

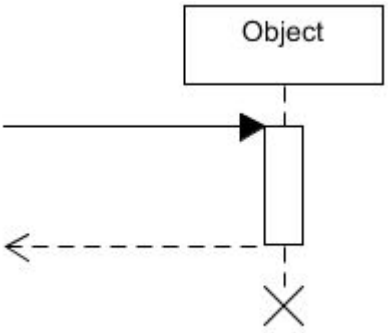
    - A reply Message has a dashed line with an open arrow head.



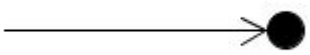
    - An object creation Message has a dashed line with an open arrow head.



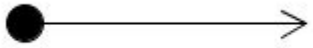
    - An object deletion Message must end in a DestructionOccurrenceSpecification.



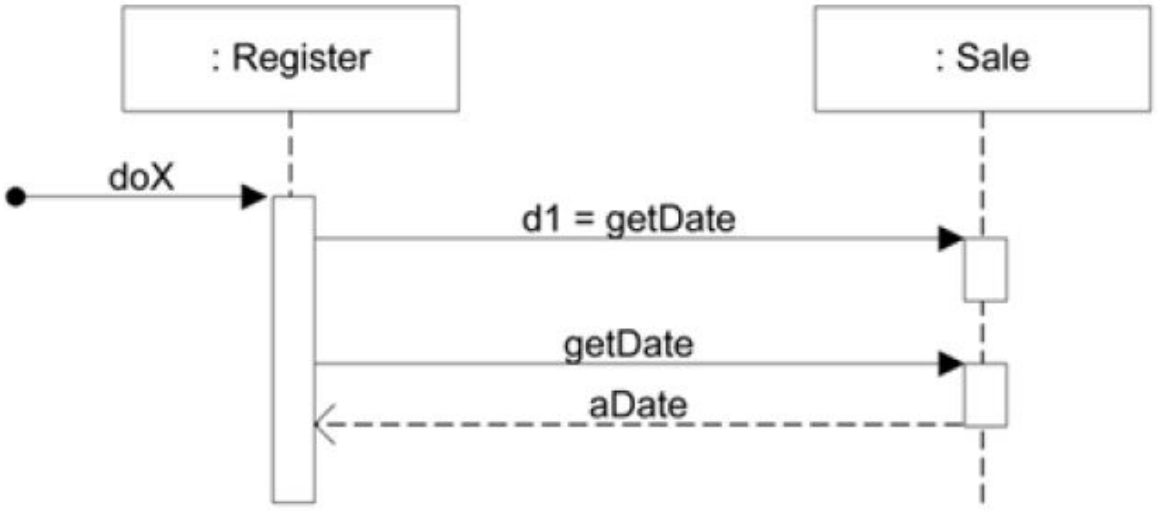
    - A lost Message is denoted with a small black circle at the arrow end of the Message.



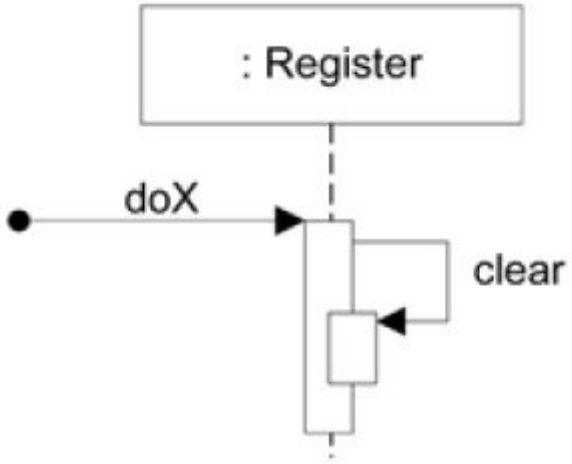
    - A found Message is denoted with a small black circle at the starting end of the Message.



