Лабораторные работы по курсу «Технологии программирования, алгоритмы и структуры данных»

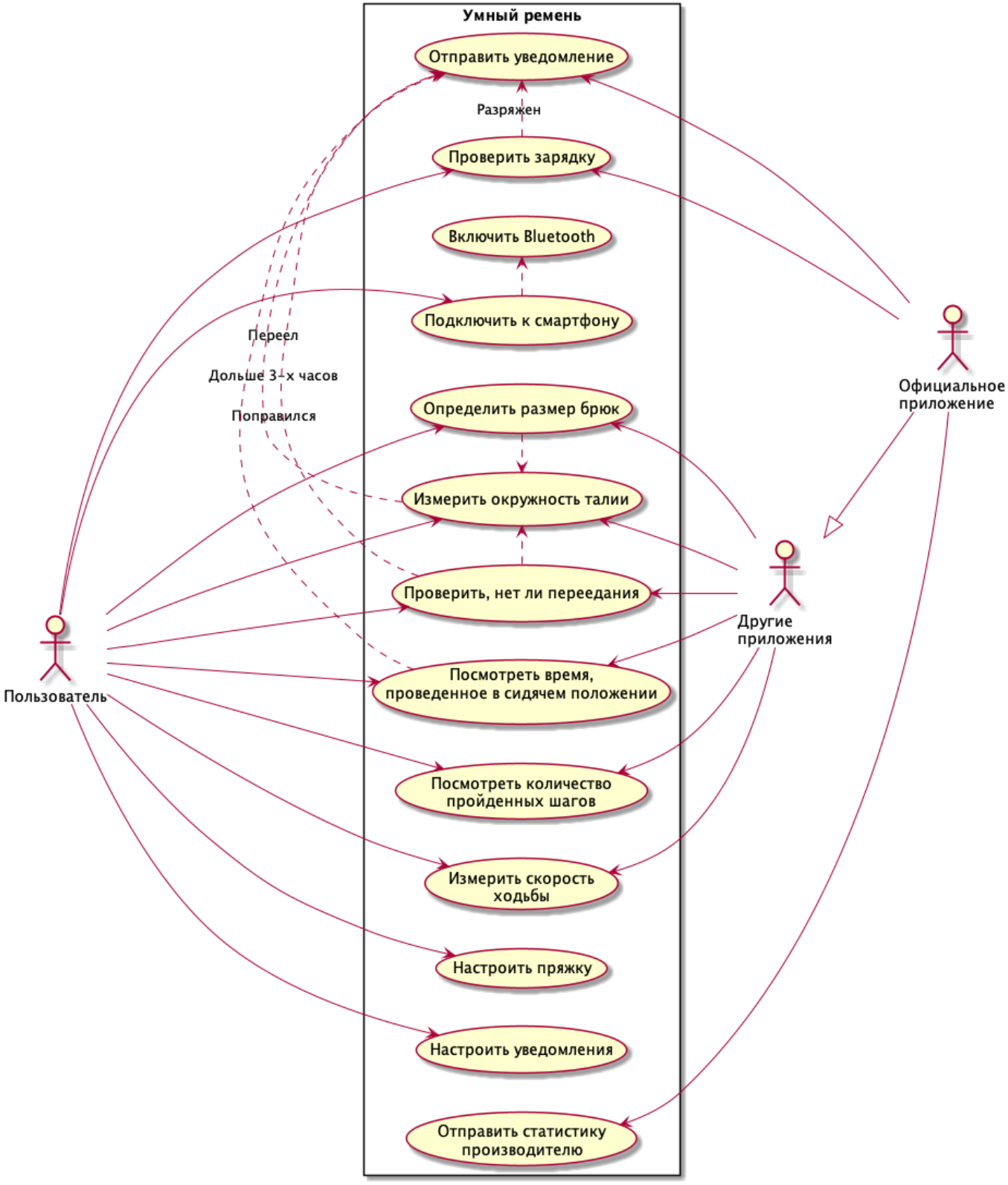
Проектирование системы работы устройства «Умный ремень» на основе UML-диаграмм

Студента группы Б18-505
Козловской Анны Андреевны

Преподаватели
Куприяшин М. А.
Борзунов Г. И.

НИЯУ-МИФИ
Москва 2019

Диаграмма вариантов использования (Use-case Diagram)

