Table of Contents

1. Development Guide

1.1. Prerequisite

- Basic knowlegde of NodeJs, Git and ROS
- Install Git & Editor

https://code.visualstudio.com/

- Install NodeJS
  https://nodejs.org/en/download/
- Install Sails

  ```
  npm install sails -g
  ```

- If on Windows:
  1. Install Ubuntu
     - Got to Microsoft store
     - Search for 'Ubuntu'
     - Click get/install
  2. Install ROS
     - Follow ROS Kinetic installation + (this might take some time) http://wiki.ros.org/kinetic/Installation/Ubuntu

       ```
       sudo apt-get install ros-kinetic-desktop-full
       ```

     - And enviroment setup

       ```
       echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
       source ~/.bashrc
       ```

     - Test the installation running ROS

       ```
       roscore
       ```

     - Done for now

       ```
       (ctrl+c)
       ```

1.2. Github

- Invite your personal Github account to acces ArtOfRobotics repos https://github.com/orgs/ArtOfRobotics/people
- Clone Git Repo

  ```
  git clone https://github.com/ArtOfRobotics/WWEB
  ```

- Switch to test branch

  ```
  git checkout -b origin/test
  ```

1.3. Compilation

Compilation is done via catkin, this is done to create a rospackage so that nodejs can run in a ROS enviroment.

```
cd WWEB/src
npm install
cd ..
source /opt/ros/kinetic/setup.bash
catkin_make
```

1.4. Testing/Debugging

Run without ros:

```
cd WWEB/src
sails lift (or node app.js)
```

Run with Ros:

1. Start Roscore
   - Open a terminal (Ubuntu app on windows) -

     ```
     cd WWEB
     source devel/setup.bash
     roscore
     ```

2. Run webplatform
   - Open a terminal (Ubuntu app on windows) -

     ```
     cd WWEB
     source devel/setup.bash
     rosrun willyweb start.sh
     ```

The rosrun command might not have acces to port 80 for this to work use sudo -s

```
sudo -s
rosrun willyweb start.sh
```

1.5. Running Scripts

In the same manner as you would do Testing/Debugging you can also run scripts. Scripts are located in the folder 'WWEB/src/scripts'.

1. Start Roscore
   - Open a terminal (Ubuntu app on windows) -

```
cd WWEB
source devel/setup.bash
roscore
```

2. Run sending script
   - Open a terminal (Ubuntu app on windows) -

```
cd WWEB
source devel/setup.bash
rosrun willyweb scripts/send.js
```

3. Run receive script
   - Open a terminal (Ubuntu app on windows) -

```
cd WWEB
source devel/setup.bash
rosrun willyweb scripts/receive.js
```

Rosrun makes it possible to communicate with ROS because it is now run as a ROS package

The 'start.sh' script consist out of a simple run script which launches the webplatform

```
#!/usr/bin/env bash
node src/app.js
```

- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

2. Software architecture document

The Software Architecture Document contains all the functional requirements, use cases and wireframes. It also includes a technical overview and explaination about the written code and how RosNodeJS is implemented. The wiki contains some more info about RosNodeJs because it lacks a proper documentation.

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

3. RosNodeJs

3.1. Introduction

RosNodeJs is a framework in which you can create real ROS Nodes. It is good to have basic knowledge about ROS before reading any further. This can be read on the Wiki and at the official website http://www.ros.org/.

RosNodeJs lacks in a proper documentation but the official wiki is available on ROS's own website, see http://wiki.ros.org/rosnodejs/overview.

RosNodeJs functions as a ROSPackage, the instance of WillyWeb is run as a RosPackage named 'willyweb'. However you are able to run multiple instances of the Rospackage 'willyweb' for example if you run multiple scripts.

3.2. Advantages/disadvantages

3.3. How to use

3.3.1. Declaration

Ros is used as a NPM package and is loaded into a controller using the following code: image::media/sad/image18.png[] Here in the first line of code RosNodeJs is loaded into the constant variable 'rosnodejs'. In the second line the type of message is defined, in this case 'sensor_msgs'.

