Team:

Nathan Bellowe
Nathan Mckenna
Yash Parekh

Title: Booze Tracker

Project Summary:

What is the high-level overview of your semester project. What are you trying to accomplish? What will your system do when you are done? This can be mostly copied from Part 1 that you submitted.

Our application serves to track a user's alcohol consumption and estimate levels of intoxication. A user will click a button every time they consume a drink, and be notified of the amount of drinks that they've had, with a cool graphic. The user will also be able to view previous drinking history. This will be useful to allow for proper analysis and planning around drinking habits.

Project Requirements:

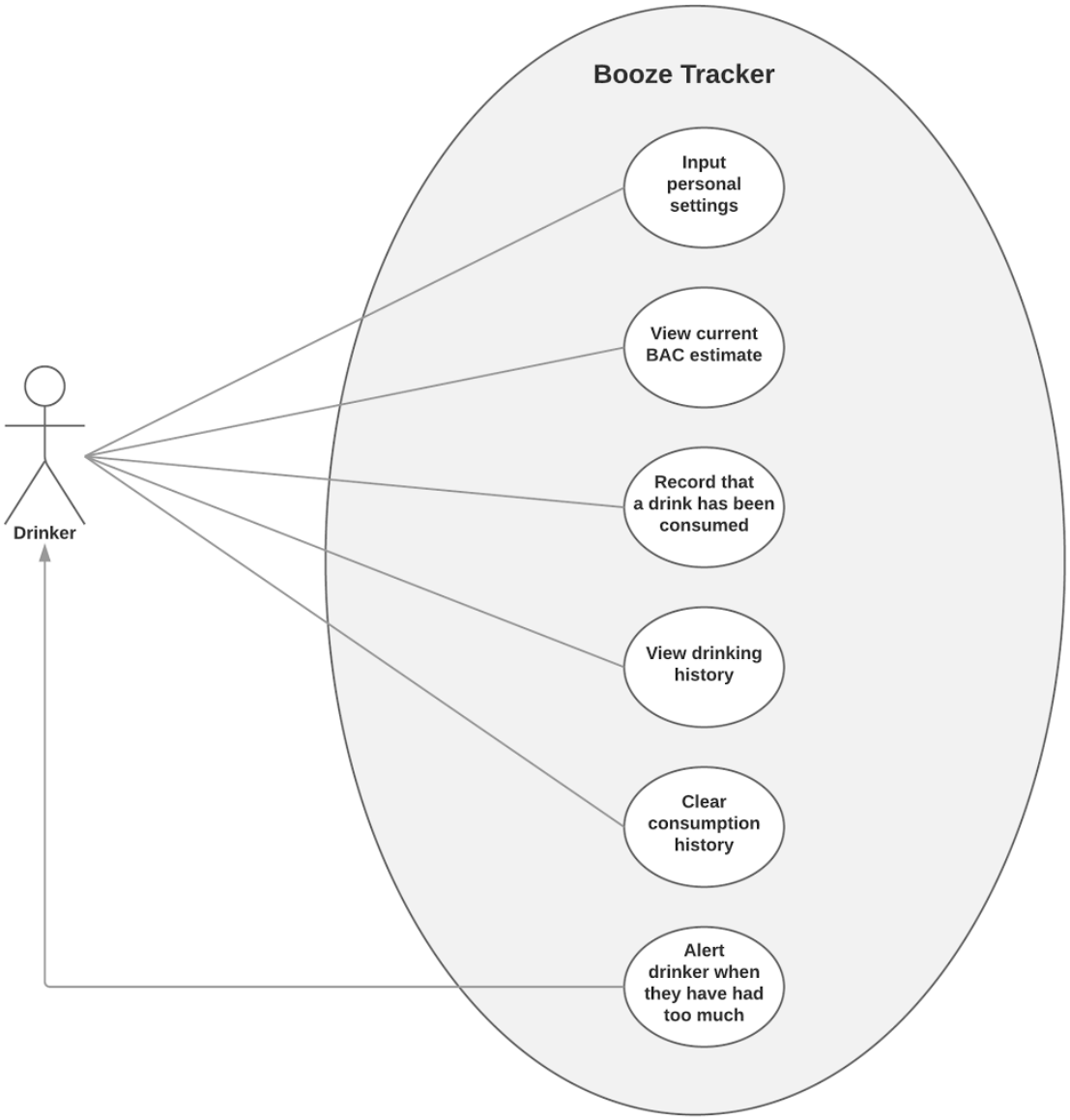
Business Requirements				
ID	Requirement	Topic Area	Actor	Priority
BR-00	The app must prominently display that this holds no legal guarantee of accuracy.			High
BR-01	The app must require users to be at least 21 years old			High

