# elliot

**ellIot** ist eine Kombination aus **App, Server** und **Internet-of-Things Geräten**, welche es dem Nutzer erlaubt sein Heim bequem fernzusteuern. Neben der leicht filterbaren Liste, welche alle angemeldeten IoT Geräte anzeigt, bietet die ellIoT App eine besonderheit. Über die **Scan-Funktion** braucht der Nutzer nur vorher eingescannte IoT Geräte in den Bildsucher der Camera holen und nach einem kurzen Scanprozess wird automatisch das passende Interface angezeigt.



**MODUL**
Interaktionstechnologien

**BETREUER**
Dipl.-Inf. (FH) Björn Plutka

**SEMESTER**
Sommersemester 2015


**AUTOR**
Dennis Timmermann - 424575

## KONZEPT

ellIoT dient als Prototyp um eine neuartige Steuerung für Internet of Things (IoT) Geräte zu evaluieren. Das Angebot an “Smart” Geräten nimmt immer weiter zu, womit auch die dazughörigen Apps immer mehr werden.

Hier soll ellIoT helfen den Weg zwischen der Intention ein Gerät steuern zu wollen und der eigentlichen Aktion zu verkürzen. Anstatt die passende App manuel aus einer langen Liste zu suchen automatisiert ellIoT diesen Prozess.

Das Smartphone wird lediglich in die Richtung des zu steuernden Gerätes gehalten, wodurch die Kamera dieses erkennt und das passende Interface heraussucht und einblendet.

## ARCHITEKTUR

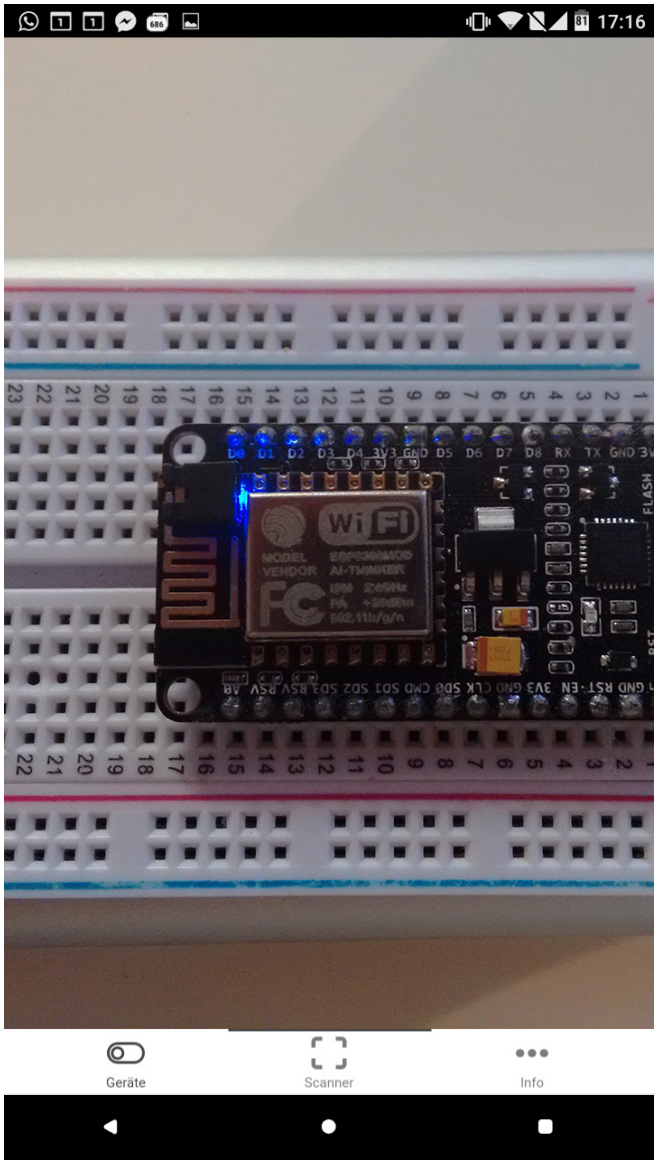
ellIoT besteht im Grunde genommen aus drei Komponenten, der App, einem zentralen Server und den IoT Geräten, im folgenen Nodes genannt.

