Dokumentation

*UrVent Eventplaner*

*12.01.2020*

Autor: Hendrik Lichtenberg

***Wichtige Hinweise:***

* ***Die in diesem Dokument aufgeführten Beschreibungen in Kursivschrift (außer einigen Abschnittüberschriften) sind beispielhaft und erläuternd und müssen aus dem fertiggestellten Bericht entfernt werden.***

# Teammitglieder

* Hendrik Lichtenberg
* Simon Micheel
* Nick Ringelmann
* Steven Geiger

# Projektbeschreibung

Motivation

UrVent ist aus der Idee entstanden es Nutzern zu ermöglichen Events anzulegen, andere Nutzer einzuladen, für diese Events QR-Code basierte Tickets zu generieren, die dann am Einlass durch andere Programme (beispielsweise eine Smartphone App) gescannt und auf Gültigkeit überprüft werden können.

Dies bietet einerseits den Vorteil, dass mit wenigen Mausklicks eine Gästeliste erstellt werden kann und der Einlassprozess bei Veranstaltungen verschlankt oder möglicherweise sogar vollkommen automatisiert werden könnte.

Zielsetzung

Ziel des Projekts ist eine leichtgewichtige Anwendung zu erstellen, die folgendes ermöglicht:

* Anlegen & Bearbeiten eines Nutzerkontos
* Authentifizierung von Nutzern (Login)
* Anlegen & Bearbeiten von Events
* Anlegen & Bearbeiten von Eventlocations (Orte)
* Einladen von anderen Nutzern zu eigenen Events
* Bestätigung und Ablehnen von Einladungen
* Darstellung von Events & Locations mithilfe Google Maps
* Generierung eines QR-Codes mit relevanten Ticketinformationen

Dabei soll zusätzlich zum Client eine Serverseitige API entwickelt werden, die auch von anderen Anwendungen verwendet werden könnte (so könnte z.B. eine Smartphone App diese Schnittstelle verwenden um QR-Tickets zu validieren und/ oder zu entwerten).

# UML-Diagramme

[**Klassendiagramm**](classDesignFinal.html) (siehe classDesignFinal.html)

