TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Lasha Amashukeli 166818IVSM

ONLINE PERCEPTION EXPERIMENTS

Master’s thesis

Supervisor: Einar Meister

Senior researcher, PhD

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Lasha Amashukeli 166818IVSM

VEEBIPÕHISED TAJUEKSPERIMENDID

Magistritöö

Juhendaja: Einar Meister
Vanemteadur

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Lasha Amashukeli

01.05.2018

Abstract

The following thesis, *Online Perception Experiments,* aimed to develop a web-based system for holding different types of perception experiments online. Online experiments have become a standard way of running experiments, in various fields: psychology, linguistics, phonetics, etc. Existing tools and frameworks, that support configuring and running these type of experiments online, don't do it in the most efficient way and lack several useful features.

To reach the aimed goal, a web application was developed which supports all the necessary functionality to properly hold such experiments online. It includes a centralized database system, backend application, and client web application. The system has individual user accounts for administrators, who design experiments and informants, who are participants of the experiments. Aggregated results are also managed through the web interface, including exporting or displaying into summarized graphical representation.

This thesis is written in English and is 42 pages long, including 6 chapters and 17 figures.

Annotatsioon

Magistritöö „*Veebipõhised tajueksperimendid*“ eesmärgiks oli luua veebipõhine rakendus erinevate tajueksperimentide läbiviimiseks. Veebiküsitlusi ja tajueksperimente tehakse mitmetes valdkondades – psühholoogias, keeleteaduses, foneetikas, jm. Olemasolevad vahendid ja keskkonnad, mis võimaldavad erinevaid teste koostada ja läbi viia, ei ole piisavalt paindlikud ja nende funktsionaalsus on sageli puudulik.

Töö raames töötati välja vajaliku funktsionaalsusega veebirakendus tajueksperimentide koostamiseks ja läbiviimiseks. See koosneb tsentraalsest andmebaasist, serverirakendusest ja kliendi kasutajaliidesest. Süsteem võimaldab luua individuaalseid administraatori (eksperimentide koostaja) ja kliendi (eksperimendis osaleja) kasutajakontosid. Eksperimentide tulemusi on võimalik hallata veebiliidese kaudu, sealhulgas on võimalik neid eksportida ja graafiliselt esitada.

Magistritöö on kirjutatud inglise keeles, see sisaldab 42 lehekülge, sealhulgas 6 peatükki ja 17 joonist.

List of abbreviations and terms

TUT	Tallinn University of Technology
OPEX	Online Perception Experiments
CI	Continuous integration
CD	Continuous deployment
WAET	Web Audio Evaluation Tool
CRUD	Create, Read, Update, Delete
UI	User interface
API	Application programming interface
Identification Experiment	Experiment to identify a single stimulus as one element of a set of categories
ABX,AXB Experiments	Experiment which consists of two known types of stimulus A and B and one unknown – X. Subject has to judge whether X is more similar to A or B.

Table of contents

List of figures

1 Introduction

Online experiments have become a standard way of running experiments, e.g. in psychology, market research, linguistics, phonetics, and other fields. In speech research, different perceptual experiments are performed by a large number of subjects. In these experiments, participants are asked for their subjective judgment of specific speech or language features, e.g. the acceptability of a given construct, length of a vowel in a word, a categorization of sounds, the perceived regional accent, etc. A number of frameworks that support configuring and running web-based experiments have been developed, although, all of them have different kinds limitations, what is the main motivation for developing the new system. Pros and cons of the existing systems are thoroughly discussed in Background and related work chapter[2] of the paper.

Currently, perception experiments in speech studies are performed in the laboratory, using open-source software "Praat". Although "Praat" is useful for constructing and running different experiments, it lacks many necessary features: is not very user-friendly – constructing experiment requires specific previous knowledge of the software, doesn't have centralized database and doesn't allow users to take tests remotely through the internet, doesn't have enough tools to manipulate with collected data etc., thus holding experiments with it is less efficient and makes process more complicated.

In order to make the process of holding perceptual experiments easier and more efficient, our research would aim to build a modern, reliable and convenient platform for this process. The platform would allow researchers to easily design a different kind of experiments which then would be stored in a centralized database and would be delivered to the clients (people who take part in the experiment) through the online portal. After collecting some data from the customers, the system should also allow the researcher to get overview/summarize results, look at the general picture and make relevant conclusions. (For detailed information about the specifications of the system, go to Technical specifications [3.1] chapter).

Building the platform described above will require designing a web-based application with two different interfaces/modules. One of them would be for the researcher and one for the client, both of them interacting with the same database and both of them having Web client-side UI, with back-end application running on cloud and functioning as a service. Collected data then should be analyzed and visualized using different data mining techniques, which must be one more module of the entire platform.

2 Background and related work

This part of the thesis contains the description of already existing work (systems and tools) in this field. When talking about a topic like audio perceptual experiment/evaluation tools, there's not a very big selection that is available online and that have proper scientific or research purpose. Thus, in this paper I will be discussing three relevant tools, which can be used for this purpose, referencing according papers:

- Praat – Desktop application, available for different platforms, including Mac, Windows, Linux and even FreeBSD and Oracle Solaris.

  Website: http://www.praat.org/

- Percy – an HTML5 framework for media-rich web experiments on mobile devices.

  Website: http://webapp.phonetik.uni-muenchen.de/WebExperiment

- Web Audio Evaluation Tool - A framework for the subjective assessment of audio

  Website: https://code.soundsoftware.ac.uk/projects/webaudioevaluationtool

- E-Prime - Commercial software solution, very comprehensive and feature-rich. Although, very highly priced and hardly affordable for research purposes.

  Website: https://pstnet.com/products/e-prime/

In the next section, I will be describing each of them separately, with advantages/disadvantages and current capabilities.

2.1 Praat

As mentioned above, Praat is an open-source desktop application developed by Paul Boersma and David Weenink from Phonetic Sciences, University of Amsterdam.

Developed mainly in C++, Praat supports basic features for generating various identification and discrimination experiments and later on executing this experiments and displaying results.

Generating experiments in Praat can be achieved by writing new Praat experiment file. This script is Praat-specific pseudo language that mainly consists of key-value pairs. Example snippet:

```
stimulusFileNameHead = "Sounds/"
stimulusFileNameTail = ".wav"
stimulusCarrierBefore = "weSayTheWord"
stimulusCarrierAfter = "again"
stimulusInitialSilenceDuration = 0.5 seconds
```

Figure 1. Praat experiment example script

This way, the basic perceptual experiment can be generated in Praat. It supports loading number of different stimuli (audio samples) and repeats them several times by random order during the experiment. Number of replications for each stimulus is one of the parameters in the experiment script as well as randomization logic type. Randomization can be:

- With replacement
- Permutation of all
- Balanced permutation

And several others, which is quite flexible and gives many possibilities.

Next step in creation experiment is entering question text and possible responses to the experiment question. A user needs to specify the number of responses and for each response enter following example script:

```
0.2 0.3 0.7 0.8 "h I d" 40 "" "i"
area, response category screen, font size, response key
keyboard, response category results
```

Figure 2. Upper line displays actual command, while lower line is an explanation of each parameter according to colors

As we can see, all of the parameters, including button location on the screen are specified in the command. It also supports some extra features, like displaying picture instead of text on the button, etc.

After specifying all the necessary options and parameters, experiment file can be saved and new experiment can be run from Praat menu.