Alle Informationen die für die steuerung gebraucht werden, werden vom Server bereitgestellt. Die Nodes melden sich lediglich mit einer Produkt-ID (EPID) und einer Einzigartigen-ID (UUID) beim Server an. Dieser ordnet der EPID dann ein vom theoretischen Hersteller bereitgestelltes Produktprofil zu. Meldet sich nun eine App beim Server an, bekommt diese die Profile der Nodes zur verfügung gestellt und generiert ein Interface.

## APPLIKATION

Die Smartphone-App wurde in Teilen mit Webtechniken umgesetzt. Aufbauen auf Cordova und Ionic wurde ein leicht bedienbares Interface geschaffen. Dieses ist in zwei Views unterteilt.

### SCAN-VIEW




Diese Ansicht stellt den Kern der Idee hinter ellIoT dar. Dem Benutzer wird ein Kamerabild geliefert, mit

dem er nach Nodes suchen kann die er steuern möchte. Es wird kontinuierlich gescannt, womit keine weitere Aktion des Nutzers erforderlich ist. Wird eine Node erkannt, wird automatisch zum dazugehörigen Interface gewechselt.

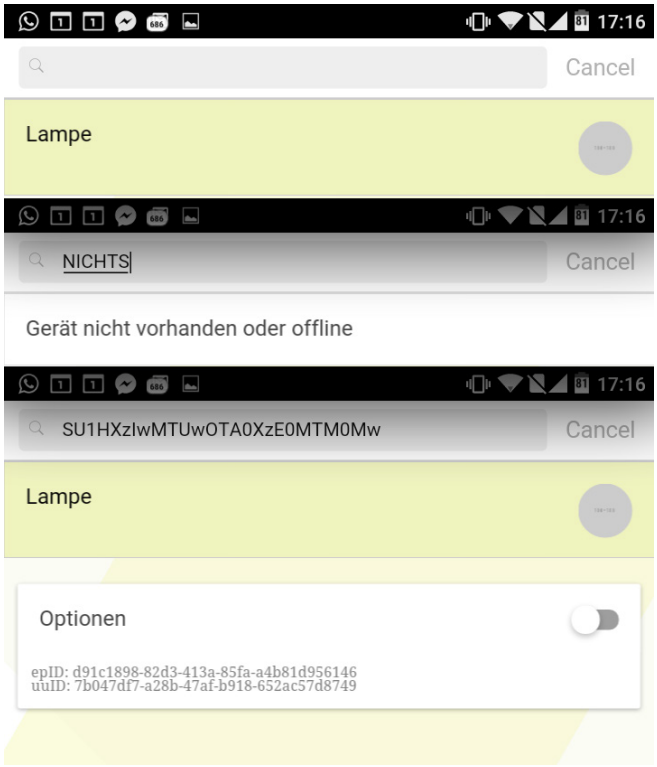
## SCAN-CONTROLLER

Wenn in die Scan-View gewechselt wird aktiviert der dazugehörige Controller automatisch den Scanprozess. Dieser wird über das native Bilderkennungs-SDK von Moodstocks in Verbindung mit einem Drittanbieter Cordova Plugin realisiert.

Beim Start der App lädt sich die App im Moodstocks Developer Account hinterlegte Vergleichsbilder herunter um diese dann kontinuierlich mit dem Kamerabild zu vergleichen. Wird eine übereinstimmung festgestellt, wird diese an den List-Controller weitergegeben.

## LIST-VIEW

Die listen Ansicht ist eine dauerhaft aktuell gehaltene Liste mit allen am Server angemeldeten und aktiven Nodes. Kommt eine Node online, ändert ihre Sensorenwerte oder geht offline wird dieses unmittelbar an die App übermittelt.
Der Nutzer hat hier die Möglichkeit sich alle Nodes in einer Liste anzuschauen oder per Suchleiste nach bestimmten zu suchen. Den Nodes können hierbei beliebige Tags zugeordnet werden, welche von der Suche berücksichtigt werden.




