Entwurf TECO

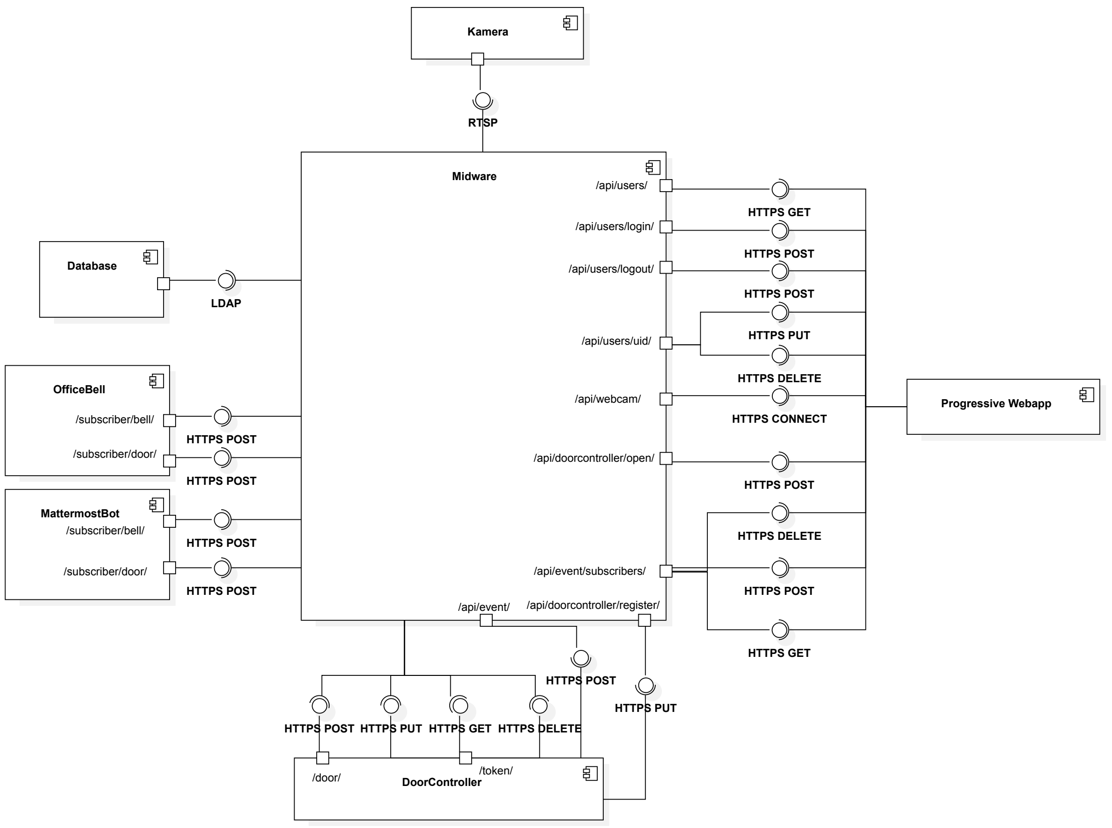
TomThalion

December 2021

Version	Date	Changes
1.0	31.01.2022	(originalversion)
1.1	20.02.2022	Update dokumentation of eventsubscriber classes
1.2	28.02.2022	Update Class and Sequence diagramms of eventsubscribers
1.3	07.03.2022	Added Class-, Sequence- and Package-Diagram to the PWA section
1.4	01.04.2022	Update classdocumentation Eventsubscriber Bots
1.5	08.04.2022	Updated Doorcontroller Class diagrams

Contents

1 System overview

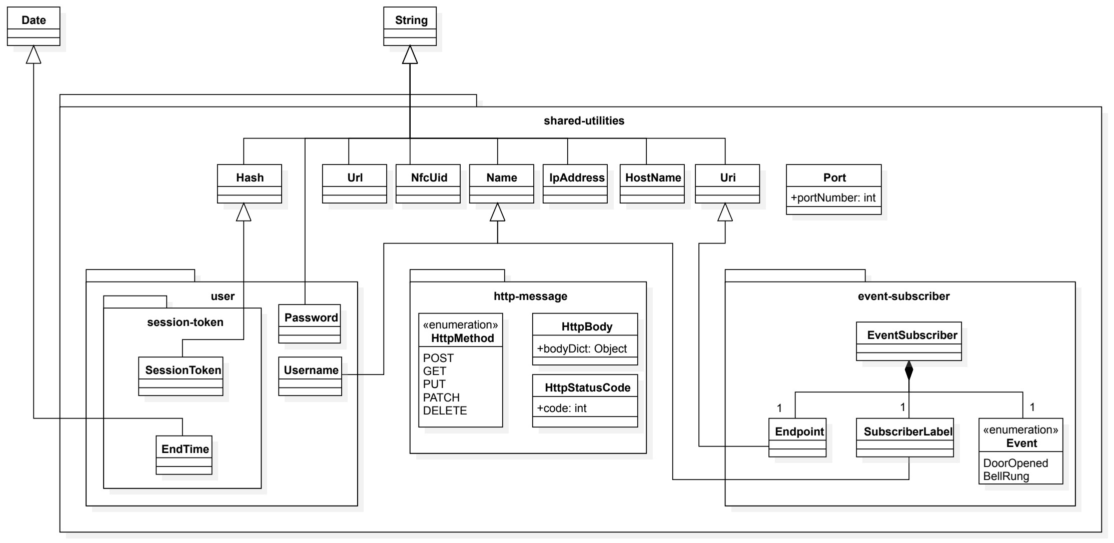


Our system is divided into various subcomponents. The midware, the door controller, a camera, the progressive webapp and event subscribers. In addition, the midware accesses a database through LDAP.

2 Subsystems

2.1 Shared Utilities

Many types are shared between different subsystems, so instead of declaring these types separately, they can simply import the following packages.



Documentation

- Hash, Url, Nfc-Token, Name, IpAddress, HostName, Uri
  These types can be used as Strings to signalize that they represent a specific type of String respectively .

- Port
  Represents a specific Port, designated by an integer

- Package: user
  Contains all user-related types.
  - Username
    A more precise type of Name, to show that it is to be used as a username.
  - Password
    Can be used as Strings to signalize that a password is represented.

- Package: session-token
  Contains all types related to the users session tokens.
  - SessionToken
    Can be used as a Hash (ergo as a string) to identify the users authority.
  - EndTime
    A subtype of Date, that signals the usage as the time that a token expires.
- Package: http-message
  Contains all types related to http messages
  - (Enum) HttpMethod
    Represents the type of the request.
  - HttpBody
    Represents the body of the http request, which might be empty
  - HttpStatus
    Represents the status of a completed request, containing the triple digit code, signaling the status.
- Package: event-subscriber
  Contains all types related to subscribers and their events
  - EventSubscriber
    Represents a subscriber with its endpoint, label and event type.
  - Endpoint
    Can be used as an URI to signal that an endpoint is represented.
  - SubscriberLabel
    A more precise type of Name, to show that it is to be used as the label of a subscriber.
  - (Enum) Event
    Represents the event that will trigger the notification of a subscriber.

2.2 Door Controller

In order to meet the requirements from the functional specification, the following functionality must be provided by the door controller:

- Door opening.
- A door bell.
- A NFC-Reader to scan for NFC-Chips.
- An HTTPServer and HTTPClient to communicate with the midware.
- A local database.

2.2.1 API

The midware communicates with the door controller over this interface.

Open Door

Opens the door.
URL: '/door/'
Method: 'POST'
Authentication required: Yes
Data constraints: No data constraints.
Success Response:
Condition: Everything is OK.
Code: '200 OK'
Content example:

```
{}
```

Error Response
Condition: Invalid Pre-Shared Key.
Code: '401 NOT AUTHORIZED'
Content example:

```
{
    "msg": "$(psk) is invalid."
}
```

Condition: Internal error.
Code: '500 INTERNAL SERVER ERROR'
Content example:

```
{
    "msg": "$(description)"
}
```

Get hash of database

Get a sha256 hash of the tokens concatenated in lexicographic order.
URL: ‘/token/‘
Method: ‘GET‘
Authentication required: Yes
Data constraints: No data constraints.
Success Response
Condition: Everything is OK.
Code: ‘200 OK‘
Content example:

```
{
    "hash": "1234567890ABCDEF1234567890ABCDEF"
}
```

Error Response
Condition: Invalid json/missing field.
Code: ‘400 BAD REQUEST‘
Content example:

```
{
    "msg": "$(description)"
}
```

Condition: Invalid Pre-Shared Key.
Code: ‘401 NOT AUTHORIZED‘
Content example:

```
{
    "msg": "$(psk) is invalid."
}
```

Condition: Internal error.
Code: ‘500 INTERNAL SERVER ERROR‘
Content example:

```
{
    "msg": "$(description)"
}
```

Put NFC-Tokens

Adds NFC-Tokens to the database or replaces NFC-Tokens in the database.
URL: ‘/token/‘
Method: ‘PUT‘
Authentication required: Yes
Data constraints:

name	Description	Type	Optional
tokens	List of objects with a toPut field and optional toReplace field.	List of Objects	No

Data example

```
{
    "tokens": [
        {
            "toReplace": "1234567890ABCD",
            "toPut": "234567890ABCDE"
        }
    ]
}
```

Success Response
Condition: Everything is OK.
Code: ‘200 OK‘
Content example: {}
Error Response
Condition: Invalid json/missing field.
Code: ‘400 BAD REQUEST‘
Content example:

```
{
    "msg": "$(description)"
}
```

Condition: Invalid Pre-Shared Key.
Code: ‘401 NOT AUTHORIZED‘
Content example:

```
{
    "msg": "$(psk) is invalid."
}
```

Condition: Internal error.
Code: ‘500 INTERNAL SERVER ERROR‘
Content example:

```
{
    "msg": "$(description)"
}
```

Delete NFC-Tokens

Adds/Replaces token to/in database.
URL: ‘/token/‘

Method: ‘DELETE‘
Authentication required: Yes
Data constraints: deleteAll or tokens must be set.

name	Description	Type	Optional
deleteAll	If true the whole database gets reset	bool	Yes
tokens	The tokens to delete.	List of String	Yes

Data example

```
{
    "deleteAll": false,
    "tokens": [
        "1234567890ABCD",
        "234567890ABCDE"
        "34567890ABCDEF,"
    ]
}
```

Success Response
Condition: Everything is OK.
Code: ‘200 OK‘
Content example: {}
Error Response
Condition: Invalid json/missing field.
Code: ‘400 BAD REQUEST‘
Content example:

```
{
    "msg": "$(description)"
}
```

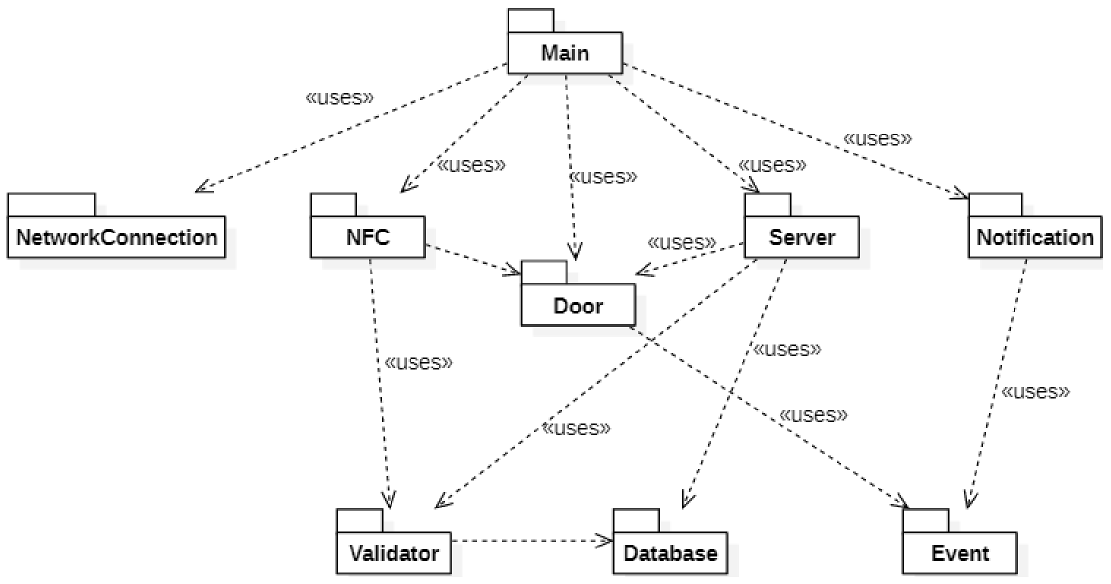
Condition: Invalid Pre-Shared Key.
Code: ‘401 NOT AUTHORIZED‘
Content example:

```
{
    "msg": "$(psk) is invalid."
}
```

Condition: Internal error.
Code: ‘500 INTERNAL SERVER ERROR‘
Content example:

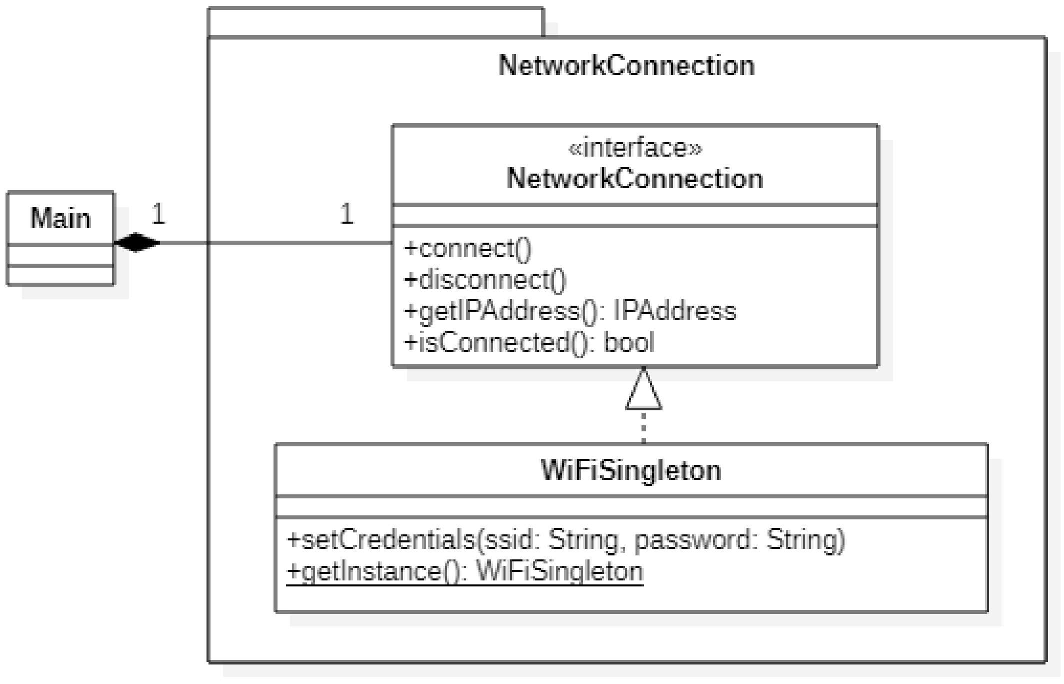
```
{
    "msg": "$(description)"
}
```

2.2.2 Architecture Overview



2.2.3 Class Diagrams

Package Network Connection

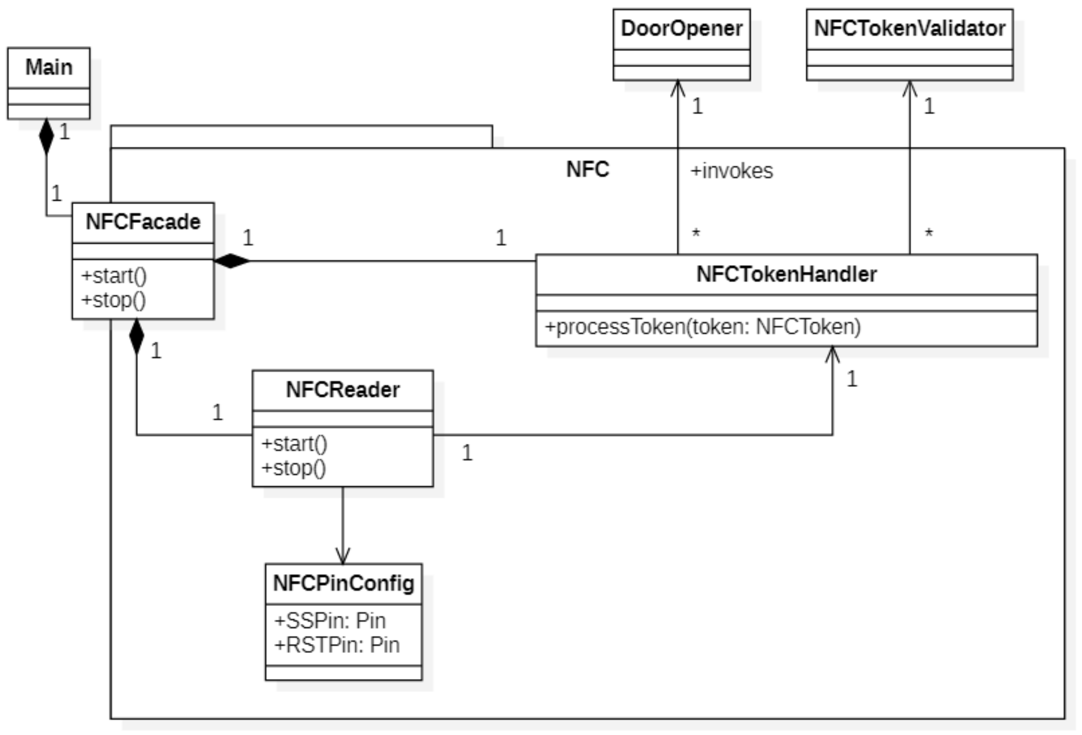


Documentation

- Interface **NetworkConnection**
  Manages a connection to a network.
  - **connect()**
    Establishes a network connection.
  - **disconnect()**
    Disconnects from the network.
  - **isConnected()**
    Signals if a network connection is present.
    return true if a network connection is present.
    return false if a network connection is not present.
  - **getIPAddress(): IPAddress**
    Returns the IP address of this device in the network.
    return The IP Address of this device in the network.
  - **isConnected(): bool**
    Checks if connected to the network. **return** true if connected to the network.