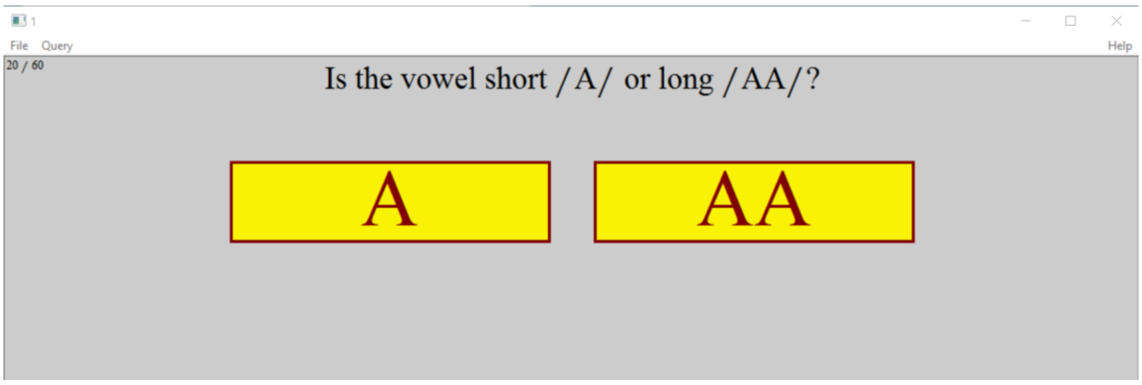


Figure 3. Example screen from running experiment

In an experiment displayed in Figure 3, there are 5 different stimuli loaded with 12 replications for each, thus 60 total listening/question for the user.

After going through all the samples and completing the experiment, results can be collected and displayed as a table or saved in a text file in the same table format as the one used for displaying.

As far as we can see, Praat gives us the ability to create various audio perceptual experiment with different audio samples and collect and save user responses in a specific way. Although, the software lacks some very necessary and user-friendly features. Firstly, creating an experiment is time-consuming and complex, requires from the user the knowledge of Praat-specific script commands, it can also very often lead to different kinds of syntax errors and runtime failures, as far as all the properties are written down manually in the experiment file and no syntax check is actually possible there. In addition to experiment creation difficulties, taking experiments isn't very user-friendly either in Praat. It doesn't support simple features, like, pausing experiments and resuming them after exiting Praat, saving the history of previous results or visualizing results in more meaningful and understandable way. Praat also doesn't support client-server interaction and remote execution of tests which is inconvenient for the researcher and makes almost impossible to held experiment against a large number of users. This would require every user to download and install Praat on a home computer and learn how to run experiments on it, which is extra effort and time, thus makes the whole process ineffective.

2.2 Percy

Percy is probably the closest example of what our thesis work aims to achieve. Unlike Praat, Percy offers perception experiments as a service, it supports the creation of experiment using simpler GUI interface and as a result, generates permanent URL which can then be distributed among the experiment participants. Percy was first deployed and used in 2012 in the Institute of Phonetics and Speech Processing, in Ludwig-Maximilian University in Munich, Germany.

For designing and configuring experiment, Percy uses PercyConfigurator, which is a single page web application, which allows an interactive definition of the experiment in the browser window. There are five easy steps to take in total to generate an experiment in Percy:

- Create experiment items

  This step includes creating a three-columned table, with item text in the first column, available options with input elements in second column and name of the media file in the third column. Available input elements include button list, slider, text field and text area.
- Edit participant form

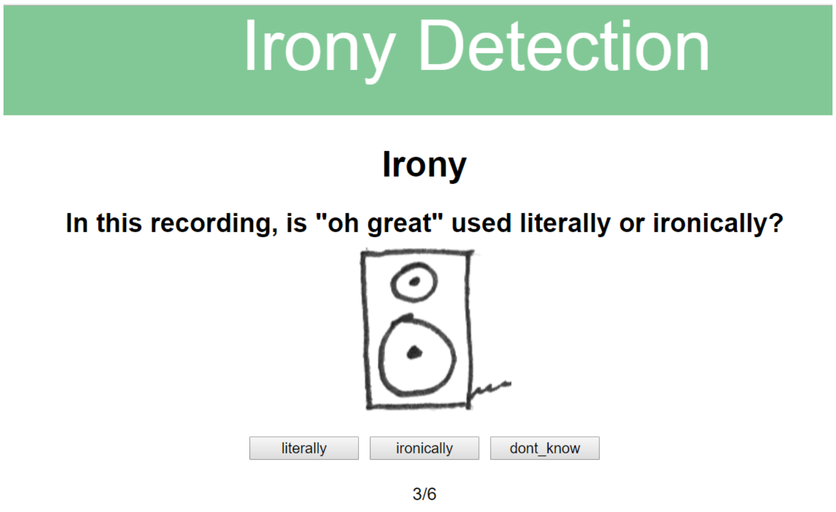
  Usually, before participants start with the experiment, they have to fill out a simple form with general information about them: age, gender, mother tongue etc. This step allows modification of this form and including necessary fields in the form.
- Test the experiment

  At this stage, you can run and test experiment displayed just as it would be when actually running in the browser.
- Upload the definition

  This step is for uploading media files and metadata about the experiment. E.g. experiment title, designer e-mail address, randomization of items and etc.
- Distribute the link

  Fifth and the last step is checking submitted experiment by Percy administrator and if accepted sending experiment link to the designer e-mail address. This step is intended for eliminating spam use of Percy.

Once the link to the experiment is available and distributed, participants can enjoy a user-friendly interface and complete experiments directly from home, their own web browsers.


Irony Detection

Irony

In this recording, is "oh great" used literally or ironically?

Figure 4. One of the available Percy experiments in progress

After completing the experiment, the summary is recorded in the Percy database and displayed to the participant screen as well.

To sum it up, Percy offers modern, interactive, software as a service solution for audio experiments. The feature that would make it even more usable and user-friendly would be personalized user system for participants and for experiment designers as well. This would allow participants to keep track of their history and always look back on how did they perform in the past, and designer-wise it would allow them to keep all their own experiments and results easily linked to their account. This is also one of the features which are aimed to be developed within the scope of our thesis work.

2.3 Web Audio Evaluation Tool (WAET)

WAET is one more open-source web-based tool developed by students of Centre for Digital Music, School of Electronic Engineering and Computer Science at Queen Mary University of London and students of Digital Media Technology Lab at Birmingham City University. Using WAET, wide range of listening test types can be easily constructed and responses visualized afterward. WAET supports different types of audio tests, including:

- AB Test
- APE Test
- ABX Test
- Pairwise Comparison (Better/Worse)

And several other types as well.

Like Percy, WAET also supports interactive test designer which is directly run into a web browser. On the test creation page using the special form, the experiment designer adds pages and audio elements one by one what leads to the creation of XML file with all the information about the experiment. Later this XML file needs to be uploaded to another page where experiments are actually run and the browser application runs the test by the instructions specified in the file.

Main advantages of WAET are an interactive designer of tests and also the capability of displaying summary after completing the experiment and visualization of results. This feature (most probably improved version) is also planned to be included in the final product of our thesis.

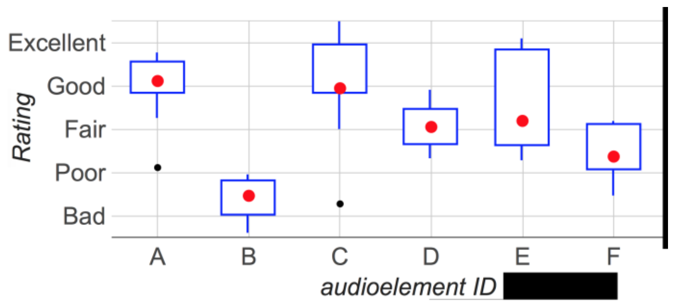


Figure 5. Box and whisker plot showing the aggregated numerical ratings of six stimuli by a group of subjects.

In general, WAET supports a wide range of test type construction and is usable basically for everyone, because of its intuitiveness and user-friendliness, although, parts of software still are not tightly connected to each other, doesn't support user accounts system and doesn't have any functionalities of a centralized platform or system in general. This is one of the key features that our work within the scope of this thesis will focus on.