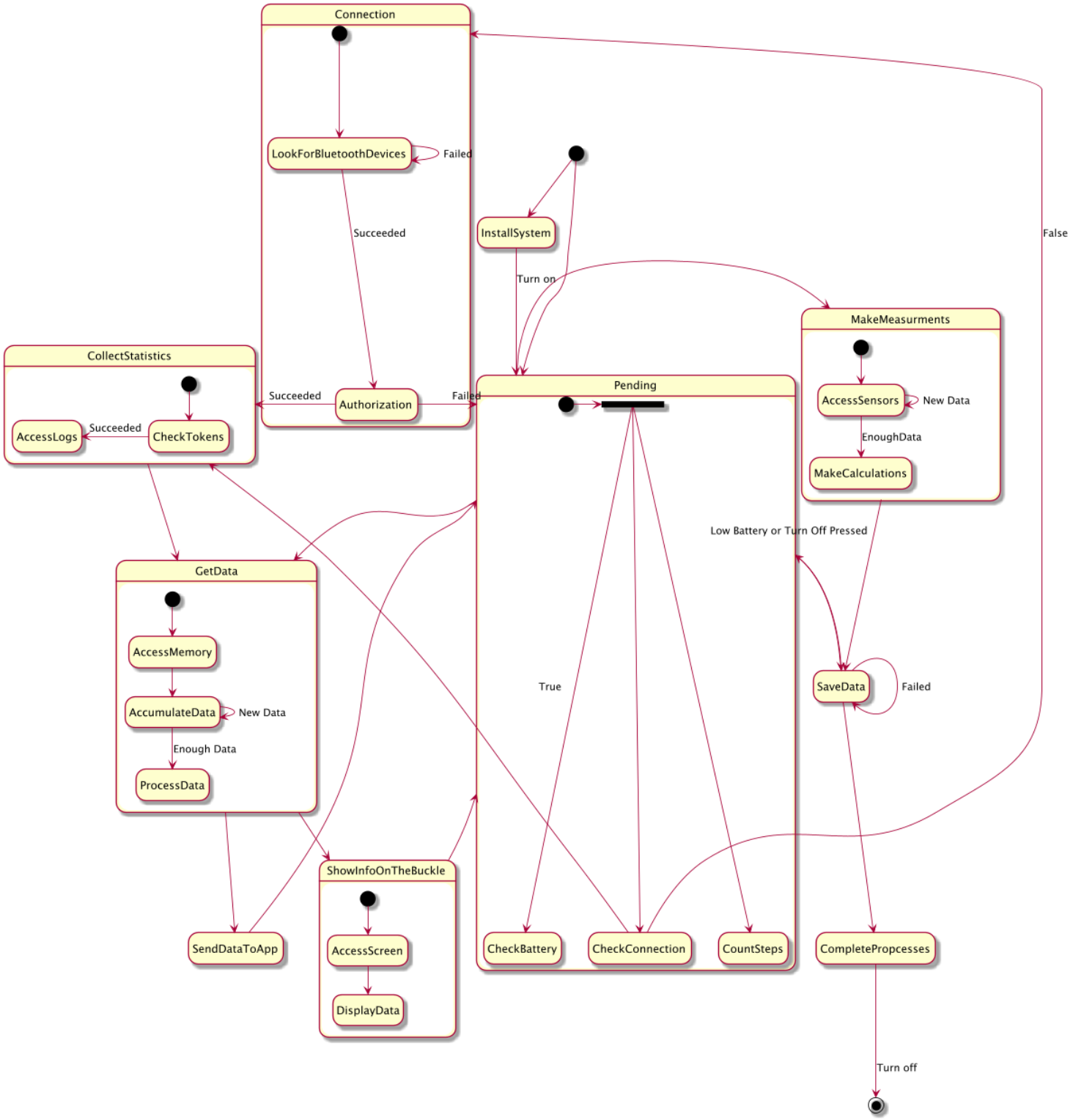


Plantuml :

```
@startuml

left to right direction

:Пользователь: as User
:Официальное\nприложение: as App
:Другие\nприложения: as Apps

Apps <|-- App

rectangle "Умный ремень"  {
    (Посмотреть количество\n пройденных шагов) as (ShowSteps)
    (Измерить скорость\n ходьбы) as (ShowSpeed)
    (Посмотреть время,\nпроведенное в сидячем положении) as (ShowSeat)
    (Измерить окружность талии) as (MeasureWaist)
    (Проверить, нет ли переедания) as (CheckOvereating)
    (Определить размер брюк) as (MeasureSize)
    (Подключить к смартфону) as (ConnectToPhone)
    (Настроить пряжку) as (SetBuckle)
    (Настроить уведомления) as (SetPush)
    (Включить Bluetooth) as (Bluetooth)

    (Отправить уведомление) as (SendPush)
    (Отправить статистику\nпроизводителю) as (SendStat)
    (Проверить зарядку) as (CheckCharge)

    (CheckCharge) .> (SendPush) : Разряжен
    (ConnectToPhone) .> (Bluetooth)
    (MeasureSize) .> (MeasureWaist)
    (CheckOvereating) .> (MeasureWaist)
    (MeasureWaist) .> (SendPush) : Поправился\n
    (CheckOvereating) .> (SendPush) : Переел\n
    (ShowSeat) .> (SendPush) : Дольше 3-х часов\n
}

User ---> (CheckCharge)
User ---> (SetBuckle)
User ---> (SetPush)
User ---> (ShowSpeed)
User ---> (MeasureSize)
User ---> (ShowSteps)
User ---> (ShowSeat)
User ---> (MeasureWaist)
User ---> (CheckOvereating)
User ---> (ConnectToPhone)

(SendStat) <-- App
(CheckCharge) <-- App
(SendPush) <-- App
(MeasureSize) <-- Apps
(MeasureWaist) <-- Apps
(CheckOvereating) <-- Apps
(ShowSpeed) <-- Apps
(ShowSteps) <-- Apps
(ShowSeat) <-- Apps

@enduml
```

Диаграмма описывает базовую функциональность будущей системы: имеется три актора, один из которых является расширением другого. Это непосредственный пользователь, приложение для смартфона и расширяющее его права официальное приложение от производителя системы. В целом, диаграмма не предполагает излишних деталей работы системы, а потому максимальна проста для восприятия. Стоит однако отметить особенности некоторых use-кейсов. Например, проверку зарядки отправку статистики производителю и отправку ведомлений, согласно схеме, по умолчанию может осуществлять только *официальное* приложение, причем отправка уведомлений происходить автоматически при соблюдении определенных условий (например, если пользователь был неподвижен в течение трех и более часов). Кроме того, пользователь не имеет доступа к включению/выключению bluetooth в устройстве, потому как он нужен исключительно для подключения к смартфону. Работу bluetooth координирует само устройство умного ремня, а пользователь может принимать или отклонять возможные подключения.