User Requirements			
ID	Requirement	Topic Area	Priority
UR-00	As a user, I want my estimated BAC to be accurately calculated based on my height, weight, and gender.	BAC	High
UR-01	As a user, I want to be able to click the “I had a drink” button so that my stored BAC estimate is updated accordingly.	UI	High

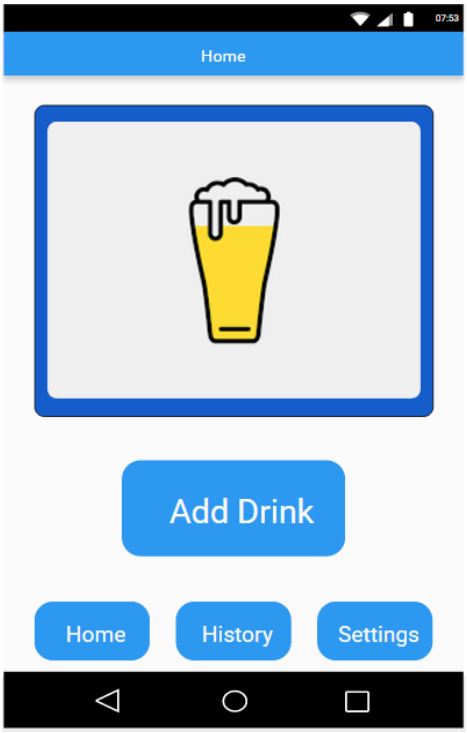
UR-02	As a user, I need to be able to view the number of drinks I've had on a previous night, so I can know how many drinks I had on a given night	UI	Medium
UR-03	As a user, I need to be able to see the estimate of my current BAC both visually and numerically so that I can track intoxication level.	UI	High
UR-04	As a user, I need to be able to enter information about my size and gender to better approximate BAC.	BAC	High
UR-05	As a user, I want to be able to view my drinking history through a graph depicting number of drinks vs. days.	UI	Low
UR-06	As a user I expect to be notified when I have had too much alcohol	Notification	Low
UR-07	As a user, I want to be able to clear the history of how many drinks I've had, so I can remove drinking history	History	Low

Non-Functional Requirements			
ID	Requirement	Topic Area	Priority
NF-00	Performance: Upon having a drink, it should be recorded in the database in order to use persist the data.	DB	High
NF-01	Works on different platforms the same. (IOS/Android)	Compatibility	High
NF-02	The user interface must be understandable, and simple to use.	UI	Medium
NF-03	The application must handle, catch, and hide, exceptions without compromising the user's experience.	UI	Medium
NF-04	The drink filling animation must be smooth.	UI	Low

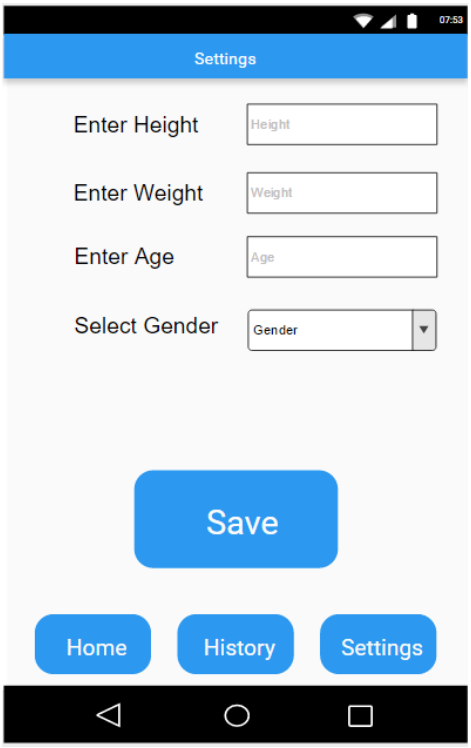
Use Case Diagram:



