Info 3

Laboratory 5

02.06.2020

Lab 5: Class Diagrams

Niklas Lengert s0563290
Pavel Tsvyatkov s0559632
Robin Jaspers s0568739
Nataliia Azarnykh s0568691

Lab Tasks:

1. You will be preparing class diagrams for your Corona-App as continued by another group in Lab 4. First **read through the changes and the sequence diagrams** the other group produced. **Do the changes make sense? Can you understand the sequence diagrams?** Note this down in your report.

   Make a list of all class candidates from the use cases and the sequence diagrams. Evaluate their suitability for classes based on the following **criteria:**

   - Do the instances of the classes have attributes that are necessary in the system?
   - Is the information about the existence of instances of the class necessary?
   - Is the 'class' essential for the system?

2. Feel free to introduce new classes or throw out old ones! Make a list of candidate methods, including setter and getter methods for important information. Assign the methods to the classes selected.

3. Now draw a class diagram using any UML-Tool. Make sure to include attributes, methods, and associations. Take your scenarios and use cases - can you walk them through the class diagram? Record any changes you needed to make!

Additional Use Cases we received from PseudoSquad(Group 4)

Use case 5

Name	Change Language
Actor	User
Summary	The user change language in the setting feature
Assumption	User has a smart device to access the app
Precondition	User downloaded the app.
Sequence of steps	1. Open App 2. Go to Main Menu 3. Choose Settings 4. Select Language 5. Choose the new language 6. Save
Trigger	User couldn’t understand the default language well so they want to change the language
Postcondition	The App restarts and changes to the chosen language
Author & Date	PseudoSquad 26.05.2020