Диаграмма состояний (State Diagram)

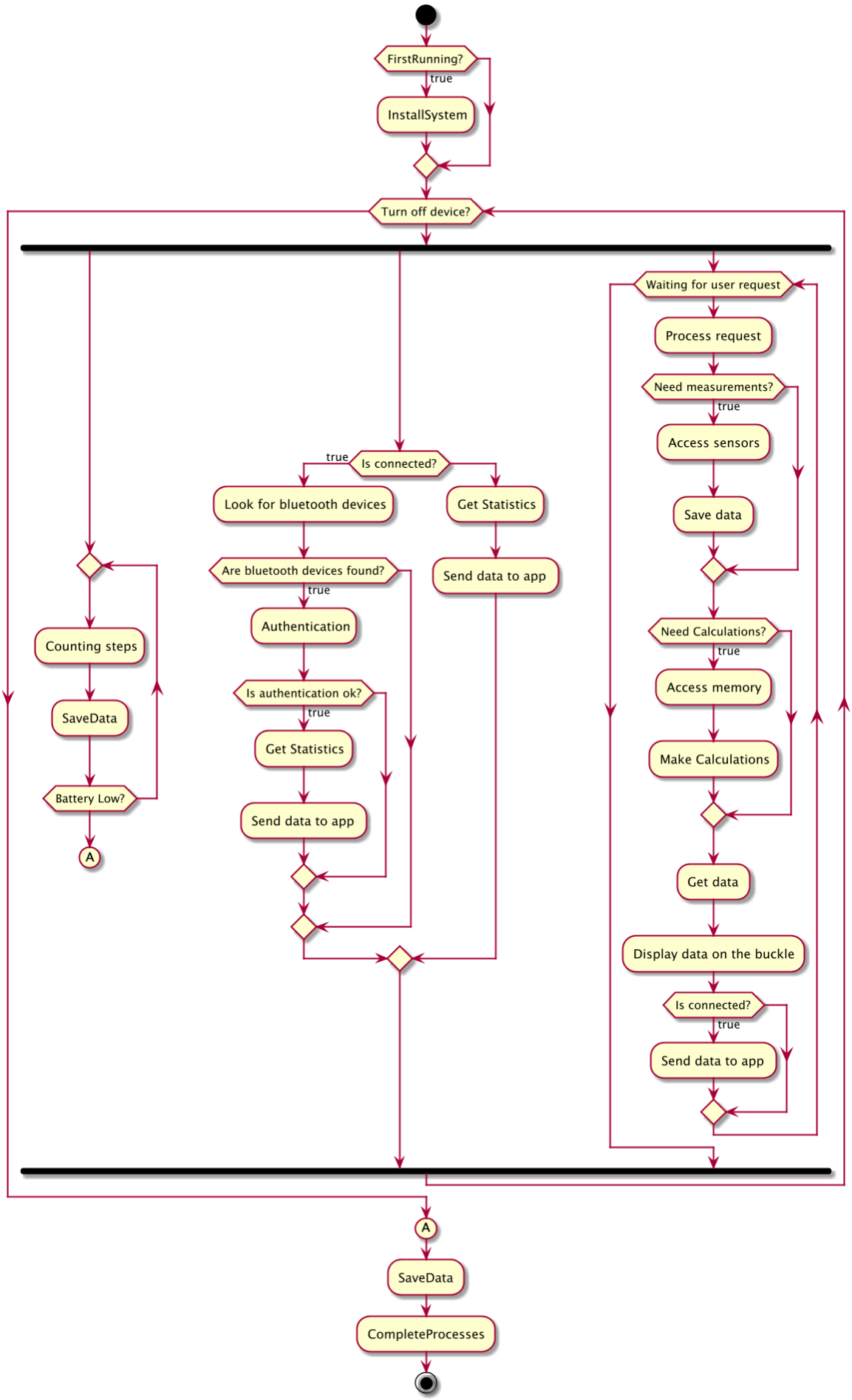


Plantuml:

```
@startuml

top to bottom direction
hide empty description
scale 1200 width

state SaveData
state Pending {
    state f <<fork>>
    [*] -> f
    f --> CheckBattery
    f --> CheckConnection
    f --> CountSteps

    CheckConnection --> CollectStatistics : True
    CheckConnection --> Connection : False
}
state GetData {
    [*] --> AccessMemory
    AccessMemory --> AccumulateData
    AccumulateData --> AccumulateData : New Data
    AccumulateData --> ProcessData : Enough Data
}
state ShowInfoOnTheBuckle {
    [*] --> AccessScreen
    AccessScreen --> DisplayData
}
state CollectStatistics {
    [*] --> CheckTokens
    CheckTokens -left-> AccessLogs : Succeeded
}
state Connection {
    [*] --> LookForBluetoothDevices
    LookForBluetoothDevices --> Authorization : Succeeded
    LookForBluetoothDevices --> LookForBluetoothDevices : Failed
    Authorization -right--> CollectStatistics : Succeeded
    Authorization -right--> Pending : Failed
}
state MakeMeasurments {
    [*] --> AccessSensors
    AccessSensors --> AccessSensors : New Data
    AccessSensors --> MakeCalculations : EnoughData
}
MakeMeasurments --> SaveData
SaveData --> SaveData : Failed
SaveData --> Pending
Pending --> MakeMeasurments
Pending --> GetData
Pending --> SaveData : Low Battery or Turn Off Pressed
GetData --> ShowInfoOnTheBuckle
ShowInfoOnTheBuckle --> Pending
CollectStatistics --> GetData
GetData --> SendDataToApp
SendDataToApp --> Pending
SaveData --> CompletePropcesses

[*] --> InstallSystem
[*] --> Pending
InstallSystem --> Pending : Turn on
CompletePropcesses --> [*] : Turn off

@enduml
```

