Bilkent University

Department of Computer Engineering

CS 491 - Senior Design Project

Project Name: Deeplay.io

High Level Design Report

Team "*Ludens*"

Project Group Members:

F. Serdar Atalay, Ekinsu Bozdağ, Gökcan Değirmenci,

Onur Sönmez, Gökçe Özkan

Supervisor: **Prof. Dr. Halil Altay Güvenir**

Jury Members: **Asst. Prof. Dr. Shervin R. Arashloo, Prof. Dr. Uğur Güdükbay**

Innovation Expert: **Mustafa Sakalsız**

Submitted at December 31, 2018

TABLE OF CONTENTS

1. Introduction

1.1. Purpose of the System

In the field of computer science, there are many companies, engineers and data scientists that extensively use deep learning. As the amount of necessary and appropriate data to be used during training increases, the performance increases with deep learning algorithms[1]. With the current advancements on machine learning, especially on its subset area of deep learning, intelligent applications become extremely data-hungry. The need for uncompromised quality, actionable data increases everyday. That is to say, data becomes the “new oil” of our era. People label their data to do training and they need more and more data to get the most accurate results for their systems. By bringing crowdsourcing and gamification concepts, our project *Deeplay* promises entertainment to people, while making them help those who are interested in or working on deep learning. *Deeplay* provides a game platform in which there will be different categories of games that mainly aims to make users label data.

1.2. Design Goals

1.2.1 Usability

- The target customer of deeplay is players (from teenagers to adults), data demanders (companies, researchers etc.) and game developers (software engineers, self-taught developers etc.). Therefore, there is a large customer base. This leads the project to require an easy to understand UI and easy to use features in the games.

1.2.2 Security

There are 2 main points that security should be regarded:

- The users need to use some personal information to use the system, such as e-mail, name-surname and company name. Therefore, the system should ensure the security of these personal data.
- The data, which is loaded into the system by labeled data demanders, can include some image or voice record of people, or other personal data such as

car plates. The security of these personal information should be guaranteed with a contract to be accepted by the data supplier. The contract should follow the provisions in KVKK.

1.2.3 Robustness

- The system should be able to cope with errors and when an error occurs, it should not affect other features.
- The system should be able to handle the problems caused by internet connection and servers.

1.2.4 Extensibility

- The data given to the games will change according to the need of the demanders. Therefore, it is highly possible for some features of the games to require further changes. The system should be easy to change according to the demands.
- The platform will be open to integration of different games. Therefore, further needs of changes should be easy to apply.

1.2.5 Reliability

- One important purpose of the system is to provide labeled data to demanders. Therefore providing correctly labeled reliable data is important.
- The data demanders can specify some requirements, such as the required times of labelling and a due date to get all data back as labelled, for the games. The reliability between game developer and data demander should be ensured.

1.3. Definitions, Keywords, Acronyms and Abbreviations

- **KVKK:** Kişisel Verileri Koruma Kanunu (Law on the protection of personal data in Turkey)
- **AWS**: Amazon Web Services
- **GCP**: Google Cloud Platform

1.4. Overview

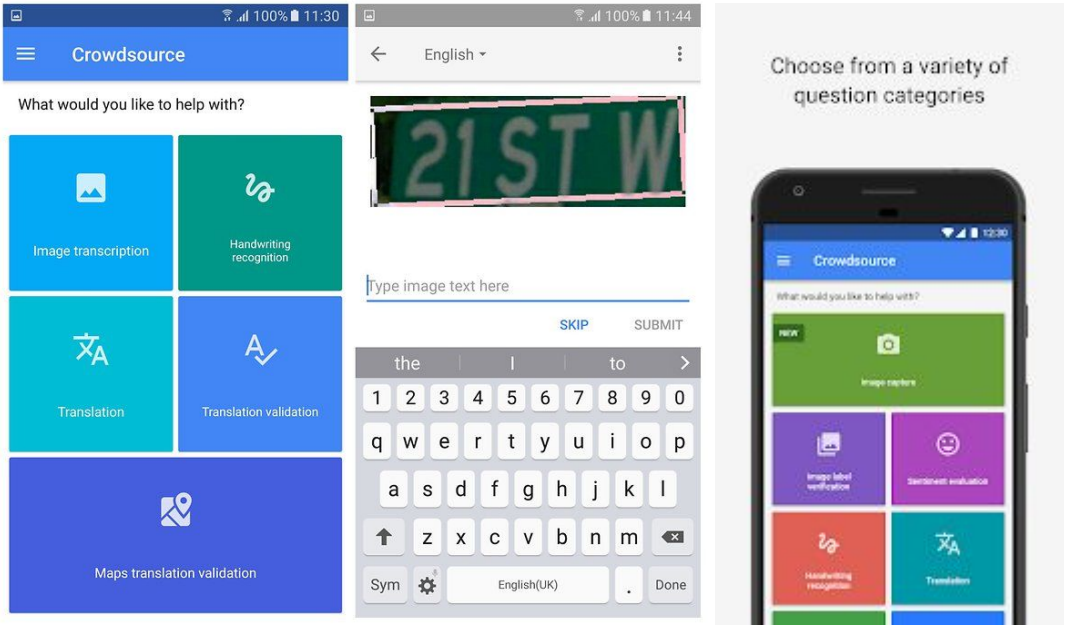
Our platform consists of different subsystems including UI layer that includes both cross-platform mobile application and progressive(offline-first) web application. Architectural overview of the proposed system are examined in Section *3.1*.

