# AI Kiosk – Contents

Contents

[AI Kiosk – Contents 1](#_Toc71895517)

[What is the AI Kiosk 4](#_Toc71895518)

[Version Control 4](#_Toc71895519)

[Website 4](#_Toc71895520)

[AI Models 4](#_Toc71895521)

[Hosting 4](#_Toc71895522)

[Website 4](#_Toc71895523)

[AI API 4](#_Toc71895524)

[Database 4](#_Toc71895525)

[Website 5](#_Toc71895526)

[Website Type 5](#_Toc71895527)

[How it Works 5](#_Toc71895528)

[Structure 5](#_Toc71895529)

[Dependencies 5](#_Toc71895530)

[App\_Code 5](#_Toc71895531)

[App\_LocalResources 6](#_Toc71895532)

[App\_Start 6](#_Toc71895533)

[Content 6](#_Toc71895534)

[Controllers 6](#_Toc71895535)

[Models 6](#_Toc71895536)

[Resources 6](#_Toc71895537)

[Scripts 6](#_Toc71895538)

[Bundle.Config 7](#_Toc71895539)

[Global.asax 7](#_Toc71895540)

[LargeScreen.aspx 7](#_Toc71895541)

[Login.aspx 7](#_Toc71895542)

[Packages.config 7](#_Toc71895543)

[Site.master 7](#_Toc71895544)

[Startup.cs 7](#_Toc71895545)

[Web.config 7](#_Toc71895546)

[Commend Intercom Client 7](#_Toc71895547)

[AI Component 8](#_Toc71895548)

[AI Structure 8](#_Toc71895549)

[Framework 8](#_Toc71895550)

[Training Data 8](#_Toc71895551)

[Retraining a new model 8](#_Toc71895552)

[AI Model Reference Tutorial 9](#_Toc71895553)

[Database 9](#_Toc71895554)

[Tables 9](#_Toc71895555)

[datClickLogging 9](#_Toc71895556)

[datConversationNodes 9](#_Toc71895557)

[datHeartbeat 9](#_Toc71895558)

[datLogging 9](#_Toc71895559)

[datSites 10](#_Toc71895560)

[jncLanguageResponses 10](#_Toc71895561)

[jncSiteKiosks 10](#_Toc71895562)

[jncSiteNodes 10](#_Toc71895563)

[LuLanguages 10](#_Toc71895564)

[MediaLocations 11](#_Toc71895565)

[Stored Procedures 11](#_Toc71895566)

[AI API 11](#_Toc71895567)

[Application Type 11](#_Toc71895568)

[Structure 11](#_Toc71895569)

[Api Calls 11](#_Toc71895570)

[Default (/) 11](#_Toc71895571)

[/Wake sounds 12](#_Toc71895572)

[/Audio 12](#_Toc71895573)

[/Videos 12](#_Toc71895574)

[/Images 12](#_Toc71895575)

[/Helpful 12](#_Toc71895576)

[/Receive 12](#_Toc71895577)

[API Functions 12](#_Toc71895578)

[Translate\_from 12](#_Toc71895579)

[Intent 12](#_Toc71895580)

[Db\_Conn 12](#_Toc71895581)

[Future\_Nodes 13](#_Toc71895582)

[Get\_Response\_text 13](#_Toc71895583)

[Is\_entry\_node 13](#_Toc71895584)

[Error Message 13](#_Toc71895585)

[Get\_audio\_file 13](#_Toc71895586)

# What is the AI Kiosk

* A virtual parking assistant to answer basic parking related questions without needing a human
* Uses windows PCs and a large touch screen to load a browser that provides the functionality
* The AI website is a single page application that processes user speech and provides a response, provides help videos & images (FAQ), and also allows direct calls to the monitoring station
* All conversations with the kiosk are logged for future training purposes of the AI
* The questions the AI handles are specific to each kiosk and need to be created for every device that we deploy

# Version Control

## Website

* **TFS Server:** 10.45.0.130
* **Path:** 10.45.0.130/PreciseParklink/Chatbotwebapp

## AI Models

* Not currently stored anywhere
* **Need to create some sort of version control system. Git repo?**

# Hosting

## Website

* IIS
* Site: Chatbot
* 172.18.1.40 (Dev)
* 172.18.1.41 (Prod)

## AI API

* IIS
* Site: ChatbotAI (Using a tunnel created by wfastcgi to act as a python environment)
* 172.18.1.40 (Dev)
* 172.18.1.41 (Prod)

## Database

* SSMS
* Database: dev-chatbot/ai\_chatbot\_kiosk
* 172.18.1.40 (Dev)
* 172.18.1.41 (Prod)
* Username: sa
* Password: precise\*3984052

# Website

## Website Type

* ASP.net Web Forms
* For this type of website instead of controllers, views, models there are layout pages (Ex: LargeScreen.aspx which have events bound to the code-behind of the page)
* To access the code-behind right click the page and ‘View Code’