- Class **WiFiSingleton** implements **NetworkConnection**
  Manages a WiFi connection. That includes attempting to reconnect whenever the connection is lost.
  - **connect()**
    Establishes a WiFi connection.
  - **disconnect()**
    Disconnects from the WiFi network.
  - **isConnected()**
    Signals if a WiFi connection is present.
    return true if a WiFi connection is present.
    return false if a WiFi connection is not present.
  - **getIPAddress(): IPAddress**
    Returns the IP address of this device in the network.
    return The IP Address of this device in the network.
  - **setCredentials(ssid: String, password: String)**
    Sets the WiFi network credentials.
    param ssid: The ssid of the WiFi network.
    param password: The password of the WiFi network.
  - **{static} getInstance(): WiFiSingleton**
    Returns the instance.
    return The instance.

Package NFC



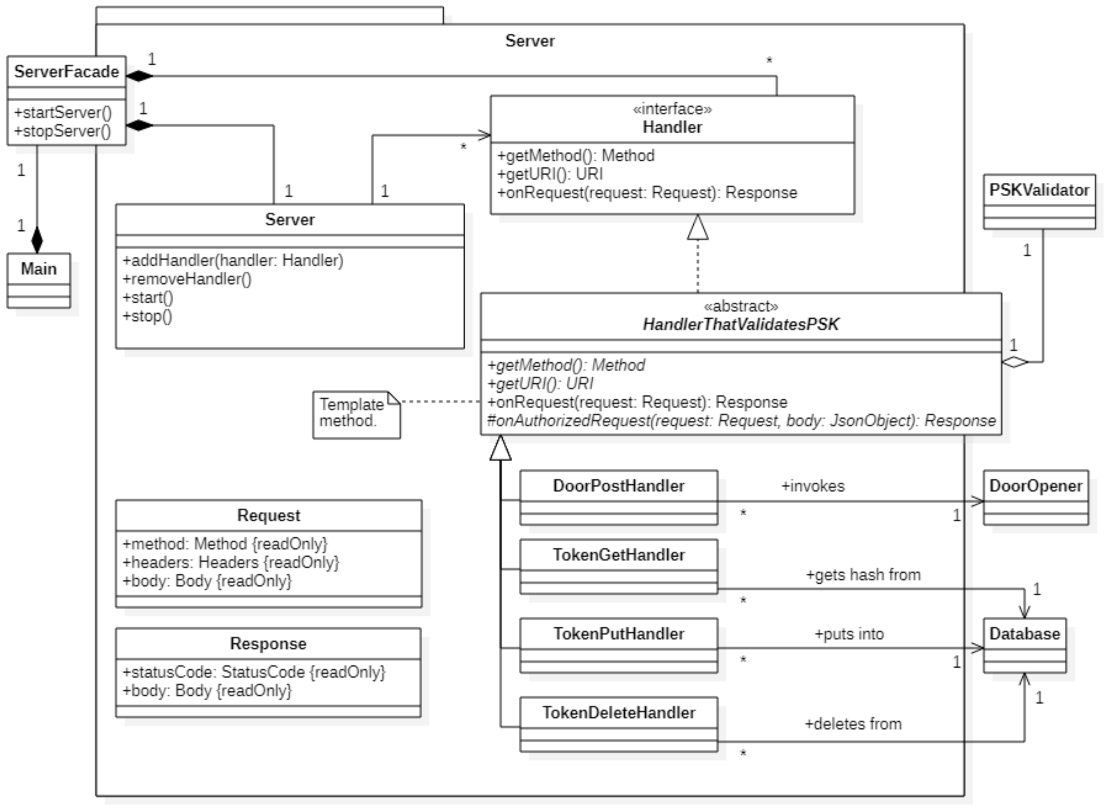
Documentation

- class **NFCFacade**
  Minimalistic interface for the NFC package.
  - **NFCFacade(doorOpener: DoorOpener, tokenValidator: KeyValidator<NFCToken>)**
    Construct a new NFC-Facade object.
    param doorOpener. A Door Opener.
    param tokenValidator A NFC-Token Validator.
  - **start()**
    The NFC-Reader starts scanning for NFC-Chips.
  - **stop()**
    The NFC-Reader stops scanning for NFC-Chips.
- class **NFCReader**
  Scans for NFC-Chips and passes their NFC-Tokens to a NFC-Token handler.
  - **NFCReader(config: NFCPinConfig, handler: NFCTokenHandler)**
    Construct a new NFC-Reader object.

    param config. A NFC Pin Config.
    param handler. A NFC-Token Handler.
  - **start()**
    The NFC-Reader starts scanning for NFC-Chips.
  - **stop()**
    The NFC-Reader stops scanning for NFC-Chips.

- class **NFCTokenHandler**
  Handles a NFC-Token
  Gets invoked when a NFC-Chip is detected by the NFC-Reader. When the NFC-Chip has a valid NFC-Token the handler shall open the door.
  - **NFCTokenHandler(doorOpener: DoorOpener, tokenValidator: KeyValidator<NFCToken>)**
    Construct a new NFC-Reader object.
    param doorOpener. A Door Opener.
    param tokenValidator. A NFC-Token Validator.
  - **processToken(token: NFCToken)**
    Check if token is valid and if so open the door.
    param token A NFC-Token.

Package Server



Documentation

- class **ServerFacade**
  A minimalistic interface to the server.
  - **ServerFacade(port: Port, pskValidator: KeyValidator<PSK>, doorOpener: doorOpener, database: Database)**
    Construct a new Server Facade object.
    param port. The server port.
    param pskValidator. Pre-Shared Key Validator initialized with the real Pre-Shared Key.
    param doorOpener. A Door Opener.
    param database. A database.
  - **startServer()**
    Starts the underlying server.
  - **stopServer()**
    Stops the underlying server.

- class **Server**
  Listens on the standard port and distributes HTTP requests to the handlers.
  - **Server(port: Port)**
    Construct a new HTTP Server
    param port. The port the server will be listening on.
  - **addHandler(handler: Handler)**
    Add a handler.
    param handler. A Handler object.
  - **removeHandler(handler: Handler)**
    Remove a handler.
    param handler. A Handler object.
  - **start()**
    Start listening on port.
  - **stop()**
    Stop listening on port.
- interface **Handler**
  Handles the HTTP requests of specific method and specific URI.
  - **getMethod(): Method**
    Return the method the Handler handles.
    return The method the Handler handles.
  - **getURI(): URI**
    Return the URI the Handler handles.
    return The URI the Handler handles.
  - **onRequest(request: Request): Response**
    Gets called when a HTTP Request meets the handler requirements.
    param request. A Request object.
    return A Response object.
- {abstract} class **HanderThatValidatesPSK** implements **Handler**
  Handles the HTTP requests of specific method and specific URI.
  - **HandlerThatValidatesPSK(pskValidator: Validator<PSK>)**
    Construct a new Handler That Validates PSK object.
    param pskValidator. A Pre-Shared Key validator.
  - **getMethod(): Method**
    Return the method the Handler handles.
    return The method the Handler handles.
  - **getURI(): URI**
    Return the URI the Handler handles.
    return The URI the Handler handles.

- **onRequest(request: Request): Response**
  A template method that validates that the supplied Pre-Shared Key is valid. Then it calls onAuthorizedRequest(request: Request: body: JsonObject): Response.
  param request. A Request object.
  return A Response object.
- **onAuthorizedRequest(request: Request: body: JsonObject): Response**
  Gets called by onRequest(request: Request): Response.
  param request. A Request object.
  param body. Parsed request body.
  return A Response object.

• class **DoorPostHandler** extends **HanderThatValidatesPSK**
Handles POST requests to /door/. Opens the door.

- **DoorPostHandler(pskValidator: Validator<PSK>, doorOpener: DoorOpener)**
  Construct a DoorPostHandler object.
  param pskValidator. A Pre-Shared Key validator.
  param doorOpener. A Door Opener.
- **onRequest(request: Request: body: JsonObject): Response**
  Opens the door.
  param request. Ignored.
  param body. Ignored.
  return See API documentation.

• class **TokenGetHandler** extends **HanderThatValidatesPSK**
Handles GET requests to /token/. Gets a hash of the database.

- **TokenGetHandler(pskValidator: Validator<PSK>, database: Database**
  Construct a new Token Get Handler object.
  param pskValidator. A Pre-Shared Key validator.
  param database. A Database.
- **onRequest(request: Request: body: JsonObject): Response**
  Sends a hash of the database.
  param request. Ignored
  param body. Ignored.
  return See API documentation.

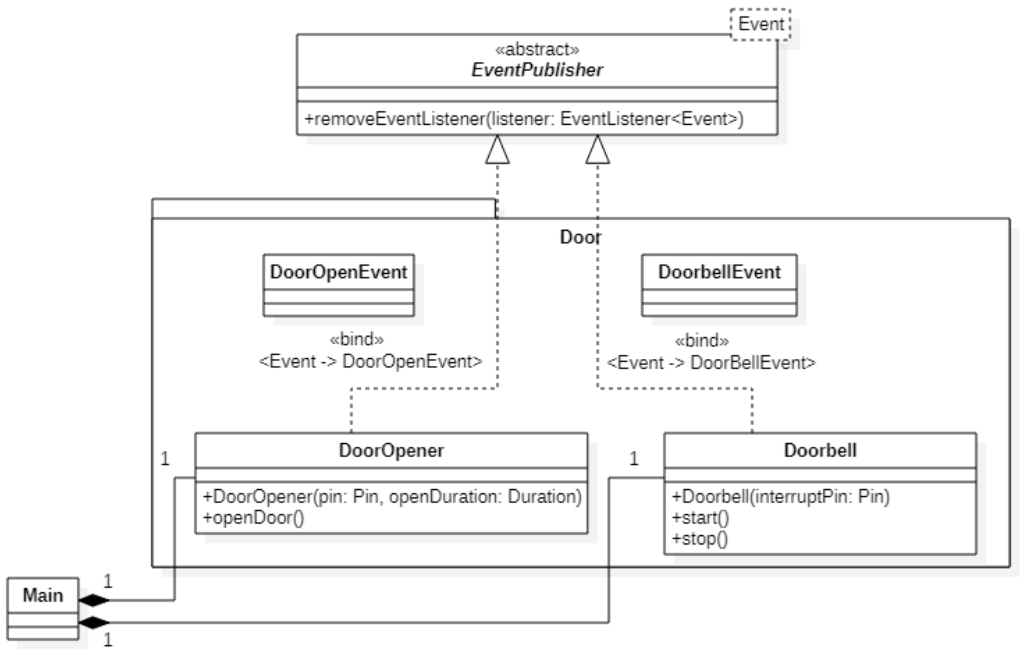
• class **TokenPutHandler** extends **HanderThatValidatesPSK**
Handles PUT requests to /token/. Inserts/Replaces NFC-Tokens.

- **TokenPutHandler(pskValidator: Validator<PSK>, database: Database**
  Construct a new Token Put Handler object.
  param pskValidator. A Pre-Shared Key validator.
  param database. A Database.

- **onRequest(request: Request: body: JsonObject): Response**
  Puts the NFC-Tokens in the body into the database.
  param request. Ignored.
  param body. A Json object.
  return See API documentation.

- class **TokenDeleteHandler** extends **HanderThatValidatesPSK**
  Handles PUT requests to /token/. Deletes NFC-Tokens
  - **TokenDeleteHandler(pskValidator: Validator<PSK>, database: Database**
    Construct a new Token Delete Handler object.
    param pskValidator. A Pre-Shared Key validator.
    param database. A Database.
  - **onRequest(request: Request: body: JsonObject): Response**
    Deletes the NFC-Tokens in the request from the database.
    param request. Ignored.
    param body. A Json object.
    return See API documentation.

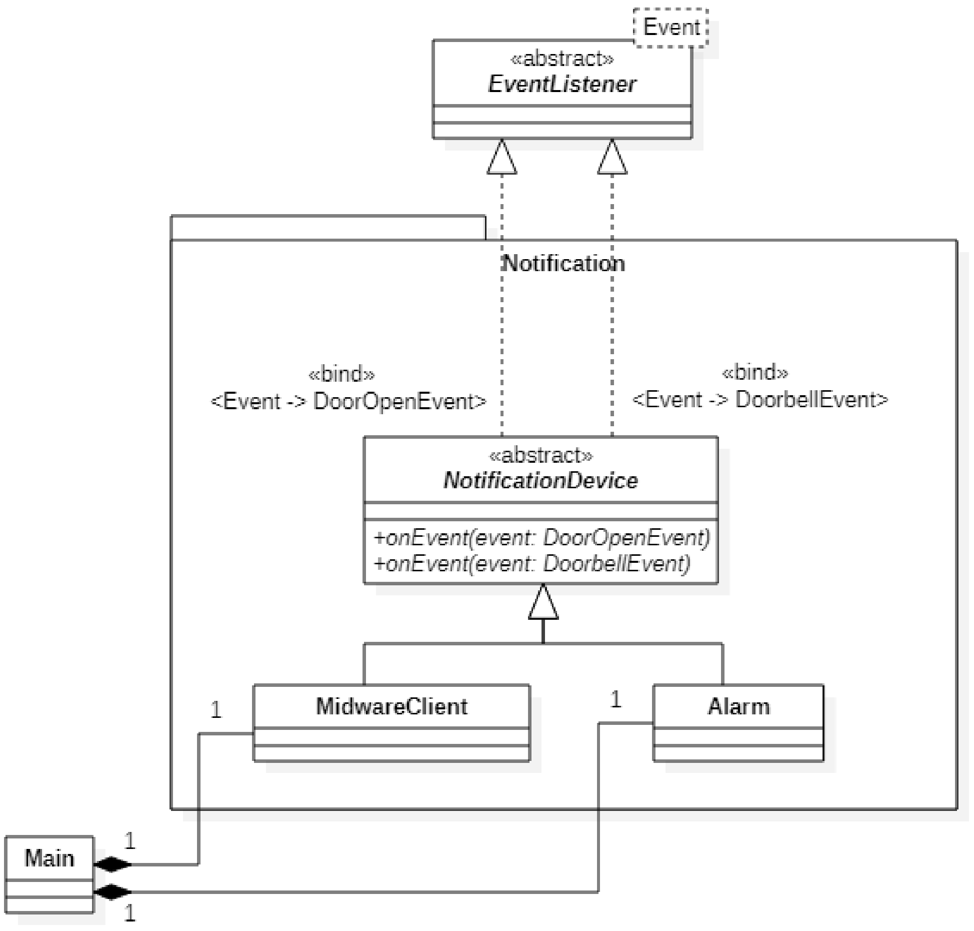
Package Door



Documentation

- class **DoorOpener** extends **EventPublisher<DoorOpenEvent>**
  Responsible for opening a door.
  - **DoorOpener(pin: Pin)**
    Construct a new Door Opener object.
    param pin. The door-open pin.
  - **openDoor()**
    Opens the door and emits a Door Open Event.
- class **Doorbell** extends **EventPublisher<DoorbellEvent>**
  Emits a Doorbell Event whenever an external button is pressed.
  - **Doorbell(interruptPin: Pin)**
    Construct a new Doorbell object.
    param pin. A interrupt pin.
  - **start()**
    Start listening for interrupts.
  - **stop()**
    Stop listening for interrupts.

Package Notification



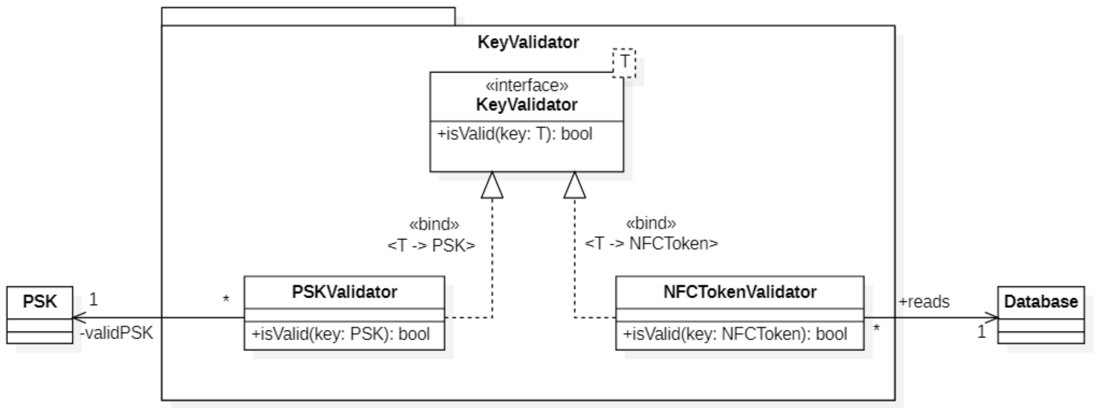
- class **MidwareClient** implements **EventListener<DoorBellEvent>**, **EventListener<DoorOpenEvent>**
  Sends information to the midware.
  - **MidwareClient(midwareBaseUrl: URL, preSharedKey: PSK)**
    Construct a new MidwareClient object.
    param midwareBaseURL The base URL of the midware.
    param preSharedKey The Pre-Shared Key
  - **registerInMidware(microcontrollerBaseURL: URL)**
    Tells the midware the URL of the microcontroller.
    param microcontrollerBaseURL The base URL of the microcontroller.
  - **onEvent(event: DoorBellEvent)**

Tells the midware that the bell was rung.
param event A Door Bell Event.

- **onEvent(event: DoorOpenEvent)**
  Tells the midware that the door was opened.
  param event A Door Open Event.

- class **Alarm** implements **EventListener<DoorBellEvent>**, **EventListener<DoorOpenEvent>**
  Starts acoustic signal when the bell is pressed.
  - **Alarm(maxRingDuration: Duration)**
    Construct a new Alarm object.
    param maxRingDuration. The maximal duration the alarm should ring.
  - **onEvent(event: DoorBellEvent)**
    Starts accoustic signal.
    param event A Door Bell Event.
  - **onEvent(event: DoorOpenEvent)**
    Stops accoustic signal.
    param event A Door Open Event.