2. Current Software Architecture

There are several applications that linked to data labeling as a service business model. The followings are some of the well known and successful examples among them include following:

2.1. Google Crowdsource

Google Crowdsource is deployed on Android and Web platforms. It uses simple UI for showing data labeling tasks that include image label verification, translation validation and sentiment evaluation. It does not support multiplayer gaming sessions. Google (big brother) obtains data to improve services and commercial products like Google Maps, Google Translate and Google Cloud Platform offerings [2].



Figure[1]: Google Crowdsource Mobile Application.

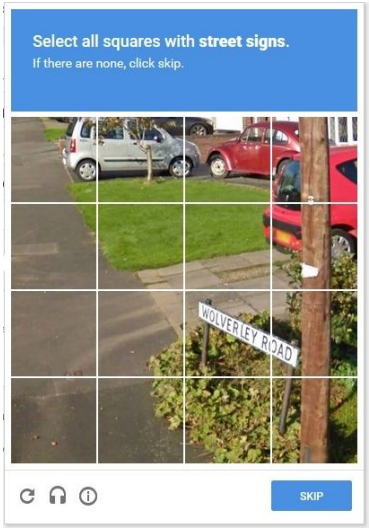
2.2. Amazon's Mechanical Turk

Amazon Mechanical Turk (MTurk) enables individuals and businesses (known as Requesters) to coordinate the use of human intelligence to perform tasks that

computers are currently unable to do. Employers are able to post jobs known as Human Intelligence Tasks (HITs), such as choosing the best among several photographs of a storefront, writing product descriptions, or identifying performers on music CDs. Workers, colloquially known as Turkers, can then browse among existing jobs and complete them in exchange for a monetary payment set by the employer. To place jobs, the requesting programs use an open API, or the MTurk Requester website.

Figure[**2**]: Amazon Mechanical Turk Workflow Diagram

2.3. reCaptcha



Figure**[3]**: Screenshot of reCAPTCHA in wild.

Google reCAPTCHA is a service to protect websites from spam and abuse. It is integratable to any client-side project by configuring reCAPTCHA widget to existing system. Basically, reCaptcha consists of two core services including **reCAPTCHA**

generator and **reCAPTCHA validator.** Architectural overflow of the reCAPTCHA is shown in the Figure [**4**]

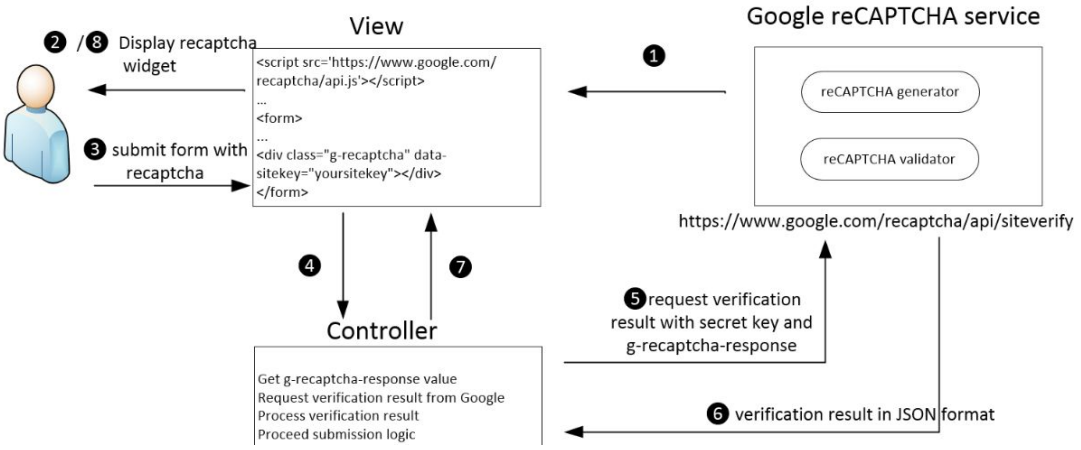


Figure **[4]**: Reference architecture of the reCAPTCHA.