Данная диаграмма отражает состояния системы и переход из одного состояния в другое без подробного описания событий, его спровоцировавших. Итак, после включения устройство попадает в состояние ожидания, в котором параллельно происходит подсчет шагов, проверка заряда аккумулятора и проверка наличия возможных подключений. При первом включении устройство попадает в состояние установки системы. Переход из одного состояния в другое кольцевой, то есть, попадая из режима ожидания в какое-либо состояние, по выходу из него устройство в большинстве случаев возвращается в режим ожидания. Отсюда можно попасть в состояние совершения измерений, осуществляемых с помощью сенсоров на корпусе и встроенной вычислительной системы, а также сделать запрос к внутренней памяти устройства за уже имеющимися данными, чтобы затем отобразить их на пряжке ремня и отправить доступным приложениям. Если устройство подключено к смартфону, оно отправляет статистику в официальное приложение, если нет - пытается осуществить подключение. И, наконец, если аккумулятор разряжен, устройство сохраняет данные, завершает все текущие процессы и переходит в конечное состояние «выключено».

Диаграмма активности (Activity Diagram)

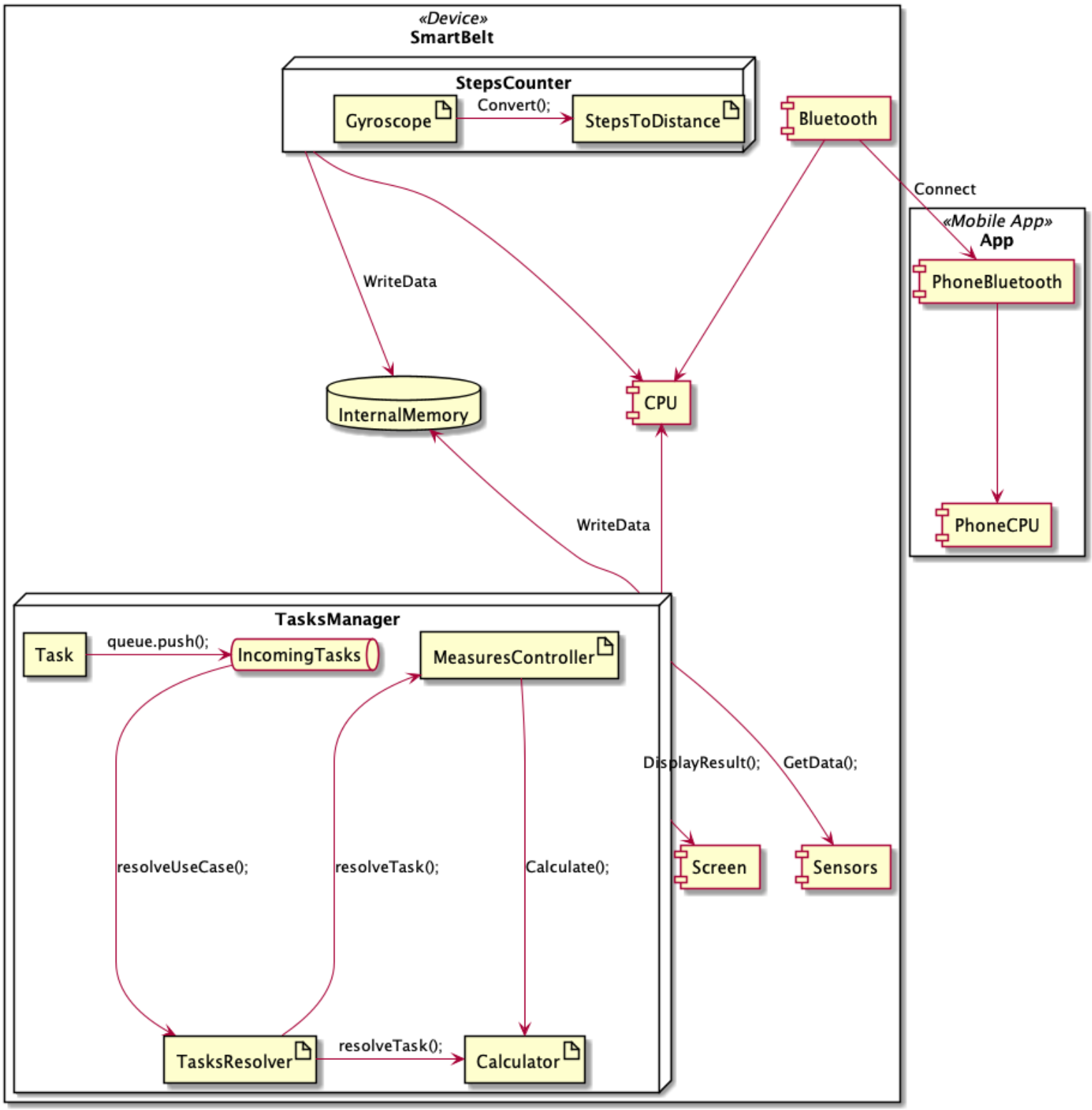


Plantuml:

```
@startuml

scale 1200 width

start
if (FirstRunning?) then (true)
    :InstallSystem;
endif
while (Turn off device?)
fork
    repeat
        :Counting steps;
        :SaveData;
    repeat while (Battery Low?)
    (A)
    detach
    fork again
        if (Is connected?) then (true)
        :Look for bluetooth devices;
        if (Are bluetooth devices found?) then (true)
            :Authentication;
            if (Is authentication ok?) then (true)
                :Get Statistics;
                :Send data to app;
            endif
        endif
    else
        :Get Statistics;
        :Send data to app;
    endif
    fork again
        while (Waiting for user request)
            :Process request;
            if (Need measurements?) then (true)
                :Access sensors;
                :Save data;
            endif
            if (Need Calculations?) then (true)
                :Access memory;
                :Make Calculations;
            endif
            :Get data;
            :Display data on the buckle;
            if (Is connected?) then (true)
                :Send data to app;
            endif
        endwhile
end fork
endwhile

(A)
:SaveData;
:CompleteProcesses;
stop

@enduml
```

