
Bilkent University

Department of Computer Engineering

# CS 492 - Senior Design Project

*Project Name: Deeplay.io*

## Low 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 on February 18, 2019**

TABLE OF CONTENTS

**Introduction** **2**
1.1 Object Design Trade-Offs 3
1.1.1 Extensibility vs. Compatibility 3
1.1.2 Functionality vs. Usability 3
1.1.3 Speed vs. Space 4
1.2 Interface Documentation Guidelines 4
1.3 Engineering Standards 5
1.4 Definitions, Acronyms and Abbreviations 5

**Packages** **6**
2.1. User Package 6
2.2. Game Network Package 7
2.3. Resource Management Package 8
2.4. Core Computation Package 9

**Class(Service) Interfaces** **10**
3.1. User Package 10
3.2. Game Network Package 12
3.3. Resource Management Package 14
3.4. Core Computation Package 16

**4. References** **19**

# 1. Introduction

**Deeplay** is an intelligent Data Labeling platform that combines machine learning and gamification with human intelligence to provide high quality training data for AI applications. The platform includes multiple software components and clients to support its unique N-way business model. We propose a scheme for (game) developers, customers that build AI applications and also our end-users who will play addictive games on our platform.

In this report, we investigate core architecture decisions and low-level design of Deeplay. Since, we adopt agile software development methodologies, we want to keep the design document as simple and concise as possible without sacrificing the quality. Our development processes heavily rely on Code Reviews, Clean Code and CI/CD principles.

## 1.1 Object Design Trade-Offs

### 1.1.1 Extensibility vs. Compatibility

Deeplay is a platform which provides addictive games to player community and actionable, high-quality data for its customers. We use the terms “User” and “Customer” for the entities that need training data for their AI applications. Our platform aims to define a new, innovative way for customers to get more meaningful training data by using human intelligence in entertaining games. While there will be a sample game available in the platform at the beginning, platform will be extensible with new (pluggable) games. Therefore, even though the compatibility of the games are important to reach more people, it is to be concerned more about the further changes and requirements. At this point, making our platform adaptable to further changes is important. Therefore, extensibility has priority over compatibility for this project.

### 1.1.2 Functionality vs. Usability

We have 2 main types of users: players and users(customers). In the platform, different parts will interest different users. The players will be interested only in the games and because the player base has a broad range of age and requires no profession, usability in games is important. On the other hand, for the system users, deeplay promises slightly complex features, compared to the features for players. In general, functionality is more important than usability, because the games are to be uploaded to the system by others and as platform developers, we are responsible to increase the usability of only the sample game. On the other hand, we should make the whole platform have high functionality.

### 1.1.3 Speed vs. Space

In the platform, clients will be sending their data to our main system to get them labeled. For sending their raw data, they will have two options. They can either let our system access their data (e.g when they store their data in a cloud) by exporting their dataset by choosing cloud providers including AWS, Microsoft Azure or Google Cloud Platform, or they can directly upload their data to our system. As the space management for the second preference, the system will let only a limited amount of data to be uploaded. When the labelling task of the data is complete, it will be returned to the data owner and removed from our system. That is, there is a control system over space requirements and usage. On the other hand, quick response in the platform and high speed in the games are essential for all user types in this project. Regarding the space management plan and importance of speed on the user side, speed overweights space in this project.

## 1.2 Interface Documentation Guidelines

The following template shows the general form of class interfaces which will be used in the following sections:

| ClassName | |
|---|---|
| **Description** | General description of the class |
| **Attributes** | • attribute1: data type<br>• attribute2: data type |
| **Functions** | • method1 (parameter1, parameter2): return type<br>• method2 (parameter1, parameter2): return type |

### 1.3 Engineering Standards

UML and IEEE standards were used to identify, visualize, construct and document the project. In order to visualize the design of the system, activity diagrams, class diagrams, scenarios and use cases were determined according to UML design principles. IEEE standards were followed in designing the reports for credible citations.[4]