3 Development

3.1 Technical specifications

Main parts of the application should include:

- A web interface for the administration of the experiments (for administrator) and collecting individual experiment responses (for informant).
- Server application
- Database

The administrator's interface allows to create new accounts and sign-in to the existing accounts using e-mail and password and to perform two tasks: (1) design an experiment, (2) manage the results.

Experiment designer should enable:

- Create new experiments, open, edit and save existing experiments
- Support of following types of experiment:
    - Identification experiment: the subject has to identify a single stimulus (e.g. sound or picture) as one element of a set of categories.
    - Rating experiment: the subject has to rate a stimulus on a discrete (e.g. Likert scale (with a maximum of nine points)) or on a continuous scale (e.g. using a slider from 0 to 100).
    - ABX and AXB experiments: a method of comparing two choices of sensory stimuli to identify detectable differences between them. A subject is presented with two known samples (sample A, the first reference, and sample B, the second reference) followed by one unknown sample X that is randomly selected from either A or B. The subject is then required to identify X as either A or B[1].

[1] https://en.wikipedia.org/wiki/ABX_test

- Define the workflow and behavior of the test. For example, automatic or manual switch to the next stimulus, interstimulus interval[1], etc.
- Upload and store necessary stimuli files.

Result management should enable:

- Access the results of the experiments created by the administrator
- Export the results and informants' data.
- Generate summaries and graphical representation of the results

The informant interface should enable:

- Creation of new informant accounts and sign-in to the existing accounts using e-mail and password.
- Personal profile information, including gender, year of birth, native language.
- Choosing the experiment in which the informant wants to participate.
- Run chosen experiment, stop and save the current progress of the experiment.
- View history of taken experiments.

3.2 General Architecture

In order to achieve the set goal, I decided to build a web application using classic client-server architecture. The main parts of this application should include following:

- Persistent layer - Database, where all the persistent data, including, experiments, results, user information etc. should be stored.
- RESTful Web API - Backend application, taking inputs as HTTP requests and processing them.
- Front-end web application – App for actual interaction by users (researchers and participants), displaying and submitting the data. Runnable on all modern browsers.

The main purpose of using REST API and building each part of the application independently is, to make the code highly reusable and extendible. For instance, the

[1] https://en.wikipedia.org/wiki/Interstimulus_interval

current architecture would easily allow developers in future to integrate native mobile application as a front-end part of the system.

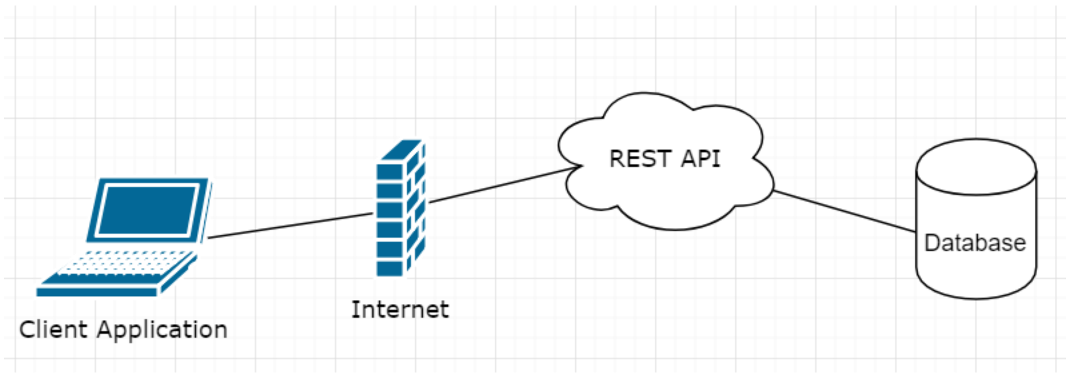


Figure 6. General architecture diagram of the system

3.3 Selected Technologies

When selecting technologies, I tried to use most up to date technologies, in order to ensure following some of the best existing practices in the process and build scalable, testable and maintainable system. These are the technologies, which were used during the development:

- ReactJS – Very powerful and trending JavaScript library for building single-page user interfaces. React aims primarily to provide speed, simplicity, and scalability.
- ASP.NET Core – Free and open-source implementation of Microsoft's famous ASP.NET framework. ASP.NET Core offers many useful and developer-friendly features, which allow building scalable, cross-platform SAAS applications.
- MS SQL Server – Database of choice for ASP.NET applications. Microsoft's powerful relational database management system.

Some other tools and technologies:

- Git – Source control management system.
- Docker containers for easier deployment process.
- Jenkins – Automation tool for building and publishing software.
- Heroku – Deployment platform. Used to publish front-end application.

3.4 CI/CD Pipeline

The very first step of the development process was setting up continuous integration and deployment pipeline (CI/CD) to avoid lots of manual work for deployments and always have a stable version up and running for testing/demonstration purposes. In order to achieve this, Jenkins automation tools and Docker images were involved in the process. The steps of the process are as follows:

Back-end Web API:

1. The code is pushed/merged into specific git branch. (Branch named "develop" was used in this case as the main development branch.)
2. Jenkins polls GitHub repository periodically by some time interval and in case of updates, builds the code, runs unit tests and if successful, packages new Docker image from the binaries. The Docker image is then pushed to Docker hub with the new version number.
3. An application server (Linux server running Docker container of our Web API), has small Docker container, watchtower[1], running which checks for a new version of the image in Docker Hub and reruns container with an updated image if a new version is available.

Thus, every push to the develop branch of GitHub repository results in an updated version of the application running on our dedicated application server.

Front-end ReactJS application:

1. The code is pushed/merged into git branch.
2. The code is pushed to Heroku repository.
3. Heroku builds and publishes a new version of the application to the publicly accessible server.

[1] https://github.com/v2tec/watchtower

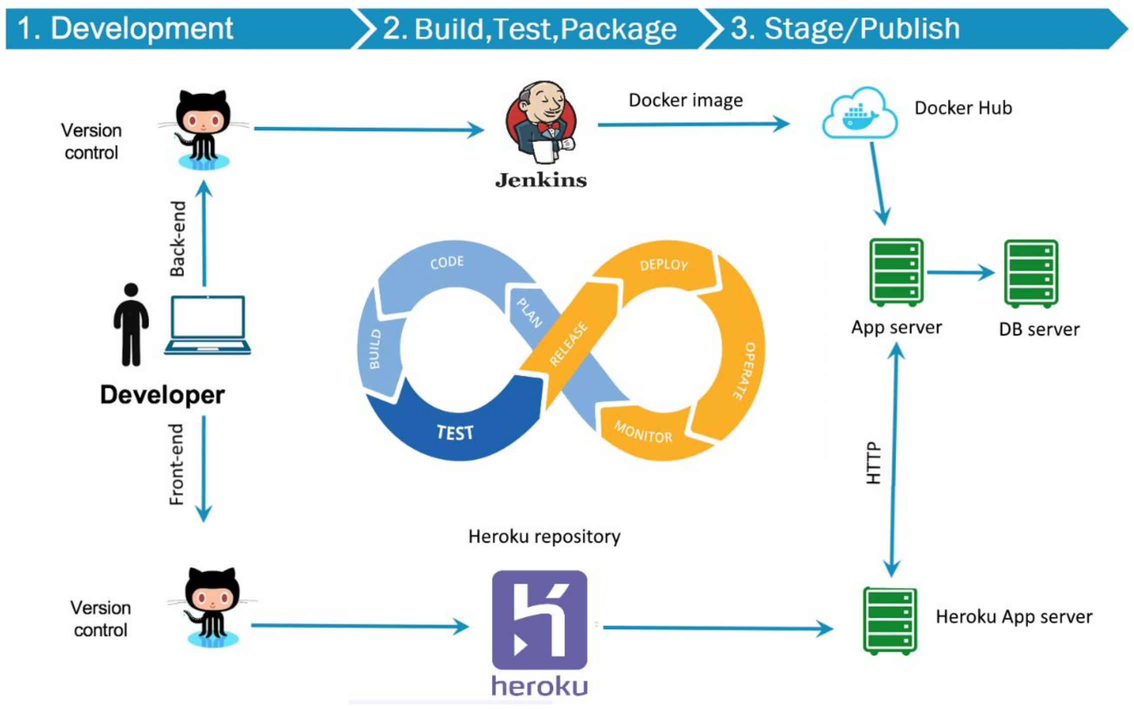


Figure 7. Diagram describing used CI pipeline used in the development process.