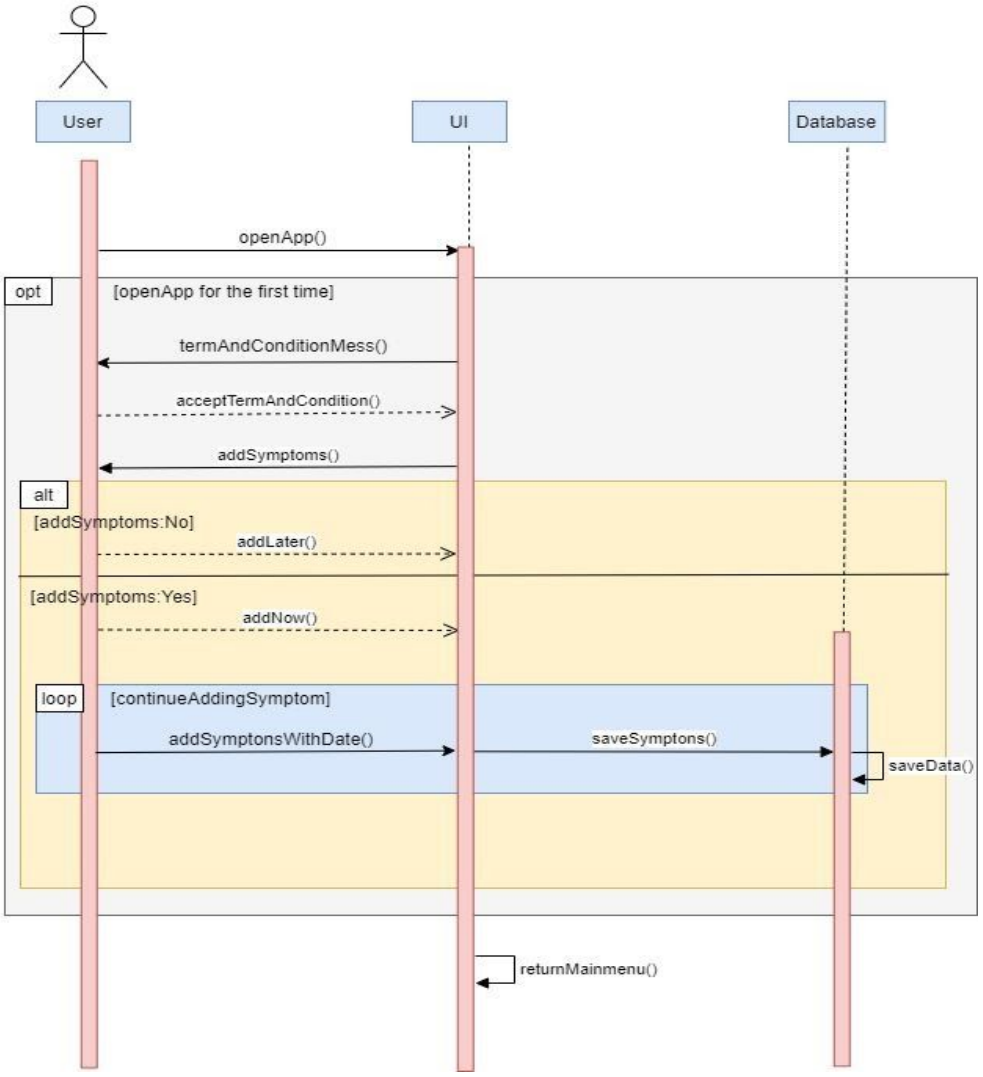
Use Case 6

Name	Read general information of the App
Actor	User
Summary	The user wants to know general information about security, privacy, creators, etc. of the App
Assumption	User has a smart device to access the app
Precondition	User downloaded the app.
Sequence of steps	1. Open App 2. Go to Main Menu 3. Choose “General Info” 4. Choose Security 5. Read the info page 6. Go back and choose Privacy 7. Read privacy information 8. Close App