### 1.4 Definitions, Acronyms and Abbreviations

**API:** Application Programming Interface

**REST:** Representational State Transfer

**AWS:** Amazon Web Services

**JWT:** JSON Web Token

**SOA:** Service Oriented Architecture

**DE:** Docker Engine - Containerization technology

**S3:** Amazon Simple Storage Service

**EC2:** Amazon Elastic Compute Cloud

**CI:** Continuous Integration

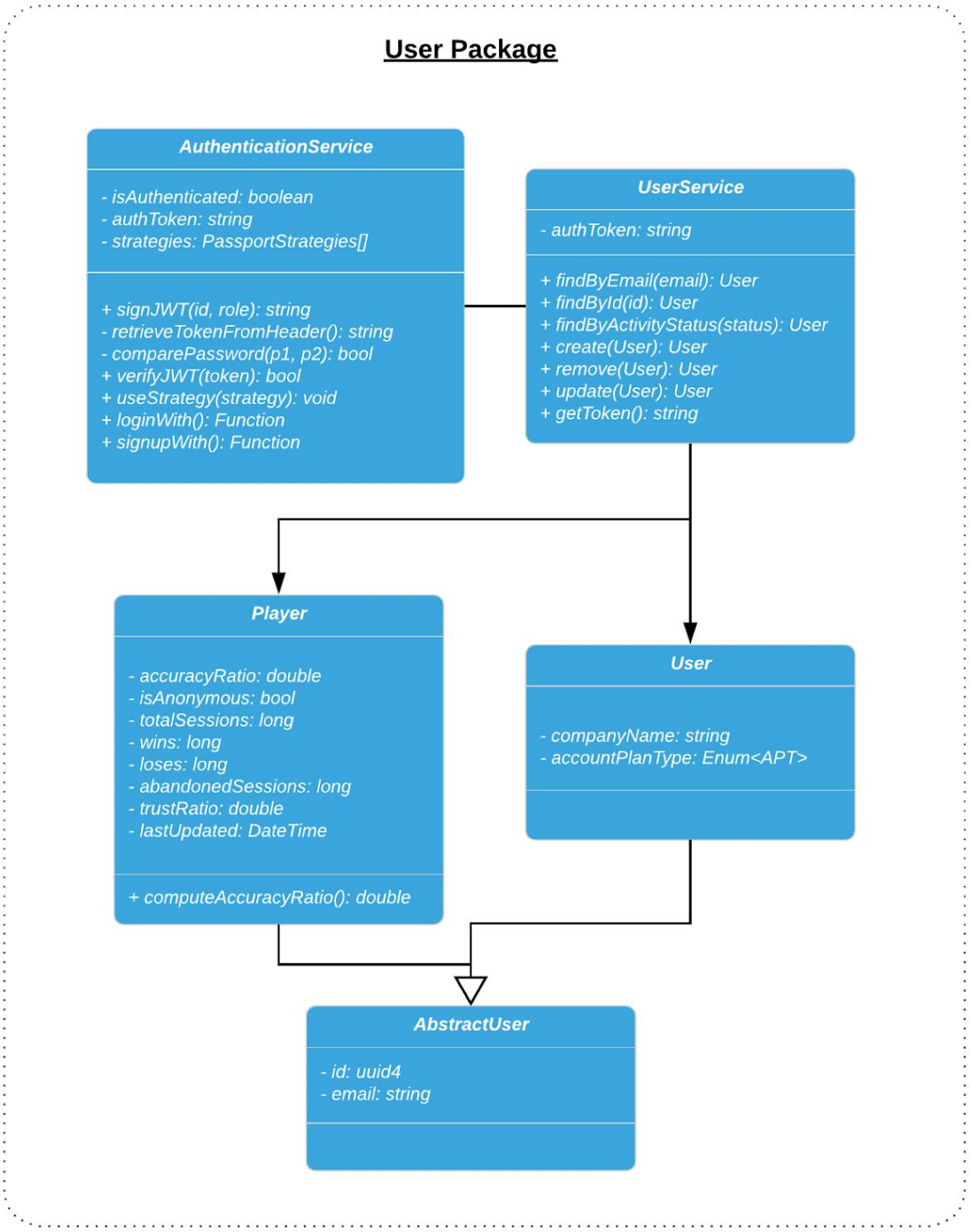
**TCP:** Transmission Control Protocol

**HTTP:** Hypertext Transfer Protocol

**CRUD:** Create, Read, Update, Delete

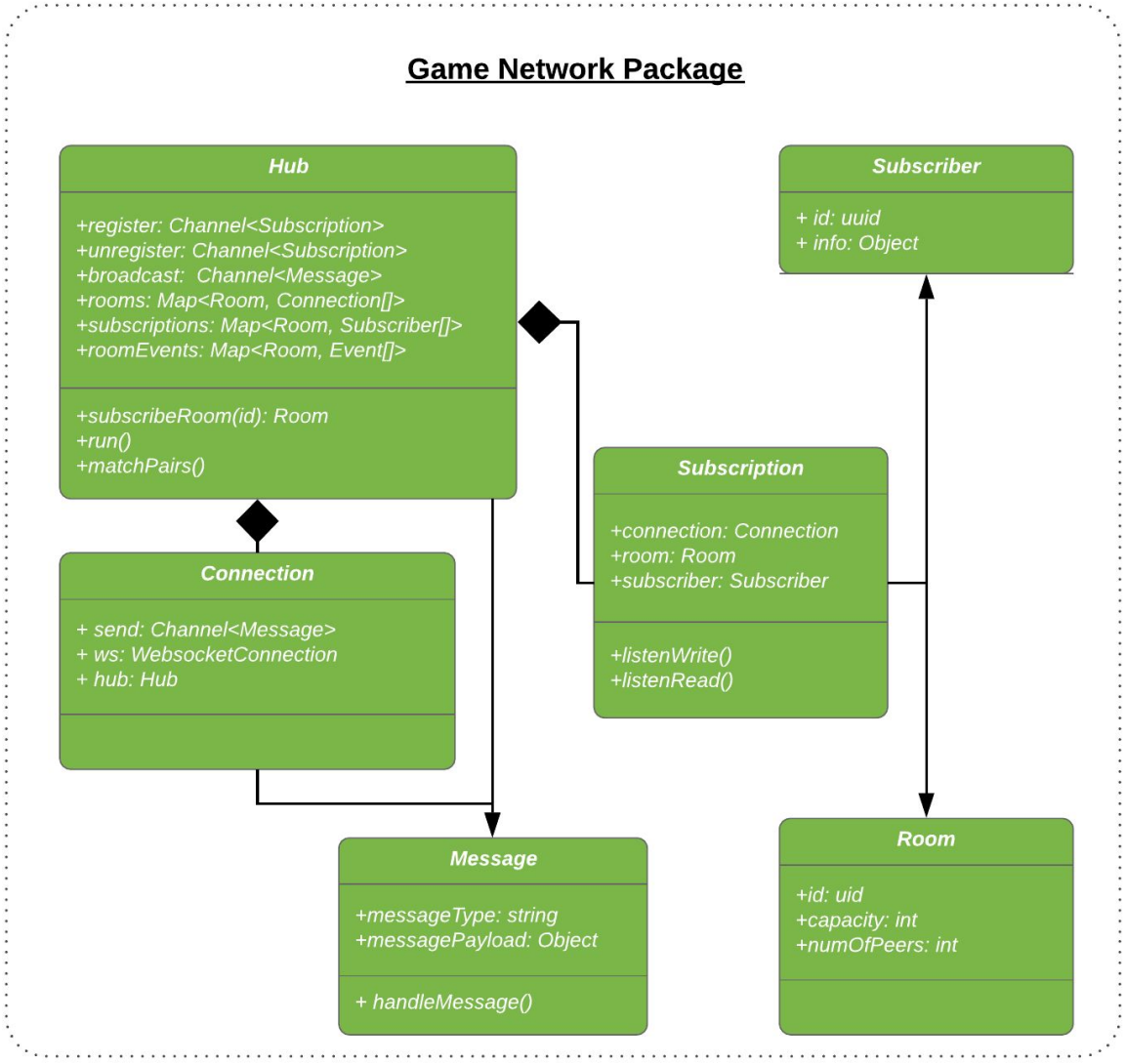