Диаграмма активности отражает деятельность устройства и связанный с ними набор действий (состояния которых описаны на диаграмме состояний), а также их координированное выполнение. При первом запуске устройство устанавливает систему, далее попадает в рабочий цикл, завершающийся только при выключении устройства или низком заряде аккумулятора. Параллельно осуществляются несколько действий: зацикленный подсчет шагов и проверка аккумулятора, подключение к смартфону и цикл ожидания запросов от пользователя. Последний включает в себя последовательную проверку на необходимость совершения измерений, необходимость вычислений и, соответственно, их выполнение в случае положительной ветки, получение данных, их отображение и отправка приложениям, если устройство подключено. При выходе из основного рабочего цикла, все данные сохраняются, процессы завершаются, и устройство выключается.

Диаграмма развертывания (Deployment Diagram)

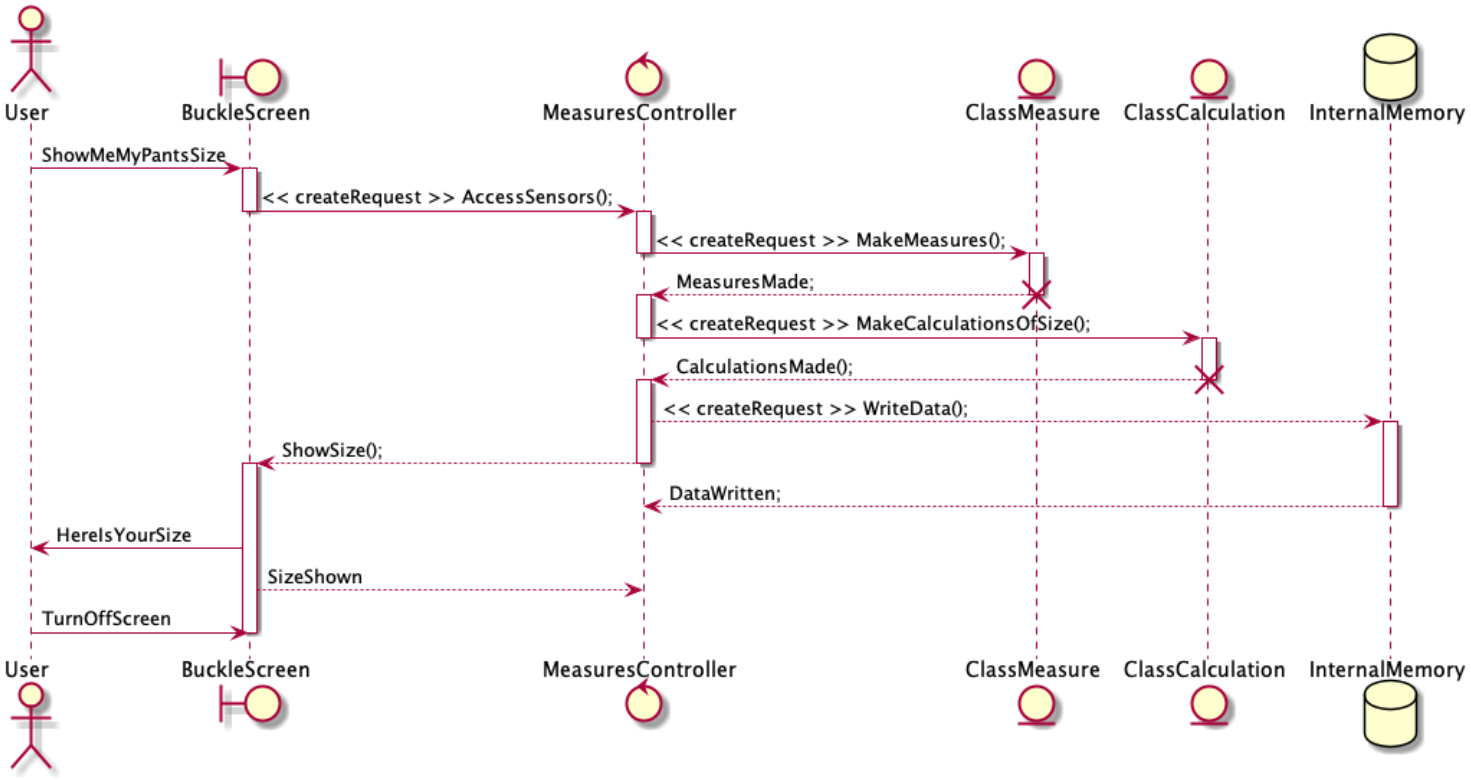


Plantuml:

```
@startuml
top to bottom direction

rectangle App <<Mobile App>> {
    Component PhoneCPU
    Component PhoneBluetooth
    PhoneBluetooth -down-> PhoneCPU
}

rectangle SmartBelt <<Device>>  {
    database InternalMemory

    node TasksManager {
        rectangle Task
        queue IncomingTasks
        artifact TasksResolver
        artifact Calculator
        artifact MeasuresController
        Task -> IncomingTasks: queue.push();
        IncomingTasks -down-> TasksResolver: resolveUseCase();
        TasksResolver -up-> MeasuresController: resolveTask();
        MeasuresController -right-> Calculator : Calculate();
        TasksResolver -> Calculator: resolveTask();
    }

    Component CPU
    Component Screen
    Component Sensors
    TasksManager --> Screen: DisplayResult();
    TasksManager --> Sensors: GetData();
    TasksManager -up-> CPU

    node StepsCounter {
        artifact Gyroscope
        artifact StepsToDistance
        Gyroscope -> StepsToDistance: Convert();
    }

    Component Bluetooth
    TasksManager -up-> InternalMemory: WriteData
    StepsCounter --> InternalMemory: WriteData
    StepsCounter -down-> CPU
    Bluetooth --> CPU
    Bluetooth --> PhoneBluetooth: Connect
}

@enduml
```