Trigger	User wants to know how the data is stored, or how their personal information is being secured.
Postcondition	User can access the information from the App.
Author & Date	PseudoSquad 26.05.2020

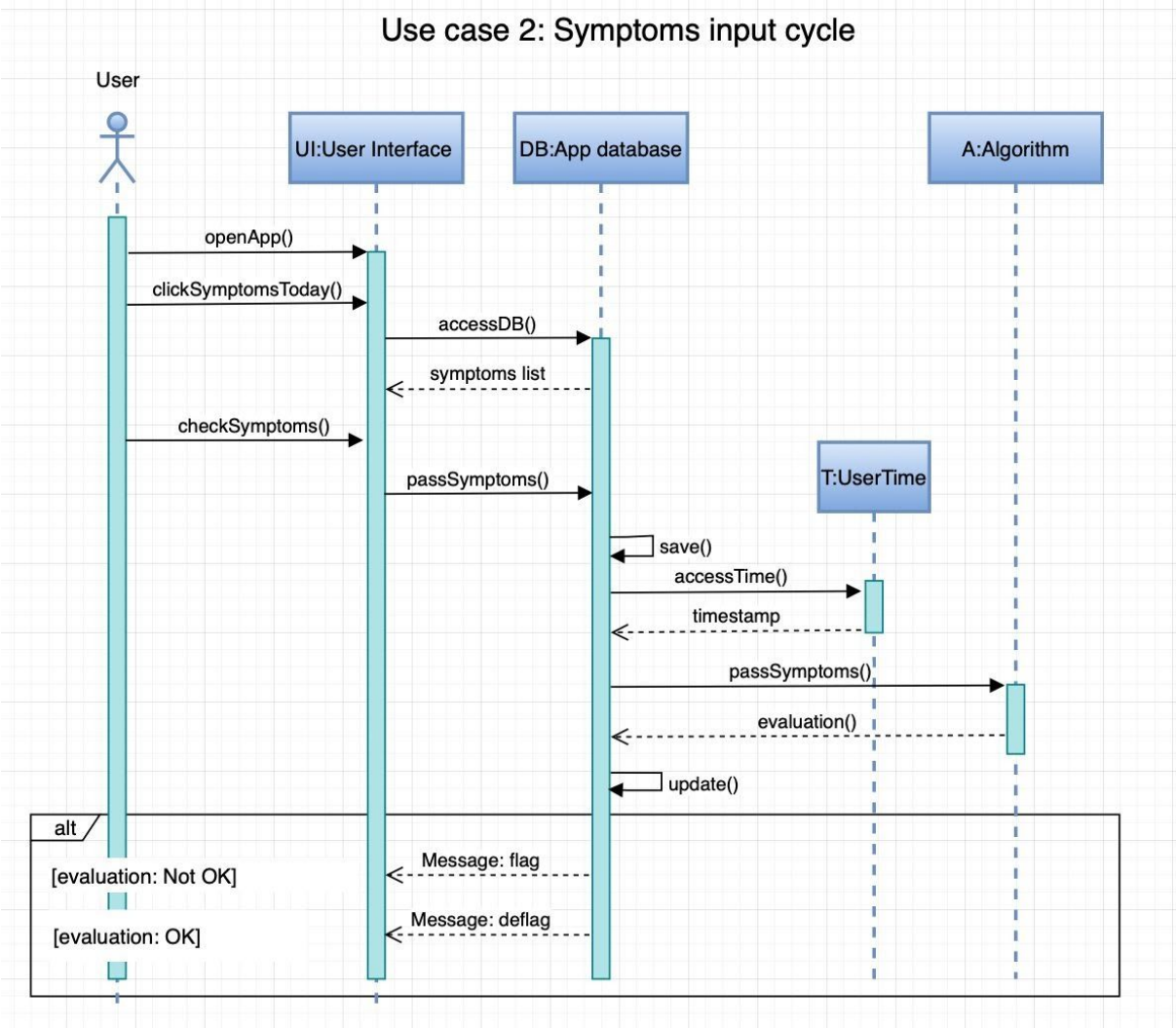
The sequence diagrams we received from PseudoSquad(Group 4)
Authors: Chung-Fan Tsai, Ba Tung Linh Pam, Konrad Ukens, Kim Ngan Le Dang