3.3.2. Node Handle

In the following section of code these variables are used as following: image::media/sad/image19.png[] In the third line of code you can see an if statement that checks if a so called Node Handle already exist. A Node Handle is the Node on which the platform operates, you could also see this as the Rospackage 'willyweb' but then in a active node. You would expect that this is not needed because you require the same RosNodeJs module across the application but research found out this does not work as of the moment of writing.

If a Node Handle already exits the node can be accessed using 'rosnodejs.nh'. If this Handle does not yet exist it awaits the creation of it using 'rosnodejs.initnode('/willyweb')'.

The main reason this is done each time RosNodeJs is used in the code is because the same instance must be used over the whole application. Multiple Node Handles cannot coexist. The official examples given are also written inside one function, this is not practical for a well designed application. Another solution was therefore to create a RosController in which all communication with Ros would be done. The main disadvantage of this is that you'll have to create a function for each and every message type used throughout the application which violates the Single responsibility principle in the SOLID principles.

3.3.3. Subscribing

If the Node Handle is initialised we can use the various functions ROS uses, for example subscribe: image::media/sad/image20.png[] In the first line the subscribe method is called with the first parameter being the topic on which it must subscribe '/sonar' and next up the message type 'msg.LaserEcho' after that the function is declared on what to do when data is published on the topic. This data is stored in the local variable 'data', in the current situation it would blast an update over the socket that new data is available. This update also contains the new data received form the topic.

3.3.4. Advertise

Before being able to publish onto a ROS topic we first need to advertise that we are going to publish data. This is done using the advertise function on the Node Handle: image::media/sad/image21.png[] First a variable is made so that the advertise topic can be reused inside the controller. image::media/sad/image22.png[] After that the variable is filled with a advertise topic, the function advertise consist out of 2 parameters in this case the topic to advertise on '/led' and a message type 'msgs.ColorRGBA'.

3.3.5. Publishing

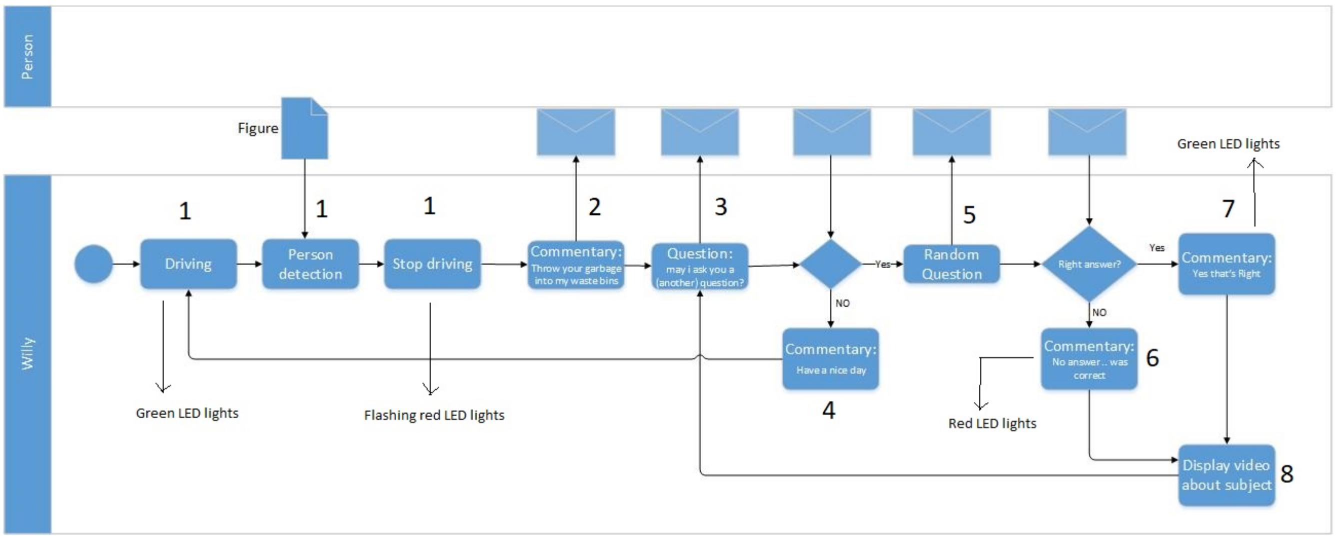
After the initialisation of the advertise topic the following function is used to publish data onto this ROS topic: image::media/sad/image23.png[] In this example the earlier variable 'pub' is reused to call the function publish on the advertise topic. This function contains the data in JSON format that needs to be published onto the topic. Make sure this data is coherent to the official format used by the specific message type which can be found at ROS official wiki: http://wiki.ros.org/common_msgs?distro=kinetic