Диаграмма развертывания описывает аппаратные компоненты работы системы и их координированную работу. Устройство непосредственно ремня включает в себя такие узлы как счетчик шагов и менеджер задач, компоненты: экран на пряжке, сенсоры, Bluetooth, CPU и внутреннюю память. Смартфон не является частью нашей системы, а потому представлен упрощенно - из частей, взаимодействующих с подключенным ремнем, в виде своего Bluetooth и своего CPU. Счетчик шагов состоит из гироскопа и программы, рассчитывающей пройденную дистанцию. Менеджер задач включает в себя очередь задач, резолвер, калькулятор и контроллер измерений. Входящая задача попадает в очередь, вынимается из очереди, затем резолвер определяет последовательность действий для ее решения, и в зависимости от этого обращается к контроллеру измерений или калькулятору. Сам узел может обращаться к экрану пряжки для отображения результата решенной задачи, к сенсорам для получения измерений, внутренней памяти для записи данных и посылать инструкции микропроцессору.

Диаграмма обмена сообщениями (Sequence Diagram)

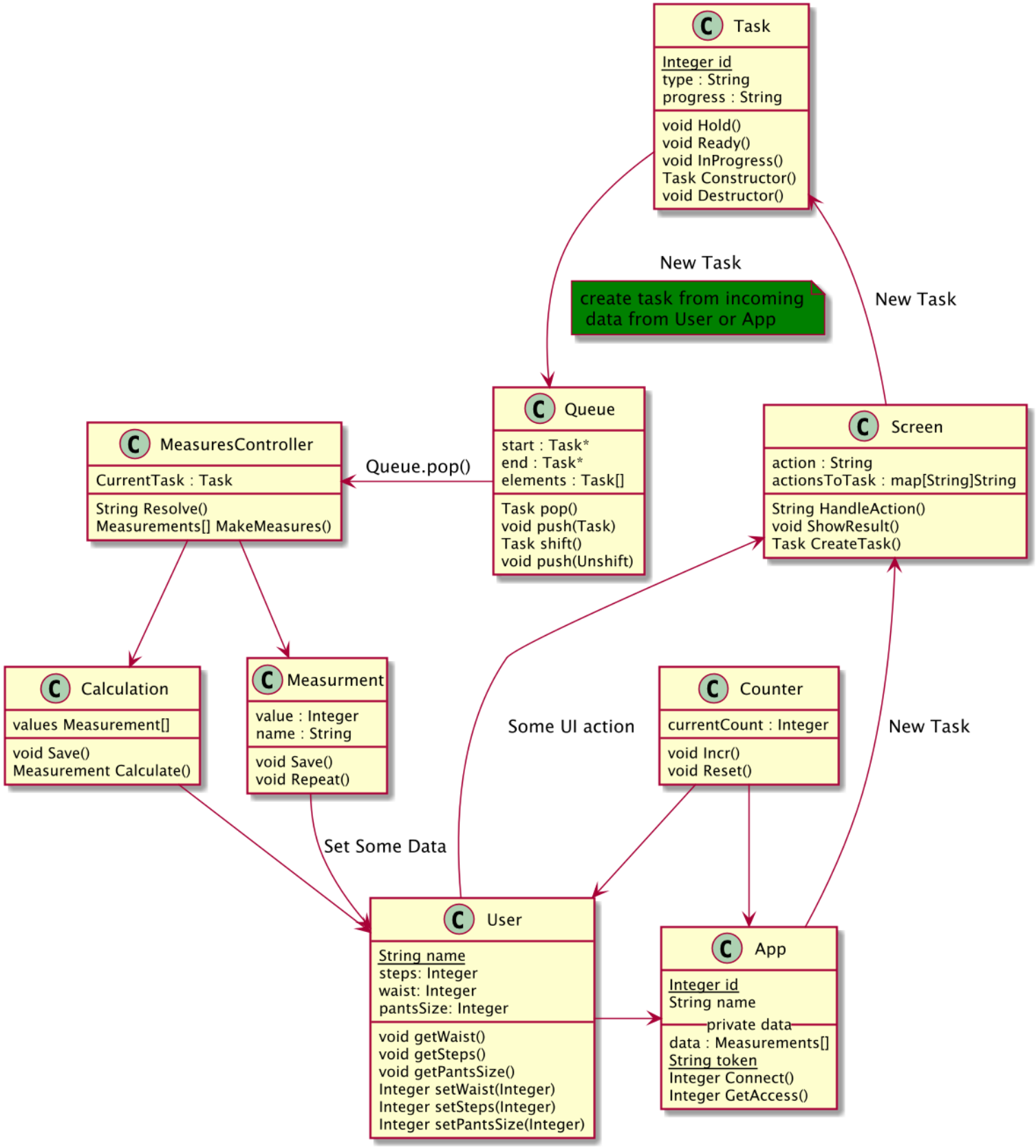


Plantuml:

```
@startuml

actor User
boundary BuckleScreen
control MeasuresController
entity ClassMeasure
entity ClassCalculation
database InternalMemory

User -> BuckleScreen: ShowMeMyPantsSize

activate BuckleScreen
BuckleScreen -> MeasuresController: << createRequest >> AccessSensors();
deactivate BuckleScreen

activate MeasuresController
MeasuresController -> ClassMeasure: << createRequest >> MakeMeasures();
deactivate MeasuresController

activate ClassMeasure
ClassMeasure --> MeasuresController: MeasuresMade;
destroy ClassMeasure

activate MeasuresController
MeasuresController -> ClassCalculation: << createRequest >>
MakeCalculationsOfSize();
deactivate MeasuresController
activate ClassCalculation

ClassCalculation --> MeasuresController: CalculationsMade();
destroy ClassCalculation

activate MeasuresController
MeasuresController --> InternalMemory: << createRequest >> WriteData();
activate InternalMemory
MeasuresController --> BuckleScreen: ShowSize();
deactivate MeasuresController
activate BuckleScreen
InternalMemory --> MeasuresController: DataWritten;
deactivate InternalMemory

BuckleScreen -> User: HereIsYourSize
BuckleScreen --> MeasuresController: SizeShown
User -> BuckleScreen: TurnOffScreen
deactivate BuckleScreen
deactivate BuckleScreen

@enduml
```

Данная диаграмма обмена сообщениями иллюстрирует жизненный цикл программных сущностей системы в рамках use-кейса «Определение размера брюк». Пользователь заявляет о своем желании узнать размер брюк с помощью пользовательского интерфейса экрана пряжки, который, в свою очередь, создает и отправляет запрос к контроллеру измерений. Создается объект класса Measure, осуществляющий измерения талии пользователя, который уничтожается после асинхронной отправки ответа контроллеру измерений. Далее последний создает объект класса Calculation, осуществляющий вычисления размера брюк, который уничтожается после асинхронной отправки ответа контроллеру измерений. Контроллер измерений делает асинхронные запросы на запись данных во внутреннюю память и запрос на отображение ответа на экране пряжки (асинхронный потому, что экран может быть занят отображением иной информации). Экран отображает запрошенный размер брюк, асинхронно отправляет ответ контроллеру измерений о том, что пользователь получил данные, пользователь читает и выключает экран.

Диаграмма классов (Static Structure Diagram)