Wenn eine Node über die Scan Funktion erkannt wurde, wird der Suchfilter mit den dem Vergleichsbild hinterlegten Informationen gefüllt. Die Trennung von der UUID ermöglicht es hier nicht nur Nodes, sondern auch “Kombinationen” festzulegen. So könnte beispielsweise bei einem Scan von der Haustür automatisch die Interfaceelemente für die Türverriegelung und das Licht im Flur eingeblendet werden. Ebenfalls können so mehrere Vergleichsbilder für nur eine Node hinterlegt werden.

## List-Controller

Der List-Controller ist für den Aufbau der Nodeliste verantwortlich. Über die socketFacory wird das List-Model dauerhaft aktuell gehalten.

### LIST-MODEL

Bei jeder Änderung des List-Models durch den List-Controler wird umgehen die List-View aktualisiert. AngularJS bietet hier eine ausgezeichnete MVC-Trennung. Änderungen an Variablen werden automatisch an alle Konsumenten weitergegeben.

### SOCKETFACTORY

Hierbei handelt es sich um einen Angular-Service, der allen Controllern zur Verfügung gestellt werden kann. Dies erlaubt die Trennung von List-Controller und der eigentlichen Kommunikation mit dem Server. Der List-Controller bedient sich der Serververbindung um auf Änderungen an Nodes reagieren zu können und das List-Model entsprechend anktualisiern zu können.

## SERVER

Der relativ schlanke Server ist der Hauptverwaltungspunkt hinter ellIot. Nodes können sich hier registrieren um dann von der App gesteru zu werden.

### NODE-SERVER KOMMUNIKATIN

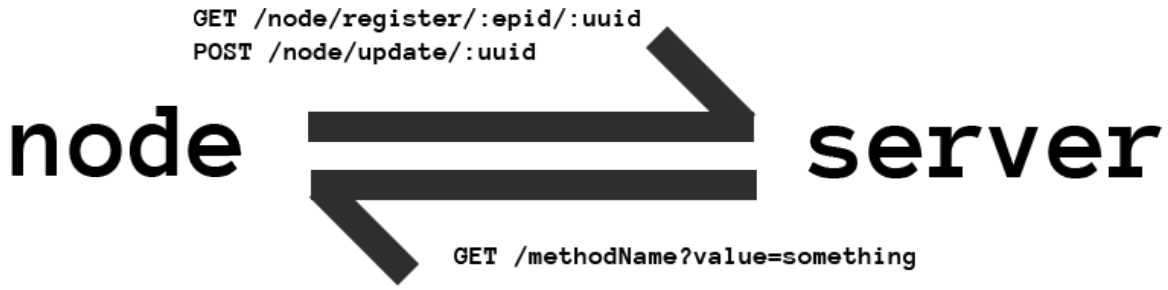
Die Kommunikation zwischen Nodes und Server erfolgt über REST, also einen simplen Vertrag von HTTP-Get und -Post befehlen. Der Server stellt den Nodes  Endpunkte zur registrierung und zum Update des eigenen Statuses bereit. Umgekehrt kann auch der Server per HTTP-Get auf die Nodes zugreifen.

### App-Server Kommunikation

Die App ist per Socket.IO mit dem Server verbunden.  Eingaben im Interface können so in Echtzeit an den Server weitergegeben werden und auch Updates der Nodes werden umgehend an die App weitegeleitet.

### METHODEN

Um sowohl die Kommunikation, als auch die Implementierung von ellIoT auf seiten der App möglichst simpel zu halten, wurde auf Steuerungslogik in der App verzichtet. Diese ist lediglich dafür verantwortlich das Interface zu generieren und die Informationen zwischen App und Server synchron zu halten.