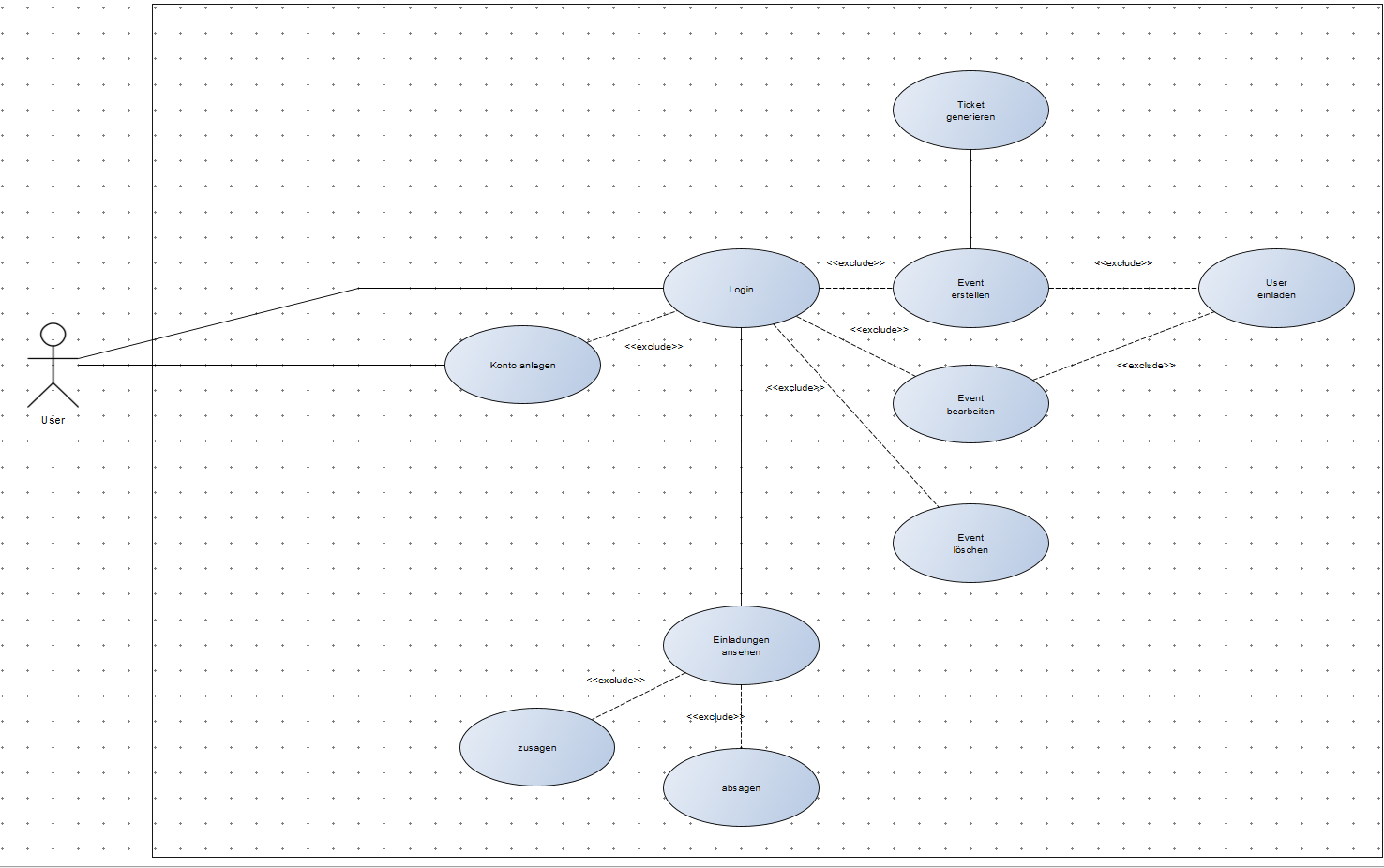


Abb. 0 Usecase Diagramm (siehe auch usecases.png)

# Anforderungen

Im folgenden werden die Funktionalen Anforderungen des Projekts in Form von User Stories erläutert:

* Als User möchte ich mit meiner E-Mail-Adresse und einem Passwort ein Konto anlegen können, um Zugang zu „UrVent“ zu erhalten.
* Als User möchte ich mich mit meiner E-Mail-Adresse und meinem Passwort anmelden können.
* Als User möchte ich eine Veranstaltung anlegen können, um eine Veranstaltung zu planen und zu veranstalten. Hierbei soll automatisch ein Ticket mit QR-Code generiert werden. Dabei wird der Veranstaltungsort mithilfe von Google Maps dargestellt.
* Als User möchte ich von mir erstellte Veranstaltungen bearbeiten können, um Änderungen vorzunehmen und um ggf. Informationen hinzufügen zu können.
* Als User möchte ich von mir erstellte Veranstaltungen löschen können, um Veranstaltungen wieder abzusagen.
* Als User möchte ich alle Veranstaltungen einsehen können, zu denen ich eingeladen worden bin, um mich über diese Einladungen und Veranstaltungen zu informieren.
* Als User möchte ich auf eine Einladung mit einer Zu- oder Absage reagieren können, um den Veranstalter zu informieren.
* Als User möchte ich andere User zu einer von mir erstellten Veranstaltung einladen können.

# Grobbeschreibung der Bedienung bzw. des Ablaufs

Der erste Schritt um UrVent zu nutzen ist die Erstellung eines Nutzerkontos, dazu startet der User das Programm und klickt den Button „New Account“.

Im darauffolgenden Screen wird der Nutzer aufgefordert seine persönlichen Daten (E-Mail, Passwort usw.) einzugeben und kann diese mit einem Klick auf „Submit“ bestätigen, woraufhin er auf den Login Screen zurückgeführt wird.

Im Login Screen gibt der Nutzer dann seine Logindaten ein, klickt auf Login und wird auf den Home Screen weitergeleitet insofern diese korrekt sind.

Im Home Screen hat der Nutzer dann die Möglichkeit die verschiedenen Funktionen von UrVent zu nutzen und z.B. ein Event zu erstellen, eine Location anzulegen usw.