3.4. Implementation

- (2017/2) Research

4. Interaction

4.1. Flow Chart


Person
Willy
Figure
Green LED lights
1
Driving
1
Person detection
1
Stop driving
2
Commentary: Throw your garbage into my waste bins
3
Question: may i ask you a (another) question?
Yes
NO
Commentary: Have a nice day
4
5
Random Question
Right answer?
Yes
NO
Commentary: No answer .. was correct
6
Red LED lights
7
Commentary: Yes that's Right
Display video about subject
8
Green LED lights
Flashing red LED lights

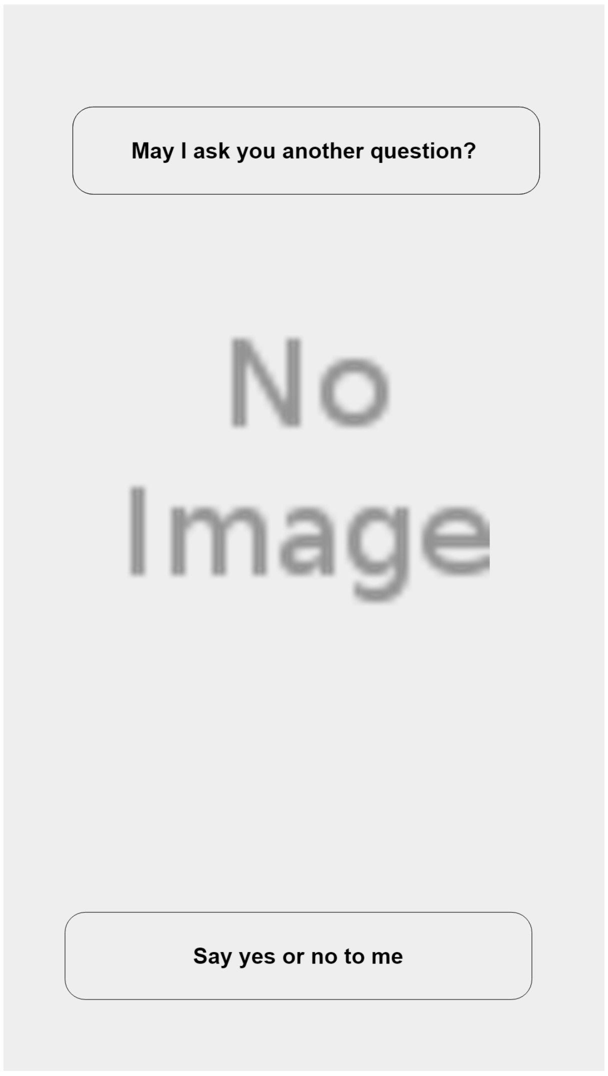
4.2. Screen Designs

4.2.1. English

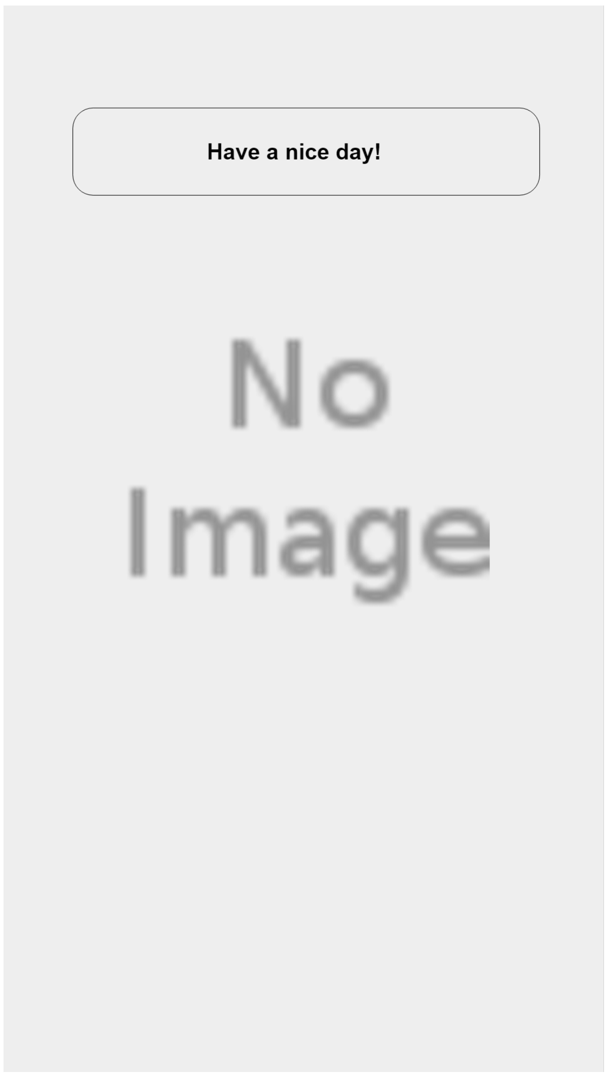
1. This is the screen shown when Willy is driving around, and there is not yet a person to interact with.

2. This screen is shown when a person is starting to interact with Willy. At the bottom there are two arroys pointing to the garbage bins.

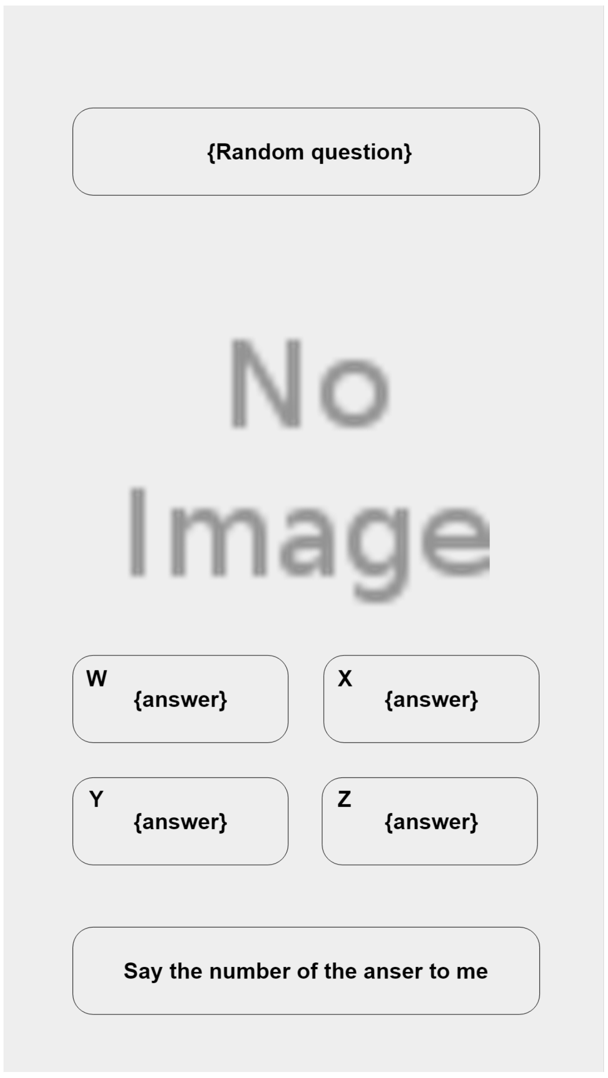
3. After that Willy asks permission to ask a question to the person in front of Willy.