- the return can be shown in to ways:
    - a=getDate on the call message;
    - a call message with getDate and then a reply message with aDate;

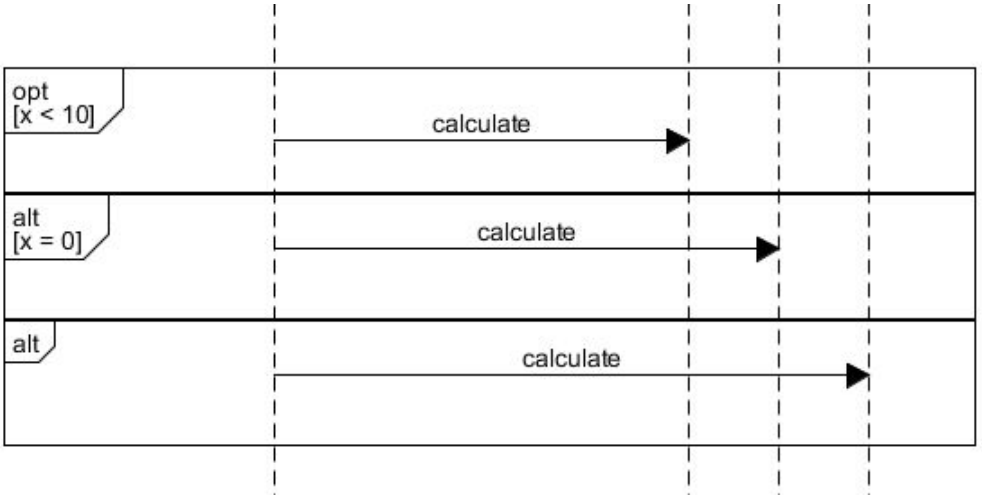


- by pointing solid, filled arrow back to objects activation bar, that has smaller activation bar on it, to show a message being sent from an object to itself;

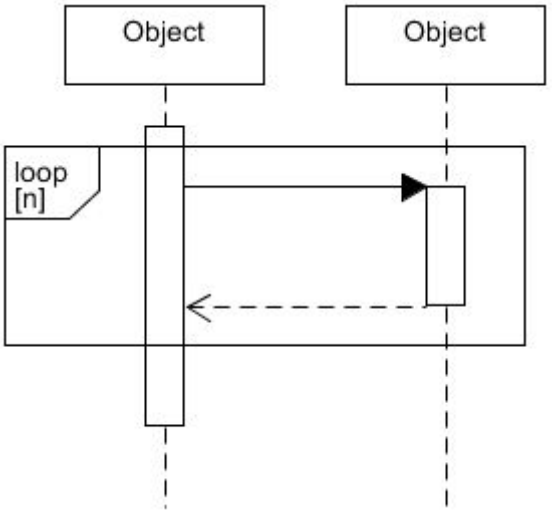


- to support conditional and looping constructs use frames, in frames left, top corner a box with type of loop and condition is shown. Loop types:
    - opt [condition] = if;
    - alt [condition] = else if;
    - alt = else;

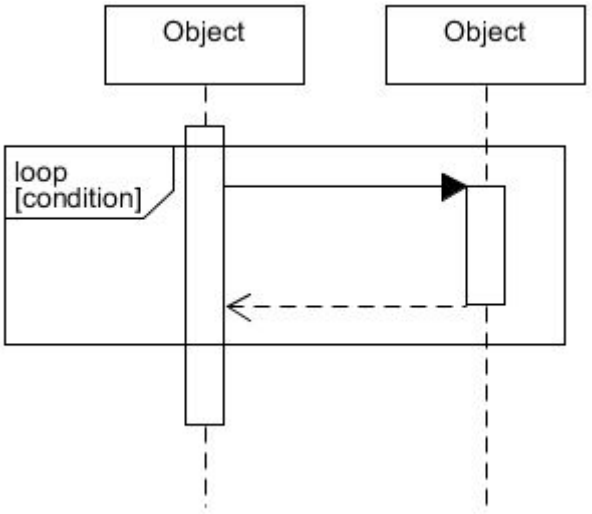
According to larman, if it's a single if, the note should read "opt", but with else-if, larman specifies it as "alt", with conditions written in the middle. We decided against this, as we find it too confusing, and is also easier to do in umlet.