Package Key Validator



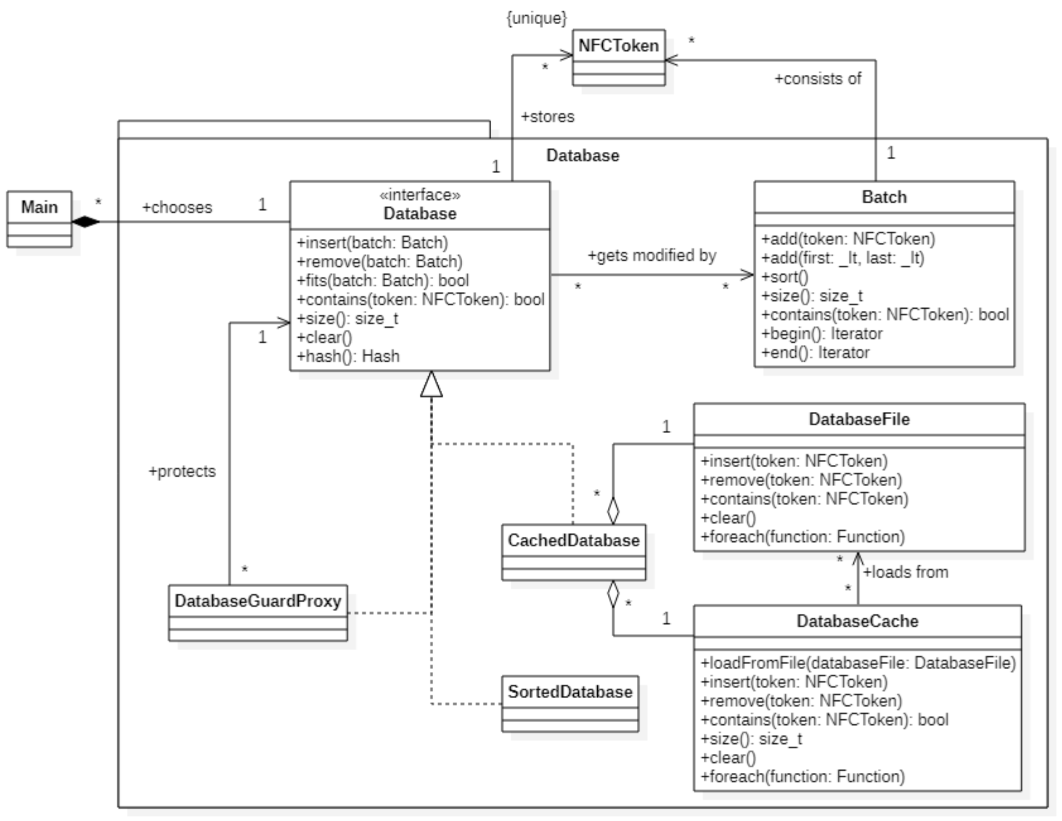
Documentation

- {generic<Key>} interface **KeyValidator**
  Validates keys.
  - **isValid(key: Key): bool**
    Checks if a key is valid.
    param key A key.

    return true if the key is valid.
    return false if the key is not valid.

- class **NFCTokenValidator** implements **KeyValidator<NFCToken>**
  Validates NFC-Tokens.
  - **NFCTokenValidator(database: Database)**
    Construct a new NFC-Token Validator object.
    param database The database used for validation.
  - **isValid(key: NFCToken): bool**
    Checks if a NFC-Token is stored in the database.
    param key A NFC-Token.
    return true if key is stored in the database
    return false if key is not stored in the database
- class **PSKValidator** implements **KeyValidator<PSK>**
  Validates Pre-Shared Key.
  - **PSKValidator(realPSK: PSK)**
    Construct a new Pre-Shared Key Validator object.
    param realPSK The real Pre-Shared Key.
  - **isValid(key: PSK): bool**
    Checks if key equals the real Pre-Shared Key.
    param key A Pre-Shared Key.
    return true if key equals the real Pre-Shared Key.
    return false if key does not equal the real Pre-Shaed Key.

Package Database



Documentation

- interface **Database**
  Stores NFC-Tokens.
  - **insert(batch: Batch)**
    Inserts NFC-Tokens.
    param batch. A batch of NFC-Tokens.
  - **contains(token: NFCToken): bool**
    Checks if the database contains a NFC-Token.
    param token A NFC-Token.
    return true if the database contains the token.
    return false if the database does not contain the token.
  - **remove(batch: Batch)**
    Removes NFC-Tokens.
    param batch A Batch of NFC-Tokens.

  - **clear()**
    Removes all NFC-Tokens from the database.
  - **hash(): Hash**
    Hashes the database
    return A hash of the database.

- class **CachedDatabase** implements **Database**
  Stores NFC-Tokens in flash memory and in RAM.
  - **CachedDatabase(fileName: FileName)**
    Constructs a new Cached Database object
    param fileName: The name of the database file.
  - **insert(batch: Batch)**
    Inserts NFC-Tokens.
    param batch. A batch of NFC-Tokens.
  - **contains(token: NFCToken): bool**
    Checks if the database contains a NFC-Token.
    param token A NFC-Token.
    return true if the database contains the token.
    return false if the database does not contain the token.
  - **remove(batch: Batch)**
    Removes NFC-Tokens.
    param batch A Batch of NFC-Tokens.
  - **clear()**
    Removes all NFC-Tokens from the database.
  - **hash(): Hash**
    Hashes the database
    return A hash of the database.

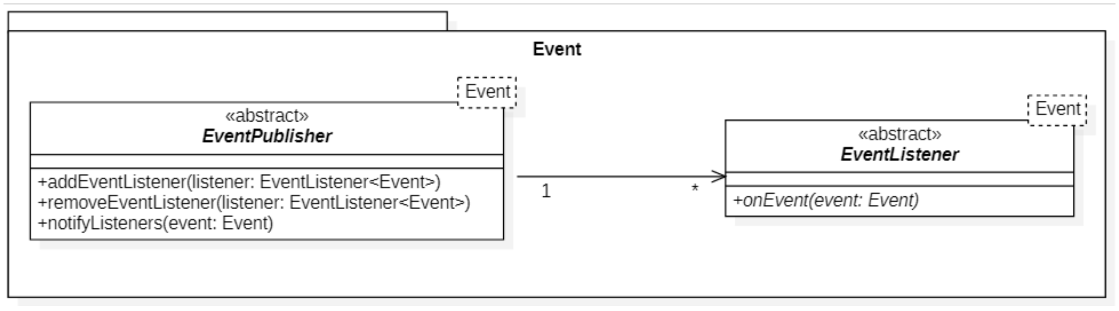
- class **SortedDatabase** implements **Database**
  Stores ordered NFC-Tokens in flash memory.

- class **DatabaseGuardProxy** implements **Database**
  Wraps a Databsase. It uses a lock mechanism on every database access to protect the database implementation from data races.
  - **DatabaseGuardProxy()**
    Constructs a new Database Guard Proxy object
  - **insert(batch: Batch)**
    Inserts NFC-Tokens.
    param batch. A batch of NFC-Tokens.
  - **contains(token: NFCToken): bool**
    Checks if the database contains a NFC-Token.

param token A NFC-Token.
return true if the database contains the token.
return false if the database does not contain the token.

- **remove(batch: Batch)**
  Removes NFC-Tokens.
  param batch A Batch of NFC-Tokens.
- **clear()**
  Removes all NFC-Tokens from the database.
- **hash(): Hash**
  Hashes the database
  return A hash of the database.

Package Event


Event
Event
«abstract»
EventPublisher
+addEventListener(listener: EventListener<Event>)
+removeEventListener(listener: EventListener<Event>)
+notifyListeners(event: Event)
1
*
Event
«abstract»
EventListener
+onEvent(event: Event)

Documentation

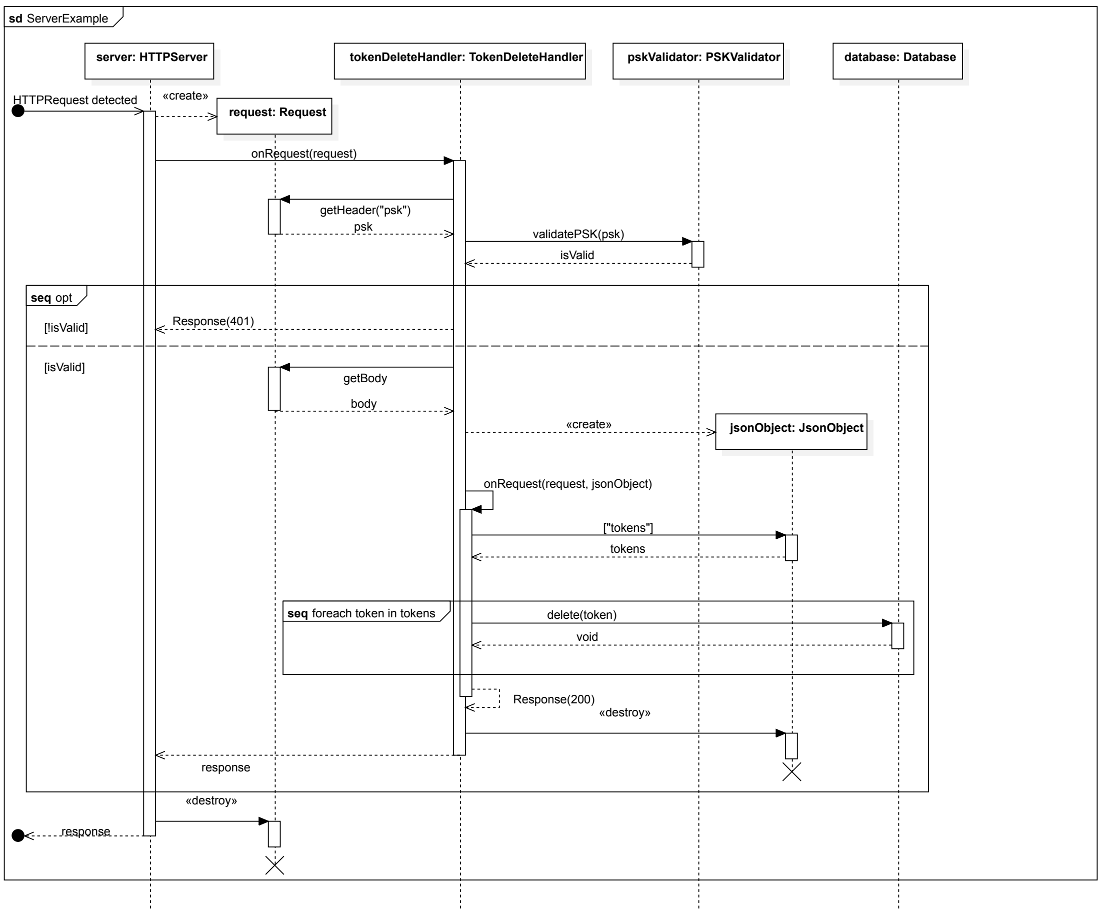
- {abstract} class **EventPublisher<Event>**
  Distributes events to it's listeners.
  tparam Event A type containing information.
  - **addListener(listener: EventListener<Event>)**
    Adds an event listener.
    param listener: The event listener
  - **removeListener(listener: EventListener<Event>)**
    Removes an event listener.
    param listener: The event listener
  - **notifyListeners(event: Event)**
    Notify all subscribed event listeners.
    param event The event to distribute.

- {abstract} class **EventListener<Event>**
  Listens to an event.
  tparam Event A type containing information.
  - **{abstract} onEvent(event: Event)**
    Action to perform when an event occurs.
    param event An event.

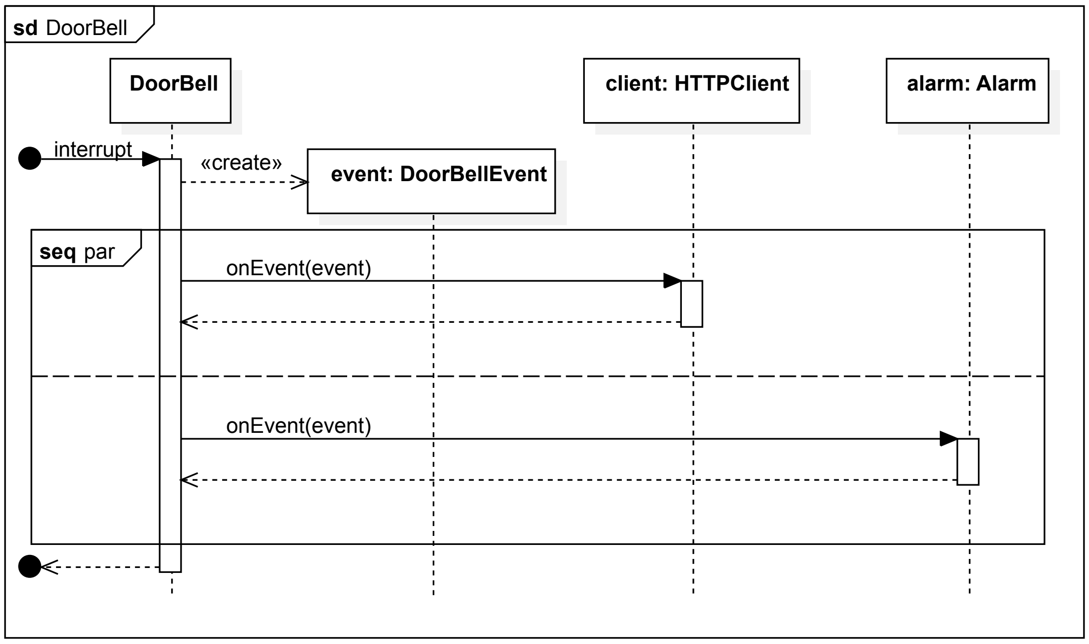
2.2.4 Sequence diagrams

Server Example

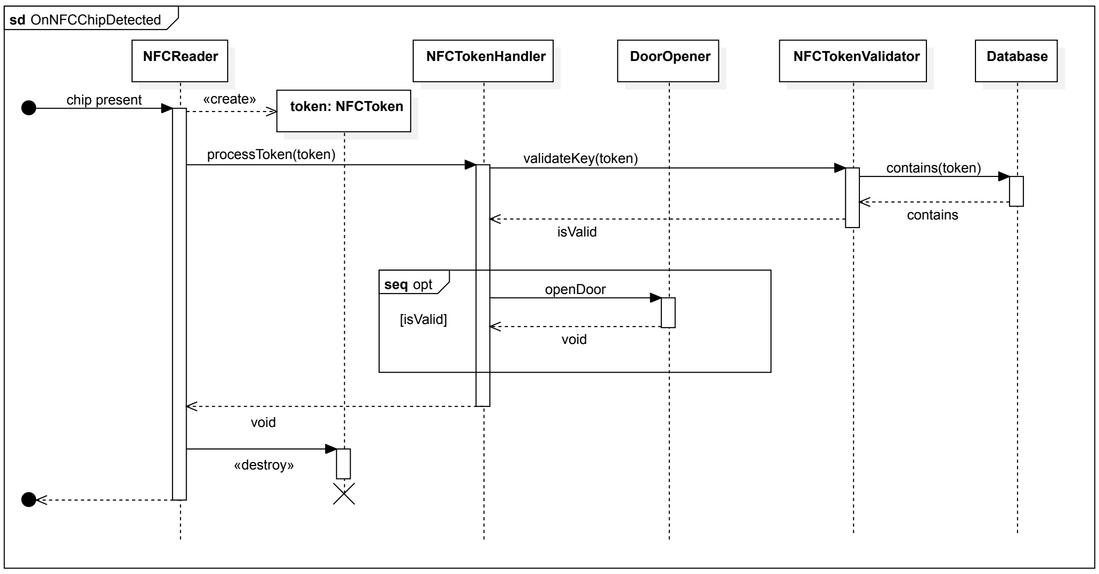
An example to illustrate how the server handles requests.



Door Bell



NFC Chip detected



2.3 Event-Subscribers

2.3.1 API

Open Endpoints: Open endpoints require an authentication.

Close Endpoints: Closed endpoints require an authentication.

Door related

- **door:** ‘POST /subscriber/door/‘

Bell related

- **bell:** ‘POST /subscriber/bell/‘

Door has been opened

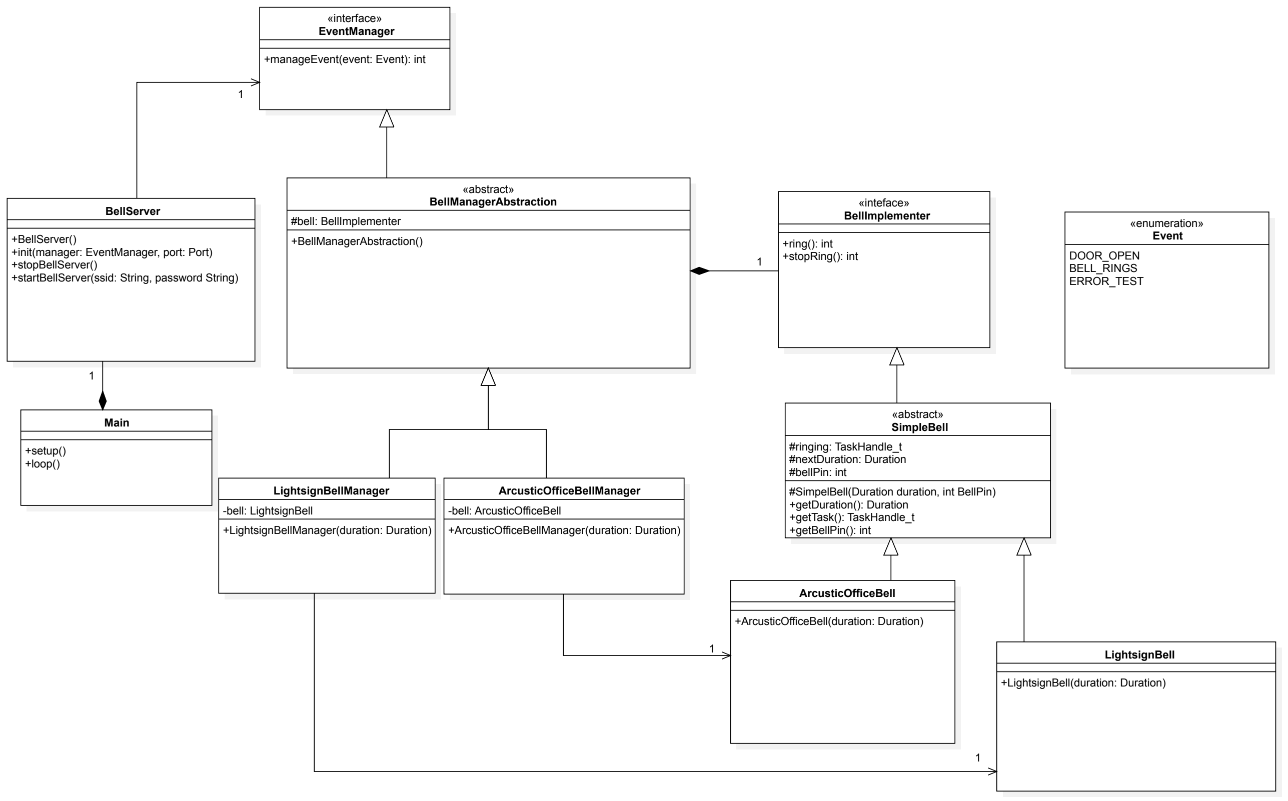
Get posted if the door gets opened by the doorcontroller.
 URL: /subscriber/door/
 Method: ‘POST‘
 Authentication required: No
Success Response:
 Condition: If everything is OK.
 URL: ‘200 OK‘
Error Response:
 Condition: Some unidentified complications.
 Code: ´400 Bad Request´

Ring the Bell

Get posted if the bell rings.
 URL: /subscriber/bell/
 Method: ‘POST‘
 Authentication required: No
Success Response:
 Condition: If everything is OK.
 URL: ‘200 OK‘
Error Response:
 Condition: Some unidentified complications.
 Code: ´400 Bad Request´

2.3.2 Office Bell

Class Diagram



Class Documentation

Main:

- Setup of the system and startingpoint of the programm. It creates a server and calls the methode startServer() to initialize the bell controller.