UI Mockups:


07:53
Home
Add Drink
Home
History
Settings


07:53
History
View last 30 days
Home
History
Settings

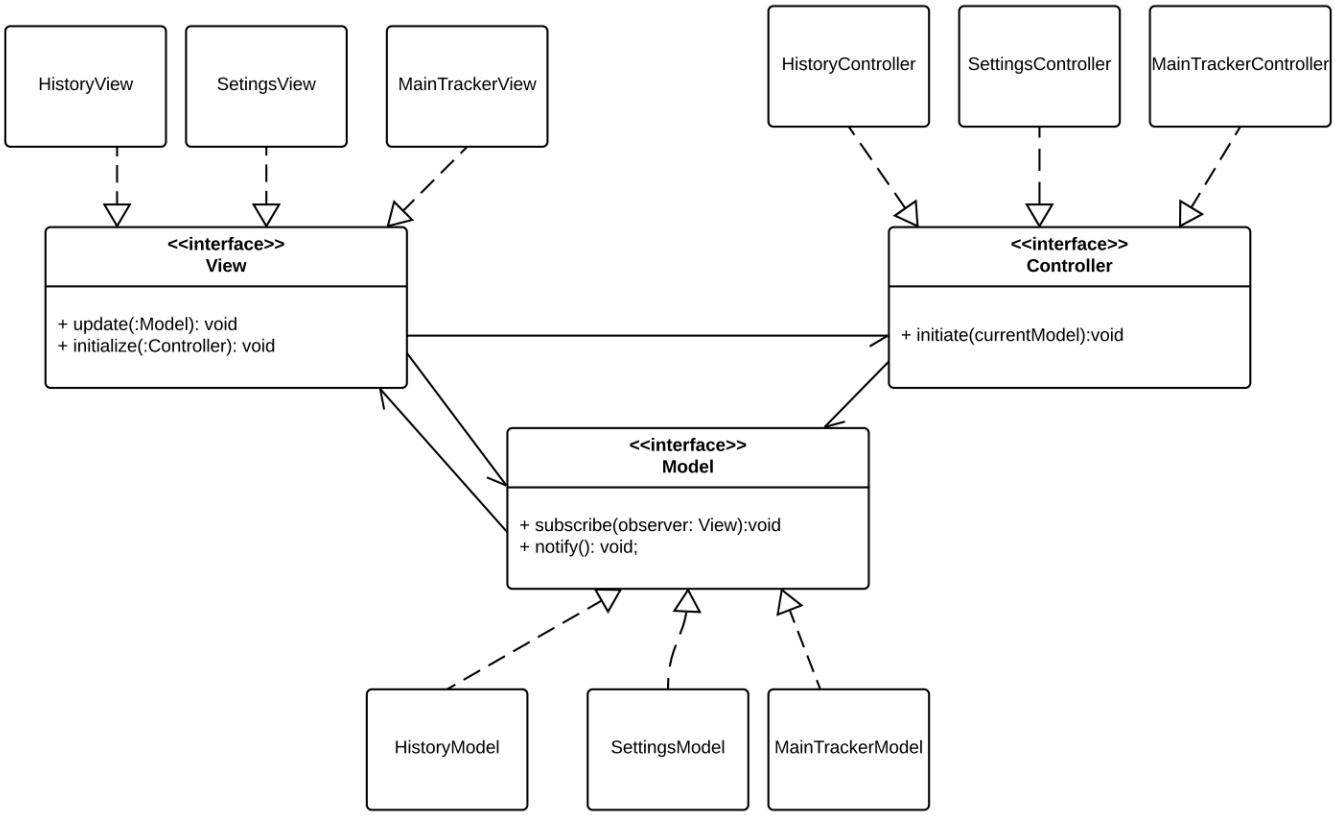

07:53
Settings
Enter Height
Height
Enter Weight
Weight
Enter Age
Age
Select Gender
Gender
Save
Home
History
Settings

