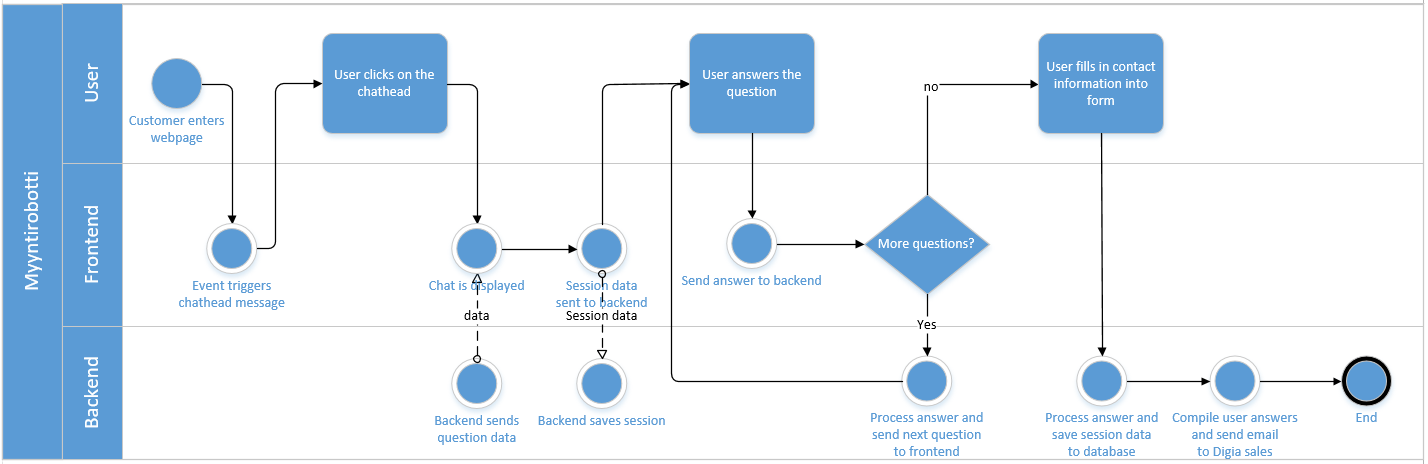
# Myyntirobotti

Myyntirobotti is a software project designed and developed by the 2016 Softala3 group in Haaga-Helia Pasila.