BellServer:

- Represents an BellServer, which receives posts from the midware, specified in the API documentation, and transmitts received Events to a given EventManager.

EventManager:

- An Interface which represents an Eventmanger, which can manage some Events of the type Event.

BellManagerAbstraction:

- Represents an abstract Bellmanager, which manage a bell according to the given Events.

ArcousticOfficeBellManager:

- Represents an Bellmanager, which uses an AcousticOfficeBell.

LightsignBellManager:

- Represents an Bellmanager, which uses an LightsignBell.

BellImplementer:

- An Interface for a BellImplementer, which is used by a BellManager.

SimpleBell:

- A special implementation of a Bell, which represents a simpleBell which should be able to set and reset the output of the microcontroller.

ArcousticOfficeBell:

- A special implementation of a Bell, which represents a Bell which should be able to set and reset the output of the microcontroller suitable for the ArcusticBell in the TECO office.

LightsignBell:

- A special implementation of a Bell, which represents a Lightsign Bell. Should be able to set an reset the output of the micro controller suitable for the LightsignBell.

Duration:

- Represents a duration in multible units.

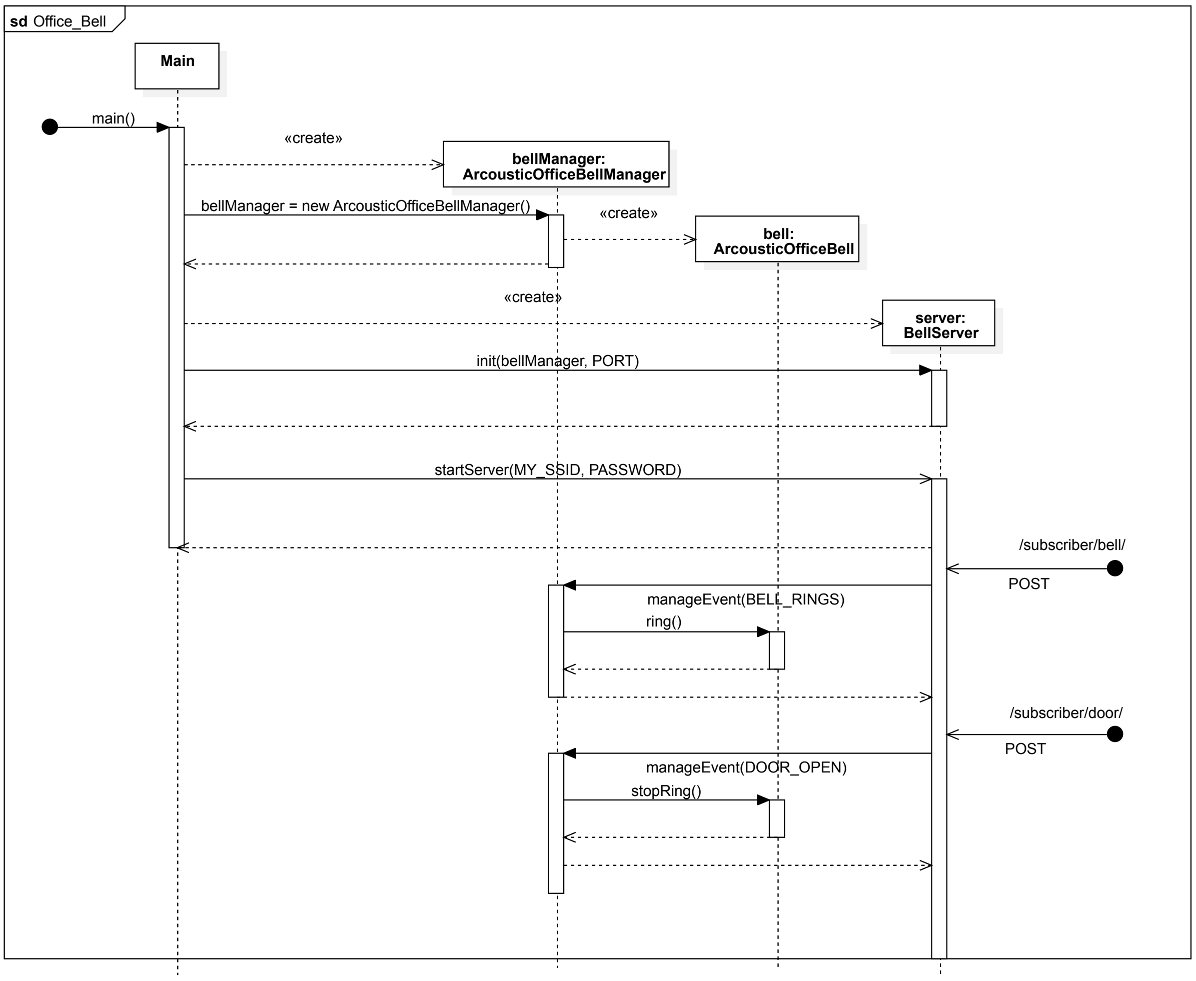
Event:

- This Enum owns all possible Events, an Eventmanager can manage.

Port:

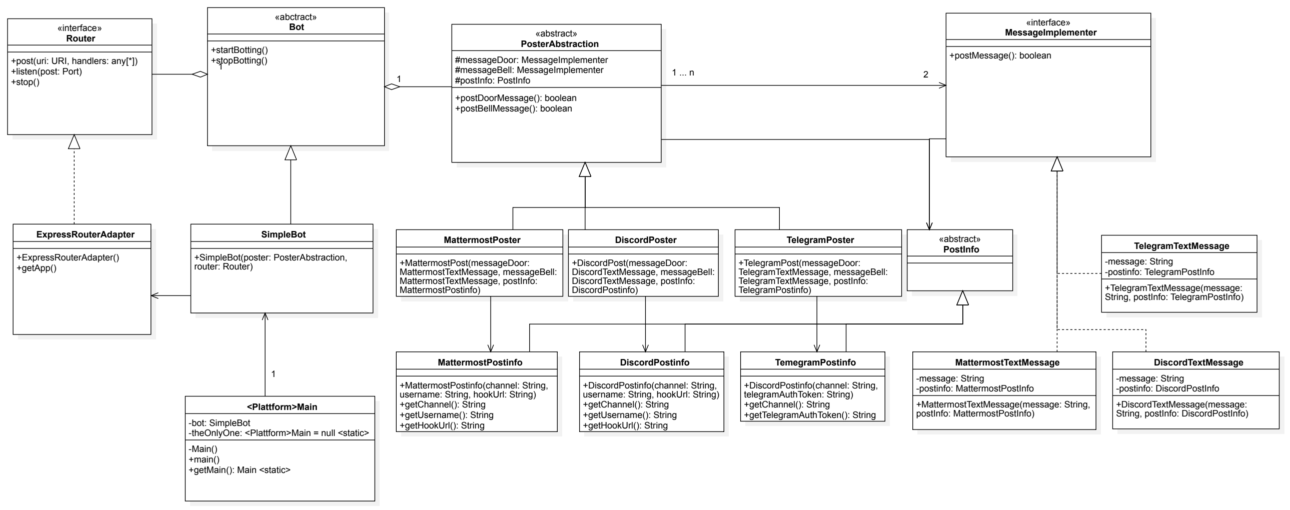
- Represents a port for a server.

Sequence Diagram



2.3.3 Bots

Class Diagram


«interface»
Router
+post(uri: URI, handlers: any[*])
+listen(post: Port)
+stop()
«abctract»
Bot
+startBotting()
+stopBotting()
1
1
«abstract»
PosterAbstraction
#messageDoor: MessageImplementer
#messageBell: MessageImplementer
#postInfo: PostInfo
+postDoorMessage(): boolean
+postBellMessage(): boolean
1 ... n
2
«interface»
MessageImplementer
+postMessage(): boolean
ExpressRouterAdapter
+ExpressRouterAdapter()
+getApp()
SimpleBot
+SimpleBot(poster: PosterAbstraction, router: Router)
1
<Plattform>Main
-bot: SimpleBot
-theOnlyOne: <Plattform>Main = null <static>
-Main()
+main()
+getMain(): Main <static>
MattermostPoster
+MattermostPost(messageDoor: MattermostTextMessage, messageBell: MattermostTextMessage, postInfo: MattermostPostinfo)
DiscordPoster
+DiscordPost(messageDoor: DiscordTextMessage, messageBell: DiscordTextMessage, postInfo: DiscordPostinfo)
TelegramPoster
+TelegramPost(messageDoor: TelegramTextMessage, messageBell: TelegramTextMessage, postInfo: TelegramPostinfo)
«abstract»
PostInfo
TelegramTextMessage
-message: String
-postinfo: TelegramPostInfo
+TelegramTextMessage(message: String, postInfo: TelegramPostInfo)
MattermostPostinfo
+MattermostPostinfo(channel: String, username: String, hookUrl: String)
+getChannel(): String
+getUsername(): String
+getHookUrl(): String
DiscordPostinfo
+DiscordPostinfo(channel: String, username: String, hookUrl: String)
+getChannel(): String
+getUsername(): String
+getHookUrl(): String
TemegramPostinfo
+DiscordPostinfo(channel: String, telegramAuthToken: String)
+getChannel(): String
+getTelegramAuthToken(): String
MattermostTextMessage
-message: String
-postinfo: MattermostPostInfo
+MattermostTextMessage(message: String, postInfo: MattermostPostInfo)
DiscordTextMessage
-message: String
-postinfo: DiscordPostInfo
+DiscordTextMessage(message: String, postInfo: DiscordPostInfo)

Class Documentation

DiscordMain:

- This is the DiscordMain class. Setup the Discordbotsystem with the main()

TelegramMain:

- This is the TelegramMain class. Setup the Telegrambotsystem with the main()

MattermostMain:

- This is the MattermostMain class. Setup the MattermostBotsystem with the main()

Bot:

- An abstract bot, which can receive Posts from the "Midware" byusing the Router and posts messages according to the Posts

SimpleBot:

- An implementation of a simple bot. It uses a Poster to Post messages at a platform specified by the poster

PosterAbstraction:

- A class with two unspecified message witch can be posted.

DiscordPoster:

- An implementation of a Poster, which post some textmessages at discord.

TelegramPoster:

- An implementation of a Poster, which post some textmessages at telegram.

MattermostPoster:

- An implementation of a Poster, which post some texts at mattermost.

MessageImplementer:

- Is an interface of an MessageImplementer, which represents a message witch can be posted by itself.

DiscordTextMessage:

- Special implementation of the MessageImplementer, which representsa text message, which can be posted at discord.

TelegramTextMessage:

- Special implementation of the MessageImplementer, which represents a text message, which can be posted at telegram.

MattermostTextMessage:

- Special implementation of the MessageImplementer, which represents a text message, which can be posted at mattermost.

Router:

- Is an inteface of an Router. The Router receives posts and run specific handlers as reaction.

ExpressRouterAdapter:

- A specific implementation of a Router which use an express router.

PostInfo:

- Info for a post of a posterAbstraction.

DiscordPostInfo:

- Info for a post at discord.

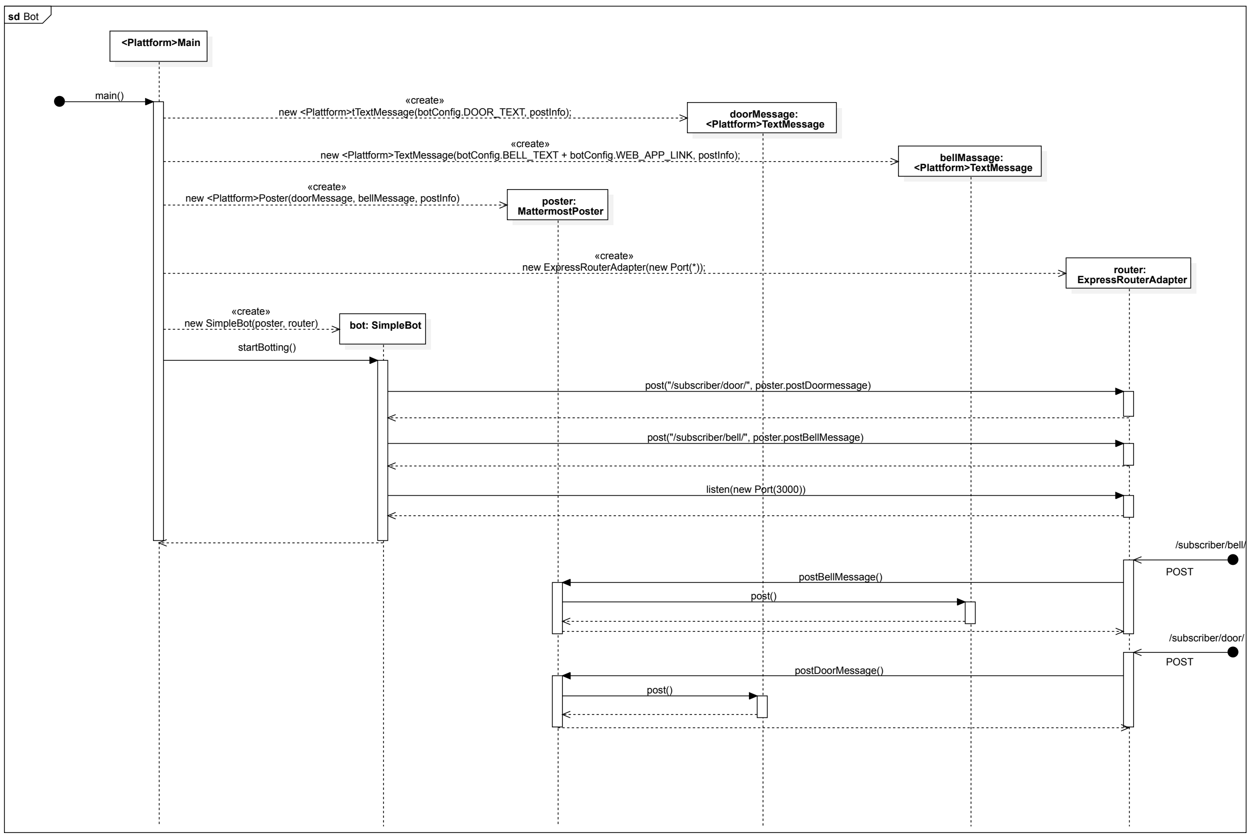
TelegramPostInfo:

- Info for a post at telegram.

MattermostPostInfo:

- Info for a post at mattermost.

Sequence Diagram

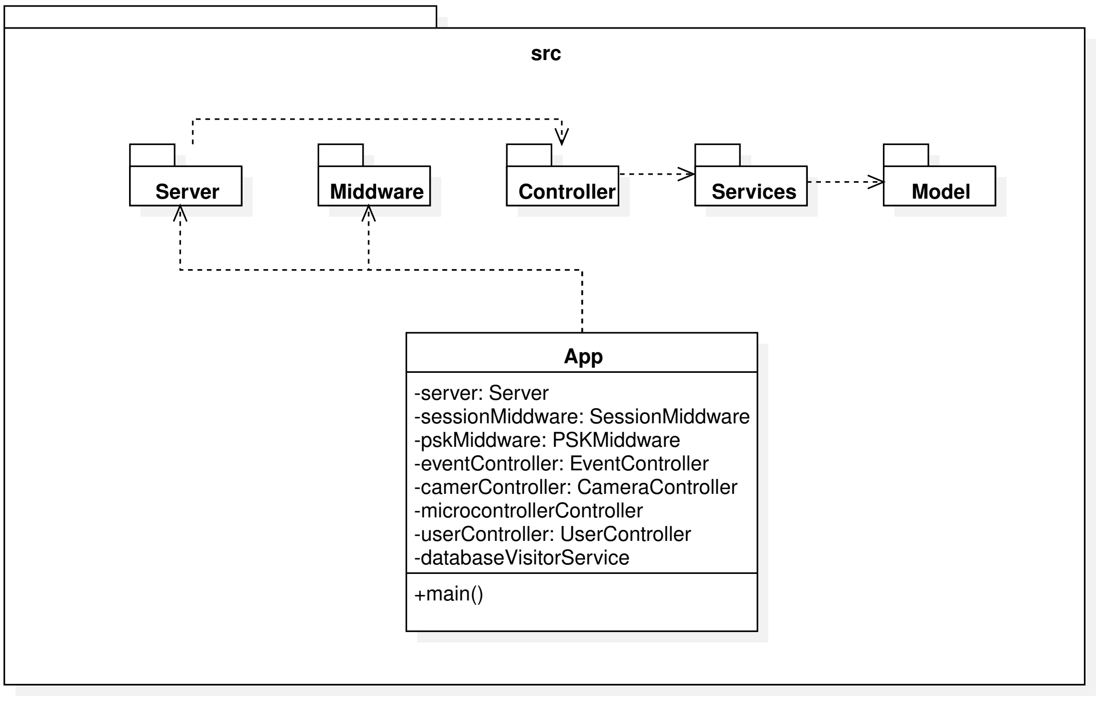

sd Bot
<Plattform>Main
main()
«create»
new <Plattform>tTextMessage(botConfig.DOOR_TEXT, postInfo);
doorMessage: <Plattform>TextMessage
«create»
new <Plattform>TextMessage(botConfig.BELL_TEXT + botConfig.WEB_APP_LINK, postInfo);
bellMassage: <Plattform>TextMessage
«create»
new <Plattform>Poster(doorMessage, bellMessage, postInfo)
poster: MattermostPoster
«create»
new ExpressRouterAdapter(new Port(*));
router: ExpressRouterAdapter
«create»
new SimpleBot(poster, router)
bot: SimpleBot
startBotting()
post("/subscriber/door/", poster.postDoormessage)
post("/subscriber/bell/", poster.postBellMessage)
listen(new Port(3000))
/subscriber/bell/
POST
postBellMessage()
post()
/subscriber/door/
POST
postDoorMessage()
post()