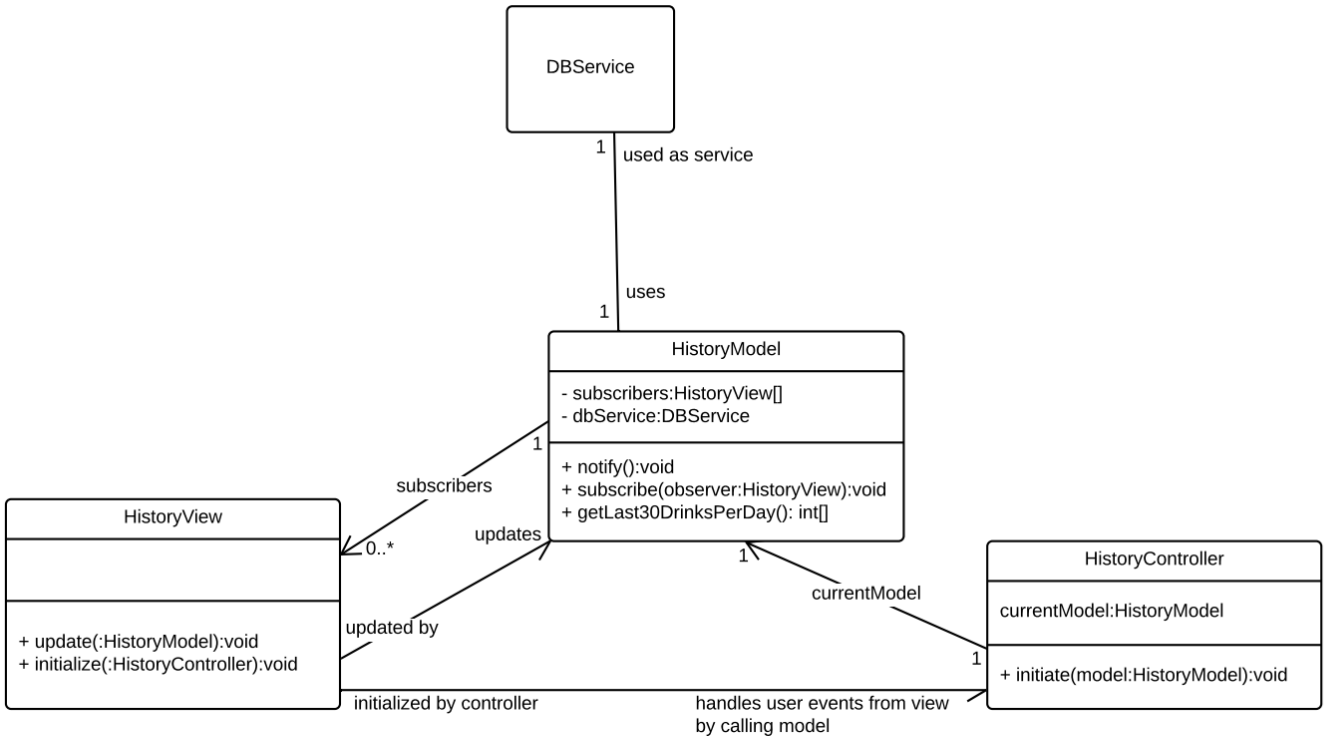
Data Storage:

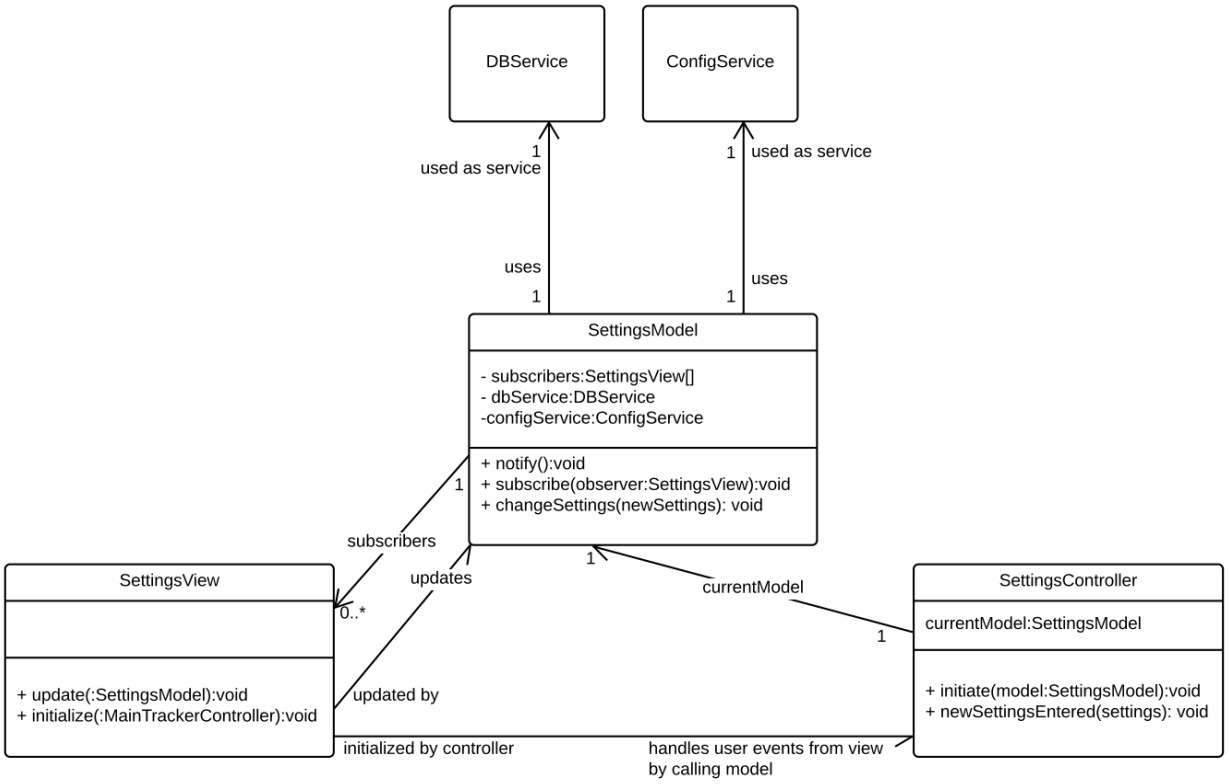
We will use a SQL database, with two tables. One table will be used to persist the current characteristics of the user, needed for BAC calculation, such as gender, age, and weight. The second database will persist data about previous drinks for future calculations, such as calculating how many drinks were recorded per day. In order to communicate with the database, we will exclusively use a database connector service, which will allow for easily changing databases in the future.