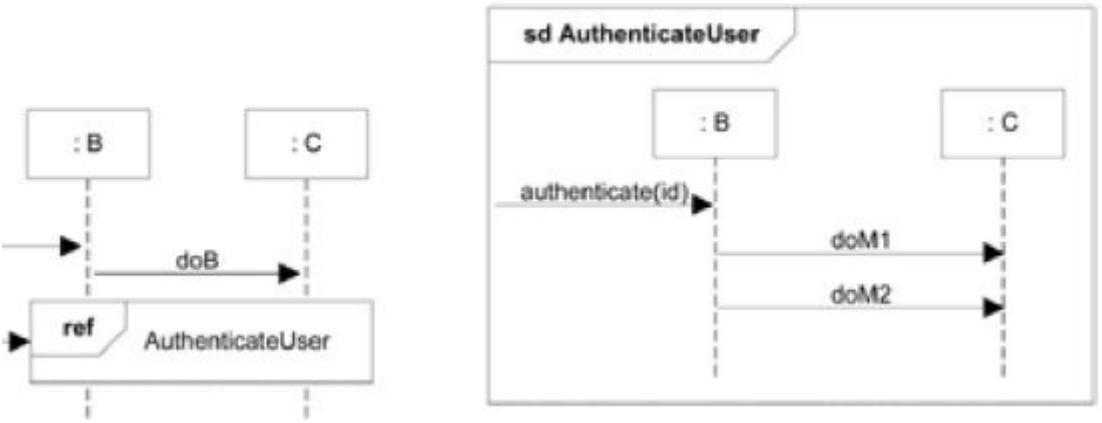


- par = parallel fragments that execute in parallel;
- region = critical region within which only one thread can run;
- loop [n times or as we do it in c#] = for n times;