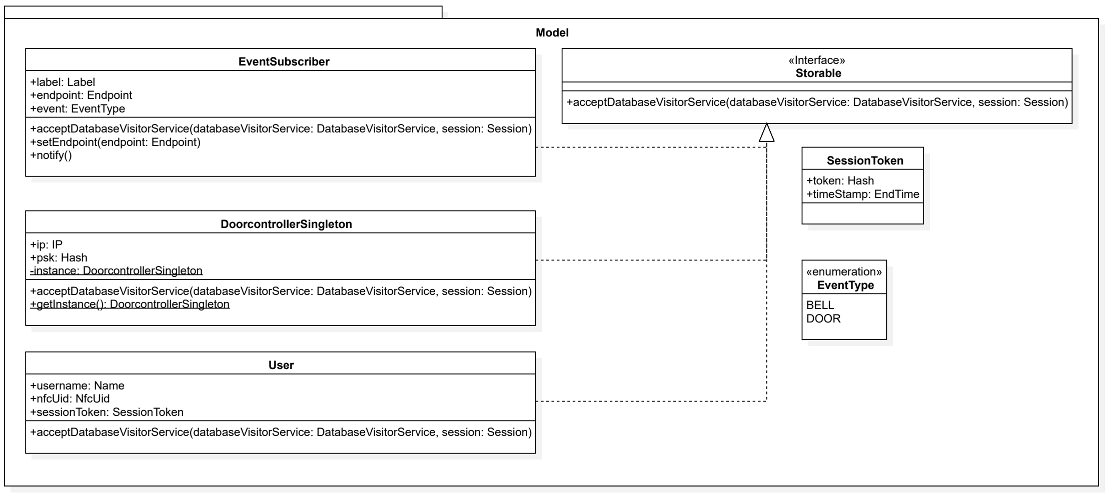
2.4 Midware

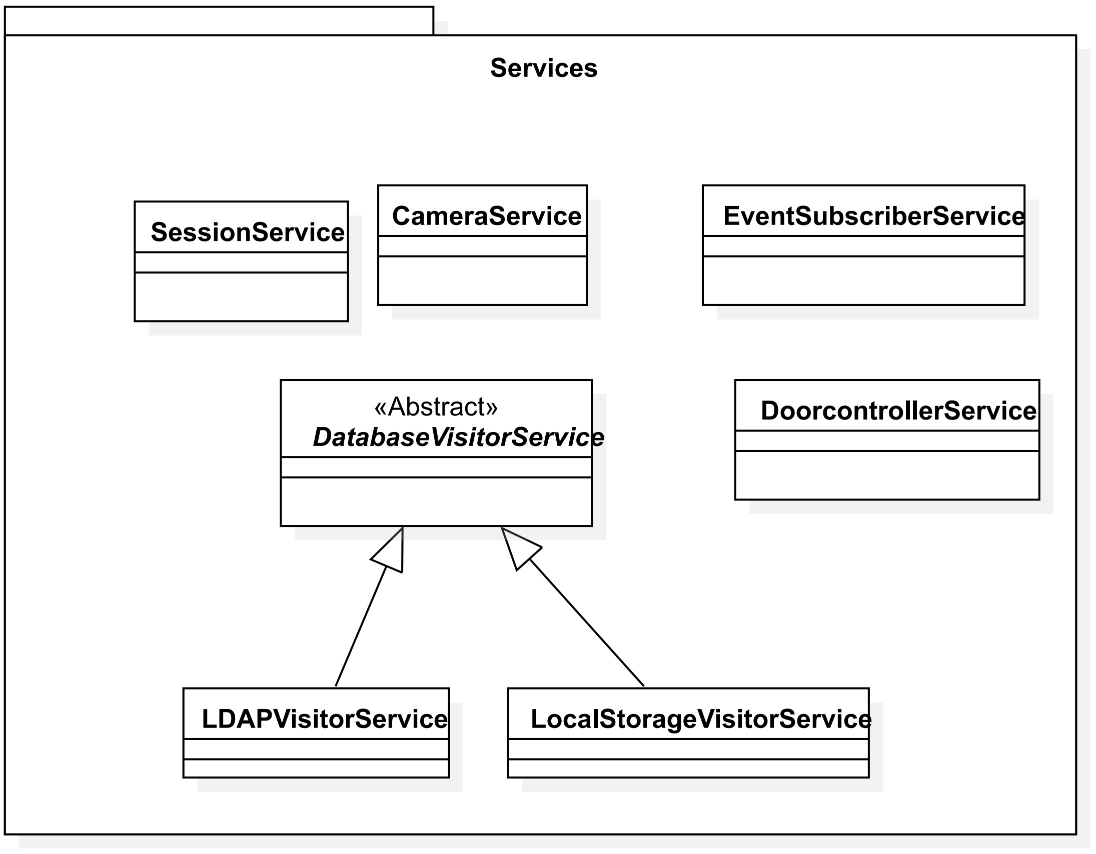
2.4.1 API

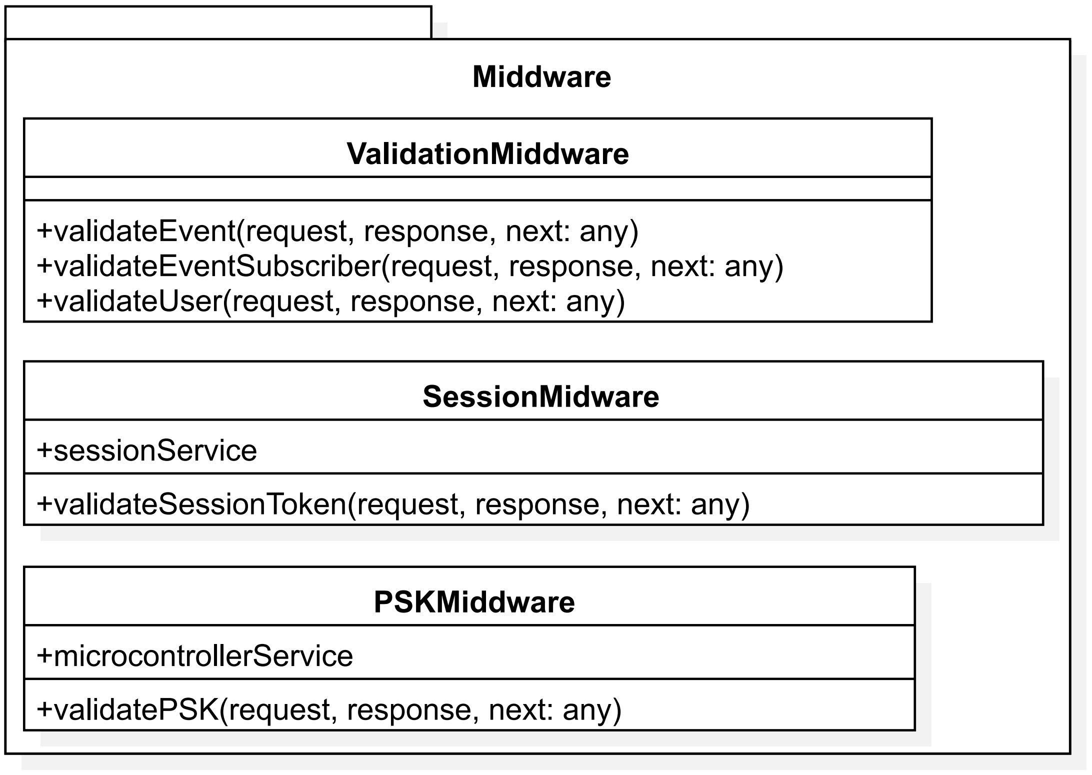
For the Midware API Documentation please read the documentation.md.

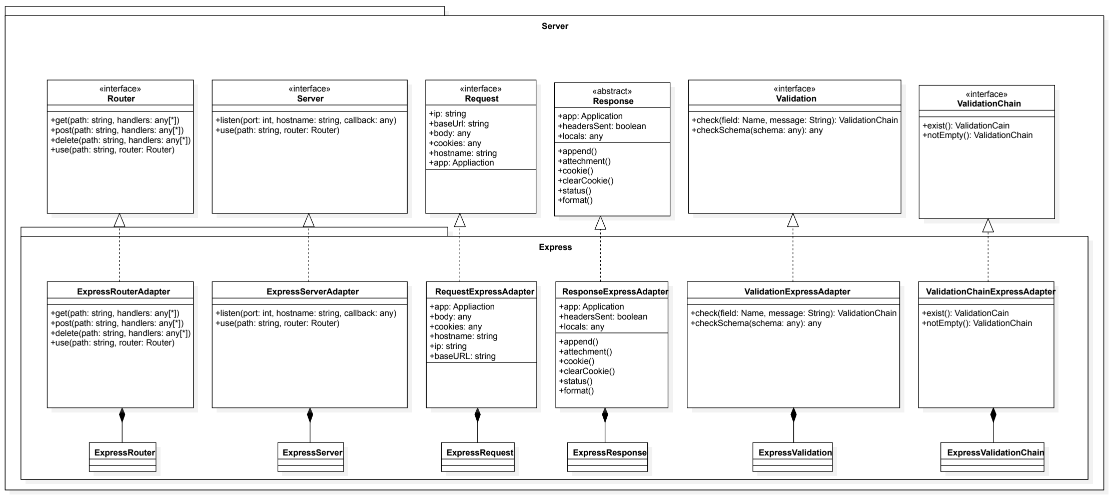
2.4.2 Package Diagrams

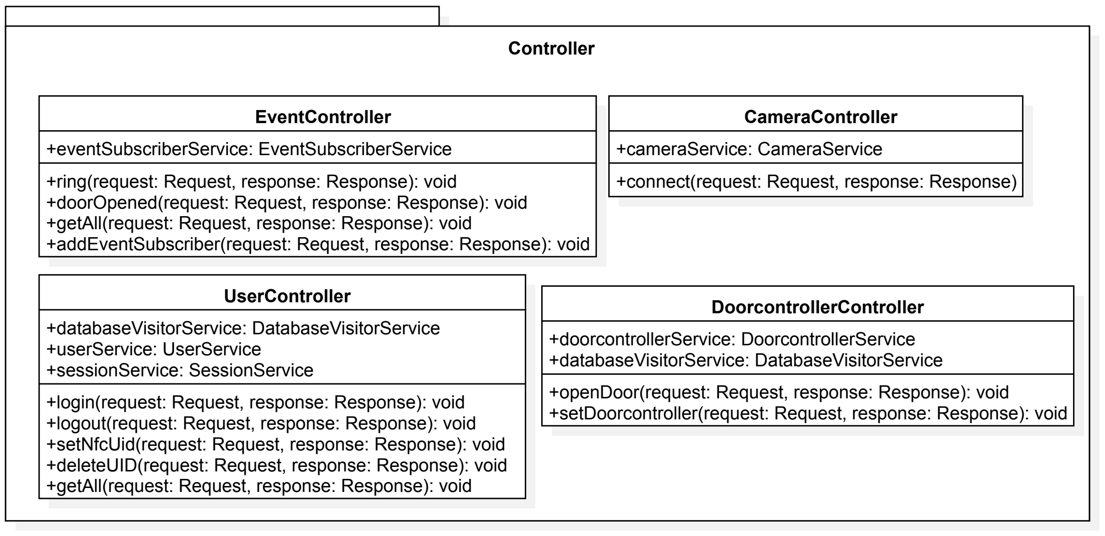

src
Server
Middware
Controller
Services
Model
App
-server: Server
-sessionMiddware: SessionMiddware
-pskMiddware: PSKMiddware
-eventController: EventController
-camerController: CameraController
-microcontrollerController
-userController: UserController
-databaseVisitorService
+main()


Model
EventSubscriber
+label: Label
+endpoint: Endpoint
+event: EventType
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)
+setEndpoint(endpoint: Endpoint)
+notify()
«Interface»
Storable
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)
SessionToken
+token: Hash
+timeStamp: EndTime
DoorcontrollerSingleton
+ip: IP
+psk: Hash
-instance: DoorcontrollerSingleton
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)
+getInstance(): DoorcontrollerSingleton
«enumeration»
EventType
BELL
DOOR
User
+username: Name
+nfcUid: NfcUid
+sessionToken: SessionToken
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)


Services
SessionService
CameraService
EventSubscriberService
«Abstract»
DatabaseVisitorService
DoorcontrollerService
LDAPVisitorService
LocalStorageVisitorService

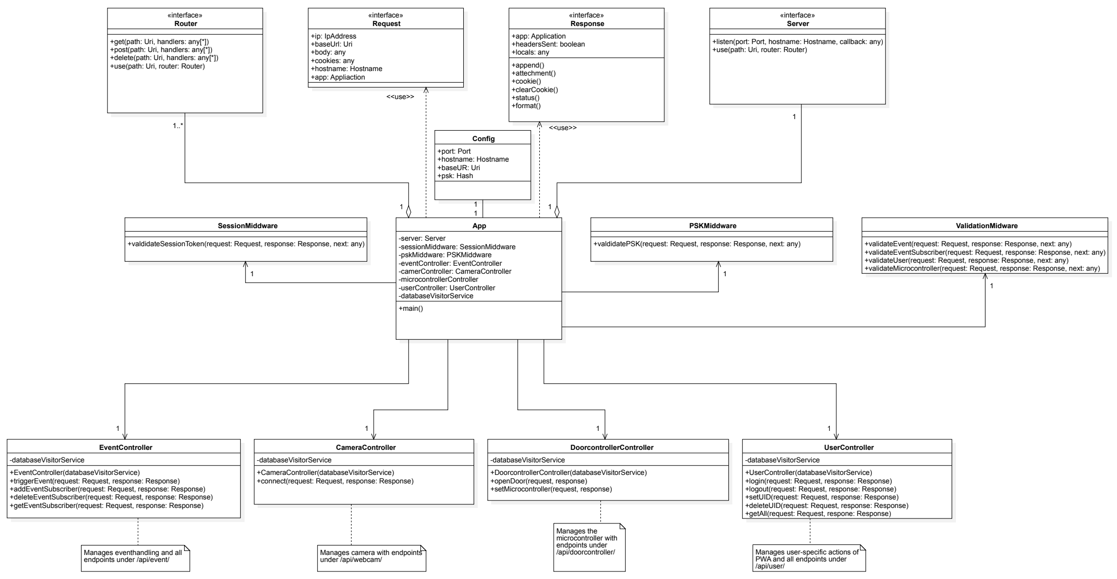

Middware
ValidationMiddware
+validateEvent(request, response, next: any)
+validateEventSubscriber(request, response, next: any)
+validateUser(request, response, next: any)
SessionMidware
+sessionService
+validateSessionToken(request, response, next: any)
PSKMiddware
+microcontrollerService
+validatePSK(request, response, next: any)


Server
«interface»
Router
+get(path: string, handlers: any[*])
+post(path: string, handlers: any[*])
+delete(path: string, handlers: any[*])
+use(path: string, router: Router)
«interface»
Server
+listen(port: int, hostname: string, callback: any)
+use(path: string, router: Router)
«interface»
Request
+ip: string
+baseUrl: string
+body: any
+cookies: any
+hostname: string
+app: Appliaction
«abstract»
Response
+app: Application
+headersSent: boolean
+locals: any
+append()
+attechment()
+cookie()
+clearCookie()
+status()
+format()
«interface»
Validation
+check(field: Name, message: String): ValidationChain
+checkSchema(schema: any): any
«interface»
ValidationChain
+exist(): ValidationCain
+notEmpty(): ValidationChain
Express
ExpressRouterAdapter
+get(path: string, handlers: any[*])
+post(path: string, handlers: any[*])
+delete(path: string, handlers: any[*])
+use(path: string, router: Router)
ExpressServerAdapter
+listen(port: int, hostname: string, callback: any)
+use(path: string, router: Router)
RequestExpressAdapter
+app: Appliaction
+body: any
+cookies: any
+hostname: string
+ip: string
+baseURL: string
ResponseExpressAdapter
+app: Application
+headersSent: boolean
+locals: any
+append()
+attechment()
+cookie()
+clearCookie()
+status()
+format()
ValidationExpressAdapter
+check(field: Name, message: String): ValidationChain
+checkSchema(schema: any): any
ValidationChainExpressAdapter
+exist(): ValidationCain
+notEmpty(): ValidationChain
ExpressRouter
ExpressServer
ExpressRequest
ExpressResponse
ExpressValidation
ExpressValidationChain


Controller
EventController
+eventSubscriberService: EventSubscriberService
+ring(request: Request, response: Response): void
+doorOpened(request: Request, response: Response): void
+getAll(request: Request, response: Response): void
+addEventSubscriber(request: Request, response: Response): void
CameraController
+cameraService: CameraService
+connect(request: Request, response: Response)
UserController
+databaseVisitorService: DatabaseVisitorService
+userService: UserService
+sessionService: SessionService
+login(request: Request, response: Response): void
+logout(request: Request, response: Response): void
+setNfcUid(request: Request, response: Response): void
+deleteUID(request: Request, response: Response): void
+getAll(request: Request, response: Response): void
DoorcontrollerController
+doorcontrollerService: DoorcontrollerService
+databaseVisitorService: DatabaseVisitorService
+openDoor(request: Request, response: Response): void
+setDoorcontroller(request: Request, response: Response): void

2.4.3 Class Diagrams

App


«interface»
Router
+get(path: Uri, handlers: any[*])
+post(path: Uri, handlers: any[*])
+delete(path: Uri, handlers: any[*])
+use(path: Uri, router: Router)
«interface»
Request
+ip: IpAddress
+baseUrl: Uri
+body: any
+cookies: any
+hostname: Hostname
+app: Appliaction
«interface»
Response
+app: Application
+headersSent: boolean
+locals: any
+append()
+attechment()
+cookie()
+clearCookie()
+status()
+format()
«interface»
Server
+listen(port: Port, hostname: Hostname, callback: any)
+use(path: Uri, router: Router)
<<use>>
<<use>>
1..*
1
1
1
1
1
Config
+port: Port
+hostname: Hostname
+baseUR: Uri
+psk: Hash
SessionMiddware
+valdidateSessionToken(request: Request, response: Response, next: any)
App
-server: Server
-sessionMiddware: SessionMiddware
-pskMiddware: PSKMiddware
-eventController: EventController
-camerController: CameraController
-microcontrollerController
-userController: UserController
-databaseVisitorService
+main()
PSKMiddware
+valdidatePSK(request: Request, response: Response, next: any)
ValidationMidware
+validateEvent(request: Request, response: Response, next: any)
+validateEventSubscriber(request: Request, response: Response, next: any)
+validateUser(request: Request, response: Response, next: any)
+validateMicrocontroller(request: Request, response: Response, next: any)
1
1
1
1
1
1
1
EventController
-databaseVisitorService
+EventController(databaseVisitorService)
+triggerEvent(request: Request, response: Response)
+addEventSubscriber(request: Request, response: Response)
+deleteEventSubscriber(request: Request, response: Response)
+getEventSubscriber(request: Request, response: Response)
CameraController
-databaseVisitorService
+CameraController(databaseVisitorService)
+connect(request: Request, response: Response)
DoorcontrollerController
-databaseVisitorService
+DoorcontrollerController(databaseVisitorService)
+openDoor(request, response)
+setMicrocontroller(request, response)
UserController
-databaseVisitorService
+UserController(databaseVisitorService)
+login(request: Request, response: Response)
+logout(request: Request, response: Response)
+setUID(request: Request, response: Response)
+deleteUID(request: Request, response: Response)
+getAll(request: Request, respone: Response)
Manages eventhandling and all endpoints under /api/event/
Manages camera with endpoints under /api/webcam/
Manages the microcontroller with endpoints under /api/doorcontroller/
Manages user-specific actions of PWA and all endpoints under /api/user/

- **App**

- **Config**
- **(Interface) Request**
  Request is an interface of a request which is sent from a client to out server.
- **(Interface) Response**
  Response is an interface for the response our server give the client.
- **(Interface) Router**
  Router is an interface to rout urls to different methods on our server.
- **(Interface) Validation**
  Validation is an interface for our main validation instance.
- **(Interface) ValidationChain**
  ValidationChain is an interface for processing validation.
- **(Interface) Server**
  Server is an interface to represent our server. The server can listen at a port and uses a router at a given path.