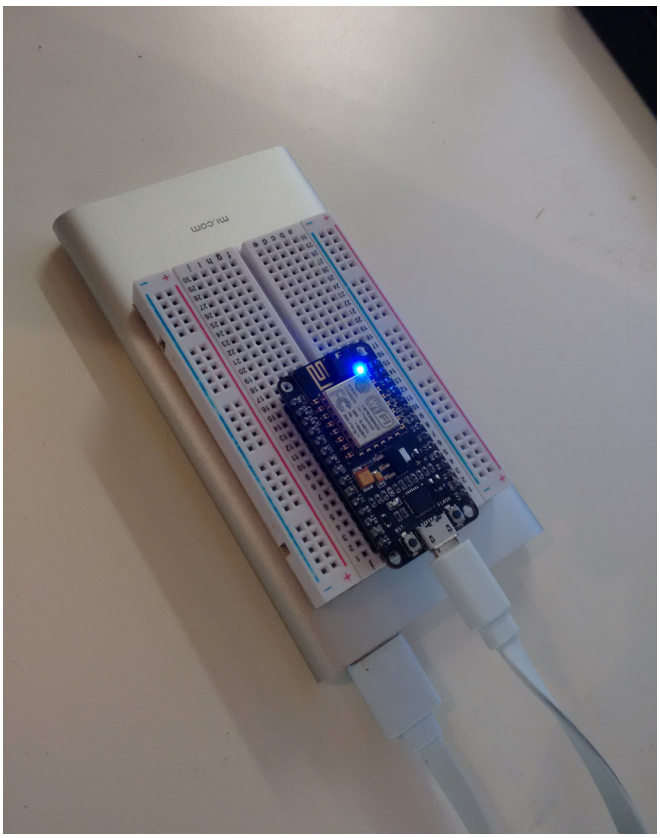

Die eigentliche Steuerungslogik ist in den Produktprofilen hinterlegt. Wird eine Änderung einer Variablen durch die App festgestellt, wird im Profil nachgeschaut, ob dies mit einer Aktion verbunden ist und gegebenenfalls ausgeführt. Diese Aktionen bestehen aus Programmlogik als auch URIs für HTTP-Endpunkte auf den Nodes, welche dann aufgerufen und ihre dazugehörige Funktion auf den Nodes ausgeführt werden.

Diese Architektur bietet den Vorteil, dass Hersteller von IoT-Geräten nur die Profile aktualisieren müssen, wenn kleinere Änderungen am Verhalten vorgenommen werden sollen, nicht das eigentliche IoT-Gerät. Auch ist die Implementierung so weitgehend unabhängig von der eigentlichen App.

Umgesetzt wurde der Server mit NodeJS unter zuhilfenahmen von Express und Socket.IO

## IOT-NODE

Beim Start verbinded sich die Node automatisch mit dem festgelegten Netzwerk, sucht nach bestehenden ellIoT-Server und registriert sich bei diesem. Hierbei wird lediglich die EPID und UUID übertragen, der eigentliche Funktionsumfang der Node ist als Profil beim Server hinterlegt, der dieses theoretisch direkt vom Hersteller beziehen könnte. In regelmäßigen Zuständen oder bei Aktionen am gerät wird der neue Zustand an den Server übertragen der diese dann an die App weitergibt.