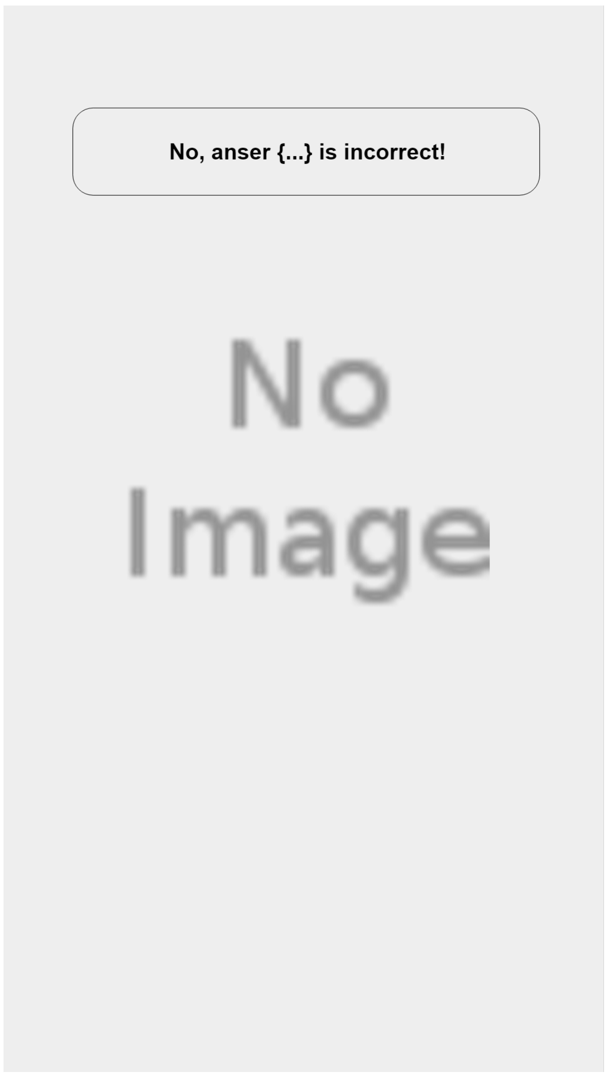
4. When the person says 'no', then this screen is shown, after wich the robot continues driving.



5. When the person says 'yes', a random question is shown with a number of multiple choise answers.



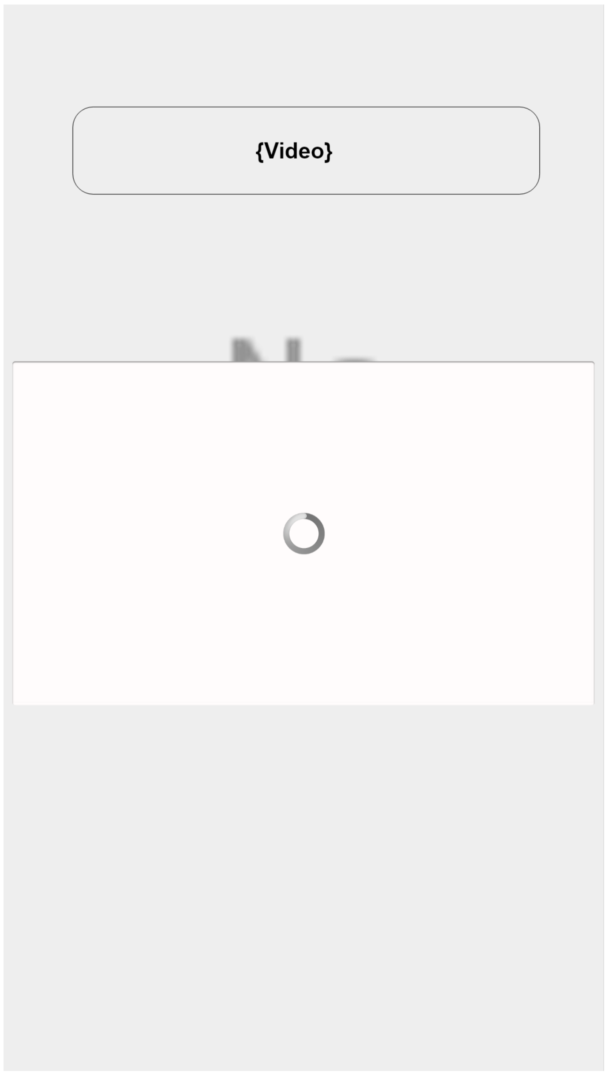
6. When the given answer is wrong, this screen is shown.