- **Sessionmidware**
  Sessionmidware is the midware to validate the session token of an authenticated user.

- **PSKmidware**
  PSKmidware is the midware to validate the psk of the microcontroller.

- **Validationmidware**
  ValidationMidware validates incoming requests and manages error handling.

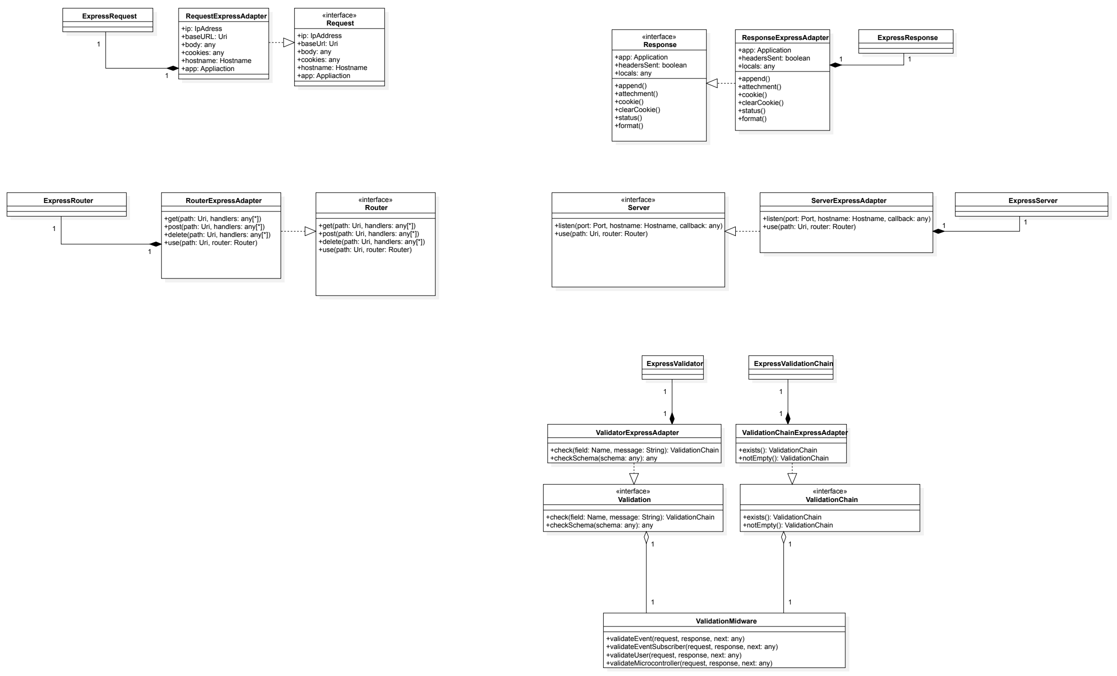
- **EventController**
  EventController controls the EventSubscribers. The class controls how the ring and doorOpen event is sent and also how EventSubscribers are added and read.

- **CameraController**
  CameraController controls the camera. A user can get the camera stream through the connect method.

- **DoorcontrollerController**
  DoorcontrollerController controls the Doorcontroller.

- **UserController**
  UserController controls the User.

Adapter


ExpressRequest
1
1
RequestExpressAdapter
+ip: IpAdress
+baseURL: Uri
+body: any
+cookies: any
+hostname: Hostname
+app: Appliaction
«interface»
Request
+ip: IpAddress
+baseUrl: Uri
+body: any
+cookies: any
+hostname: Hostname
+app: Appliaction
«interface»
Response
+app: Application
+headersSent: boolean
+locals: any
+append()
+attechment()
+cookie()
+clearCookie()
+status()
+format()
ResponseExpressAdapter
+app: Application
+headersSent: boolean
+locals: any
+append()
+attechment()
+cookie()
+clearCookie()
+status()
+format()
ExpressResponse
1
1
ExpressRouter
1
1
RouterExpressAdapter
+get(path: Uri, handlers: any[*])
+post(path: Uri, handlers: any[*])
+delete(path: Uri, handlers: any[*])
+use(path: Uri, router: Router)
«interface»
Router
+get(path: Uri, handlers: any[*])
+post(path: Uri, handlers: any[*])
+delete(path: Uri, handlers: any[*])
+use(path: Uri, router: Router)
«interface»
Server
+listen(port: Port, hostname: Hostname, callback: any)
+use(path: Uri, router: Router)
ServerExpressAdapter
+listen(port: Port, hostname: Hostname, callback: any)
+use(path: Uri, router: Router)
ExpressServer
1
1
ExpressValidator
ExpressValidationChain
1
1
1
1
ValidatorExpressAdapter
+check(field: Name, message: String): ValidationChain
+checkSchema(schema: any): any
ValidationChainExpressAdapter
+exists(): ValidationChain
+notEmpty(): ValidationChain
«interface»
Validation
+check(field: Name, message: String): ValidationChain
+checkSchema(schema: any): any
«interface»
ValidationChain
+exists(): ValidationChain
+notEmpty(): ValidationChain
1
1
1
1
ValidationMidware
+validateEvent(request, response, next: any)
+validateEventSubscriber(request, response, next: any)
+validateUser(request, response, next: any)
+validateMicrocontroller(request, response, next: any)

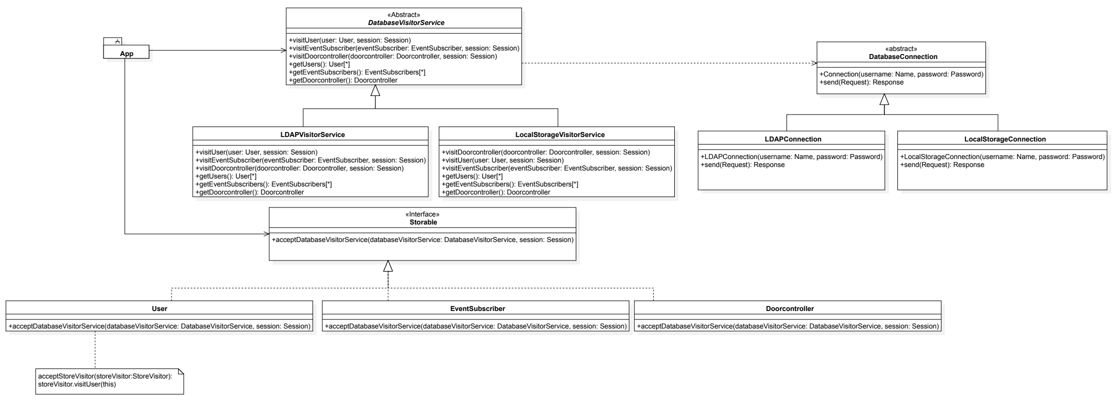
- **(Interface) Request**
  See App.
- **RequestExpressAdapter**
  RequestExpressAdapter adapts the Request from Express to the Request interface.
- **ExpressRequest**
  ExpressRequest is the Request class from Express.
- **(Interface) Response**
  See App.
- **ResponseExpressAdapter**
  ResponseExpressAdapter adapts the Response from Express to the Response interface.
- **ExpressResponse** ExpressResponse is the Response class from Express.
- **(Interface) Router**
  See App.
- **RouterExpressAdapter**
  RouterExpressAdapter adapts the Router from Express to the Router interface.
- **ExpressRouter**
  ExpressRouter is the router from Express.
- **(Interface) Server**
  See App.
- **ServerExpressAdapter**
  ServerExpressAdatper adapts the Server from Express to the Server interface.
- **ExpressServer**
  ExpressServer is the Server from Express.
- **Validationmidware**
  See App.
- **(Interface) Validation**
  See App.
- **(Interface) ValidationChain**
  See App.
- **ValidationChainExpressAdapter**
  ValidationChainExpressAdapter adapts the validation chain object from express-validator to our project.

- **ExpressValidator**
  ExpressValidator is the Validator from express-validator.

- **ExpressValidationChain**
  ExpressValidatonChain is the ValidationChain from express-validator.

- **ValidatorExpressAdapter**
  ValidatorExpressAdapter adapts the validation object from express-validator to our project.

DatabaseVisitorService


App
«Abstract»
DatabaseVisitorService
+visitUser(user: User, session: Session)
+visitEventSubscriber(eventSubscriber: EventSubscriber, session: Session)
+visitDoorcontroller(doorcontroller: Doorcontroller, session: Session)
+getUsers(): User[*]
+getEventSubscribers(): EventSubscribers[*]
+getDoorcontroller(): Doorcontroller
«abstract»
DatabaseConnection
+Connection(username: Name, password: Password)
+send(Request): Response
LDAPVisitorService
+visitUser(user: User, session: Session)
+visitEventSubscriber(eventSubscriber: EventSubscriber, session: Session)
+visitDoorcontroller(doorcontroller: Doorcontroller, session: Session)
+getUsers(): User[*]
+getEventSubscribers(): EventSubscribers[*]
+getDoorcontroller(): Doorcontroller
LocalStorageVisitorService
+visitDoorcontroller(doorcontroller: Doorcontroller, session: Session)
+visitUser(user: User, session: Session)
+visitEventSubscriber(eventSubscriber: EventSubscriber, session: Session)
+getUsers(): User[*]
+getEventSubscribers(): EventSubscribers[*]
+getDoorcontroller(): Doorcontroller
LDAPConnection
+LDAPConnection(username: Name, password: Password)
+send(Request): Response
LocalStorageConnection
+LocalStorageConnection(username: Name, password: Password)
+send(Request): Response
«Interface»
Storable
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)
User
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)
EventSubscriber
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)
Doorcontroller
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService, session: Session)
acceptStoreVisitor(storeVisitor:StoreVisitor):
storeVisitor.visitUser(this)

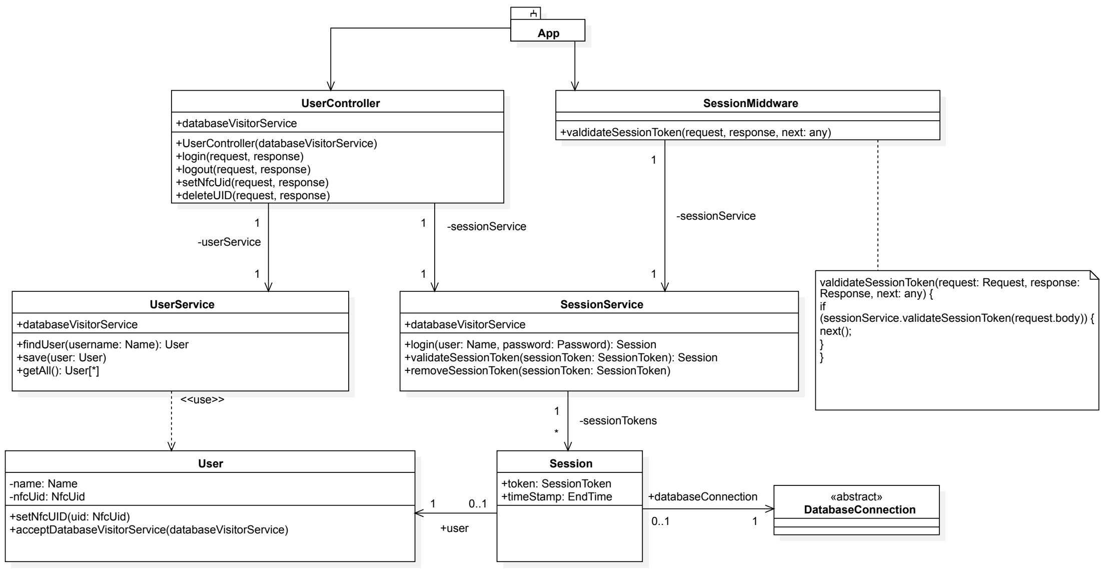
- **DatabaseVisitorService**
  DatabaseVisitorService is a abstract class to visits and stores implementations of the storable interface. Subclasses realize different database types.

- **LDAPVisitorService**
  LDAPVisitorService is a generalisation of the DatabaseVisitorService. It uses the LDAP as database.

- **LocalVisitorService**
  LocalVisitorService is a generalisation of the DatabaseVisitorService. It uses a local database for testing reasons.

- **(Interface)Storable**
  Storable is an interface for storable classes, that can accept DatabaseVisitorService.

User


App
UserController
+databaseVisitorService
+UserController(databaseVisitorService)
+login(request, response)
+logout(request, response)
+setNfcUid(request, response)
+deleteUID(request, response)
SessionMiddware
+valdidateSessionToken(request, response, next: any)
1
-userService
1
1
-sessionService
1
1
-sessionService
1
UserService
+databaseVisitorService
+findUser(username: Name): User
+save(user: User)
+getAll(): User[*]
SessionService
+databaseVisitorService
+login(user: Name, password: Password): Session
+validateSessionToken(sessionToken: SessionToken): Session
+removeSessionToken(sessionToken: SessionToken)
<<use>>
1
*
-sessionTokens
User
-name: Name
-nfcUid: NfcUid
+setNfcUID(uid: NfcUid)
+acceptDatabaseVisitorService(databaseVisitorService)
1
0..1
+user
Session
+token: SessionToken
+timeStamp: EndTime
+databaseConnection
0..1
1
«abstract»
DatabaseConnection

```
valdidateSessionToken(request: Request, response:
Response, next: any) {
if
(sessionService.validateSessionToken(request.body)) {
next();
}
}
```

- **UserController**
  UserController controls the User. The class controls how the user is logged in and out, how the uid is set and deleted.

- **Sessionmidware**
  Sessionmidware is the midware to validate the session token of an authenticated user.

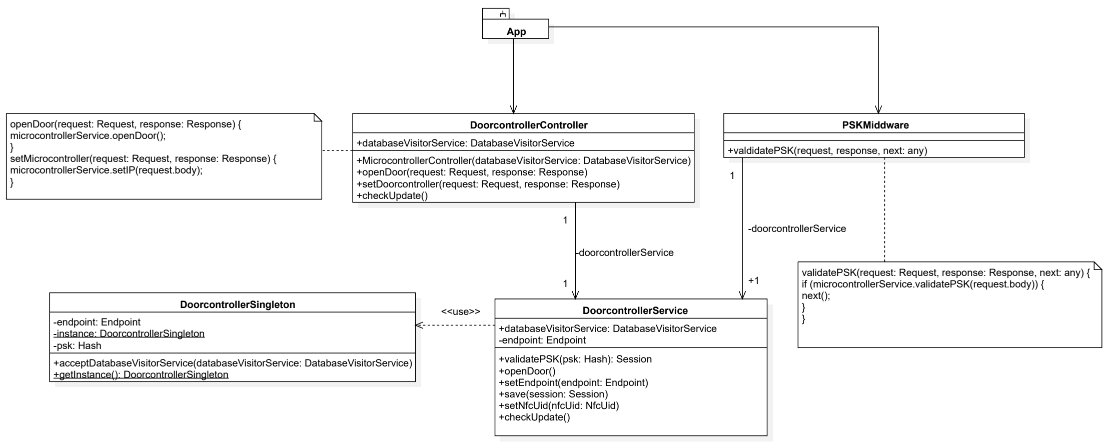
- **UserService**
  UserService is the service for the user.

- **SessionService**
  SessionService is the service for the session.

- **User**
  User is the class model of a User. A user has a unique username and a session token if he is validated. The user can save a unique id (uid or Nfc-Token) what is supposed to be the Nfc-Token of the NFC-Tag. This data get stored in a databased by DatabaseVisitorService

- **SessionToken**
  SessionToken is the model class of the Session token. Every authenticated user gets a SessionToken to prove that he is authenticated. The creating time of that token is saved to validate the age of the token.