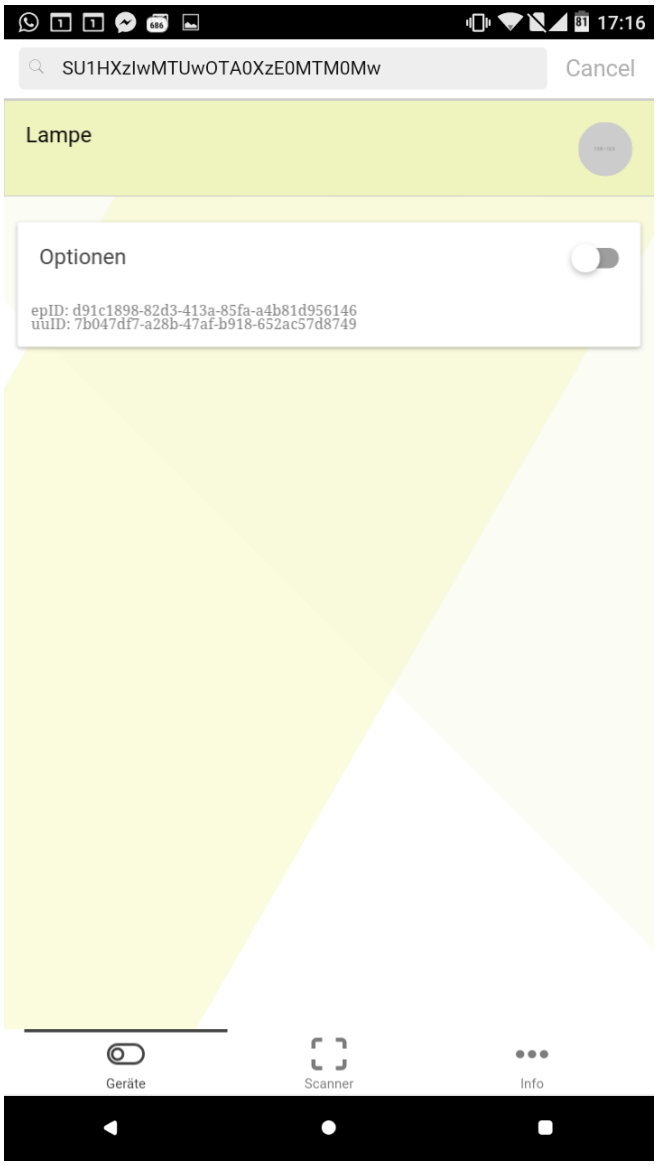
### ERWEITERBARKEIT

Neben den bereits angesprochenen Vorteilen der Profile. namentlich der leichten Aktualisierbarkeit und der auslagerung der Steuerungslogik auf den Server bieten diese sich auch für den Produktvertrieb an. Anstatt bei Auslieferung eines neuen Produkts eine Smartphone-App aktualisiern zu müssen kann der Server sich einfach ein neues Profil vom Herstellerserver laden. Aus diesem wird dann automatisch ein Interface in der App generiert ohne diese Anpassen zu müssen. Sollte der Funktionsumfang der App doch einmal nicht reichen, können sehr einfach neue Interfacelemente Implementiert werden. Diese haben zum einen die Form eines Eintrages ins Controls Feld im Produktprofil, als auch eine simple Erweiterung des devices.html Pratials in der App. Mit wenig Aufwand kann ellIoT so um neue Elemente erweitert werden.

Durch die Trennung von EPID und UUID können mehrere Nodes des gleichen Typs unabhängig voneinander gesteuert werden, diese teilen sich lediglich das Profil, nicht die Methodenaufrufe.

## PROBLEME

Aufgrund der Wahl des Moodstocks-SDKs für ellIoT musste auf ein Drittanbieter Cordova Plugin zurückgegriffen werden. Dieses setzte leider die Verwendung von Cordoba in der älteren Version 3.6 vorraus, was jedoch den Funktionsumfang nicht mindert.




Da es sich bei ellIoT noch um einen frühen Prototypen handelt, müssen viele Einstellungen manuell vorgenommen werden. Diese sollen später einmal durch den Nutzer über die App vorzunehmen sein.Hierzu gehört die festlegung des WLAN-Netzwerkes in der IoT-Node. Diese smuss zum jeztigen Zeitpunkt noch während der kompilation gesetzt werden, ebenso die Host-Adresse des Servers.

Neben dem Api-Key und -Secret des Moodstocks Cloudservices muss diese Adresse auch bei der App gesetzt werden.

## BEWERTUNG

Das Konzept hinter ellIoT betrachte ich als durchaus Verwendbar. Mit einer entsprechend guten Kamera können auch aus der Ferne IoT-Geräte erkannt werden. DIes bietet dem Nutzer beispielsweise die Möglichkeit von der Couch aus das Licht zu schalten oder das Thermostat einzustellen. Die Verbindung zwischen Hardware und Software wird durch das Element des Zeigens etwas greifbarer, was einigen Nutzern das abstrakte Konzept des Internet of Things eventuell etwas näher bringen kann. Auch Moodstocks-SDK bietet bei guten Vergleichsbildern eine schnelle erkennung, wodurch die Benutzung nicht frustrierend wird.