- Generator is for integrating and configuring client widget.
- Validator is for the verification of the user submissions and it sends JSON formatted verification results to the client.

3. Proposed Software Architecture

3.1. Overview

Deeplay will use Microservices Architecture instead of monolithic and tiered architecture. It will consists of **small**, **self-contained**, **autonomous** services, modeled around a business logic. The main purpose for applying this architecture:

- It is autonomous in implementing a minimal unit of work and persisting its data.
- Each of microservices can be written in any programming language.
- Services can be versioned and deployed independently.
- Services communicate with each other over the network by using APIs.

We will use lightweight protocols such as HTTP/REST and WebSockets to implement our services.

3.2. Subsystem Decomposition

Deeplay consists of several subservices including **User Interface**, **Application Logic, User Management**, **Data Management**, **Task Management**, **Game Session** and **Machine Learning** services.

- Access control management and authentication of users are controlled with **User Management** service.
- **Task Management** service are used for monitoring the analytics of labeling tasks and creating, cancelling annotations.
- **Data Management** service is a bidirectional channel for processing and streaming data to other services including game session and machine learning model.
- **Game Session** includes game services for real-time multiplayer gaming, pair matching and game content management.
- **Machine Learning** service is a mechanism for validating the user labeling tasks, creating pre-trained raw data and assuring the quality of crowdsourced data.
- **API Gateway** is the single entry point for client requests and it dispatches each request to relevant microservice for providing transactions and data flow.