Doorcontroller



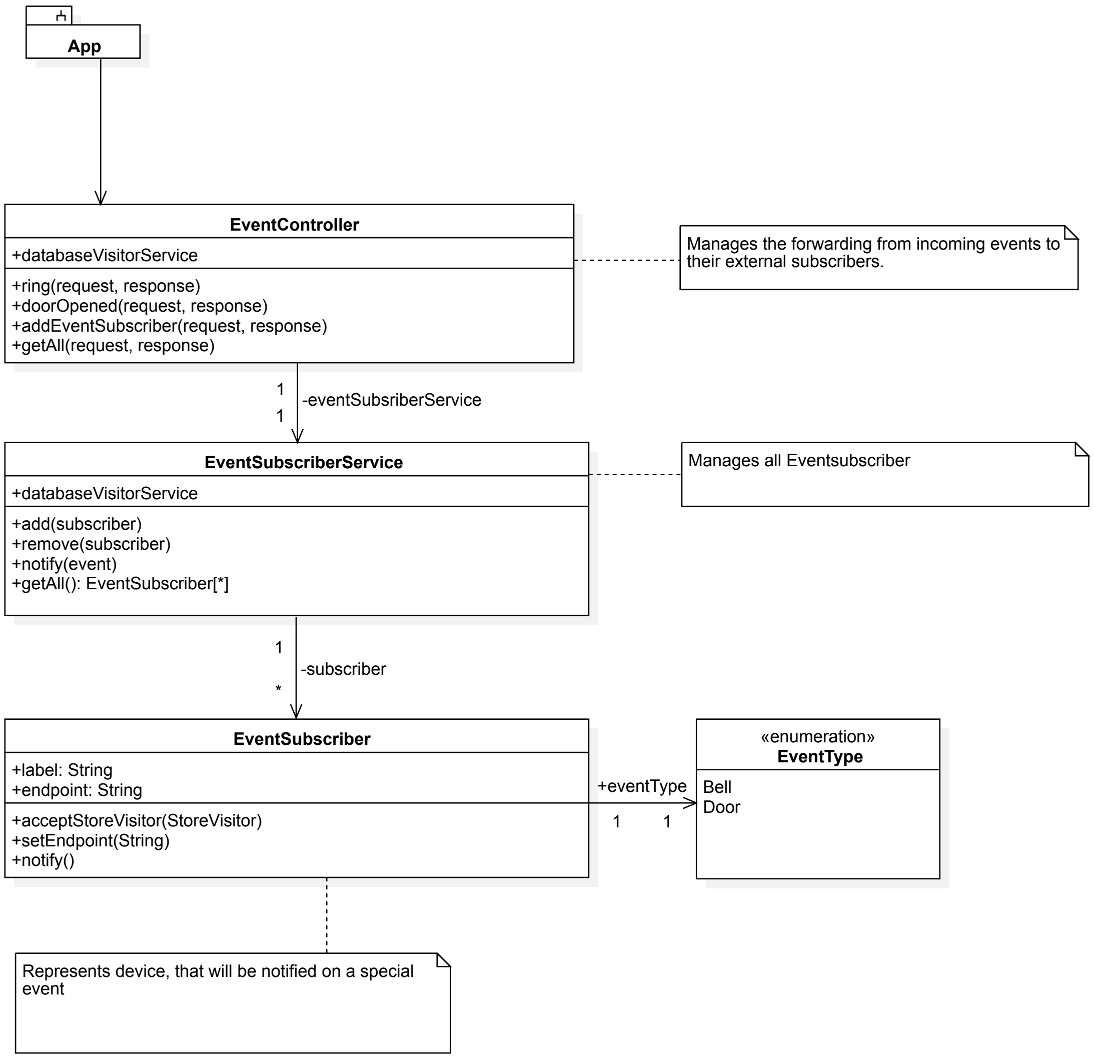
```
openDoor(request: Request, response: Response) {
microcontrollerService.openDoor();
}
setMicrocontroller(request: Request, response: Response) {
microcontrollerService.setIP(request.body);
}
```

```
validatePSK(request: Request, response: Response, next: any) {
if (microcontrollerService.validatePSK(request.body)) {
next();
}
}
```

App
DoorcontrollerController
+databaseVisitorService: DatabaseVisitorService
+MicrocontrollerController(databaseVisitorService: DatabaseVisitorService)
+openDoor(request: Request, response: Response)
+setDoorcontroller(request: Request, response: Response)
+checkUpdate()
PSKMiddware
+valdidatePSK(request, response, next: any)
1
-doorcontrollerService
1
1
-doorcontrollerService
+1
DoorcontrollerSingleton
-endpoint: Endpoint
-instance: DoorcontrollerSingleton
-psk: Hash
+acceptDatabaseVisitorService(databaseVisitorService: DatabaseVisitorService)
+getInstance(): DoorcontrollerSingleton
<<use>>
DoorcontrollerService
+databaseVisitorService: DatabaseVisitorService
-endpoint: Endpoint
+validatePSK(psk: Hash): Session
+openDoor()
+setEndpoint(endpoint: Endpoint)
+save(session: Session)
+setNfcUid(nfcUid: NfcUid)
+checkUpdate()

- **DoorcontrollerController**
  DoorcontrollerController controls the Doorcontroller. It controls how the door is opened and the endpoint of the doorcontroller is set.
- **PSKmidware**
  PSKmidware is the midware to validate the psk of the microcontroller.
- **DoorcontrollerService**
  DoorcontrollerService is the service for the doorcontroller.
- **DoorcontrollerSingleton**
  Doorcontroller is the model class of the doorcontroller and a singleton. Only one doorcontroller for the door control is allowed per system. The doorcontroller has an ip and a pre shared key (psk) for registration.

EventController

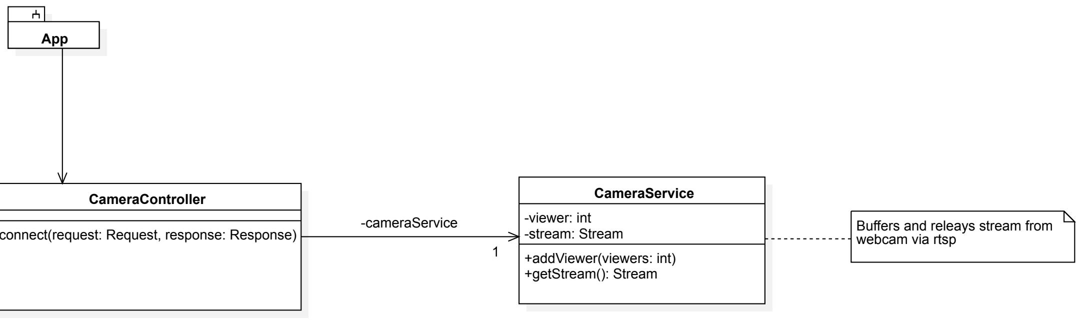


App
EventController
+databaseVisitorService
+ring(request, response)
+doorOpened(request, response)
+addEventSubscriber(request, response)
+getAll(request, response)
Manages the forwarding from incoming events to their external subscribers.
1
1
-eventSubsriberService
EventSubscriberService
+databaseVisitorService
+add(subscriber)
+remove(subscriber)
+notify(event)
+getAll(): EventSubscriber[*]
Manages all Eventsubscriber
1
*
-subscriber
EventSubscriber
+label: String
+endpoint: String
+acceptStoreVisitor(StoreVisitor)
+setEndpoint(String)
+notify()
+eventType
1
1
«enumeration»
EventType
Bell
Door
Represents device, that will be notified on a special event

- **EventController**
  EventController controls the EventSubscribers. The class controls how the ring and doorOpen event is sent and also how EventSubscribers are added and read.
- **EventSubscriberService**
  EventSubscriberService is the service for our EventSubscribers.
- **EventSubscriber**
  EventSubscriber is the model class of an Event Subscriber. It represents a EventSubscriber with an endpoint to send messages to and a label for better reading by humans. This data get stored in a database by the DatabaseVisitorService

- **(Enumeration) Event**
  Enumaration of events, which notifies their subscriber.

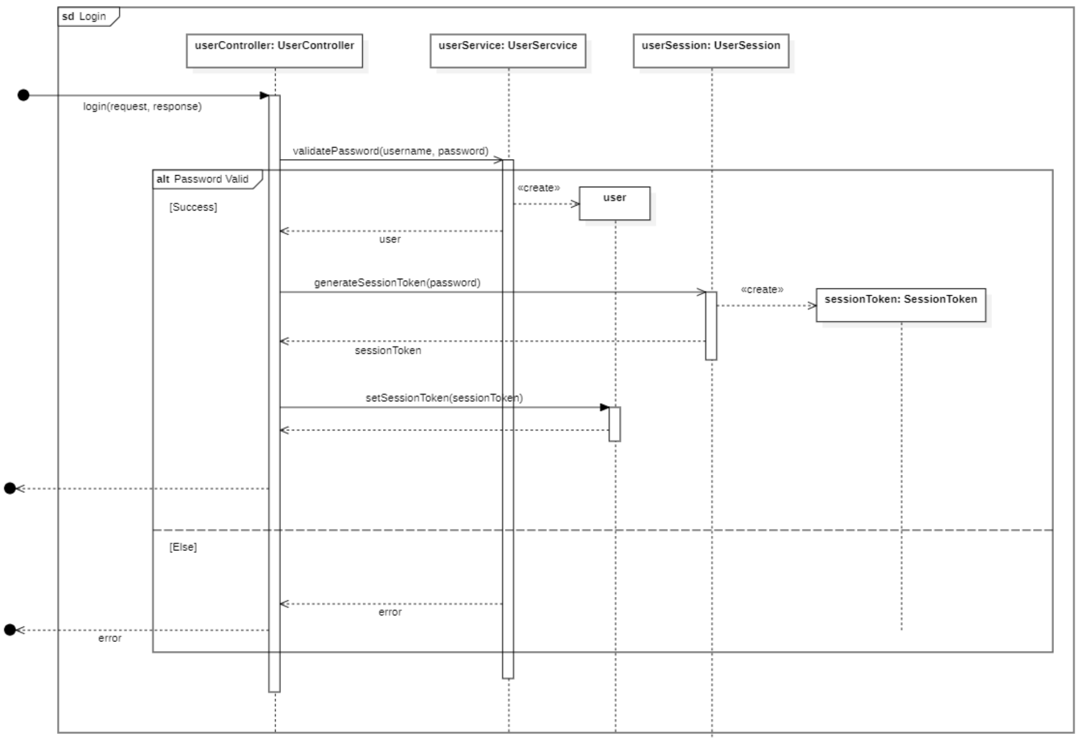
CameraController



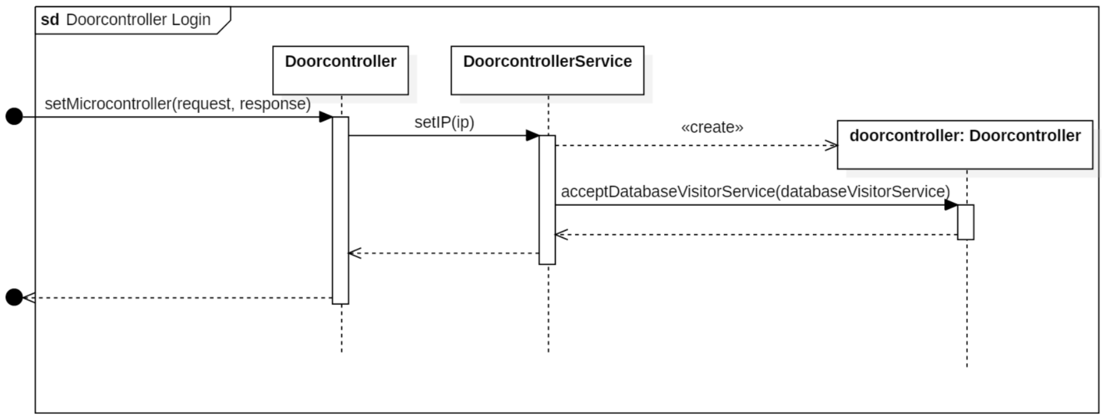
- **CameraController**
  CameraController controls the camera. A user can get the camera stream through the connect method.
- **CameraService**
  CameraService is the service of our camera. It handles the viewers and sent them the camera stream.

2.4.4 Sequency Diagrams

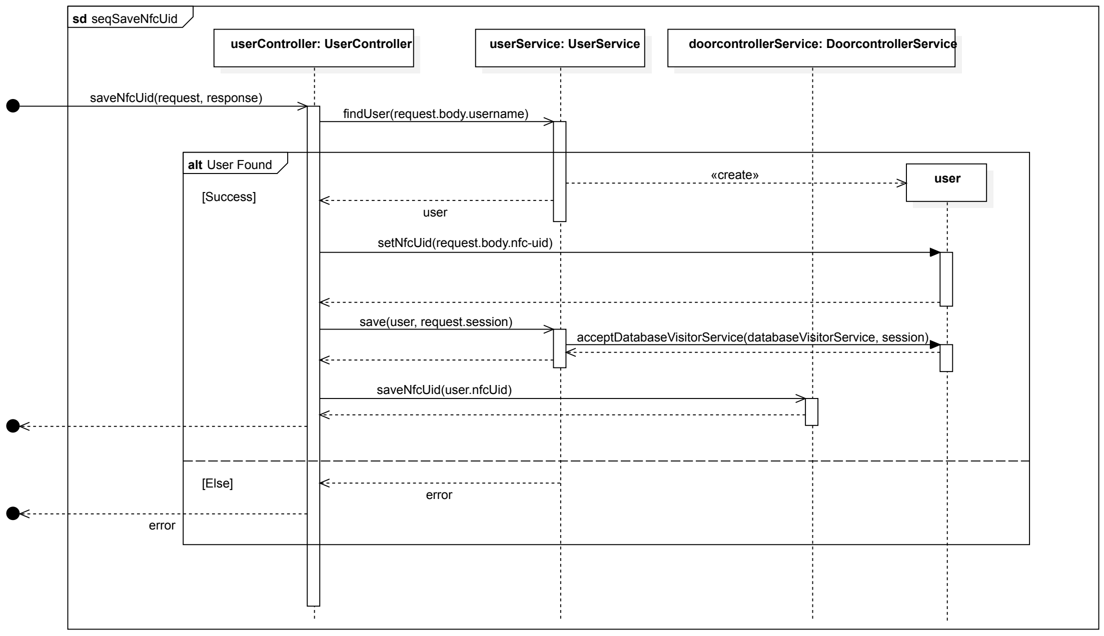
Login


sd Login
userController: UserController
userService: UserSercvice
userSession: UserSession
login(request, response)
validatePassword(username, password)
alt Password Valid
[Success]
«create»
user
user
generateSessionToken(password)
«create»
sessionToken: SessionToken
sessionToken
setSessionToken(sessionToken)
[Else]
error
error

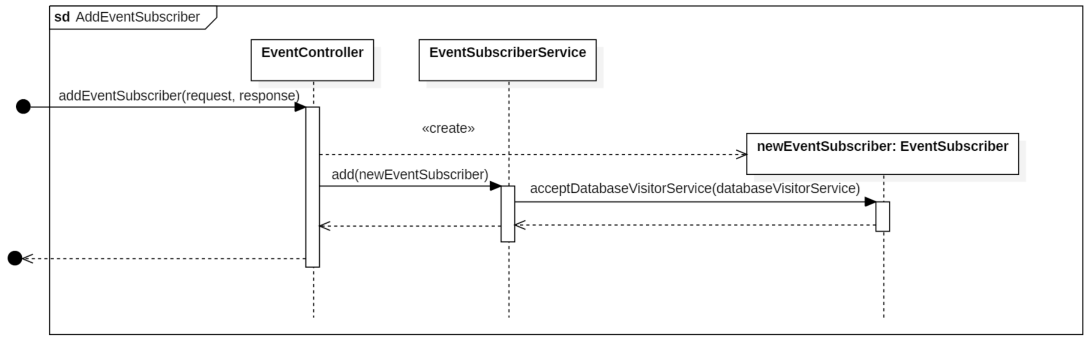
Doorcontroller Login


sd Doorcontroller Login
Doorcontroller
DoorcontrollerService
setMicrocontroller(request, response)
setIP(ip)
«create»
doorcontroller: Doorcontroller
acceptDatabaseVisitorService(databaseVisitorService)

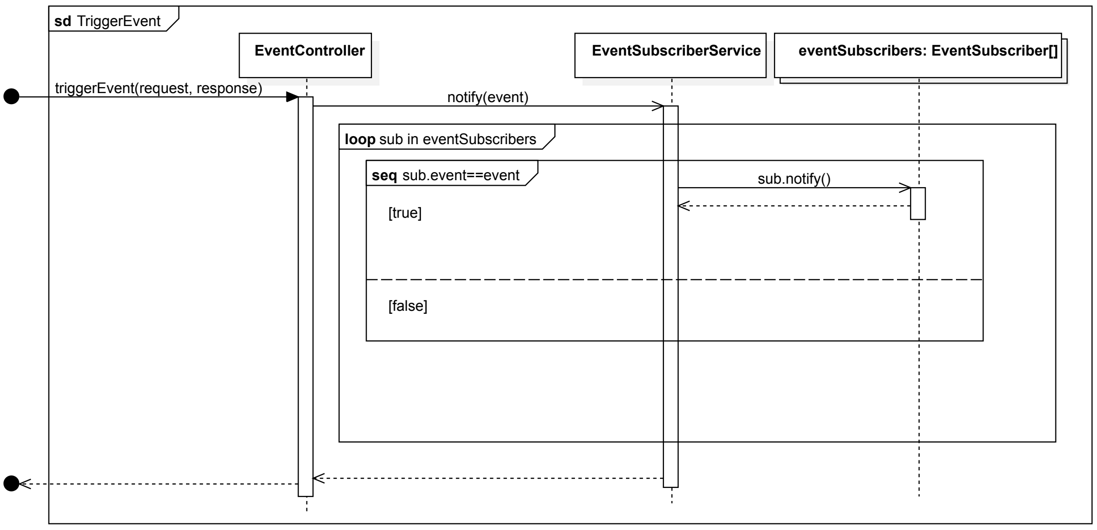
Save UID


sd seqSaveNfcUid
userController: UserController
userService: UserService
doorcontrollerService: DoorcontrollerService
saveNfcUid(request, response)
findUser(request.body.username)
alt User Found
[Success]
«create»
user
user
setNfcUid(request.body.nfc-uid)
save(user, request.session)
acceptDatabaseVisitorService(databaseVisitorService, session)
saveNfcUid(user.nfcUid)
[Else]
error
error