No, anser {...} is incorrect!

7. When the given answer is right, this screen is shown.

8. After the question, Willy starts to play an informative video about garbage.



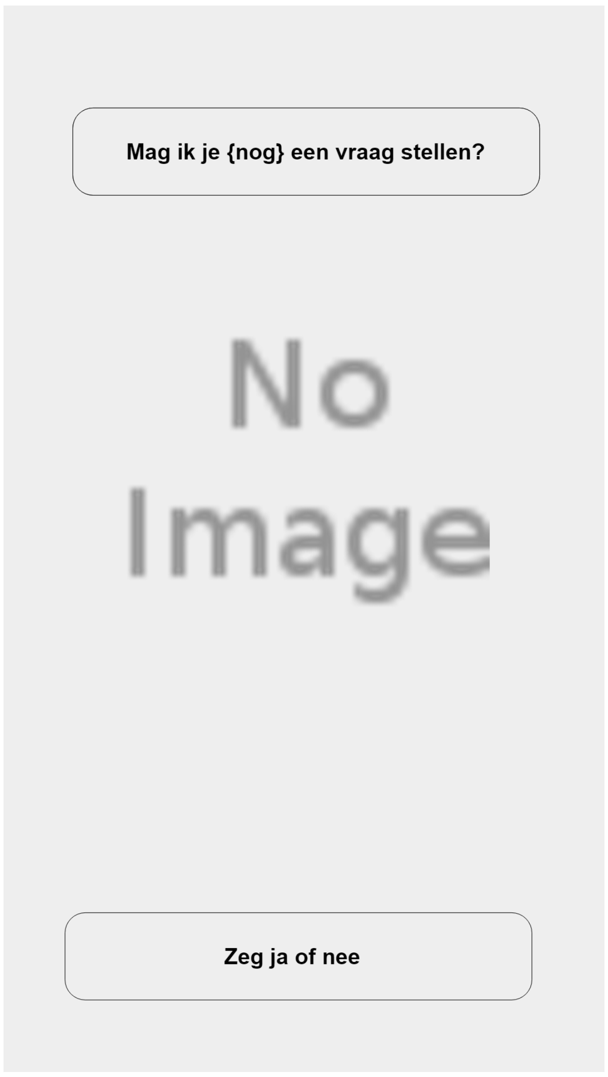
4.2.2. Dutch

For comments at each image, see the English version above.

1.