1.)

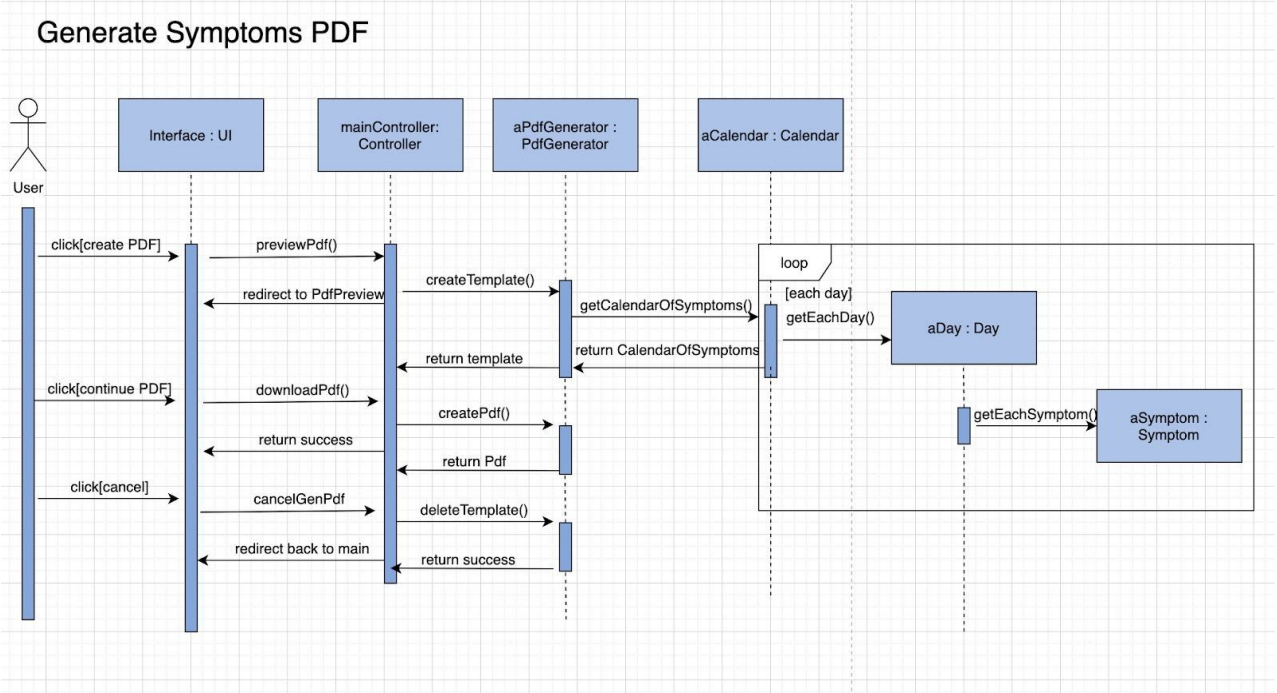
First Startup



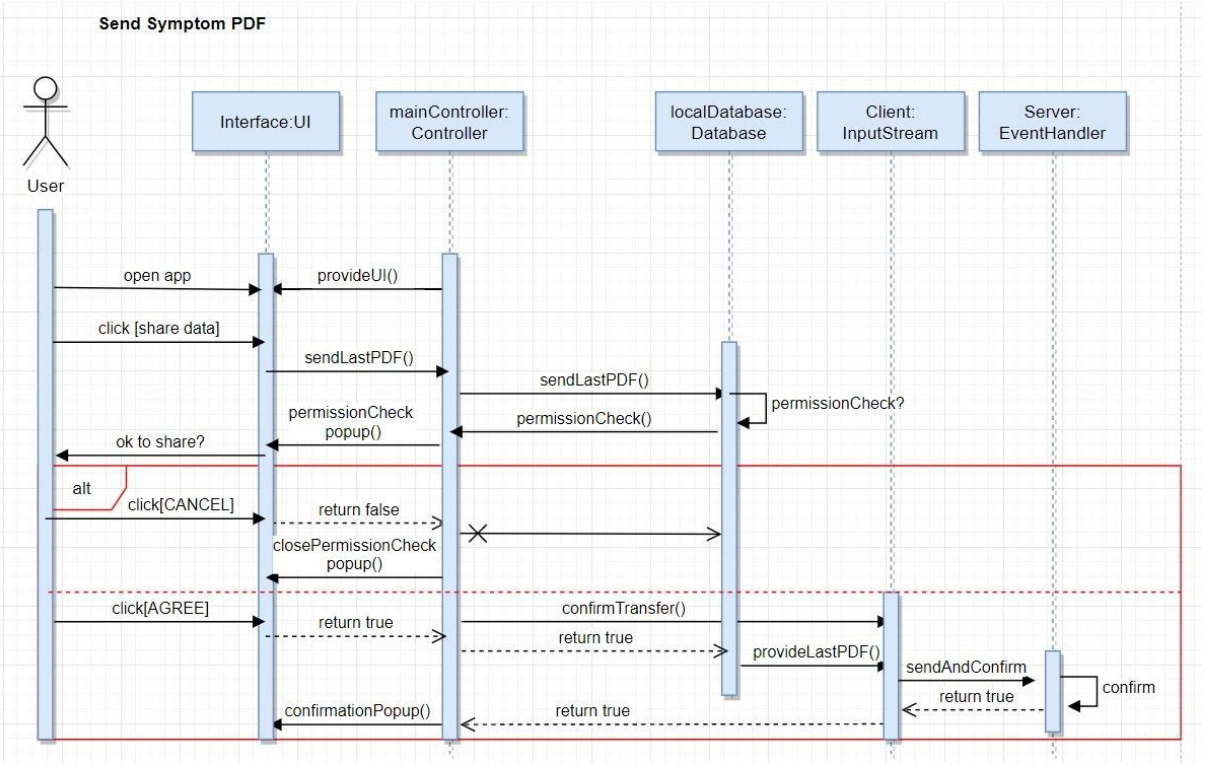
2.)


Use case 2: Symptoms input cycle
User
UI:User Interface
DB:App database
A:Algorithm
T:UserTime
openApp()
clickSymptomsToday()
accessDB()
symptoms list
checkSymptoms()
passSymptoms()
save()
accessTime()
timestamp
passSymptoms()
evaluation()
update()
alt
[evaluation: Not OK]
Message: flag
[evaluation: OK]
Message: deflag

3.)


Generate Symptoms PDF
User
Interface : UI
mainController: Controller
aPdfGenerator : PdfGenerator
aCalendar : Calendar
click[create PDF]
previewPdf()
redirect to PdfPreview
createTemplate()
getCalendarOfSymptoms()
return CalendarOfSymptoms
return template
loop
[each day]
getEachDay()
aDay : Day
getEachSymptom()
aSymptom : Symptom
click[continue PDF]
downloadPdf()
createPdf()
return success
return Pdf
click[cancel]
cancelGenPdf
deleteTemplate()
redirect back to main
return success

4.)


Send Symptom PDF
User
Interface:UI
mainController: Controller
localDatabase: Database
Client: InputStream
Server: EventHandler
open app
provideUI()
click [share data]
sendLastPDF()
sendLastPDF()
permissionCheck?
permissionCheck()
permissionCheck popup()
ok to share?
alt
click[CANCEL]
return false
closePermissionCheck popup()
click[AGREE]
confirmTransfer()
return true
return true
provideLastPDF()
sendAndConfirm
confirm
return true
confirmationPopup()
return true

Task 1:

Read through the changes and the sequence diagrams the other group produced. Do the changes make sense? Can you understand the sequence diagrams? Note this down in your report.