Add EventSubscriber

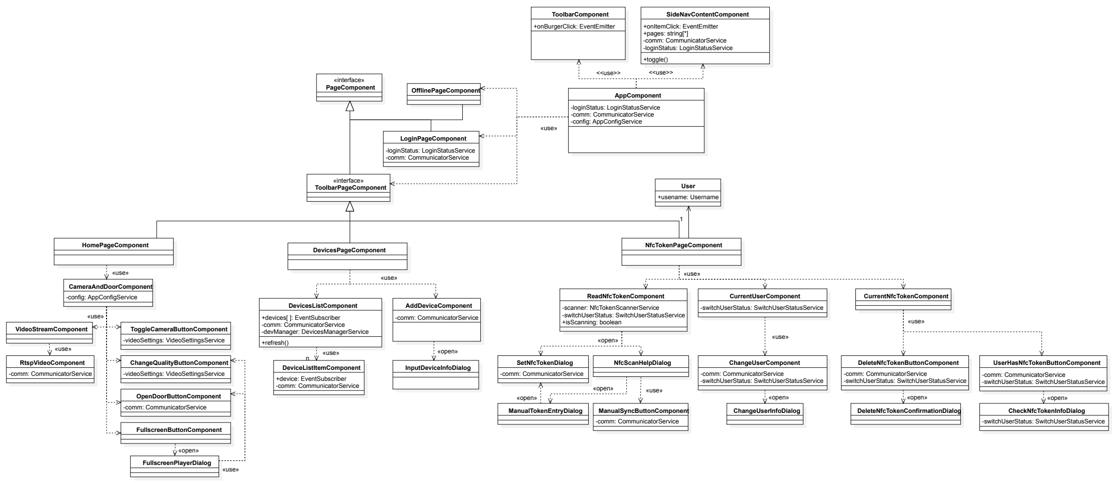

sd AddEventSubscriber
EventController
EventSubscriberService
addEventSubscriber(request, response)
«create»
newEventSubscriber: EventSubscriber
add(newEventSubscriber)
acceptDatabaseVisitorService(databaseVisitorService)

TriggerEvent

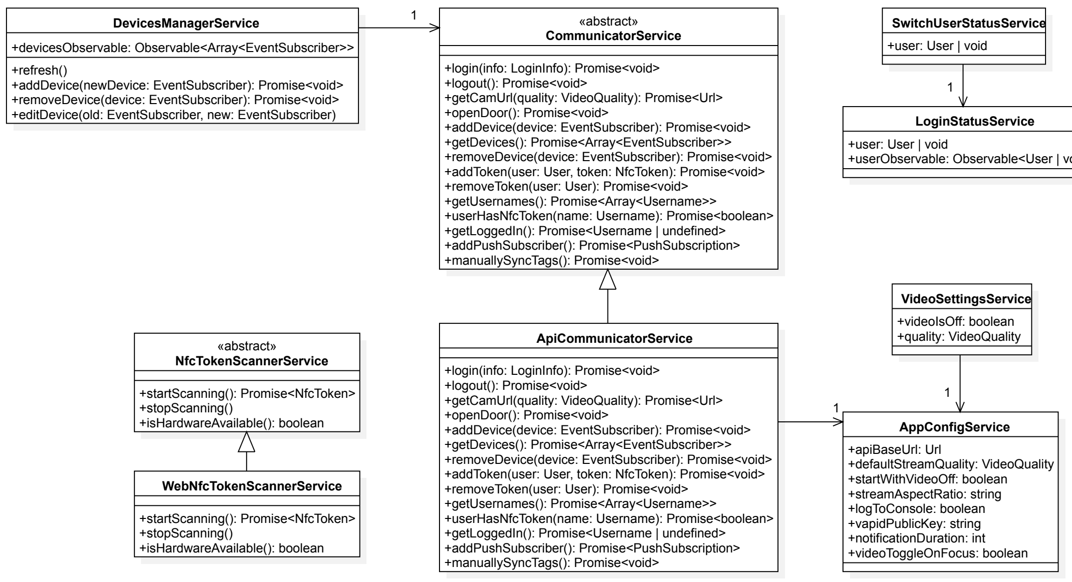


2.5 Progressive Web-App

2.5.1 Class Diagrams



ToolbarComponent
+onBurgerClick: EventEmitter
SideNavContentComponent
+onItemClick: EventEmitter
+pages: string[*]
-comm: CommunicatorService
-loginStatus: LoginStatusService
+toggle()
<<use>>
<<use>>
«interface» PageComponent
OfflinePageComponent
AppComponent
-loginStatus: LoginStatusService
-comm: CommunicatorService
-config: AppConfigService
«use»
LoginPageComponent
-loginStatus: LoginStatusService
-comm: CommunicatorService
«interface» ToolbarPageComponent
User
+usename: Username
1
HomePageComponent
DevicesPageComponent
NfcTokenPageComponent
«use»
CameraAndDoorComponent
-config: AppConfigService
«use»
VideoStreamComponent
«use»
RtspVideoComponent
-comm: CommunicatorService
ToggleCameraButtonComponent
-videoSettings: VideoSettingsService
ChangeQualityButtonComponent
-videoSettings: VideoSettingsService
OpenDoorButtonComponent
-comm: CommunicatorService
FullscreenButtonComponent
«open»
FullscreenPlayerDialog
«use»
«use»
DevicesListComponent
+devices[ ]: EventSubscriber
-comm: CommunicatorService
-devManager: DevicesManagerService
+refresh()
n
«use»
DeviceListItemComponent
+device: EventSubscriber
-comm: CommunicatorService
AddDeviceComponent
-comm: CommunicatorService
«open»
InputDeviceInfoDialog
«use»
ReadNfcTokenComponent
-scanner: NfcTokenScannerService
-switchUserStatus: SwitchUserStatusService
+isScanning: boolean
«open»
SetNfcTokenDialog
-comm: CommunicatorService
NfcScanHelpDialog
«open»
«open»
«use»
ManualTokenEntryDialog
ManualSyncButtonComponent
-comm: CommunicatorService
CurrentUserComponent
-switchUserStatus: SwitchUserStatusService
«use»
ChangeUserComponent
-comm: CommunicatorService
-switchUserStatus: SwitchUserStatusService
«open»
ChangeUserInfoDialog
CurrentNfcTokenComponent
«use»
DeleteNfcTokenButtonComponent
-comm: CommunicatorService
-switchUserStatus: SwitchUserStatusService
«open»
DeleteNfcTokenConfirmationDialog
UserHasNfcTokenButtonComponent
-comm: CommunicatorService
-switchUserStatus: SwitchUserStatusService
«open»
CheckNfcTokenInfoDialog
-switchUserStatus: SwitchUserStatusService



DevicesManagerService
+devicesObservable: Observable<Array<EventSubscriber>>
+refresh()
+addDevice(newDevice: EventSubscriber): Promise<void>
+removeDevice(device: EventSubscriber): Promise<void>
+editDevice(old: EventSubscriber, new: EventSubscriber)
1
«abstract» CommunicatorService
+login(info: LoginInfo): Promise<void>
+logout(): Promise<void>
+getCamUrl(quality: VideoQuality): Promise<Url>
+openDoor(): Promise<void>
+addDevice(device: EventSubscriber): Promise<void>
+getDevices(): Promise<Array<EventSubscriber>>
+removeDevice(device: EventSubscriber): Promise<void>
+addToken(user: User, token: NfcToken): Promise<void>
+removeToken(user: User): Promise<void>
+getUsernames(): Promise<Array<Username>>
+userHasNfcToken(name: Username): Promise<boolean>
+getLoggedIn(): Promise<Username | undefined>
+addPushSubscriber(): Promise<PushSubscription>
+manuallySyncTags(): Promise<void>
SwitchUserStatusService
+user: User | void
1
LoginStatusService
+user: User | void
+userObservable: Observable<User | void>
ApiCommunicatorService
+login(info: LoginInfo): Promise<void>
+logout(): Promise<void>
+getCamUrl(quality: VideoQuality): Promise<Url>
+openDoor(): Promise<void>
+addDevice(device: EventSubscriber): Promise<void>
+getDevices(): Promise<Array<EventSubscriber>>
+removeDevice(device: EventSubscriber): Promise<void>
+addToken(user: User, token: NfcToken): Promise<void>
+removeToken(user: User): Promise<void>
+getUsernames(): Promise<Array<Username>>
+userHasNfcToken(name: Username): Promise<boolean>
+getLoggedIn(): Promise<Username | undefined>
+addPushSubscriber(): Promise<PushSubscription>
+manuallySyncTags(): Promise<void>
VideoSettingsService
+videoIsOff: boolean
+quality: VideoQuality
1
1
AppConfigService
+apiBaseUrl: Url
+defaultStreamQuality: VideoQuality
+startWithVideoOff: boolean
+streamAspectRatio: string
+logToConsole: boolean
+vapidPublicKey: string
+notificationDuration: int
+videoToggleOnFocus: boolean
«abstract» NfcTokenScannerService
+startScanning(): Promise<NfcToken>
+stopScanning()
+isHardwareAvailable(): boolean
WebNfcTokenScannerService
+startScanning(): Promise<NfcToken>
+stopScanning()
+isHardwareAvailable(): boolean

Documentation

- PageComponents
  - (Interface) PageComponent
    Represents an entire page that spans the size of the screen

- LoginPageComponent
  Represents the page containing buttons and textboxes allowing the user to log into the system

  onLogin.emit(user) is called when the used has entered their details and tapped the Log in button.
- (Interface) ToolbarPageComponent
  Represents a page that is to be displayed with a toolbar above it

  The title is what will be displayed in the side-navigation.
- HomePageComponent
  Represents the page on which the user can activate and view the camera, as well as open the door
  * OpenDoorButtonComponent
    Consists of only a button and sends a signal to the communicator to open the door when tapped
  * CameraAccessComponent
    Works as a container for the VideoStreamComponent, and as a button that activates the camera stream
  * (Interface) VideoStreamComponent
    Displays a video feed in a certain format
  * RtspVideoComponent
    Implements the display of a livestream in the RTSP format
- StatisticsPageComponent
  Represents the page displaying the graphed statistics
  * GraphListComponent
    Displays a vertically scrollable box containing a specific list of graphs
  * (Interface) Graph
    Represents a graph with its image
- DevicesPageComponent
  Represents the page used to view and manage notification devices. If the AddDevicesComponent onAdd emitter is emitted, devicesListComponent.refresh() is run.

  The DevicesPageComponent is not responsible for adding a device when onAdd is signaled.
  * DevicesListComponent
    Displays an itemized list of currently added ListedDeviceComponents, gotten from the communicator. If the ListedDeviceComponents onDelete event emitter emits, the list of devices is refreshed.
  * ListedDeviceComponent
    An element in the DevicesListComponent assigned to one device. Displays

the devices label, url as well as a button to remove its subscription using the communicator. onDelete.emit() ist called when the delete button has been tapped.

    * Device
      Represents a notification device with a label, a string and the execution event to trigger the notification.
    * (Enum) NotificationType
      Notification types are distinguished by the event needed for the notification to be delivered.
        · DoorBell signals a notification that is sent when the Doorbell has been pressed.
        · DoorOpened signals a notification that is sent when the door has been opened.
    * addDeviceComponent
      Contains certain components to allow the user to add a notification device to the system. When the Add Device button is tapped, the device is added and onAdd.emit() is run.

- UidPageComponent
  Represents the page used by a user to manage their uid, or an admin to manage another users uid.

  The currently managed user is saved here, and can be updated by the onEnter emitter in ChangeUserComponent.

  If the AddUidComponent onAdd signal is given, run currentUidTextComponent.refresh(). UidPageComponent is not responsible for adding the UID to the user.

    * ReadUidComponent
      contains an image to signal that a UID can now be scanned. This component, when visible, constantly checks if a UID can be scanned.
    * (Interface) UidScannerService
      Can scan for, and return a UID. Child classes will implement a specific type of scan.
    * NfcUidScannerService
      Uses the current device to scan for an NFC chip, returns the NFC tag when scan() is called.
    * ChangeUserComponent
      Contains components allowing admins to change the user whose UID they are managing. onEnter.emit(username) is called when the admin inputs the username and taps the Change User button.
    * AddUidComponent
      If a UID has been read, it is displayed here, and can be saved to the

    current user when the Add UID button is tapped. When that happens, onAdd.emit() is run, to notify other components to refresh.

    * DeleteUidComponent
      Contains components to allow the user to see and delete the current users UID.

      onDelete.emit() is called when the onDelete emitter in DeleteUidButtonComponent is signaled.

      DeleteUidComponent is not responsible for deleting any UIDs. That is done in DeleteUidButtonComponent

    * DeleteUidButtonComponent
      Contains only a button. If it is pressed, the current users UID is deleted, and onDelete.emit() is run.

    * CurrentUidTextComponent
      Gets the current users UID from the Communicator on init() and refresh(), and displays it.

- Other Components
  - AppComponent
    The component on the highest level, displaying a toolbar, a side-navigation, and all pages.

    The AppComponent has 0 or 1 logged in user.

    The AppComponent is responsible for switching the current page, when receiving a signal from SideNavComponents onItemClick or LoginPageComponents onLogin.

  - User
    Users consist of a username, a UID and a token. The user's password is not saved.

    Users are used to save who is logged into the app, and to save whose UID is being edited by an admin in the UidPageComponent

  - ToolbarComponent
    Displays a toolbar with a "burger-menu" icon-button on the left, and an optional text on the right. onBurgerClick.emit() is run when the "burger-menu" icon-button is tapped. This should send a signal to the AppComponent to open the side-navigation.

  - SideNavComponent
    The side-navigation is the overlaying menu on the left-hand side. It will be opened by the AppComponent, when it receives the onBurgerClick signal from the ToolbarComponent.

    The items displayed are the titles of ToolbarPageComponents in the pages list.

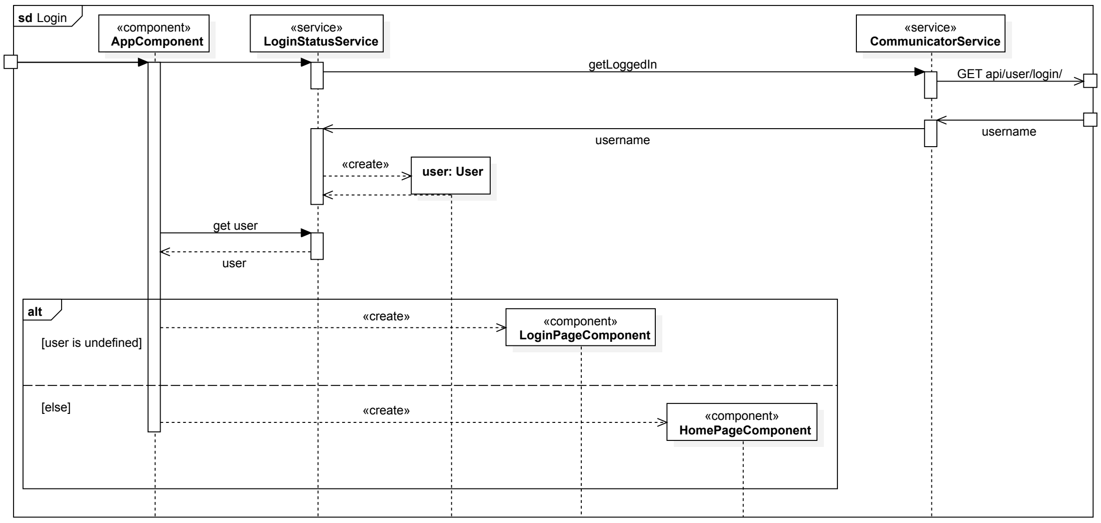
toggle() can be used to toggle the SideNav open and closed.

onItemClick(toolbarPageComponent) is called when the user clicks one of the items. The ToolbarPageComponent associated with the item will be sent as an argument, to inform the AppComponent which Page is to be displayed.

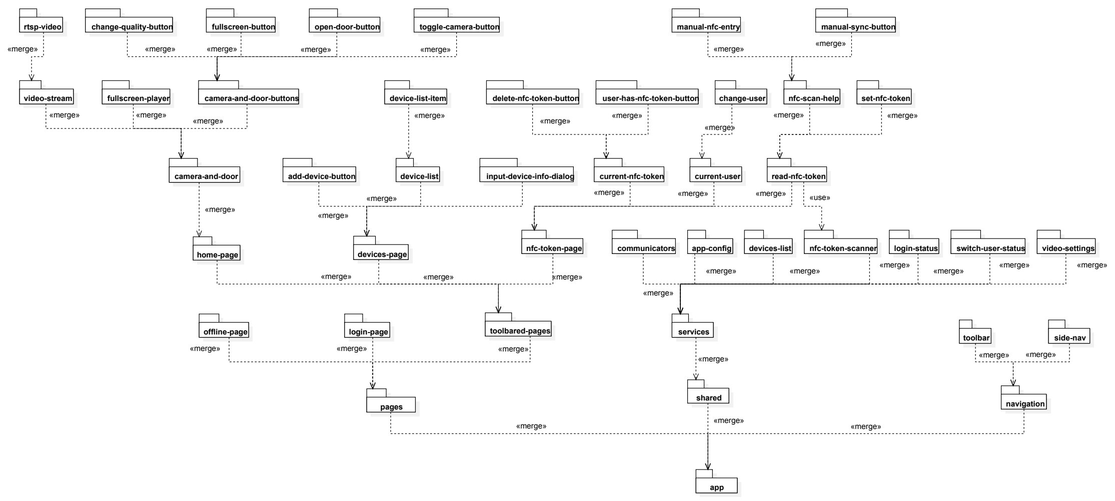
- Communicators
  - (Interface) Communicator
    An interface for the entire PWA to interact with the midware.
  - ApiCommunicatorService
    Implements Communicator methods by converting the arguments to json and sending them in requests to the REST Api. Json responses are usually unpacked and returned.

2.5.2 Sequence Diagrams

- Detect Login



2.5.3 Package Diagrams


rtsp-video
change-quality-button
fullscreen-button
open-door-button
toggle-camera-button
manual-nfc-entry
manual-sync-button
video-stream
fullscreen-player
camera-and-door-buttons
device-list-item
delete-nfc-token-button
user-has-nfc-token-button
change-user
nfc-scan-help
set-nfc-token
camera-and-door
add-device-button
device-list
input-device-info-dialog
current-nfc-token
current-user
read-nfc-token
home-page
devices-page
nfc-token-page
communicators
app-config
devices-list
nfc-token-scanner
login-status
switch-user-status
video-settings
offline-page
login-page
toolbared-pages
services
toolbar
side-nav
pages
shared
navigation
app
«merge»
«use»