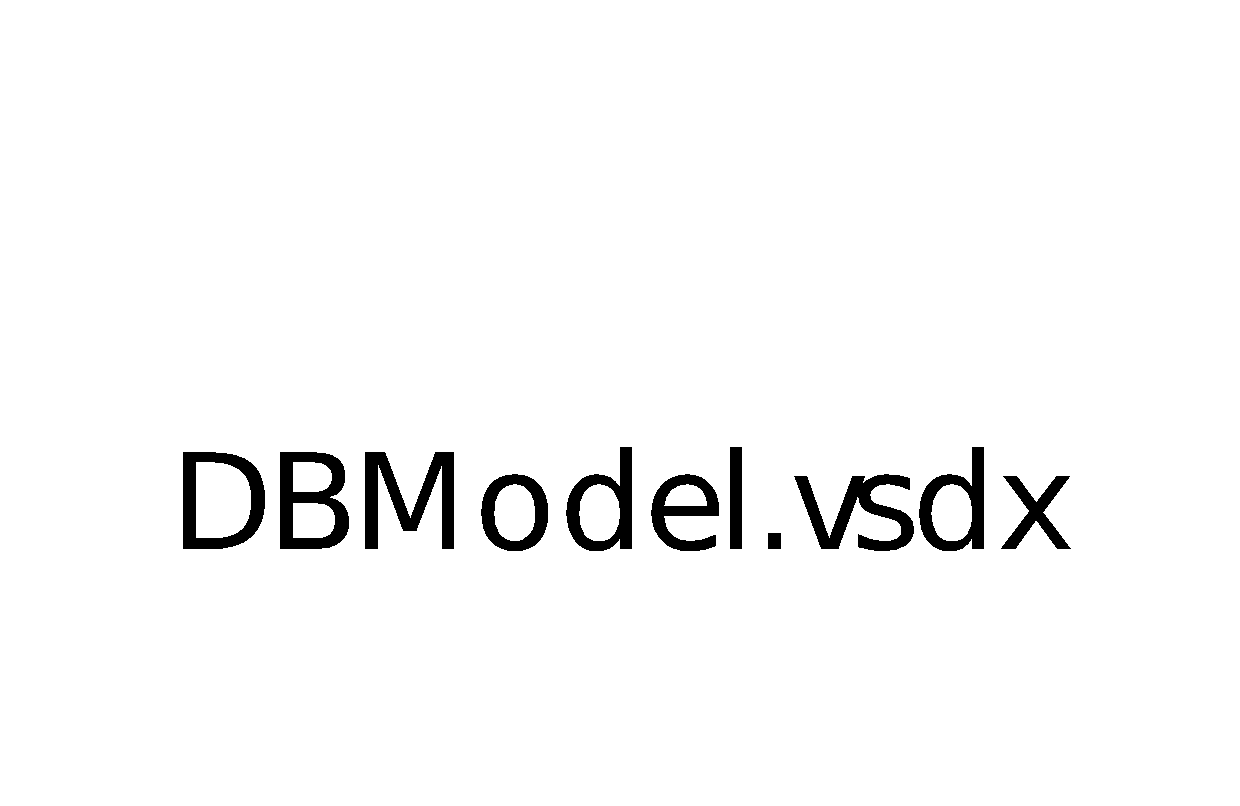
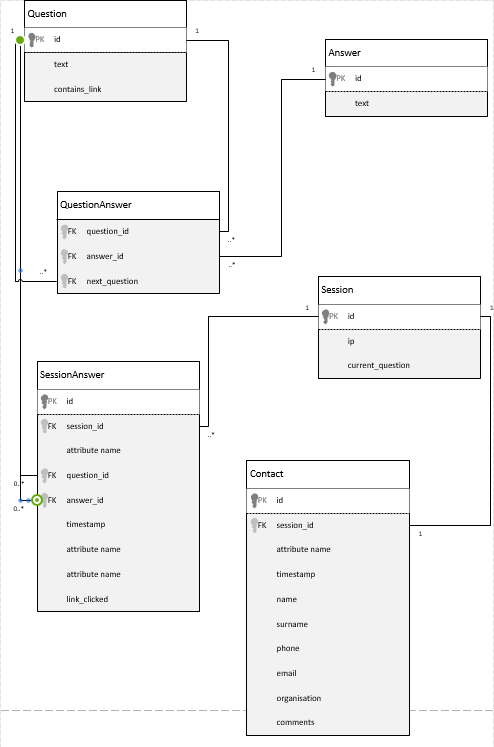
The project was designed for Digia, and is a chatbot designed for marketing, sales and data gathering purposes. It’s designed to be implemented into Digia’s own website and serves customers by asking a series of pre-defined questions. The questions are saved to a custom database, and the data is saved as reports for custom timeframes. A mailing address is used to store the reporting data.

# Process flow

## 



# Database structure



# Myyntirobotti files

## mainController.js

MainController.js is the controller for the Myyntirobotti project. It’s a javascript object that controls the data of the AngularJS applications. It handles session manipulation, answer processing and link formatting.

**Session fetching and creation**

Fetch the existing session or creates a new one upon starting the application. If a session exists the app moves to the question user was at when they last used the application.