Im Folgenden werden einzelnen Menüpunkte des Home Screens erläutert:

## My Events

Hat der User noch kein Event erstellt ist auf diesem Screen lediglich die Schaltfläche „Create Event“ zu sehen. Ein Klick auf diese leitet den Nutzer auf einen Screen weiter auf dem er die relevanten Daten eingeben kann um ein Event zu erstellen.

Hat der Nutzer mindestens ein Event erstellt wird es hier angezeigt und ein Klick auf es leitet den User auf einen Screen weiter, der die Daten des Events inklusive einer Google Map, die auf die Koordinate der Eventlocation zentriert ist, anzeigt und die Möglichkeit bietet Gäste einzuladen.

## My Invitations

Wurde der Nutzer zu einem Event eingeladen werden hier die noch nicht bestätigten Einladungen aufgelistet, die der Nutzer dann entweder annehmen oder ablehnen kann.

## Acceptet Invites

Hier werden die vom Nutzer bestätigten Einladungen angezeigt, ein Klick auf den Event Namen leitet den Nutzer auf den Eventinformationsscreen weiter und ein Klick auf Get Ticket öffnet ein neues Fenster mit dem, zur Einladung gehörenden Ticket, in QR-Code Form, dass dann in der momentanen Version von UrVent als Screenshot gespeichert werden kann, eine Konvertierung in ein PDF Dokument wäre jedoch in einer späteren Iteration von UrVent denkbar.

## My Account

Der Menüpunkt „My Account“ zeigt die Kontoinformationen des Nutzers an und gibt ihm die Möglichkeit diese über „Edit“ zu bearbeiten.

## My Calendar

Dieser Button dient als Platzhalter und hat noch keine Funktionalität.

## My Locations

Dieser Menüpunkt ermöglicht es dem Nutzer Locations anzulegen, dies ist erforderlich bevor ein Event angelegt werden kann, da jedes Event eine Location benötigt. In der aktuellen Version von UrVent muss dafür der Längen- und Breitengrad eingegeben werden, in einer späteren Version von UrVent wäre eine Auflösung von Adressen zu Koordinaten durchaus denkbar. Bereits angelegte Locations können mit einem Klick auf den Namen angezeigt werden.

# Entwurf und Umsetzung

Die Architektur von UrVent orientiert sich lose am MVVM-Pattern, das heißt es soll eine möglichst lose Kopplung zwischen Model, View & ViewModel erreicht werden, was die Testbarkeit von einzelnen Modulen stark vereinfacht, eine starke Modularisierung sowie effektive Datenkapselung ermöglicht.

Jeder „Screen“, der dem Nutzer angezeigt werden kann setzt sich aus folgendem zusammen:

## ViewModel

Das ViewModel ist der zentrale Bestandteil jedes Screens und dient dazu den View (.fxml) zu initialiseren und benötigte UI-Elemente als Objekt abzubilden, Models von der API anzufragen sowie zu initialisieren sowie diese dann im View darzustellen (Präsentationslogik) und Befehle an die API auszuführen. Außerdem initialisiert das ViewModel interaktive wie Buttons oder zur Laufzeit zu erzeugende UI-Elemente.

Die Klasse ViewModel selbst ist abstrakt und erbende Klassen implementieren die individuelle Funktionalität jedes Screens.

## Model

Models bilden Objekte (Rows), die von der API aus der Datenbank geliefert wurden als Java-Objekte, ab. Sie selbst werden aus JSONObjects konstruiert, die als String aus der API eingelesen wurden. Models verfügen nur über minimale Funktionalität und sind zur Organisation von Daten gedacht.