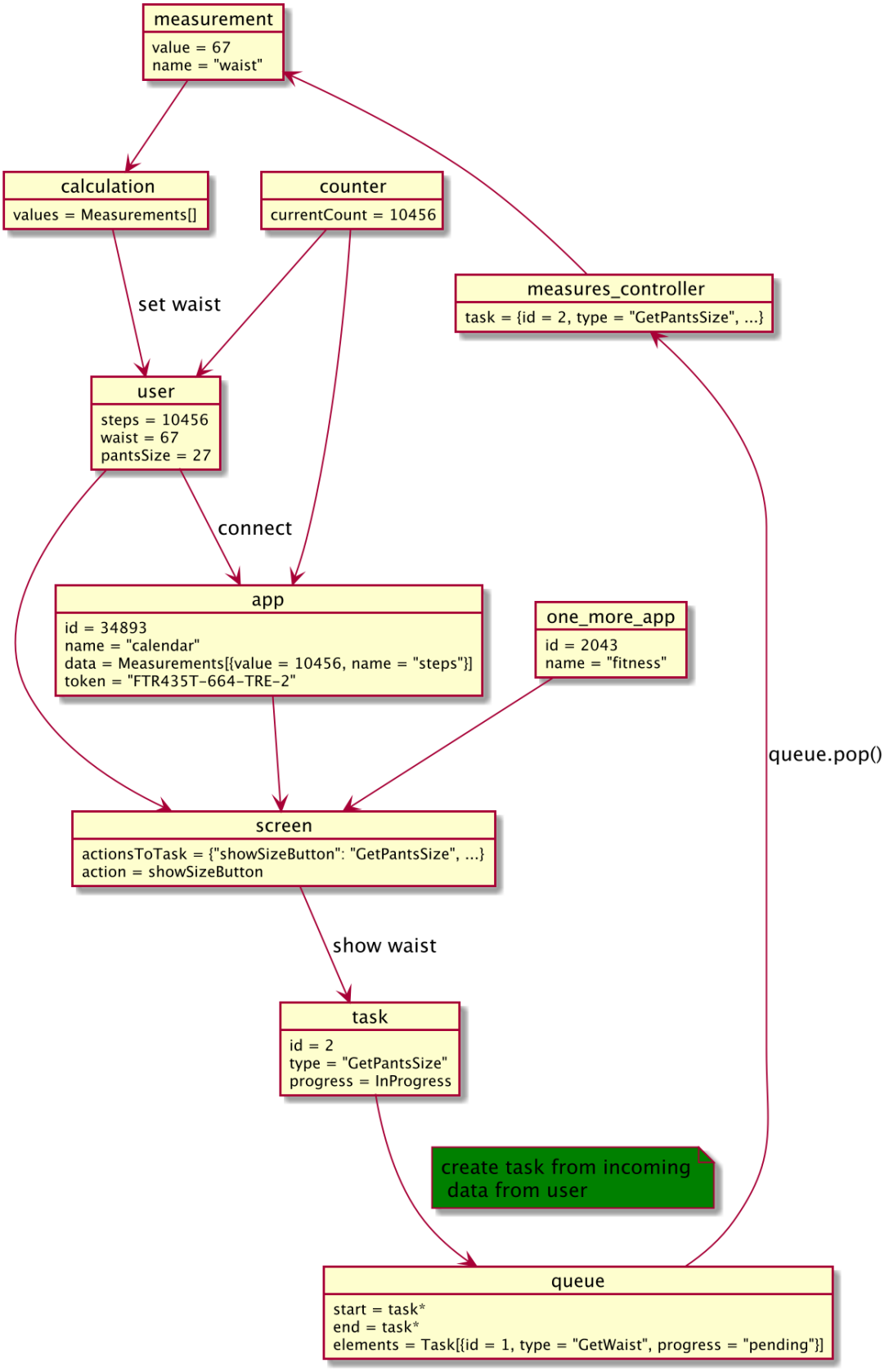


Plantuml:

```
@startuml

scale 1200 width

Class Calculation {
   void Save()
   values Measurement[]
   Measurement Calculate()
}

Class User {
   {static} String name
   steps: Integer
   waist: Integer
   pantsSize: Integer

   void getWaist()
   void getSteps()
   void getPantsSize()
   Integer setWaist(Integer)
   Integer setSteps(Integer)
   Integer setPantsSize(Integer)
}

Class App {
   {static} Integer id
   String name
   __ private data __
   data : Measurements[]
   {static} String token
   Integer Connect()
   Integer GetAccess()
}

Class Task {
   {static} Integer id
   type : String
   progress : String
   void Hold()
   void Ready()
   void InProgress()
   Task Constructor()
   void Destructor()
}

Class Queue {
   start : Task*
   end : Task*
   elements : Task[]
   Task pop()
   void push(Task)
   Task shift()
   void push(Unshift)
}

Class Counter {
   currentCount : Integer
   void Incr()
   void Reset()
}

Class Measurment {
   value : Integer
   name : String
   void Save()
   void Repeat()
}

Class MeasuresController {
   CurrentTask : Task
   String Resolve()
   Measurements[] MakeMeasures()
}

Class Screen {
   action : String
   actionsToTask : map[String]String
   String HandleAction()
   void ShowResult()
   Task CreateTask()
}

Task --> Queue : New Task
note on link #green: create task from incoming\n
data from User or App
User -left-> Screen : Some UI action
Screen -up-> Task : New Task
App -left-> Screen : New Task
Queue -right-> MeasuresController : Queue.pop()
MeasuresController --> Measurment
MeasuresController --> Calculation
Measurment --> User : Set Some Data
Calculation --> User
User -right-> App
Counter --> User
Counter -right-> App

@enduml
```

Диаграмма классов демонстрирует общую структуру иерархии классов системы и их структуру. Стоит начать с классов, описывающих акторов. User содержит поля, описывающие внешние параметры пользователя, а также гетеры и сеттеры для них. User или App могут через интерфейс могут создавать объекты класса Task, предоставляющий набор методов для установки стадии выполнения той или иной задачи (Hold(), Ready(), InProgress()). Объекты типа Task попадают в структуру данных Queue - очередь - где ожидают выполнения. Структура класса Очередь стандартная - указатели на начало и конец, элементы очереди в виде массива Tasks[] и методы push() и pop(). В контроллере измерений производится определение типа задачи и получение данных путем создания объектов измерений и/или вычислений. Класс измерений