## 2. Packages

### 2.1. User Package



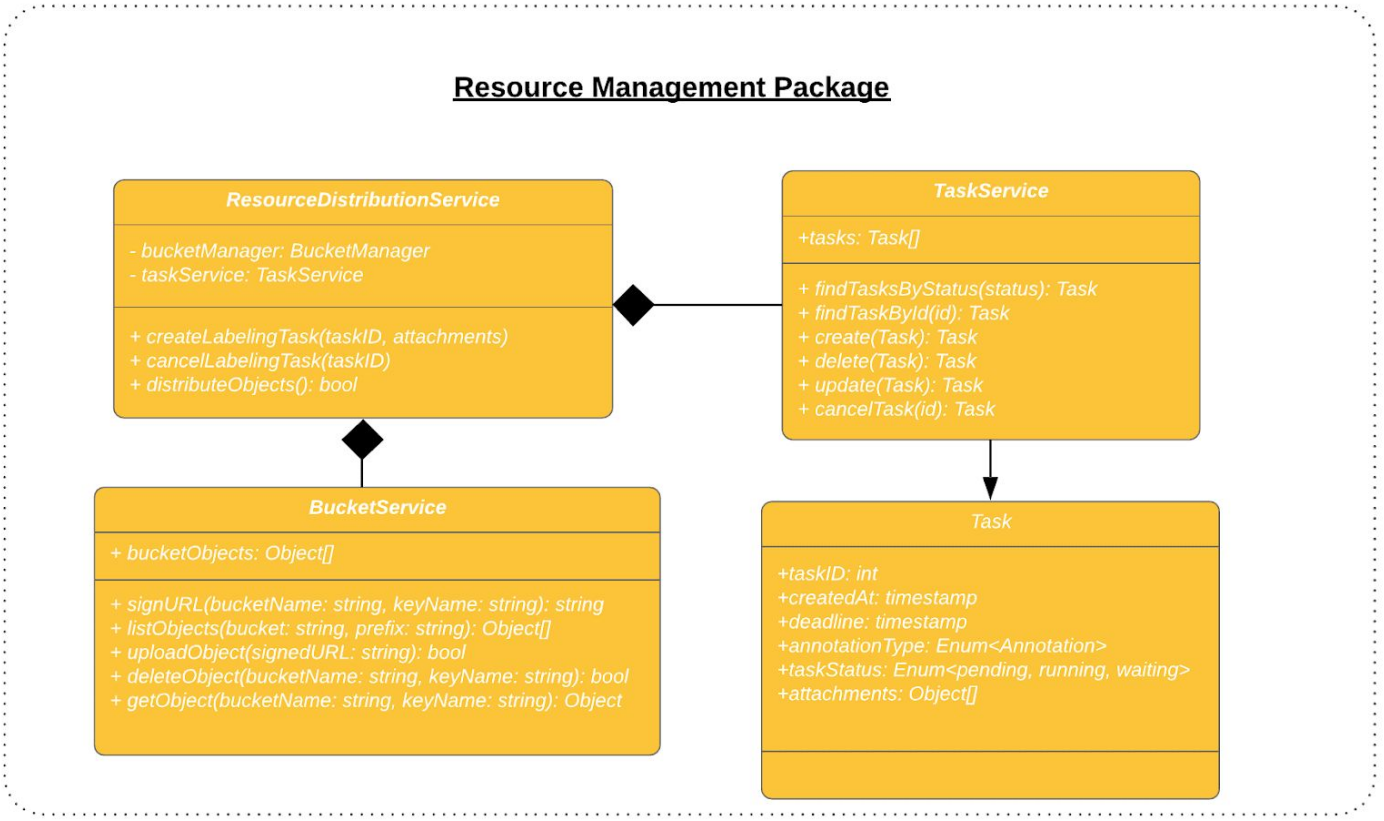
User package provides **authentication** and **CRUD operations** for all types of user entities. It also signs and verifies JWTs for our authentication process.