This kind of approach gives an opportunity to constantly have code tested, deployed and running throughout the development process. Having tuned CI/CD pipeline is an essential part of Agile Software Development methodology.

3.5 ASP.NET Core (back-end application) architecture

ASP.NET Core framework implements MVC (model-view-controller) pattern, thus the architecture of our Web API, mostly, follows this pattern. It consists of following layers:

- Models – Contains all the domain model classes used in the application.
- Persistent – Contains logic to interact with the storage (MS SQL Server database), create, update and delete rows in the database. Internally uses Entity Framework Core[1] as an ORM to interact with the database.
- Services – This layer contains the main business logic of the application. Interacts with the Persistent layer to read/update data.
- WebApi – This is the presentation layer of the application. It contains Controller classes with Action methods which are mapped to the specific routes. This layer

[1] https://github.com/aspnet/EntityFrameworkCore

interacts with the Service layer to process the request and return the response to the client.

- Tests – Unit test projects that are executed on every build of the application.

The project uses DI containers to resolve dependencies, thus all the layers/classes of the application interact with abstractions of each other. The domain model of the application consists of 8 different classes which are mapped to corresponding 8 tables in the database.

3.5.1 Domain Model

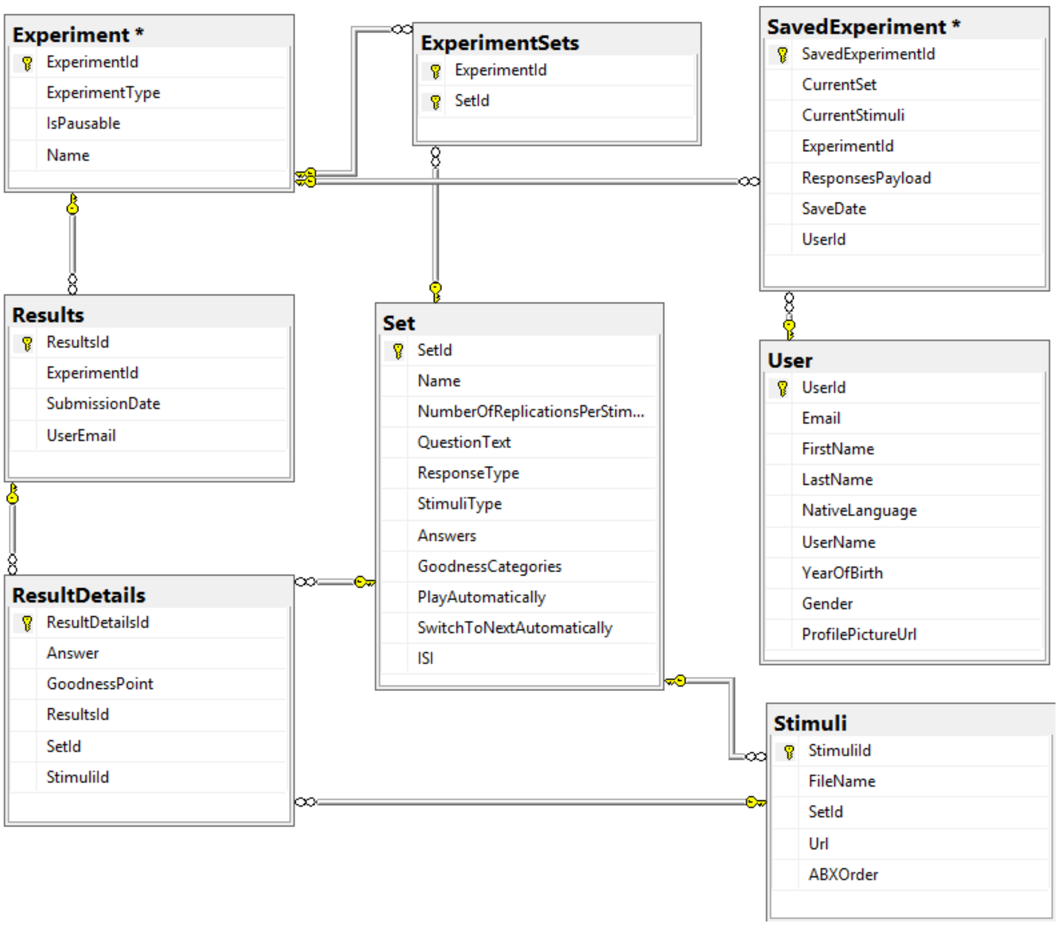


Figure 8. Relational model of OPEX database

Experiment, Set, Results, SavedExperiment and User tables are primary tables which directly map to the business objects.

- Experiment – table contains experiment metadata: Name, type, option of pause/resume.

- Set – contains set of stimuli, question text, possible answer and several other stimuli related properties. Has a many-to-many relationship with Experiments table.
- Results – keeps records of completed experiments with answers to each question. Details per set are kept inside ResultDetails table, which has a one-to-many relationship with Results.
- SavedExperiment – This table keeps track of in-progress experiments which were chosen to be saved and continued later.
- User – Contains registered user data. Passwords are not stored in this table though; they are managed by Auth0 service. (See detailed information in Authorization[3.6.2] chapter).

Other tables are supportive tables which help to correctly reflect schema of the domain model. ExperimentSets, for example, is an associative entity, which gives possibility having a many-to-many relationship between Experiments and Sets. This, in turn, makes Sets reusable across different experiments.

3.5.2 Persistent layer

All interactions with the database are done using Entity Framework (object-relation mapper) in the persistent layer of the application. Entity Framework (the code-first approach in this case) uses entity classes to generate database schema and generates migrations for every future schema change. These migrations are run automatically after deployment to the staging server, thus removes the necessity of manually running SQL scripts on the database machine. It also helps to avoid hardcoded queries in the application code and generates them based on LINQ[1] queries. This logic is encapsulated inside Repository classes and their abstractions are exposed publicly to the service layer.

[1] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

```
dbContext.Results.Include(x => x.ResultDetails)
                              .ThenInclude(x => x.Stimuli)
                              .Include(x => x.ResultDetails)
                              .ThenInclude(x => x.Set)
                              .Include(x => x.Experiment)
              .SingleOrDefault(x=>x.ResultsId==results.ResultsId);
```

Figure 9. Sample code of a LINQ query used in Results Repository class.

Above code represents LINQ query, which is used by Entity Framework to retrieve necessary records from the Results table. It tells the compiler to retrieve Result record joined with Experiment and ResultDetails record. ResultDetails, in turn, must be joined with Stimuli and Set tables. The final line of the query tells the compiler to apply the filter by ResultsId column of the table. As seen above, this quite complex query was written by much simpler and shorter C# code. Avoiding necessity to create stored procedures in the database or hardcode SQL queries in the application code.