The first thing they point out is that we did not include names for the characters in our scenarios. However, in our sequences of steps in the use case descriptions we did use names instead of user. This is a valid point which we understand although it is not very important.
After that they are saying that our first two scenarios are so similar that we can just treat them as one sequence of steps. Our group also discussed and agreed with this.
They went through our scenarios and tried to add some details. There are some valid points, but on some places our opinions differ and that was how to present the application or database etc. They say we should not treat them as actors. While we do think they are not persons (and can't want something themselves) they are still their own instances in the sequence diagrams. We made sure to research about whether the database should be included in sequence diagrams. We found out that it's sometimes okay to include it, even if it's external to the system.

As we looked through their feedback we noticed for example that they comment under our last use case that a user should manually enter the time of the symptom himself.
We made sure to double-check and saw that we wrote that: "The system automatically sets a timestamp for each reported symptom."

At the end they rewrote almost all our sequences of steps and we do agree that somewhere we wrote more than needed.(e.g. John opens the app → Open App) When we were writing the sequence of steps, we tried to stick to the example we had with the ATM Cash Withdrawal, while also not making it hard to understand and follow the steps.

The first two sequence diagrams looked similar to the way we worked on them and we were able to follow them and understand them well. However, we had some issues understanding the last two. They added some new entities, about which we didn't think about at that point. We were able to clear up any confusion around that and it actually helped us with the class diagram, because now we had some more ideas about the app.

Make a list of all class candidates from the use cases and the sequence diagrams:

1. User
2. User Interface
3. AppController
4. (Client)InputStream
5. PDFGenerator
6. ShareData(a way to interact with the global(company) database)
7. EventHandler
8. Database
9. SymptomsList

We also tried to think about the app in a "Client->Server" way.

Client → Server(Application) → DatabaseClass (SymptomsToday())
→ CreatePdfClass (CreatePDF())
→ Interface
→ GlobalDatabaseClass (Share())

Even though it was challenging for us, we were having a really nice discussion about how we should go and pick the classes. We decided it would probably be a nice idea to have a class Symptoms, because that way we can have attributes like timestamp and methods like getName or getDescription for a certain symptom.

We were not quite sure how to represent the database exactly, becuase it doesn't really have any attributes and it just serves the user to store data.

PDF-Creator should have just one job and only do something like:
For each recorded symptom on each day, add it to the pdf and output it using the example blueprint for tracking symptoms on a day-to-day basis and present it to the user so that it's ready to be downloaded and shared.

Another thing we thought about was that from a privacy point of view, everything is stored on the client's local database. That has a trade-off though, because if the user loses his phone or deletes the app for example, all the data is lost.

While we were discussing, we noticed that we need some sort of identification, in case a user decides to share their data with the company's database. For that reason we decided to include ID,mail and Name for each user. While every user who downloads and opens the app has an ID already, the E-Mail and Name are only asked for when a user decides to share data.

Task 2

Assign the methods to the classes selected

User:

_____

Attributes

-name:string
-id:int
-eMail:string
-previousCondition:boolean
-isFlagged:boolean
-status: HealthStatus

_____

Methods

+getName():string
+getId():int
+getEMail():string
+setName(string name):void
+setID(int id):void
+setEMail(string eMail):void
+hasPreviousCondition():boolean
+setPreviousCondition(boolean previousCondition):void

PDFCreator:

_____

Attributes

_____

-createPDF():file

AppController:

Attributes
-languageID: char

+closeApp()
+openApp()
+setLanguage(char languageID)

ShareData:

Attributes

+createEMail(Object): String
+deleteUser(User): void
+shareWithCompany(Object):void

HealthStatus:

Attributes
-currentTime:date
-healthScore:int
-symptoms[]: Symptom

+addSymptoms(Symptom symptom):void
+deflagUser(User):void
+editSymptoms():void
+evaluateHealthScore():void
+flagUser(User):void
+statistics():void
+symptomsToday():void

Symptom:

Attributes
-intensity:int
-isPreviousCondition:boolean

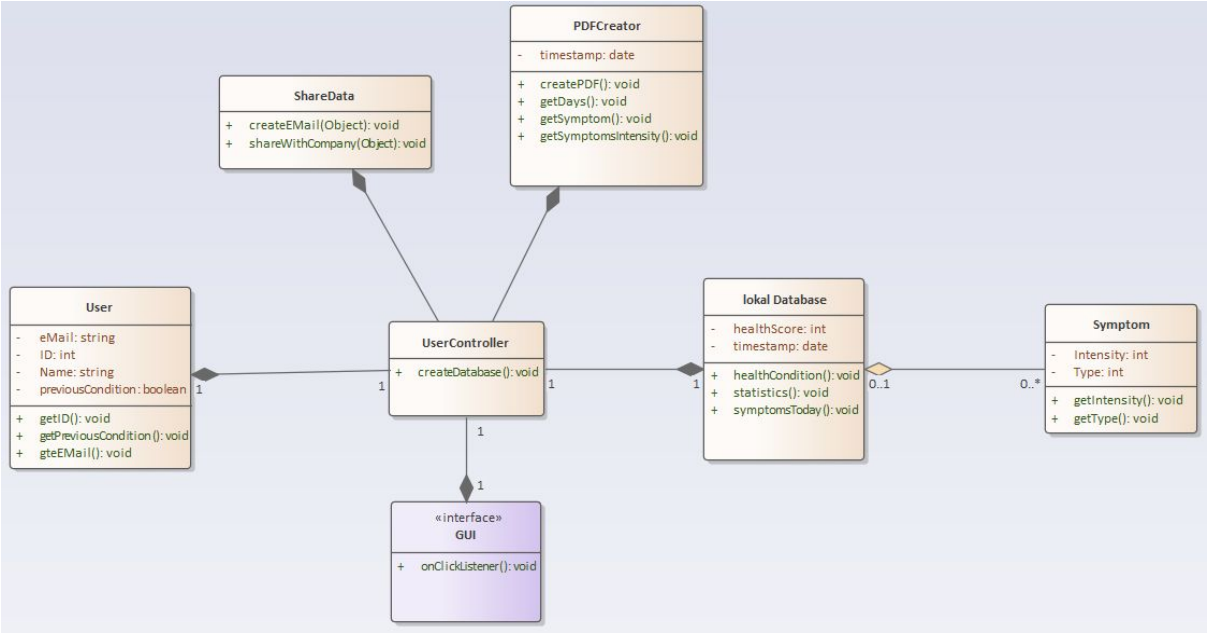
-name:string
-timeStamp:date

_____
+getIntesity():int
+getPreviousCondition():boolean
+getName():string
+getTimeStamp():date
+setIntensity(int intensity):void
+setPreviousCondition(boolean):void
+setName(String name):void
+setTimeStamp(date timestamp):void