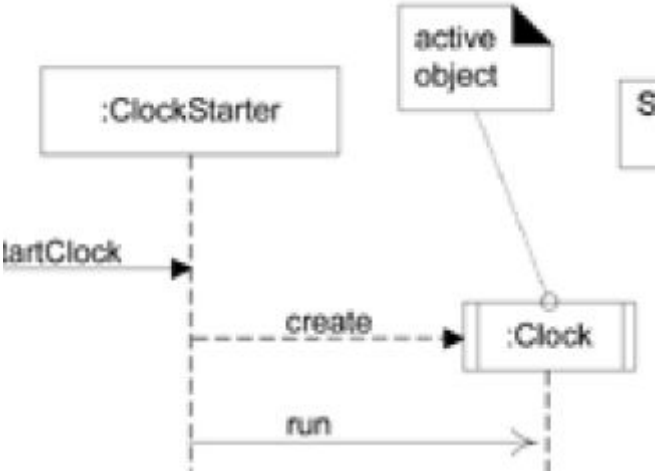
- loop [condition] = while

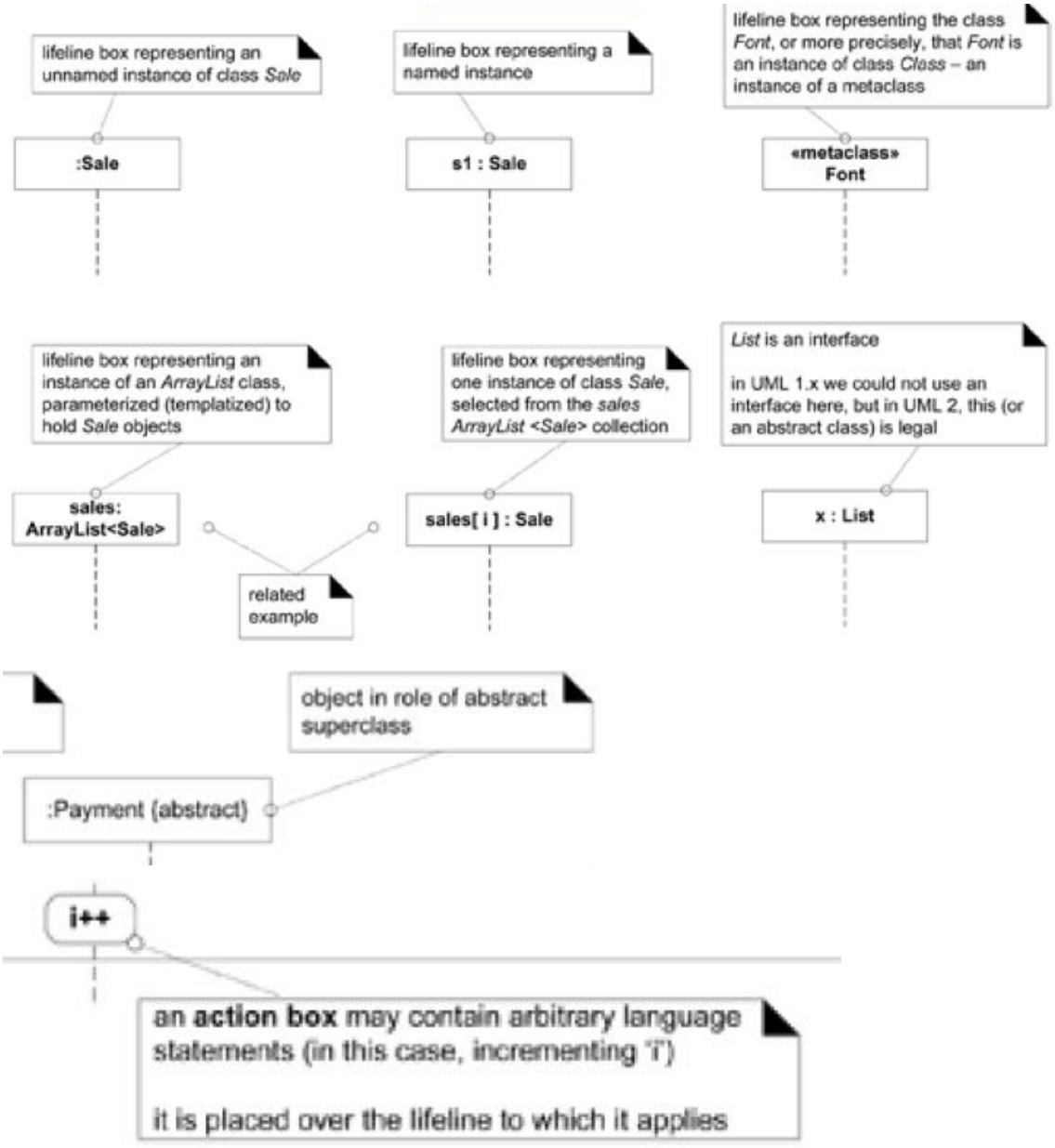




- ref = referenc from other SD, the name of SD is written in the box;

- For threads a create function goes to the active thread object and then run method arrow starts the lifebox;