## View

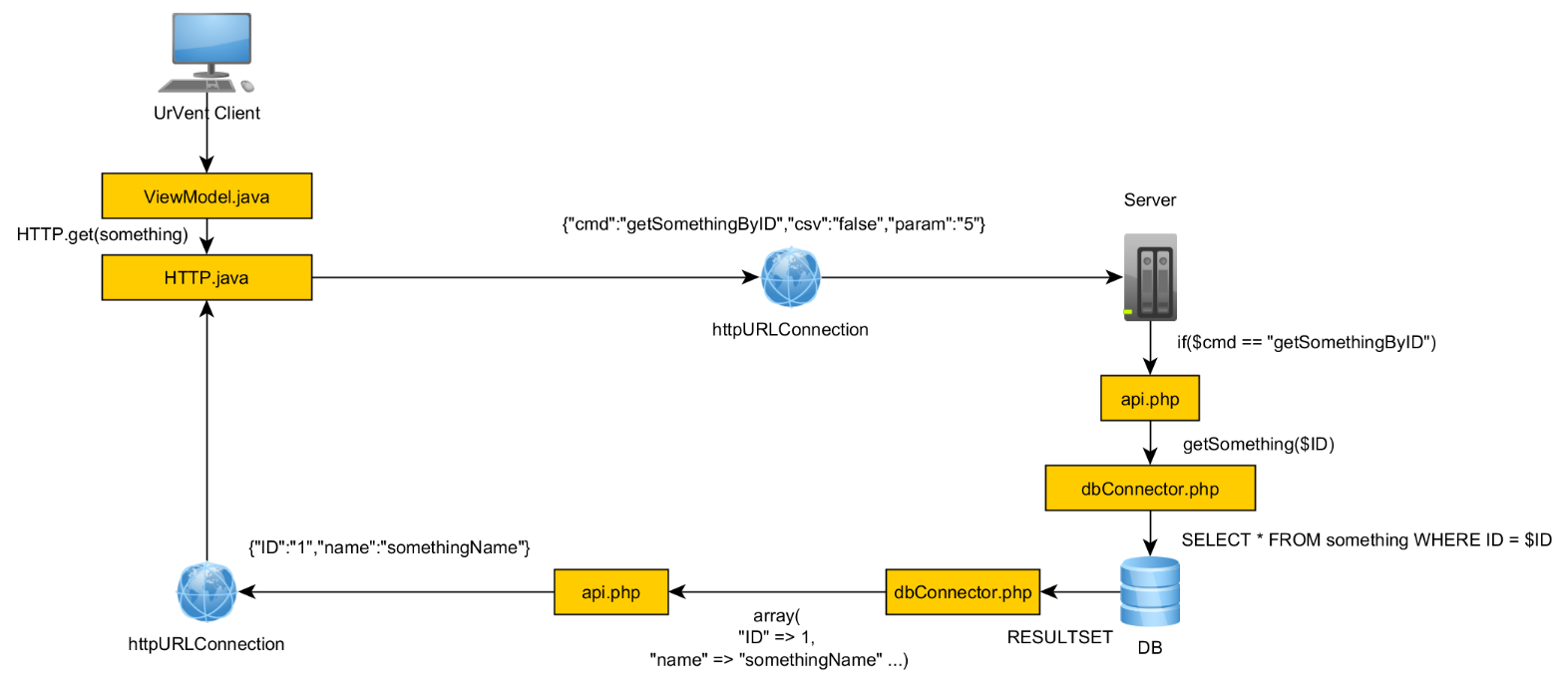
Als View werden .fxml Dokumente bezeichnet, die das Layout des UIs definieren. Aus den „fx:id“ Attributen werden im ViewModel entsprechende Objekte erzeugt.

## VMEventHandler

Von der abstrakten Klasse „VMEventHandler“ erbende Klassen, implementieren die Funktionalität aller Buttons und werden vom ViewModel erzeugt, sowie ihren entsprechenden Buttons zugewiesen.

## Client – Server – API - DB

Im folgenden wird die Kommunikation zwischen Client, Server, API und Datenbank erläutert.



*Abb. 1 Client fragt den Eintrag aus der Tabelle „something“ mit der ID 1 an.*

### JSON

JSON oder „JavaScript Object Notation“ ist ein Datenformat, das es vereinfacht Objekte über Netzwerke zu transportieren und wird von der UrVent API verwendet um Objekte zwischen dem Client und der API zu kommunizieren. JSON verwendet Key-Value Paare, die als String abbildbar, was sich gut eignet um UrVent Models in Textform abzubilden.

Ein Beispiel: {„ID“ : „1“, „name“ : „A Name“} würde ein Java Objekt mit den Feldern int ID = 1 und String name = „A Name“ abbilden.