3.5.3 Service layer

Services layer contains the main business logic of the application. Main responsibilities include translating model objects submitted from the controllers into domain objects and vice versa and calling corresponding repository methods in order to persist the data or retrieve one existing in the database already. It uses Google Cloud Storage API[1] as an external service to save and manage the stimuli files uploaded by the users. Only URLs of the files are stored in SQL database.

3.5.4 Presentation layer

Web API (presentation) layer, as a RESTful service, contains the separate controller for each resource (entity). Each controller has an instance of the service class registered which is used to process actions requested by the client. Following five controllers are used to handle all the requests:

- ExperimentController – Responsible for CRUD operations for Experiments. In some cases, requests may handle persisting nested Set objects which are part of the Experiment.
- SetController – Separate controller for managing Set records. Contains actions responsible, for example, retrieving Sets by parent Experiment Id, etc.

[1] https://cloud.google.com/products/storage

- ResultsController – Responsible for saving/retrieving completed experiment result objects.
- SavedExperimentController – Separate controller for not complete experiments, which are still in progress and which were saved in the current state to continue later.
- UserController – Controller responsible for creating and updating application user data.

```
[Produces("application/json")]
[Route("api/Experiment")]
public class ExperimentController : Controller
{
  private readonly IExperimentService experimentService;

  public ExperimentController(IExperimentService experimentService)
  {
    this.experimentService = experimentService;
  }
  // POST: api/Experiment
  [HttpPost]
  public async Task<IActionResult> Post([FromBody]Experiment experiment)
  {
      if (ModelState.IsValid)
      {
       var addedExperiment = await
                                experimentService.SaveExperiment(experiment);
       return Ok(addedExperiment);
      }
      return BadRequest(ModelState);
  }
}
```

Figure 10. Example of ASP.NET Core controller method with POST action method

Above code, sample demonstrates part of ExperimentController with single POST action method inside. C# attribute classes are used to define the route of the controller "api/Experiment" and the format of the response "application/json". The class inherits from AspNetCore.MVC base controller class. Dependency injection pattern is used to inject single dependency of the controller, which is ExperimentService class, containing all services that can be performed on the experiment objects. This service is defined as a read-only field in the controller and is initialized from the constructor. The action method is itself very simple and contains only single call to the service method. It also uses the attribute, in this case defining verb of the request – POST. As far as it has no route attribute defined, POST requests by default will be directed to this action, meaning POST to “api/experiment” would invoke method described in the code. The method logic itself only validates the model state and if all the required fields are present invokes the service method call which then later creates posted Experiment object in the database. If the model state is invalid, error code 400, bad request, is returned to the caller.

All action methods, which return data, format responses in JSON format and all POST actions, which take complex objects as a parameter, use JSON formatted objects as well.

```
{
    "experimentId":5038,
    "experimentType":2,
    "isPausable":false,
    "name":"Identification with rating experiment",
    "sets":[
        {
            "setId":3006,
            "name":"Set1",
            "stimuliType":0,
            "questionText":"Did you hear A or O?",
            "responseType":0,
            "numberOfReplicationsPerStimuli":2,
            "playAutomatically":true,
            "switchToNextAutomatically":false,
            "isi":1,"stimulis":[],
            "possibleResponses":["A","O"],
             "goodnessCategories":["1","2","3","4","5"],
            "stimulisInfo":[
                {
                    "stimuliId":2035,
                    "fileName":"a-to-o1.wav-636581015460797782",
                   "url":"https://storage/a-to-o1.wav",
                    "setId":3006,"abxOrder":null
                }
            ],
            "stimuliAInfo":null,
            "stimuliBInfo":null,
            "shuffledStimulisId":[2035,2036,2035,2036]
        }
    ],
    "setsCount":1
}
```

Figure 11. Example JSON body of Experiment PUT (update) method

3.6 React JS (front-end application)

Front-end application, as a classic React/Redux application, consists of Store, Reducers, Components and Actions. Components contain main UI elements of the application and user interactions in these components, like submitting forms, pressing buttons, selecting different options, filtering lists etc., invoke specific Actions. Actions, in most cases, make some HTTP requests to the back-end Web API, and using results returned from the requests update store. Store contains global state of the application. All the persistent data which was loaded from the back-end database is saved in the store. Reducers are responsible for updating the store.

3.6.1 Components

Components are grouped hierarchically according to their functionality. There is a total of 10 component groups:

- Common – contains general, non-feature-specific components like text and number inputs, Boolean checkboxes, dropdown lists and several other types of inputs.
- Experiments – components for managing experiments, like existing experiments list and edit forms.
- Sets – set forms for creating different kinds of sets. (Identification, ABX, AXB, etc.)
- Experiment player – components for executing created experiments and collecting results from the user. Contains components like answer selectors, value sliders etc.
- Results – contains list components for results and export component to allow users exporting data in the spreadsheet file.
- History – Single component for displaying all previously taken experiment of the user.
- Statistics – Contains controls for displaying various graphs based on experiment results. Uses Chart.Js[1] internally.
- Profile – contains controls for updating the user profile.

[1] https://www.chartjs.org/

- Documentation – Single component mainly containing markup text.

React router is used for navigation between different parts of the application. As mentioned before, React uses a single-page application approach, thus all the parts of the page are loaded dynamically using react-router and JavaScript. Unauthorized users aren't able to access any part of the application.

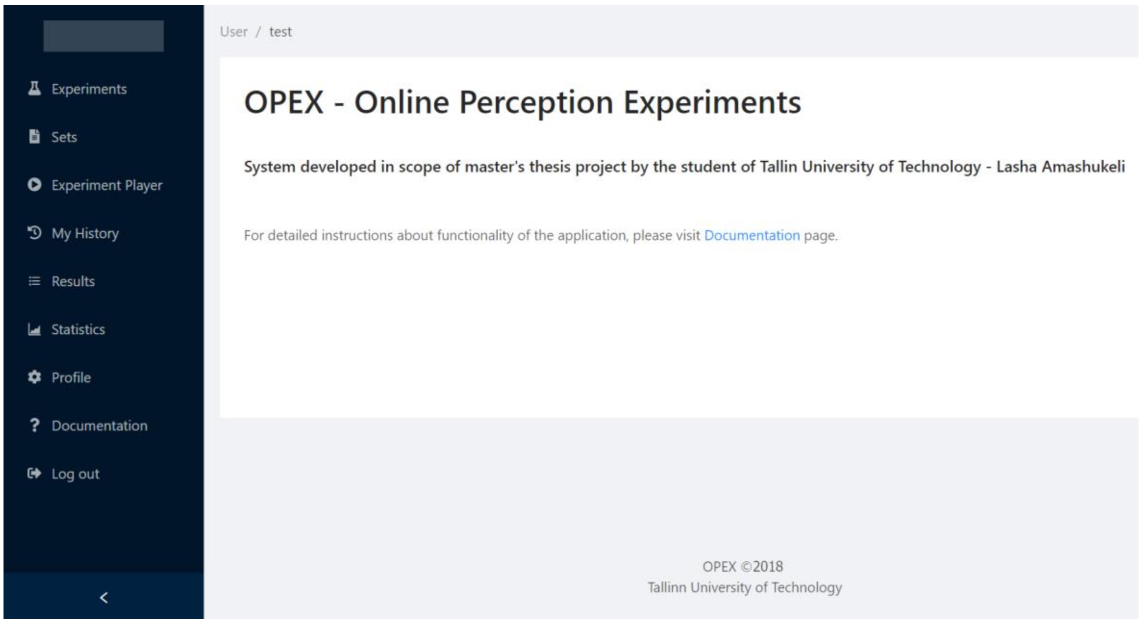
3.6.2 Authorization

Registration and Authorization part is handled by the Auth0[1] service. Auth0 is an authentication-as-a-service solution which integrates easily into almost every type of application and helps developers to avoid implementation of their own, complicated and less secure authorization mechanism. Auth0 uniquely identifies its users using the E-mail address, which in our case is also used to map authenticated users with the user records in the OPEX database.