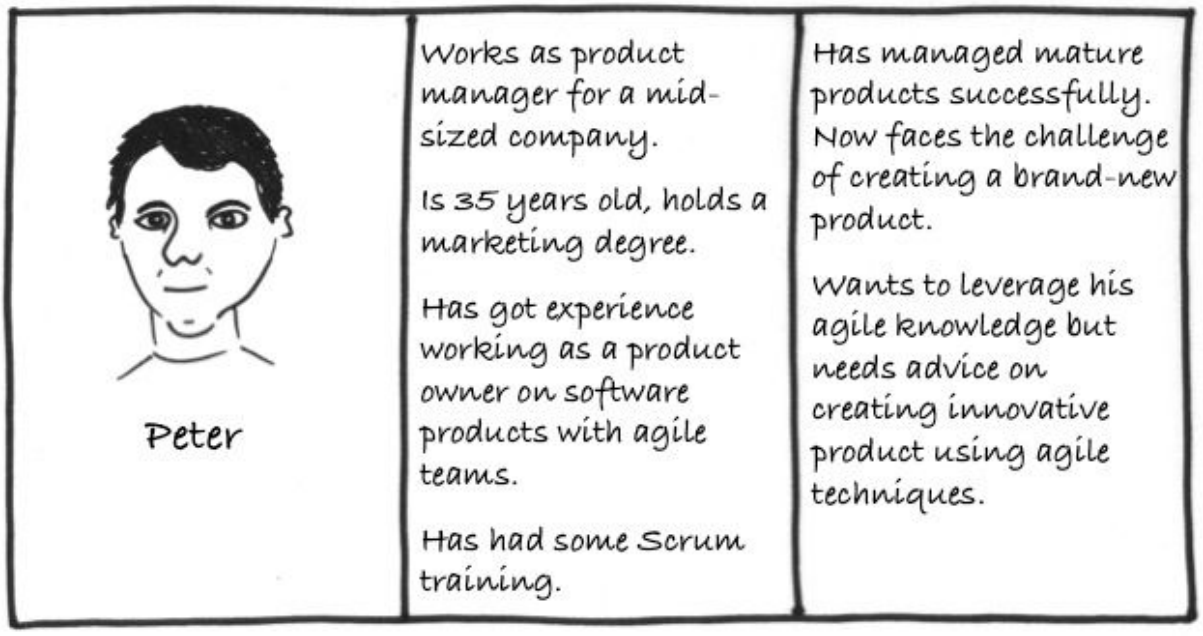


11.5. Personas

- It's not a real person, but based on information about the usertype.
- Described in great detail
- The personas should be different age groups and different type of people so we capture.
- Great for user centric design
- Should help you catch alternative flows in a use-case.

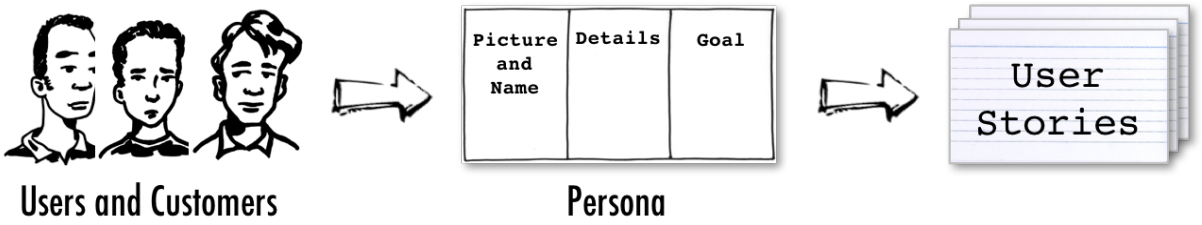
PICTURE & NAME	DETAILS	GOAL
What does the persona look like? What is its name? Choose a picture and a name that are representative, and that allow you to develop sympathy for the persona.	What are the persona's relevant characteristics and behaviours? Consider demographics, job, lifestyle, spare time activities, attitudes, and common tasks, for instance.	Why would the persona want to buy or use the product? What problems should the product solve? What benefits does the persona want to achieve? If there are multiple problems or benefits, identify the main one and put it at the top.

http://www.romanpichler.com/



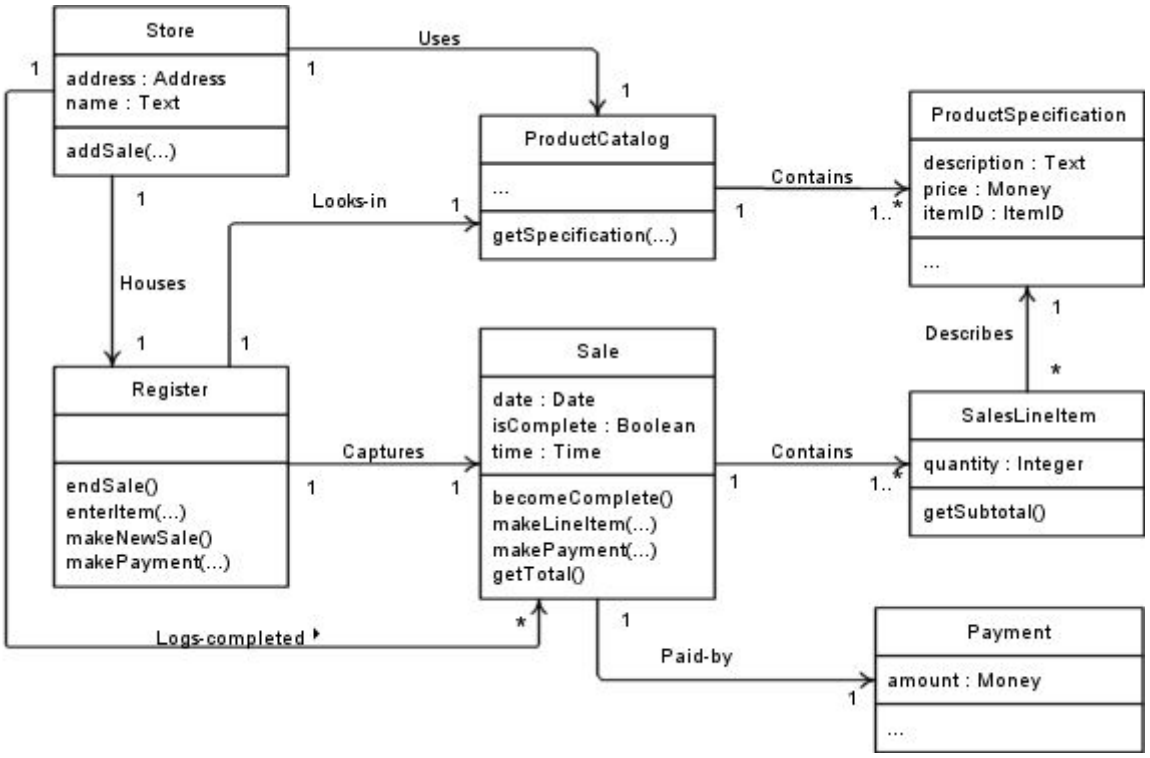
Personas are fictional characters that are based on first-hand knowledge of the target group. They usually consist of a name and a picture; relevant characteristics, behaviours, and attitudes; and a goal. The goal is the benefit the persona wants to achieve, or the