## How it Works

* The UI loads a page with a specific Kiosk ID in the url (Ex: kiosk=9953). The kiosk id is crucial to identify which device is loading the page in terms of rendering FAQ content, sending requests to the AI, connecting the correct device using Commend, etc.
* On the page itself there are 3 main components:
  + Joy, the AI, can be used by saying ‘Hey Joy’ followed by your question. Example: ‘Hey Joy, where do I park?’. Google speech to text will be used to parse what you said, then a request will be sent to the AI API. The response generated by the API will be displayed on the screen and also played through the kiosk speakers
  + FAQ, which generates a categorized UI version of the questions and answers supported by the kiosk.
  + Live Agent, which uses the Commend Intercom Client running in the background of the PC to directly send a voice call to our Monitoring Station

## Structure

### Dependencies

* Nuget packages used by the website

### App\_Code

* Class files used for various functionality
* Commend.cs – Unused
* FilesAndUtil.cs – Utility class to handle url parameters, read/write error logs, send email alerts when the kiosk goes offline, etc.
* Helper class used to generate the FAQ section

### App\_LocalResources

* Various language files to generate strings for the UI
* Default.aspx.fr.resx – Deprecated. Placeholder for French strings
* Default.aspx.resx – Deprecated. English FAQ content
* LargeScreen.aspx.fr.resx – Placeholder for French strings
* LargeScreen.aspx.resx – English UI content strings
* OldKiosk.aspx.fr.resx – Deprecated. Placeholder for French strings
* OldKiosk.aspx.resx – Deprecated. English FAQ content

### App\_Start

* BundleConfig.cs – Defines the js files to be bundled and minified
* IdentityConfig.cs – Unused. Autocreated auth class
* RouteConfig.cs – Defines the URL routes for the website
* Startup.Auth.cs – Handles kiosk login and cookie management
* WebApiConfig.cs – Defines URL routes for API calls

### Content

* Css files for the website

### Controllers

* ClickLogController – Manages writing user clicks of the UI to the database for Heatmap generation
* ErrorController – Manages various error states of the kiosk (microphone, heartbeat, camera, ec.) and sends alert emails. Also manages the kiosk heartbeat
* WebcamController – Used for managing inbound and outbound camera feeds when the kiosk is connected to the Monitoring station talking to a live agent so each user can see one another

### Models

* IdentityModel.cs – Used for creating the auth cookie for the kiosk

### Resources

* Images, videos, icons, etc. used throughout the website

### Scripts

* Js libraries for the website. Custom js functions are defined in functions.js

### Bundle.Config

* Defines the css classes to be bundled (not minified)

### Global.asax

* App startup code that configures the URL routing defined in other classes

### LargeScreen.aspx

* Primary UI page for the website

### Login.aspx

* Manages login for a new kiosk
* **Login accounts are not currently stored in database, they are stored in the code behind for this page in arrays**
* Creates and stores the auth cookie and redirects to UI page with kiosk ID in the URL

### Packages.config

* Defines all of the imports and libraries to be used

### Site.master

* Master page for the website, individual pages (LargeScreen.aspx) get loaded into
* Defines title, imports bundled css/js files

### Startup.cs

* Default startup config, registers the SignalR routes to be used in Monitoring Station connections

### Web.config

* Connection strings, url paths, build settings, etc.

# Commend Intercom Client

* For each physical kiosk device + pc, the Commend Intercom Client must be installed and configured with a unique device number. This number **must** match the kiosk ID assigned to the kiosk and placed in the URL
* The intercom client runs in the background on the pc, and when the user clicks to call a live agent the following occurs:
  + The website sends a request to the Commend API to generate a call between the monitoring station and the kiosk (using provided device ID)
  + The intercom client on the desktop will automatically start connecting to the monitoring station
  + When the monitoring station answers, handles, and ends the call the Intercom Client on the kiosk pc will return to an idle background state until the next call
* For configuring the Intercom on new kiosks refer to Herbert

# AI Component

* The AI model is currently accessed, trained, and managed using a Jupyter notebook
* The file to use for development is: chatbot.ipynb in Chatbot/Snipless\_Chatbot

## AI Structure

### Framework

* Neural network using TFLearn and softmax activation
* Intent based classification model
* Saved and restored using pickle

### Training Data

* Training data is stored in Intents.json
* Questions are grouped by category (Site staff, parking, payment, etc.), and given a tag to define them
* Training data is listed under Intents.json. **Note: This training data must be unique to each kiosk. Reusing the same Intents file will train the model using all intents in the file unless an additional feature is added**
* The neural net layers, epoch, batch size, etc. have already been configured for best performance on the training data set. **In the future if performance is poor this may need to be updated**

### Retraining a new model

* Create a new Training.json file
* Delete and previous training checkpoints (by default the model will continue to train and overfit unless you delete the previous state)
* Run chatbot.ipynb
* Run chatbot2.ipynb and ensure the accuracy of the model is still acceptable
* Deploy the trained model wherever it needs to go

### AI Model Reference Tutorial

* <https://chatbotsmagazine.com/contextual-chat-bots-with-tensorflow-4391749d0077>