**Functions**

|  |  |  |
| --- | --- | --- |
| **Name** | **Input/Output** | **Function** |
| ChatWindow.success | Input: question data | Main function of mainController.js. Handles session manipulation, and contains all functions mentioned below. This function runs if the first question has been fetched. |
| formSubmit | Input: form data  Output: N/A | Run when form is submitted. Sends the form information to backend for handling. |
| answerFunction | Input: current question id, current answer id,  current url (pathname) | Run when answer is pressed. Gets the next question from the database using the input values. |
| clearSession | Input: sessionid  Output: N/A | Clears the current session. |
| linkClicked | Input: Web or file link  Output: N/A | Run when a link is clicked. Sets the $scope.link value to true, which is sent to the backend to determine if the link has been clicked. Also opens the link in a new browser window. |
| findLink | Input: current question  Output: link | Takes in the current question and formats it as a link using linkTypes and fileFormats functions |
| linkTypes | Input: current question, type of link  Output: formatted link | Formats the given question’s link if the link contains “www” or “http” |
| fileFormats | Input: current question  Output: formatted link | Formats the given question’s link if the link contains “.pdf”, “.html”, “.jpg” or “.png” |

## appService.js

Fetches the first question and its answers.

## index.html

Index.html is the only webpage of the myyntirobotti project. Myyntirobotti is a single-page application, and index.html is manipulated through jQuery and Angular.js.

## style.css

Contains the styling of the webpage.

## bootstrap.css

Bootstrap.css is a style file that contains the default styling for bootstrap components.

## bootstrap.js / bootstrap.min.js

Bootstrap configuration is stored in these files.

## chatheads.js

Contains the configuration for the chathead. Uses javascript and jQuery functions to handle drag-and-drop functionalities and animations.

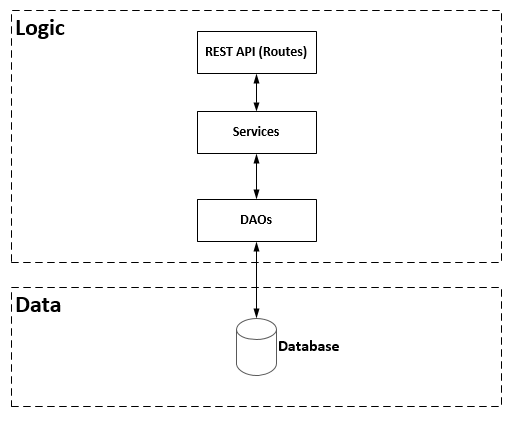
# Software Architecture

## Frontend

The frontend is built according to the principles of MVC using AngularJS.

## Backend

The backend architecture is similar to MVC but the presentation layer is handled using REST. The backend application is two tier architecture in which PostgreSQL is used for the data tier and the application tier is separated into routes, services and DAOs (database access objects). Routes handle the requests and respond accordingly. Services handle the business logic. DAOs access the database. Models are used to pass data between the different layers uniformly. However the model objects do not force type safety, which leaves some of the burden to the developer using the object. Other files include setup-file for the database, environment variables writer (setup.js) and the index file for starting the server and setting up rest of the backend.



## API Calls

|  |  |  |
| --- | --- | --- |
| **URL** | **Input/Output** | **Description** |
| GET /question/:id | INPUT: INTEGER (ID of the question)  OUTPUT: JSON (question, answers, contains\_link) | Retrieves a single question with its answer options from the database. |
| GET /create-session | INPUT: None  OUTPUT: INTEGER | Creates a session and saves it to the database. |
| GET /get-session/:id | INPUT: INTEGER (ID of the session)  OUTPUT: JSON (answers, current\_question) | Retrieves a session from the database using the given ID. Creates a new session if it does not exist. |
| GET /delete-session/:id | INPUT: INTEGER (ID of the session)  OUTPUT: HTTP Status 200 | Deletes a session with the given ID. Removes the reference to the session from the given answers. |
| POST /save-answer | INPUT: JSON (example:  {  “session\_id”: 1,  “question\_id”: 1,  “answer\_id“: 1,  “link\_clicked”: true  }  )  OUTPUT: HTTP Status 200 | Saves the user’s answer to the database. |
| POST /send-contact-request | INPUT: sessionId, name, surname, phone, email, organisation, comments  OUTPUT: HTTP Status 200 | Sends a contact request email. |
| GET /send-email-list | INPUT: None  OUTPUT: HTTP Status 200 | Sends a list of email addresses via email. |
| GET /send-report | INPUT: None  OUTPUT: HTTP Status 200 | Sends a report of answers via email. |