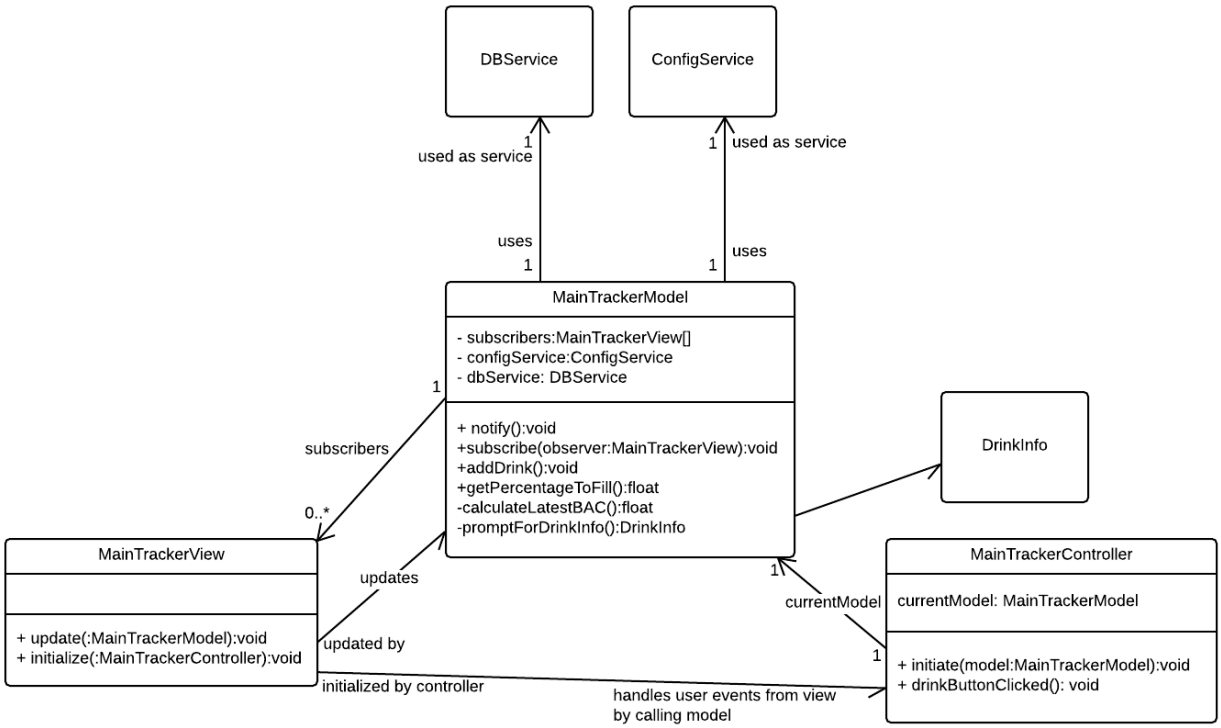
Class Diagram

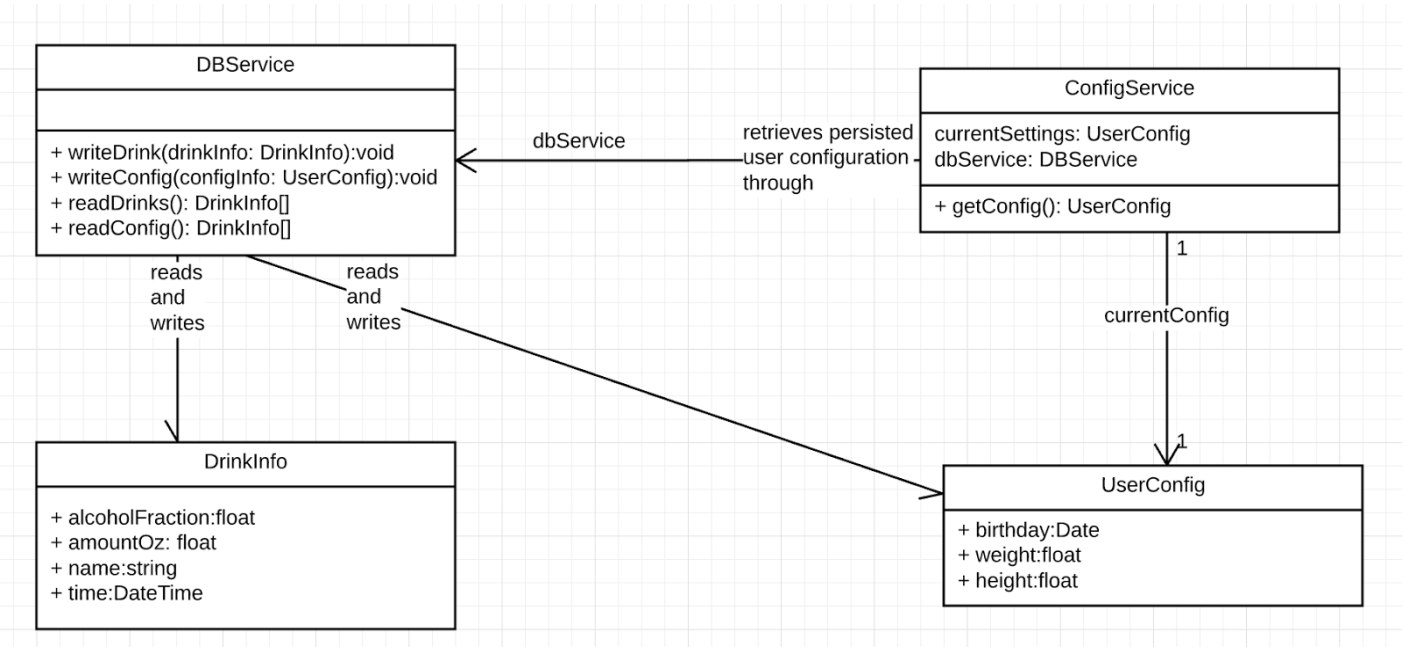
Our program will utilize three instances of the MVC pattern, one for each screen in the program. Note that we use interfaces for the Model, View and Controller, rather than abstract classes. We also use two services that the models can utilize for communicating with the database, and storing and loading the current user configuration.