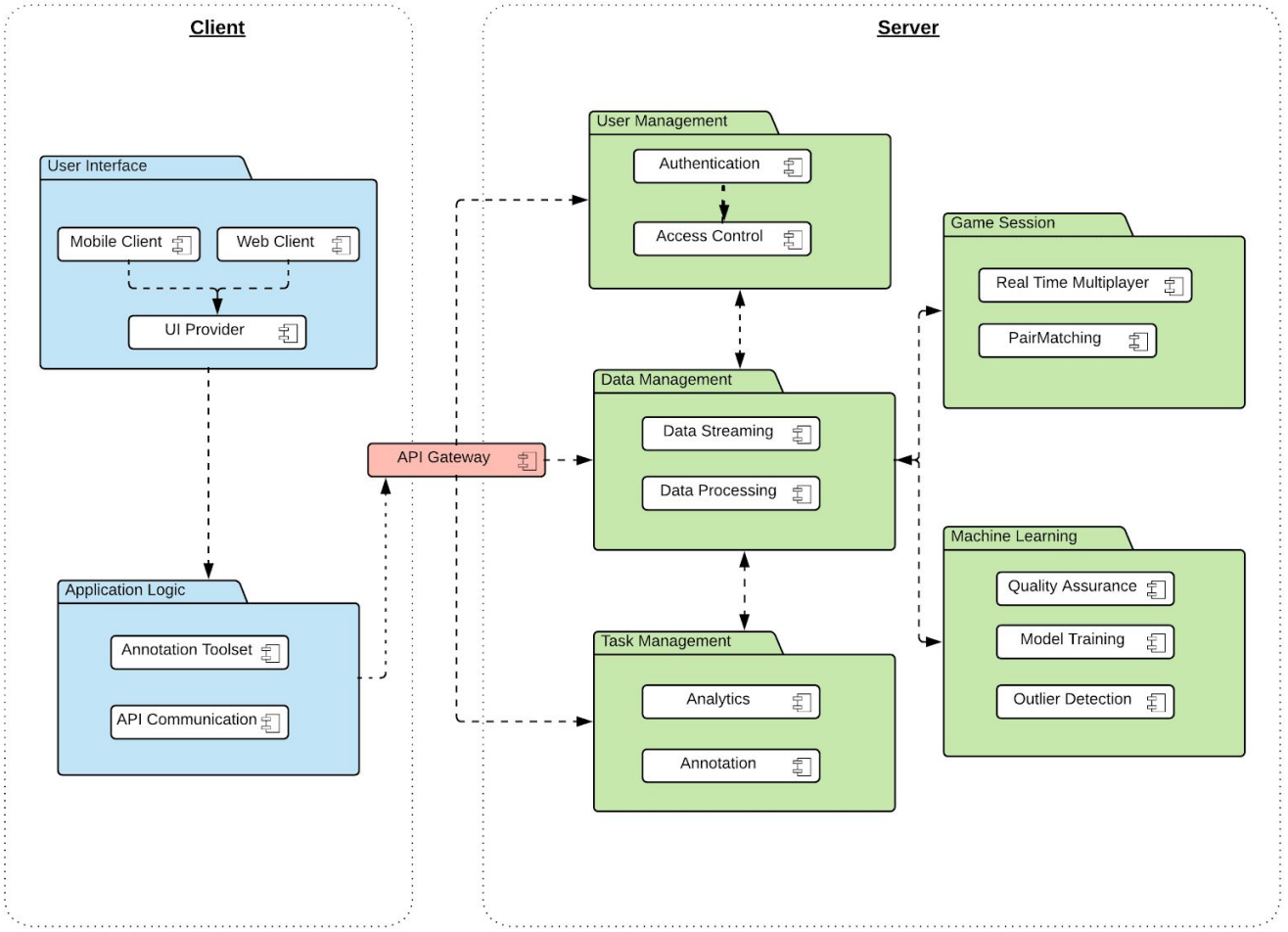


Figure [**5**]: Subsystem Decomposition Diagram of Deeplay

3.3. Hardware/Software Mapping

- General system performance should be fast and reliable enough to ensure that all data processing, multiplayer and overall backend operations work as expected.
- Backend(ML) operations that require the power of distributed computing should be run in an asynchronous, non-blocking pipeline.
- In order to have a nice experience in real-time multiplayer sessions, players should have a stable and high-bandwidth internet connection. Any connectivity issue may lead to a cancellation of the game session. For the best experience, latency(ping) should be less than 100ms. Also recommended download and upload speeds are 2 Mbps and 1 Mbps, respectively.
- Backend services of our platform will run on the cloud such as Google Cloud Platform(GCP) or Amazon Web Services(AWS).

- Cross-platform mobile application of Deeplay will run in smartphones that run the latest version of Android or iOS.

3.4. Persistent Data Management

Deeplay is ultimately a data platform that uses and manipulates raw data to output actionable insights. The web client of the platform will have a separate section for uploading (business) user provided datasets. We will support different cloud-based data storage services such as Google Cloud Storage and AWS S3 buckets to retrieve and store these datasets. One of the backend services of the platform will create base ML models for different categories and train(improve) the models with the supplied datasets. These models will be stored also in the cloud on the preferred region of the user. On the other hand, the platform will use aggressive caching mechanisms to reduce the computation bottlenecks.

Here are some planned-to-use technologies for our persistent data layer:

- Redis *as a* **key-value pair storage** for backend side request caching.
- GCS and/or AWS S3 *as a* bucket (object) storage for ML models.
- PostgreSQL *as a* relational storage for user information, authentication and persistent game state (Leaderboard, user achievements, etc).
- AsyncStorage and IndexedDB *as a* client side offline storage.

3.5. Access Control and Security

As *Deeplay* is a data-intensive platform, taking high-level of security measures to **protect** the privacy of its users should be our top priority. We will investigate best practices to implement security precautions in our system. In the end, we want to establish a secure software architecture to **prevent** *Unauthorized Data Disclosure(confidentiality)* and *Unauthorized Data Modification (integrity), Denial of Service(DOS)* attacks and incidents. Here are some planned security procedures to implement in our services:

- Sensitive information and credentials of the platform users will be handled according to the protocols suggested by OWASP [4].

- Passwords in length are limited to (not less than) 160 characters. After processing and cleaning the password, it will be hashed by *SHA-512* or similar hash and then will be stored along with a cryptographically strong user-specific salt. Even the rare case of data breach happens, it is **"theoretically" impossible** to encode the stored string if our services adopt this protocol.
- Our platform will adopt the usage of encrypted *HTTP/2* as a secure connection layer to communicate between the subsystems.
- Players (mobile client) and business users (web client) should be authenticated before using the platform. We will prefer anonymity for our game players but still there will be a need to authenticate and track them with seamless authentication methods such as device based "*iOS GameKit"* authentication. On the other side, business users and have enough privileges to use the platform and start labeling tasks.
- All communications with the REST API (Backend) must be authenticated and authorized to access protected resources and endpoints. We will use JSON Web Tokens(JWT) based authentication. Tokens will have a certain lifecycle for being expired and refreshed, regularly.