Measurement и вычислений Calculation максимально просты - измерение можно сохранить - Save() и повторить - Repeat(), вычисления - Calculate() и Save(). Класс счетчик имеет методы Increase() и Reset(). Класс «Приложение» принадлежит непосредственно системе, то есть лежит в памяти ремня и служит лишь для упорядочивания взаимодействий со всеми приложениями, подключаемыми к устройству, но ни в коем случае не описывает структуру самого приложения. У App есть id и имя, ему можно отправить массив с измерениями, присоединиться к нему и предоставить доступ к определенным данным (это может осуществлять пользователь через интерфейс).

Диаграмма объектов (Object Diagram)



Plantuml:

```
@startuml

scale 1200 width

object calculation
    calculation : values = Measurements[]
object user
    user : steps = 10456
    user : waist = 67
    user : pantsSize = 27
object app
    app : id = 34893
    app : name = "calendar"
    app : data = Measurements[{value = 10456, name = "steps"}]
    app : token = "FTR435T-664-TRE-2"
object one_more_app
    one_more_app : id = 2043
    one_more_app : name = "fitness"
object screen
    screen : actionsToTask = {"showSizeButton": "GetPantsSize", ...}
    screen : action = showSizeButton
object task
    task : id = 2
    task : type = "GetPantsSize"
    task : progress = InProgress
object queue
    queue : start = task*
    queue : end = task*
    queue : elements = Task[{id = 1, type = "GetWaist", progress = "pending"}]
object counter
    counter : currentCount = 10456
object measurement
    measurement : value = 67
    measurement : name = "waist"
object measures_controller
    measures_controller : task = {id = 2, type = "GetPantsSize", ...}

measures_controller -up-> measurement
task --> queue
note on link #green: create task from incoming\n data from user
screen --> task : show waist
queue -left-> measures_controller : queue.pop()
measurement --> calculation
calculation --> user : set waist
user --> screen
app --> screen
one_more_app --> screen
user --> app : connect
counter --> user
counter --> app

@enduml
```

Диаграмма объектов в языке моделирования предназначена для демонстрации совокупности моделируемых объектов и связей между ними в фиксированный момент времени. На данной диаграмме пользователь прошел 10456 шагов, обхват талии = 67, размер брюк = 27. В процессе постановки в очередь - измерение размера брюк. Выполняется - измерение обхвата талии. Устройство в данный момент подключено к приложению «Календарь», которому переданы данные о пройденных шагах, b «Фитнес».