[1] https://auth0.com/

4 Product

Application user interface, from the very beginning, was meant to be simple and intuitive, achieving necessary functionality without overcomplicated views and forms. Application navigation system is divided into 7 main items and one documentation page, describing how to use the basic functionality of the system.



OPEX - Online Perception Experiments

System developed in scope of master's thesis project by the student of Tallin University of Technology - Lasha Amashukeli

For detailed instructions about functionality of the application, please visit Documentation page.

OPEX ©2018
Tallinn University of Technology

Figure 12. Homepage of the OPEX front-end application

4.1 User management

As already mentioned in the previous chapter, authorization feature is implemented with the help of Auth0 service, which keeps passwords and emails of each user and returns the unique identifier to the client application after login is successfully completed. These unique identifiers are then mapped with the user records inside OPEX database. In the "Profile" section of the application, users can update their personal information, including first name, last name, year of birth, gender, and native language. This information is optional for every user and in case of presence can be used in further analysis of the experiment results. Every experiment and experiment result is linked to the specific user in the database. This lets us keep the history of taken experiments for each user and also let the researcher know, who was the author of particular trial in the experiment results data.

4.2 Create/Edit experiments

This part of the application allows users creating new experiments and managing existing ones. Creating experiment includes entering experiment metadata and sets of stimuli. Experiment metadata form includes following fields:

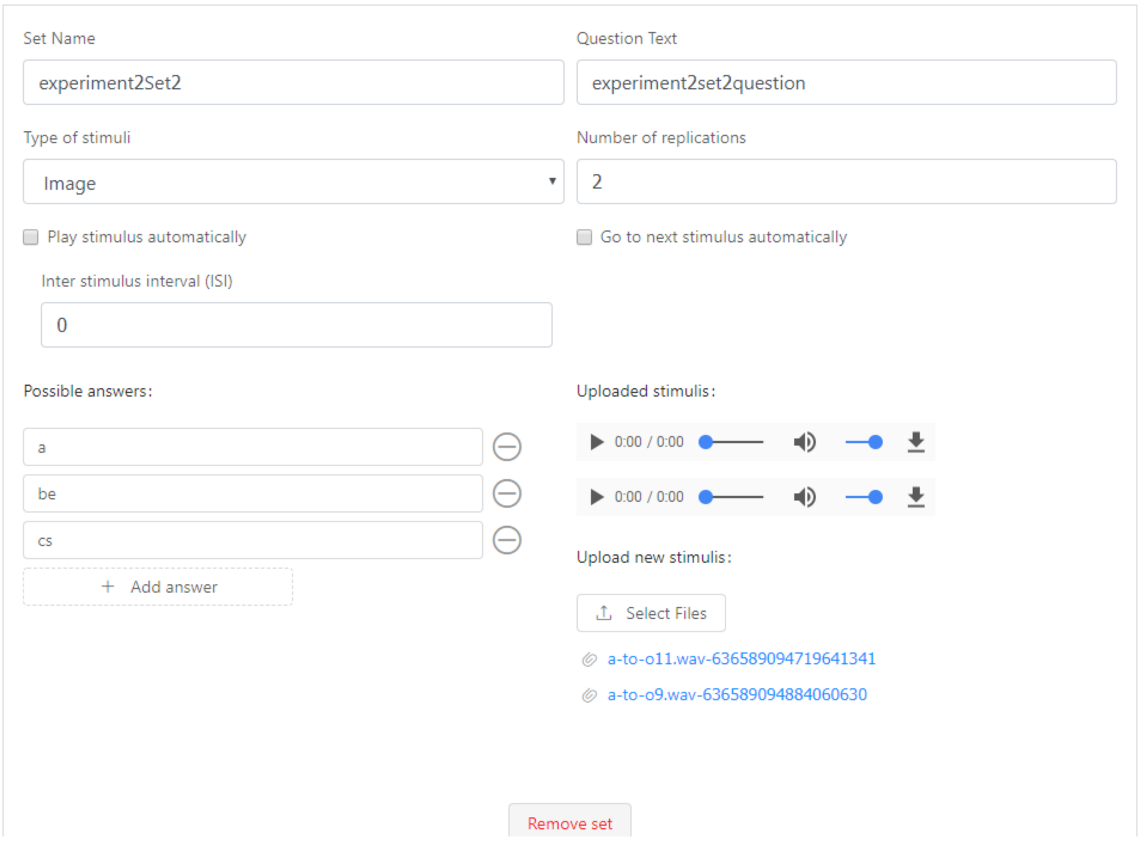
- Experiment name - Name of the experiment, can be anything, helps to easily identify saved experiment in the future.
- Experiment type - Type of the experiment, this can be following 4 currently: identification, identification with the rating, ABX, AXB.
- Can be paused - Possibility of pausing and resuming experiment later

Pre-selecting experiment types allows the system to render relevant form according to the type i.e. if the user selects "ABX" as an experiment type, form with separate controls for A, B, and X stimulus upload will be rendered.

One experiment can have multiple stimuli Sets attached and each of this Sets can be reused in other experiments, as well as explored/edited independently. Exploring/editing sets are possible from the “Sets” section of the navigation.

4.3 Sets

Sets form lets the user enter set of stimuli and related question with possible answers and several other properties. Set forms are divided per experiment type, including Identification, Identification with rating, ABX and AXB types. Set forms are embedded inside experiment form, when building new experiment or editing existing one and can also alternatively be edited as an independent object in the “Sets” section. This section also displays all available “Sets” from the database.



Set Name: experiment2Set2

Question Text: experiment2set2question

Type of stimuli: Image

Number of replications: 2

☐ Play stimulus automatically

☐ Go to next stimulus automatically

Inter stimulus interval (ISI): 0

Possible answers:

- a
- be
- cs

Uploaded stimulis:

Upload new stimulis:

a-to-o11.wav-636589094719641341

a-to-o9.wav-636589094884060630

Figure 13. Screenshot of the Set edit form in OPEX

Following inputs are required in order to create the Set:

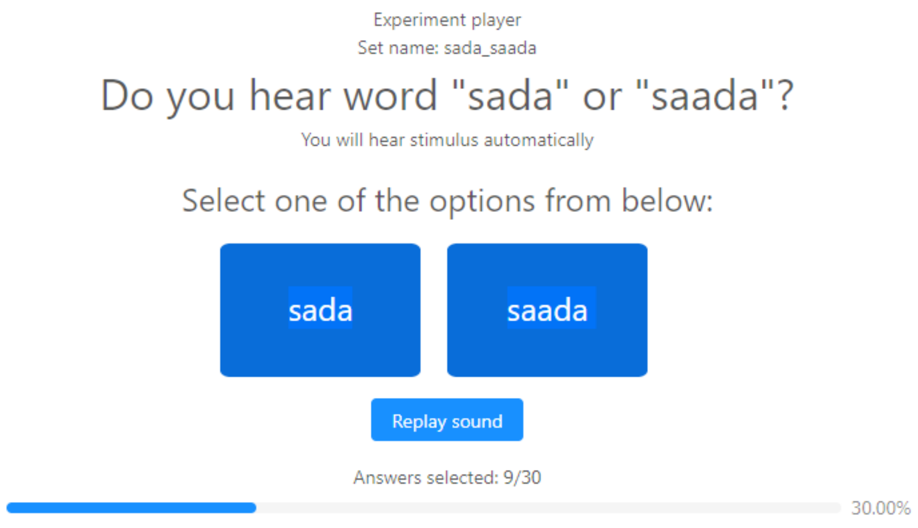
- Set Name - Name of the set, can be anything, helps to easily identify saved set in the future.
- Question Text - Question text of the set.
- Type of stimulus - Type of stimuli: either Sound or Picture. (Only sounds supported in the current version).
- Number of replications - Indicates the number of how many times each stimulus should be played.
- Play stimulus automatically - Indicates whether stimulus should be played automatically when the question is displayed. In case of false, button click will be required to play the one.
- Go to next stimulus automatically - Indicates whether the system should switch to next stimulus automatically after the answer to the current one is selected. In case of false, Next button click will be required.
- Possible answers - Possible answers to the question of the set. Can be added as many as required. Applicable for Identification and Identification with rating type experiments.