# Database

## Tables

### datClickLogging

* Logging table for all click events that user’s perform on the kiosk
* ClickLogId – Generic autonumber PK
* KioskId – Kiosk that was clicked
* LanguageId – Active language the kiosk was in during the click (Ex: English)
* XCoordinate – x coordinate of the click
* YCoordinate – y coordinate of the click
* ClickControl – which button was clicked, if any
* Parentcontrol – Screen user was on when they clicked
* DateEntered – Datetime of click

### datConversationNodes

* Legacy content. Was used to store the relation between nodes for multi-step assistance
* NodeId - Generic autonumber PK
* NodeName – intent name
* ParentId – Id of parent node, if any
* ResponseId – ID of the intent for determining response

### datHeartbeat

* Status tracker to ensure kiosks are all online
* heartbeatId - Generic autonumber PK
* Kioskid – kiosk who’s heartbeat is being tracked
* Lastheartbeat – Last time the kiosk updated the server
* isMonitored – whether or not to send email alerts

### datLogging

* Logs user interaction with the AI assistant
* LogId - Generic autonumber PK
* KioskId – kiosk used for the request
* NodeId – The intent believed to answer the user’s question
* Userspeech – what the user said
* Probability – the confidence in the NodeId being the correct intent
* DateEntered – datetime ai was asked the question
* WasHelpful – optional user feedback on if it solved their issue

### datSites

* List of sites that have kiosks deployed to them
* SiteId - Generic autonumber PK
* SiteName – description of the site

### jncLanguageResponses

* Binding between intent, response text, response speech
* LanguageResponseId - Generic autonumber PK
* ResponseId – link to intent for this response
* Languageid – Which language this response is applicable to (English vs French)
* Responsetext – text to display on screen
* Responsespeech – text to generate into speech using google cloud
* usesMedia – Deprecated. Whether or not to attach an image file

### jncSiteKiosks

* Assigns kiosks to sites
* Sitekioskid - Generic autonumber PK
* Siteid – id of the site the kiosk is deployed to
* Kioskid – id of the deployed kiosk

### jncSiteNodes

* Assignment of intents to certain sites
* SNID - Generic autonumber PK
* Siteid – the site the intent belongs to
* Nodeid – the id of the intent to use
* Responseid – the id of the response to assign to this site/intent

## LuLanguages

* Lookup table of supported languages
* Languageid - Generic autonumber PK
* LanguageName – Description of the language
* WebSpeechKitCode – Language code to use for browser text to speech
* googleApiCode – language code to use for google cloud text to speech

### MediaLocations

* Deprecated. Was used to map file folder paths to be used in responses for the AI when images or videos needed to be included

## Stored Procedures

* spCheckHeartbeats – Gets a list of kiosks that haven’t submitted a heartbeat in the last hour
* spGetClickLog – Gets the click logs to be used for generating heat maps
* spGetErrorMessage – returns the default error message for the AI
* spGetFutures – deprecated. Returned the list of connected nodes to the provided intent
* spGetResponse – generates the AI response for the provided intent
* spInsertClickLog – inserts the array of clicks into the logging table
* spInsertLog – Inserts the users speech/ai response into the logging tale
* spIsEntryNode – deprecated. Returns whether or not this is a starting intent
* spQueryToHtmlTable – converts the sql query string to html for generating email templates
* spSendHeartbeatEmailAlerts – if any kiosks are in spCheckheartbeats result, this will send the alert email
* spUpdateHeartbeat – updates the last heartbeat time for the provided kiosk id
* spWasHelpful – logs whether the AI was helpful for the given log id

# AI API

## Application Type

* Python based Flask API hosted in IIS using wfastcgi

## Structure

* App.py is the API itself
* AI Chatbot-XXX is the google cloud API key
* Requirements.txt are the python libraries required to run the api
* ./ppl/api/nlu\_engine is where the AI model is stored
* Everything else (I think?) is deprecated from other versions of the kiosk

## Api Calls

### Default (/)

* Returns a ‘connected’ string if the api is running properly

### /Wake sounds

* Returns the wake sound mp3 to be played by the browser

### /Audio

* Returns the pre-generated mp3 file of the audio response to be played by the browser
* These files are precreated to decrease response time of the AI

### /Videos

* Returns the mp4 video file that was requested to be played by the browser

### /Images

* Returns the png image file to be displayed by the browser

### /Helpful

* Submits the user’s response to the database for the provided kiosk

### /Receive

* Processes a request for the AI to help a user
* Takes user speech, language code, kiosk id
* Parses the speech through the AI model to determine intent, gets response from db, generates/loads the audio file and returns the text/audio response to the calling browser

## API Functions

### Translate\_from

* Takes the raw speech and uses Google cloud to translate into English

### Intent

* Uses the provided text to classify the intent using the AI model

### Db\_Conn

* Handles database connection

### Future\_Nodes

* Deprecated
* Was used to determine which possible intents the AI can navigate to on the next step

### Get\_Response\_text

* Queries the database using the intent, language code, and kiosk id to get the HTML response of the AI to display in the browser

### Is\_entry\_node

* Deprecated
* Determines if the current intent is considered an entry node for the conversation

### Error Message

* Gets the default effort message for this language from the database

### Get\_audio\_file

* Deprecated
* Gets the file path to the mp3 for the provided intent