DBService
1
used as service
1
uses
HistoryModel
- subscribers:HistoryView[]
- dbService:DBService
+ notify():void
+ subscribe(observer:HistoryView):void
+ getLast30DrinksPerDay(): int[]
1
subscribers
0..*
updates
HistoryView
+ update(:HistoryModel):void
+ initialize(:HistoryController):void
updated by
1
currentModel
HistoryController
currentModel:HistoryModel
+ initiate(model:HistoryModel):void
1
initialized by controller
handles user events from view
by calling model


DBService
ConfigService
1
used as service
1
used as service
uses
1
1
uses
SettingsModel
- subscribers:SettingsView[]
- dbService:DBService
-configService:ConfigService
+ notify():void
+ subscribe(observer:SettingsView):void
+ changeSettings(newSettings): void
1
subscribers
0..*
updates
1
currentModel
SettingsView
+ update(:SettingsModel):void
+ initialize(:MainTrackerController):void
updated by
1
SettingsController
currentModel:SettingsModel
+ initiate(model:SettingsModel):void
+ newSettingsEntered(settings): void
initialized by controller
handles user events from view
by calling model


DBService
ConfigService
1
used as service
1
used as service
uses
1
1
uses
MainTrackerModel
- subscribers:MainTrackerView[]
- configService:ConfigService
- dbService: DBService
+ notify():void
+subscribe(observer:MainTrackerView):void
+addDrink():void
+getPercentageToFill():float
-calculateLatestBAC():float
-promptForDrinkInfo():DrinkInfo
1
subscribers
0..*
DrinkInfo
MainTrackerView
+ update(:MainTrackerModel):void
+ initialize(:MainTrackerController):void
updates
updated by
initialized by controller
1
currentModel
1
MainTrackerController
currentModel: MainTrackerModel
+ initiate(model:MainTrackerModel):void
+ drinkButtonClicked(): void
handles user events from view
by calling model


DBService
+ writeDrink(drinkInfo: DrinkInfo):void
+ writeConfig(configInfo: UserConfig):void
+ readDrinks(): DrinkInfo[]
+ readConfig(): DrinkInfo[]
dbService
retrieves persisted
user configuration
through
ConfigService
currentSettings: UserConfig
dbService: DBService
+ getConfig(): UserConfig
1
currentConfig
1
reads
and
writes
reads
and
writes
DrinkInfo
+ alcoholFraction:float
+ amountOz: float
+ name:string
+ time:DateTime
UserConfig
+ birthday:Date
+ weight:float
+ height:float