problem the character wants to see solved by using the product.



11.6. Design Class Diagram

Example:



DCD = Design Class Diagram

Purpose:
Further your understanding of how the software should be developed and use it to communicate it to other developers. Helps to get an overview of the system and to see what design decisions could be made. Needs to be updated during the development phase as you expand on your system.

Criterias[2]:

[2] https://courses.cs.washington.edu/courses/cse403/11sp/lectures/lecture08-uml1.pdf

- The connection between 2 software classes needs a word between them that describe the relationship. See Discussion [3]
- The number of items contained in an object is shown on the opposite side of the line, except with inheritance.*(One ProductSpecification can have infinite amount of SalesLineItem, but one SalesLineItem can have one ProductSpecification)*

ProductSpecification 1 ——— 0..n *SalesLineItem*

- Interface class has <<interface>> in front of the name and the name starts with I.
- Exceptions should be written in a custom compartment called “Exceptions Thrown” [4]
- Enum class has <<enum>> in front of the name.
- aggregation: “is part of” – symbolized by a clear white diamond
- composition: “is entirely made of” – stronger version of aggregation – the parts live and die with the whole – symbolized by a black diamond
- dependency: “uses temporarily” – symbolized by dotted line – often is an implementation detail, not an intrinsic part of that object's state
- Operations / Methods should not include inherited methods
- visibility markers:
    - + public
    - # protected
    - - private
    - ~ package (default)
    - / derived (not stored, but can be computed from other attribute values)
- Static attributes & methods are underlined
- Abstract classes are italics
- Omit return type for constructors ~~and when void~~[5]
- Parameters are listed as (name: type)
- Comments are folded notes, attached via a dashed line

General

- In contrast to the domain model showing real-world classes, this diagram shows software classes.

Attributes

Methods

[3] https://github.com/jona8690/Project-Web/issues/11#issuecomment-342468441
[4] As seen in Larman Section 16.19
[5] https://github.com/jona8690/Project-Web/issues/11#issuecomment-346802509

Examples:

- attribute
    - visibility name : type [count] = default_value
    - - balance : double = 0.00
- class operations/ methods
    - visibility name (parameters) : return_type
    - + distance(p1: Point, p2: Point) : double
- multiplicity (how many are used)
    - * ⇒ 0, 1, or more
    - 1 ⇒ 1 exactly
    - 2..4 ⇒ between 2 and 4, inclusive
    - 3..* ⇒ 3 or more (also written as “3..”)

12. Database Design

Choosing our database SQL vs firebase:
https://github.com/jona8690/Project-Web/issues/32

Fields required (unnormalized form):

US1 - Lecture Plan:

LecturePlanName	EntryDueDate	EntryAssignedDate	EntryDescription
O12345	null	13 Oct 2017	Read Chapter 5
O12345	15 Oct 2017	13 Oct 2017	Repeat “Jeg elsker Danmark” 500 times
J678	null	13 Oct 2017	Read Chapter 9
J678	15 Oct 2017	13 Oct 2017	Prepare sentences about home country
J678	null	13 Oct 2017	Research pancake recipes

Composite Keys

LectrePlanName, EntryAssignedDate, EntryDescription

Normalization:

1st normal form: atomic values.
Our table does not contain non-atomic values, hence complies to 1st normal form.