### Client -> Server

Der Client kommuniziert mit dem Server über die Klasse HTTP, die mithilfe einer httpURLConnection PHP Skripte über den HTTP Server von XAMPP ausführt und diesen Kommandos im JSON Format übergibt. (siehe Abb. 1)

### Server -> API -> DB

Wird das Skript api.php aufgerufen, so wird die angefragte Methode im dbConnector.php ausgeführt, der dann das in JSON form vorliegende Kommando in eine SQL-Query umwandelt und das entsprechende Ergebnis als Array zurückgibt. (siehe Abb. 1)

### API -> Client

Wurde ein Ergebnis gefunden, wird dieses als JSONString über die bestehende httpURLConnection zurückgegeben, wird clientseitig in ein JSONObject (Java Objekt) konvertiert, aus welchem danach ein Model erzeugt und vom ViewModel verarbeitet werden kann. (siehe Abb. 1)

## Eingesetzte Technologien (Framework, etc.) inkl. deren Einbindung

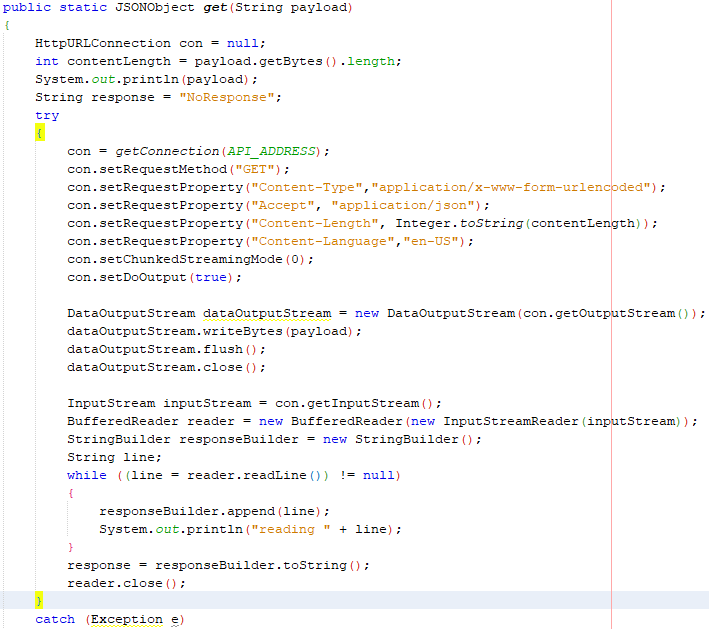
Der UrVent Client wurde mit Java 1.8\_221 SE entwickelt und nutzt das in dieser Java Version enthaltene JavaFX Framework zur Darstellung und zum Management des UIs.

Des weiteren nutzt UrVent die Libraries SimpleJSON [1] und ZebraCrossing [2], wobei erstes der Konvertierung von Objekten in JSON Format und zweites dem Generieren von QR-Codes dient.

Der Serverseitig verwendet UrVent XAMPP zum hosten der API und der Datenbank (MariaDB). Die API nutzt PHP 7.2.9

## Codesnippets & Erläuterungen

### Datentransfer



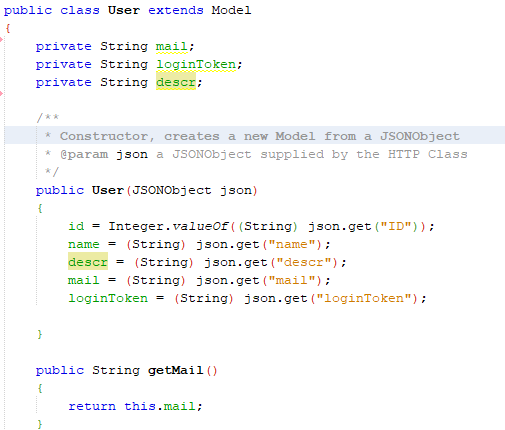
*Abb. 2 tools.HTTP.get(String Payload)*

Die Methode „HTTP.get()“ Stellt den zentralen Knotenpunkt zwischen Client und API dar, sie akzeptiert einen String als Eingabe, der als Kommando an die API gesendet wird und liefert die Antwort als JSONObject zurück. Dieses Kommando liegt in der Klasse APICommand als vorgefertigter Get vor, so kann Beispielsweise über die Zeile:

„JSONObject obj = HTTP.get(APICommand.getEventByID(1));“

unkompliziert das Event mit ID = 1 aus der Datenbank abgefragt werden.