3.6. Global Software Control

- Since Microservices architecture is applied, each service has its own control flow. Client requests are distributed from API Gateway which is a single control point for client-server communication.
- Subsystems use asynchronous callbacks for communicating with each other.
- In order to automatically propagate changes through the data flow, Reactive Programming Paradigm will be used. Thus control of the data flow will be held by using data streams.

3.7. Boundary Conditions

3.7.1. Initialization

Game developer should first sign up to the platform, so as to be able to integrate his/her game and a data demander should sign up to upload his/her data to the system. They can authenticate with their current email addresses, such as Google account. They are required to create a unique password.

A player can reach a game from the platform or application stores of IOS and Android. Because differents games will be integrated to the system, some games can be supported by Android whereas some of them are supported by IOS. In the first version of the platform, there will be game(s) supported by Android. The player needs to download a game he/she selected.

3.7.2. Termination

For the games available in the first version of the project, which has no outer integration but only has games developed by the project team, there will be only multiplayer mode, so as to provide reliability of the labels by comparing the players performances. The users will be anonymous to each other. There will not be pause option because it is multiplayer game. The scores of the users will be updated as they play, and their success of labelling will be recorded for further data. As the game continues, whether the players are labelling completely wrong will be checked with algorithms. Therefore, one player's performance will be compared with both the algorithm's result and other players' labels. As the success of label is guaranteed regarding the demand, related data will be removed from the system and will be sent to the players no longer.

3.7.3. Failure

The games in the first version of the project requires internet connection because they are multiplayer. Thus, disconnection from internet causes failure during the game.

The data will be taken from servers. The problems in connection to the servers causes the games to crash.

4. Subsystem Services

4.1. Application Logic

- **Annotation Toolset:** Client tool for creating custom annotations and labeling tasks.
- **API Communications:** Client interface for communicating with Deeplay microservices over HTTPs using basic REST operations.

4.2. User Management

- **Authentication:** Sign-on protocol for permitting user access to Deeplay services. Deeplay will use stateless JSON Web Token (JWT) for authenticating users.
- **Access Control:** Access and permission control of the users. For each client request, permission of the users will controlled with access control service.

4.3. Data Management

- **Data Streaming**: High-performance data streaming service for managing and distributing large chunk of raw data from different sources.
- **Data Processing:** High-performance data processing service for sending labeled data to the client.

4.4. Task Management

- **Annotation**: Service to create, manage and cancel annotation tasks. It stores the details of the requested labeling tasks.
- **Analytics**: Service to monitor the analytics, details and current status of the scheduled annotation.

4.5. Game Session

- **Real Time Multiplayer:** Multiplayer game server which provides a two-way communication channel between the user's browser and a server.
- **Pair Matching:** Service to match pairs according to their common feature set for the created game session.

4.6. Machine Learning Service

- **Quality Assurance**: Confidence check service to validate user labelings. Player-labeled data will be sent to quality assurance service for checking the accuracy rate and quality of the task done. Data under some rate will be reprocessed for reaching high quality results.
- **Model Training**: Service to train raw dataset for creating initial labels. After processing it with human interaction, more accurate results will be collected and combined with initials results.
- **Outlier Detection:** Anomaly detection mechanism for identifying outliers in the given dataset.

5. Glossary

Salt random data that is used as an additional input to a one-way function that "hashes" data, a password or passphrase.

Reactive Programming is asynchronous programming paradigm based on data streams and the propagation of change.

Representational State Transfer (REST) is a software architecture that defines set of constraints for creating web services.

6. References

[1] J. Brownlee, "What is Deep Learning?", *Machine Learning Mastery*, 2018. [Online]. Available: https://machinelearningmastery.com/what-is-deep-learning/

[2] Perez, Sarah (2016-08-29). *TechCrunch*."Google's new app Crowdsource asks users to help with translation, transcription & more".

[3] "Apply Google reCAPTCHA into Your Site", 9 Dec 2015. [Online]. Available https://www.codeproject.com/Articles/1063051/Apply-Google-reCAPTCHA-into-Your-Site

[4] https://www.owasp.org/index.php/Authentication_Cheat_Sheet