- Stimuli - List of stimuli to be played. Can be added as many as required.
- Goodness categories - Values for goodness point slider. Applicable for Identification with rating type experiments.
- Stimulus A - A stimulus for an ABX experiment. Applicable for ABX and AXB type experiments.
- Stimulus B - B stimulus for an ABX experiment. Applicable for ABX and AXB type experiments.

4.4 Experiment Player

Experiment player section is the place from where saved experiments can be executed. Experiments can be started either by selecting one from the list or by going to specific URL for the selected experiment. URLs are also displayed on the list of experiments on the “Experiment Player” page.

The player component itself is a simple view, displaying the Set question, Possible answers and some instruction notes for the user. In case of the specific experiment type, some extra controls can be displayed, for example, slider to select rating point for the answer in case of Rating type experiments. For every new trial, stimuli are shuffled in the random order and played by the times entered in the set form, meaning set with 6 stimuli and 3 repetitions would contain 18 questions in total. Progress bar control is also presented on the player page to visualize progress during the experiment process.


Experiment player
Set name: sada_saada
Do you hear word "sada" or "saada"?
You will hear stimulus automatically
Select one of the options from below:

Answers selected: 9/30

Figure 14. Example screenshot of Identification Experiment in progress

Experiment player also allows saving current progress and continuing it later, if this option was ticked as “true” by experiment designer.

4.5 History

Every user in the system has its own history record of taken experiments. History section displays the full list of previously taken experiments of the current user, including information like Experiment name, type, id, and date of submission. History list also displays experiments which weren't hadn't been completed yet, but have been saved by the user to complete later.

4.6 Results

Results page aggregates all experiment results taken by any user. Result records contain all the details of the individual experiment results, including experiment metadata and all stimuli alongside with chosen answers. The system also supports exporting desired records into a spreadsheet format and downloading them.

4.7 Statistics

All the individual results which are recorded in the system need to be aggregated and analyzed in some way. Statistics page groups results by experiments and displays summarized results for each stimulus.

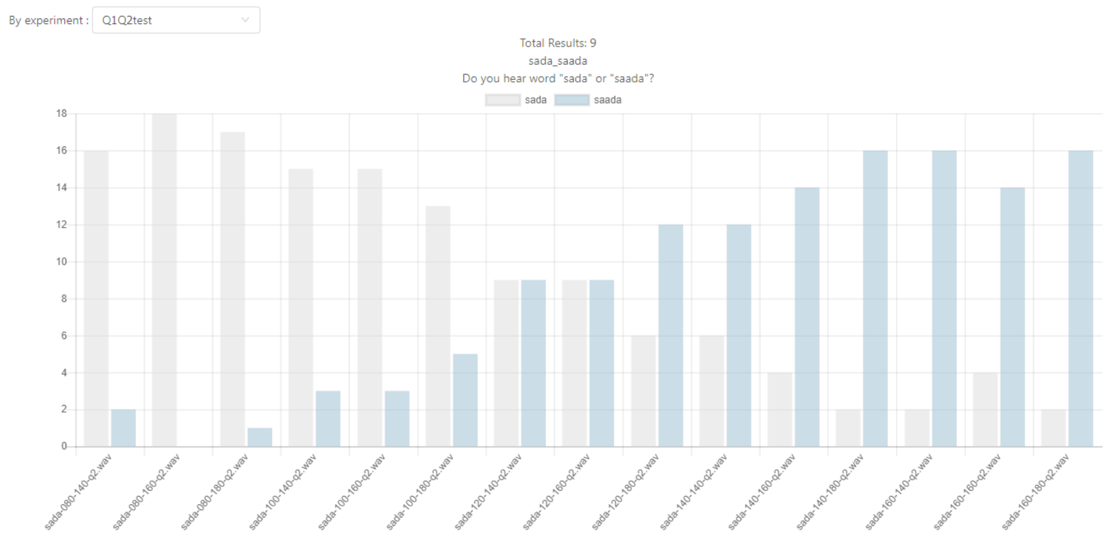


Figure 15. Graph from statistics page, visualizing Identification experiment results.

In the figure above, we can see a graph of Identification experiment results. There are nine different results recorded by different users for this experiment (indicated on top of the screen). The aim of this sample experiment is to distinguish between two Estonian words: “sada” and “saada”, containing nine different audio stimuli. Each column on the graph represents the stimulus file, ordered in respective order. In this example, the most left one represents the one closest to “sada” and most right one is closest to “saada". Each column has 2 bars differentiated by colors, representing two different answers for this set: grey one represents "sada" and blue one represents "saada". Looking at the graph, the researcher can clearly see the pattern. The number of "sada" responses decreasing and the number of "saada" responses increasing from left to right. Middle columns have an almost identical number of answers for both, as far as they aren't very close to either of the possible answers.

In case of experiments containing two or more Sets, the separate graph for each set is displayed.

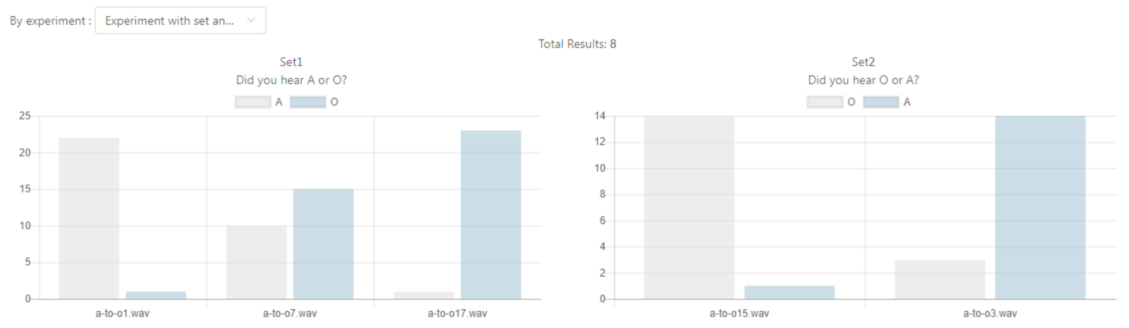


Figure 16. Example statistic page for experiment containing two stimulus sets

This module of the system can be improved in many ways and can become very useful when making some conclusions based on results collected. Currently, it only supports basic summaries, grouped by experiments. This can be further extended to include visualization of results for specific users, filtering by native language, age, gender etc. or for specific sets only, in case the set is reused in several different experiments. Results stored in the database have direct relations with users, sets, and their other properties, what makes it very easy and convenient to work with them and execute different queries.

5 User feedback

For evaluating carried out work and receiving feedback from users, a number of people, who were unfamiliar with the system, were asked to interact with it.

5.1 Test experiments

Two sample experiments were created for this purpose: one Identification experiment with rating points and one simple Identification one. The first experiment contained one set with audio samples of Estonian vowels “ö” and “õ”, containing 9 stimuli files with 3 repetitions for each of one, thus 27 questions in total. The second experiment contained also one set, with words as a stimulus, instead of single vowels. Purpose of the second one was to differentiate length of the vowel "a" in words "sada" and "saada". This set contained 15 different audio files and each of them was played 2 times during the experiment.