Die Methode baut eine httpURLConnection zur API unter der Adresse „http://localhost/urvent/api.php“ auf und sendet den von APICommand generierten JSONString an diese.



*Abb. 3 Model.User*

Die Klasse User stellt Beispielhaft ein Model dar, das aus einem Datenbankeintrag erzeugt wird. Wie im Konstruktor in Abb. 3 zu sehen akzeptiert dieser, so wie alle anderen Models, ein JSONObject als Parameter, wie bereits erwähnt können diese unkompliziert aus der API abgefragt werden, sodass ein Objekt aus der Datenbank sehr einfach als Model initialisiert werden kann. So würde zum Beispiel die Zeile:

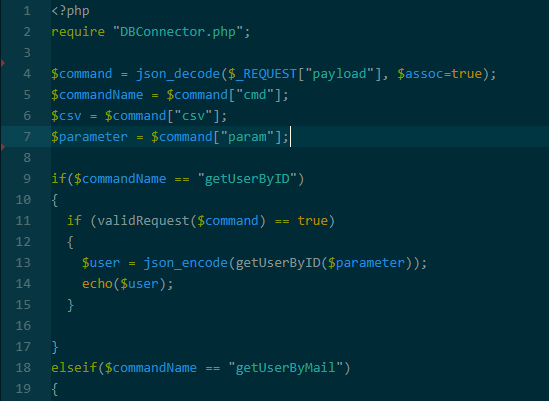
„User u = new User(HTTP.get(APICommand.getUserByID(1)));“

den User mit der ID 1 abfragen und als User Model initialisieren, sodass dieser nun einfach vom ViewModel verarbeitet werden kann.

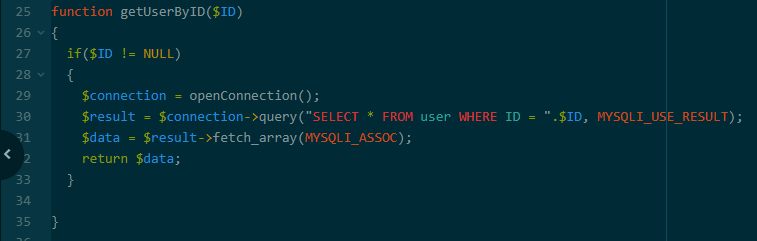


*Abb. 4 tools.QRGen.generateQRCode()*

Die Klasse tools.QRGen stellt dank der ZebraCrossing[2] Library eine sehr einfach zu verwendende Methode zur Verfügung, mit der Tickets in Form von QR-Codes in Form eines BufferedImage dargestellt werden.



*Abb. 5 api.php*



*Abb. 6 DBConnector.php*

Die Klasse api.php stellt die Serverseitige Schnittstelle zwischen Client und Datenbank dar, wie in Zeile 4-5 zu sehen nimmt die API Kommandos in JSON Form entgegen, dekodiert diese zu einem PHP Array und entnimmt diesem dann die relevanten Parameter, die dann an di entsprechenden Methoden der DBConnector.php weitergegeben werden. (z.B. Z.13, siehe auch Abb. 6)

# Evaluation

Das Projekt wurde auf mehreren Rechnern erfolgreich ausgeführt und Anhand von Testcases auf Funktionalität überprüft. Dabei konnte der ursprüngliche Entwurf verbessert werden und eine große Zahl potentieller Fehler verbessert werden.

Insgesamt kann davon ausgegangen werden, dass sich das Produkt zuverlässig ausführen lässt und die Funktionen größtenteils Fehlerfrei sind.

# Fazit

Insgesamt konnte das Projektziel erreicht werden, die im Vorfeld formulierten Anforderungen konnten erfüllt werden und das Minimal Feature Set wurde implementiert, jedoch ist zu bemängeln, dass effizienteres Projektmanagement und bessere Teamarbeit ein potentiell hochwertigeres Produkt ermöglicht hätte.

# Literatur & Quellen

[1] https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple/1.1.1

[2] https://github.com/zxing/zxing