As per our 2nd normal form, we should drop EntryAggiendDate and EntryDescroption down to its own table. This leaves EntryDueTake with the LecturePlanName, as its not part of the composite key. This still violates 2nd normal form, hence we also drop EntryDueDate down, leaving us with:

Table: LecturePlans

LecturePlanName
O12345
J678

Table: LecturePlanEntries

LecturePlanName	EntryDueDate	EntryAssignedDate	EntryDescription
O12345	null	13 Oct 2017	Read Chapter 5
O12345	15 Oct 2017	13 Oct 2017	Repeat “Jeg elsker Danmark” 500 times
J678	null	13 Oct 2017	Read Chapter 9
J678	15 Oct 2017	13 Oct 2017	Prepare sentences about home country
J678	null	13 Oct 2017	Research pancake recipes

MOAR NORMALIZATION

12.1. Final Schema

Table: LecturePlans

ID *	Name
1	O12345
2	J678

Table: LecturePlanEntries

ID *	LecturePlanID	DueDate	AssignedDate	Description
1	1	null	13 OCt 2017	Read Chapter 5

2	1	15 OCt 2017	13 Oct 2017	Repeat “Jeg Elsker Danmark” 500 times
3	2	null	13 OCt 2017	Read Chapter 9
4	2	15 Oct 2017	13 Oct 2017	Prepare sentences about home country
5	2	null	13 Oct 2017	Research pancake recipies

12.1.1. Data Structure Diagrams

Giving us this Data Structure for US1

Table: LecturePlans

Column Name	**Data Type**
ID	Primary Key; Integer; Not Null
Name	NVarChar; Not Null

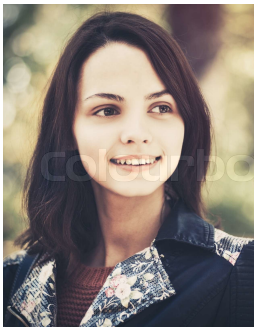
Table: LecturePlanEntries

Column Name	**Data Type**
ID	Primary Key; Integer; Not Null
LecturePlanID	Foreign Key (LecturePlans); Integer; Not Null
DueDate	DateTime; Null
AssignedDate	DateTime; Not Null
Description	LongText; Not Null

13. Personas

13.1. Students

Name: Kathleen Berry



Kathleen is a 17 year old student, moving to Denmark from America, to start studying. Her dad is Danish, but her mother is from America. She has no prior language skills in Danish.

On her first day in LærDansk she is given access to the google site, and is asked to prepare for her next lessons. As she is navigating the google site, she gets increasingly frustrated by the lack of organisation, and her inability to find anything, as the site is completely in Danish.

Unable to find her lecture plan, she meets up to the next lesson, unprepared.

By having a phone app, with an easy to navigate interface, Kathleen will be able to more easily find her lecture plans, and do some exercises when she has time.

13.2. Teachers

Name: Gustav Krogh

Gustav, 27, has been a teacher at LærDansk for 3 months. Both his parents are Danish, and he has lived in Denmark his entire life. He is studying for Business at Lillebaelt Academy. He both speaks and writes fluent Danish.

He regularly updates his lecture plan, but many students always seems to come unprepared. He also finds that updating the lecture plan can be quite a tedious process, with changing documents, and uploading them over and over.

By students getting access to a phone app, Gustav personally believes students will become more engaged in the lectures, as he knows students prefer to use their phones over human interaction, and badly designed websites.

13.3. Admin

Name: Lily Thomsen

Lily is 40 years old, and is an administrative worker at LærDansk. Both her parents are Danish citizens, but her mother immigrated from Germany. She lived her entire life in Denmark herself.

Her normal duties are assigning classrooms to the teachers, and ensuring teachers show up. Sometimes she also deals with other administrative work, such as funding, finances, etc.

She often finds students wandering the corridors, trying to figure out which classroom they are in today, because they did not read their lecture plan from home. She believes that the phone app will solve this, giving students easy access to the information they need.