5.2 Process of testing

About 10 selected people, were asked to take part in the experiments and to give their opinion about the system. The main focus of the feedback was made on how user-friendly and usable the system was and what general impressions would unfamiliar people have on it. The process of the testing was as follows:

- Users had to create an account in the system.
- Both experiments mentioned above had to be completed fully.
- After completing experiments, participants had to fill a special form (5.3), with questions about their experience with the system.

5.3 Feedback Form

Feedback form contained 5 different questions about usability/user-friendliness with an option to select 5 different rating points: from 1 to 5. And, additionally, one open question for any suggestions or remarks from the users.

Following list displays 5 criteria of assessment and 5 according questions which were used for this assessment:

- **Intuitive design** - How effortless was it to understand architecture and navigation of the site.
  Question: How easily did you understand the architecture and the navigation of the site? -
- **Ease of learning** - how fast a user who has never seen the user interface before can accomplish basic tasks.
  Question: How fast were you able to accomplish basic tasks (listening to samples, selecting answers)?
- **Memorability** - after visiting the site, if a user can remember enough to use it effectively in future visits.
  Question: How well would you remember how to use the site in case of a future visit?
- **Error frequency and severity** - how often users make errors while using the system, how serious the errors are, and how users recover from the errors.
  Question: How often did you make errors while using the system? (pressed wrong button, forgot to select answer)
- **Subjective satisfaction** - If the user likes using the system.
  Question: How would you rate overall user-friendliness of the system?

And one open question: Any specific remarks/suggestions, how to improve user experience?

URL to the actual google form which was used in the process: https://goo.gl/forms/G4S1KnXLKuVVjVje2

5.4 Results

To summarize the results received after the survey, overall user-experience and satisfaction seem to be positive.

Overall user-friendliness of the system was rated 5 points by the majority of the participants (58.3%) and 4 by (33.6%) with the average point 4.5 out of 5. The most positive feedback was received for the memorability question, which received 5 points from about 66% of the respondents and 4 points by only 25% of them, average point for this question was almost 4.6. Error frequency and severity and architecture and the navigation were also rated very positively with 75% rating architecture and navigation with maximum points and 50% stating that their error frequency was very low. A relatively low point was given to ease of learning question, compared to others, although the average point is still high, with 4.42 out of 5.

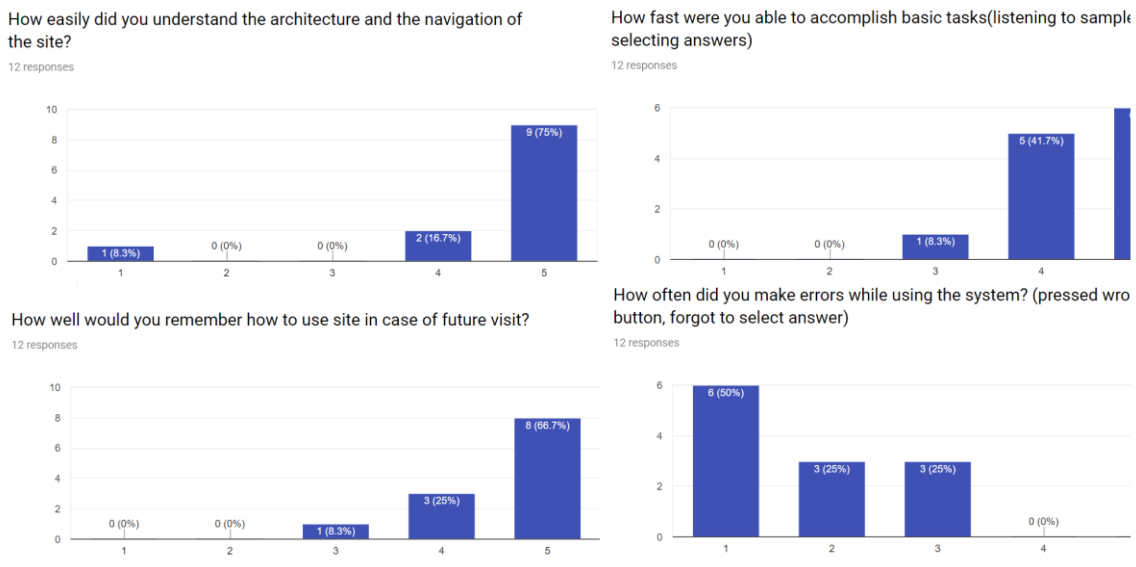


Figure 17.Summary page of feedback form responses

6 Summary

The following graduation thesis, *Online Perception Experiments*, aimed to develop a web-based system for holding different types of perception experiments online. Existing systems and applications which accomplish the same goal as the current thesis does, don't do it in the most efficient way and lack several useful features which could make creating, holding and analyzing different kinds of perceptual experiments lot easier and convenient. The main functions and features on which the carried out work focused included: individual user account system for researchers and general users, centralized database for storing all experiments and results of these experiments, web-based user-friendly UI runnable on all modern browsers, which supports managing experiments and running them, and finally, possibility of summarizing and visualizing collected results.

To reach the aimed goal, a classic web application was developed, which includes a standalone relational database system, backend application, and client web application. The development process was done using modern tools and technologies, as well as with modern development approach, including continuous integration and continuous deployment methodologies. Main technologies used for the system development were: MS SQL Server for the database, ASP.NET Core and C# for RESTful Web API and React JS for a single-page client-side application.

The final product, developed within the scope of the thesis, contains all the main modules and features which were initially aimed to be present in the system, including:

- Individual user account system
- Module for creating and editing experiments, which supports reusable stimuli sets and several different types of experiment templates.
- Module for holding/playing experiments in the browser, supporting features like saving and resuming sessions.
- Possibility for users to track their experiment history.
- The possibility of researchers to overview and export results, as well as an option to take a look at a summarized picture, including statistical graphs based on all results recorded in the database.

Besides functional requirements, results of the user feedback showed, that initial aim of building user-friendly system was mainly accomplished as well. Although, there are still different features that could be added to the system and that couldn’t be fully fit in scopes of this thesis work, including statistical analysis of results, which, potentially, could include several other types of reports and graphs, based on the need of the specific experiment or its author.

References

[1] C. Draxler, “Percy – an HTML5 framework for media rich web experiments on mobile devices,” in Proc. Interspeech, Florence, Italy, 2011, pp. 3339–3340.

[2] ——, “Online experiments with the Percy software framework – experiences and some early results,” in Proc. LREC, Reykjavik,

[3] ——, “PercyConfigurator – Perception Experiments as a Service.”, INTERSPEECH 2017: Show & Tell Contribution August 20–24, 2017, Stockholm, Sweden

[4] Nicholas Jillings, Brecht De Man, David Moffat, Joshua D. Reiss, Ryan Stables – “Web Audio Evaluation Tool: A framework for subjective assessment of audio”

[5] Boersma, Paul & Weenink, David (2018). Praat: doing phonetics by computer [Computer program]. Version 6.0.39, retrieved 3 April 2018 from http://www.praat.org/

[6] ABX-Testing - https://en.wikipedia.org/wiki/ABX_test

[7] Summary Praat: ExperimentMFC - http://www.coli.uni-saarland.de/~juegler/courses/praatscripting/SoSe13/summary_MFC.pdf

Appendix 1 – Source code and access to the system

Source code repositories:

- Backend application: https://github.com/Duke-fleed/OPEX
- Frontend application: https://github.com/Duke-fleed/OPEX-React

Public URLs to access the application:

- OPEX portal (main client application): http://opex-portal.herokuapp.com/
- REST API Swagger endpoint: http://193.40.251.30/swagger/