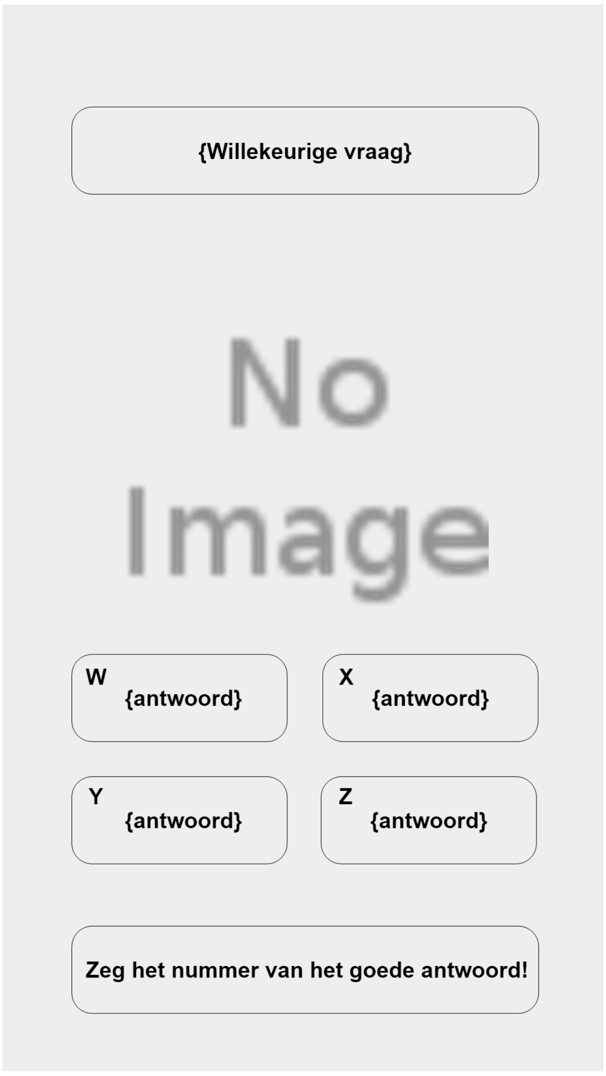
2.

3.



4.

5.

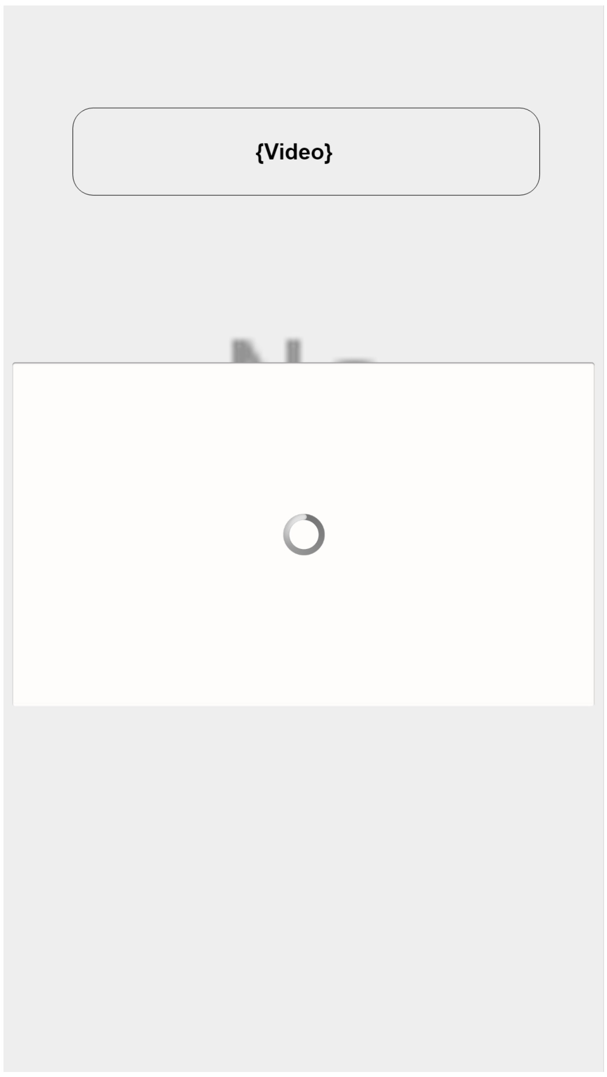

{Willekeurige vraag}
No Image
W {antwoord}
X {antwoord}
Y {antwoord}
Z {antwoord}
Zeg het nummer van het goede antwoord!

6.

Nee, antwoord {...} is fout.

7.

8.



4.2.3. Speech configuration

```
curl -sL https://deb.nodesource.com/setup_9.x | sudo -E bash -

sudo apt-get install -y nodejs

sudo apt-get install -y build-essential

sudo apt-get install python-pyaudio python3-pyaudio sox

pip install pyaudio

sudo apt-get install libmagic-dev libatlas-base-dev

git clone https://github.com/ArtOfRobotics/Speechtest.git

cd Speechtest

node server.js
```