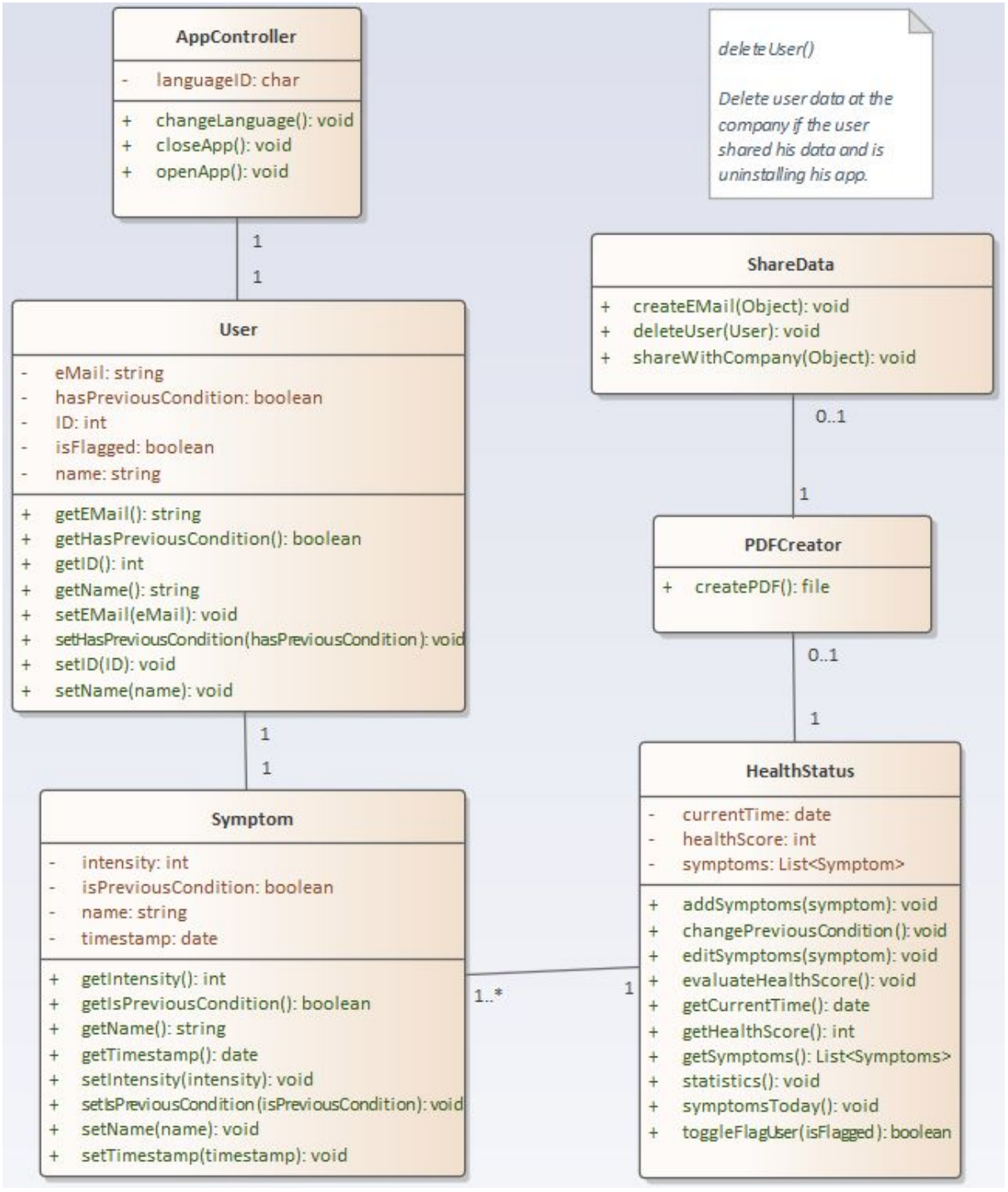
Task 3:

Now draw a class diagram using any UML-Tool. Make sure to include attributes, methods, and associations.

At first we had the UserController connected to and related to all other classes. Someone in our group noted that from a developer point of view this is not a good idea, because if we need to change UserController at some point, then we will need to change ALL other classes as well. Which is something we definitely **don't** want.



We decided to change the Class Diagram and this is the end result.



Reflection:

Niklas: This was really confusing since we basically had to make a blueprint of our application with all the classes and what they contain which is a lot to think about as newbies like we are. We did the work as far as we could understand everything. At some point we thought it was really hard to proceed further because we just did not know what we could do to make our uml diagram more effective/efficient.

Pavel: Making the class diagram was challenging, but it was great that we all could share our opinion about something, either agreeing or disagreeing, then looking at something from a different point of view and finding out that it would be better if we make it this or that way. I learned a lot from the discussions we had and the process we had to go through to draw the class diagram.

Robin:
This exercise was the hardest yet. We not only had to get into the result that the other group achieved for us. Also we had to get into the quite challenging task in creating a Class Diagram since we lack in experience. We are still not quite sure about the relations we used in our Class Diagram. Also the relation that Enterprise Architect provides for Class Diagrams are a bit different from some of Wikipedia. But those from Wikipedia are a bit clearer for us so we photoshopped a bit. Also we were not sure about the general structure and changed it significant in the process. In the end i mainly thought about our Exercise 3 idea and left the Sequence Diagrams a bit left behind.

Nataliia:
I'm happy to be in this group, as always. We really did our best to create an UML diagram. This exercise is challenging for beginners, as writing an application architecture is a difficult but necessary part of the development process. I am not sure that we have been able to bring the UML diagram into line with the sequence diagrams that another group drew for us. But I am sure that our UML reflects the main functions and goals of our application: to allow the user to record symptoms, save symptoms, share saved symptoms. I personally like that our diagram is not too complicated and overloaded.

Time needed:

Task 1	**3h**
Task 2	**5h**
Task 3	**5h**