## 2.2. Game Network Package



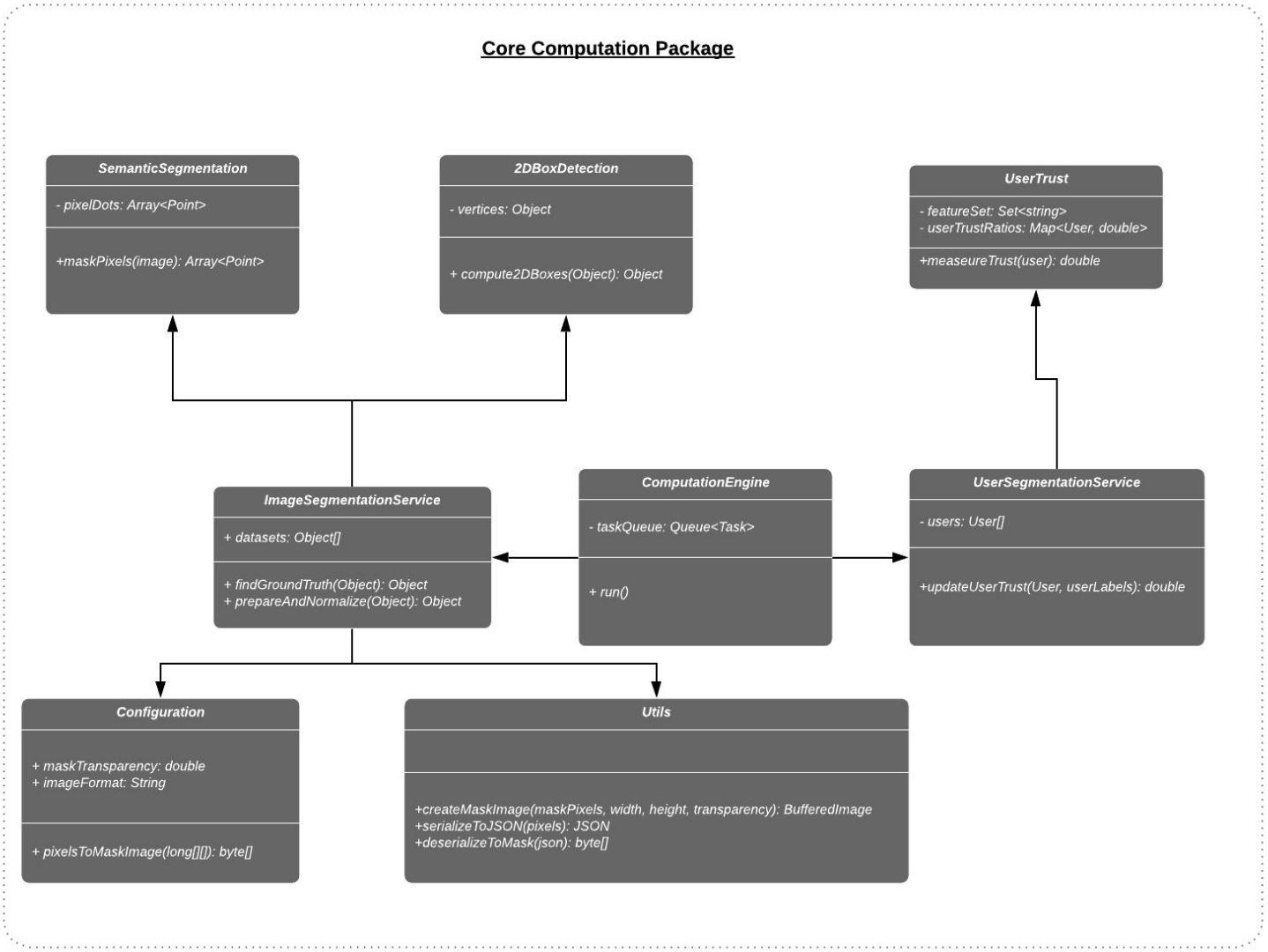

**Game network package/service** provides full-duplex communication channels over a single TCP connection between the client and server. Network service handles inbound messages from client by dispatching custom events and broadcasts outbound messages to each pair.

## 2.3. Resource Management Package




**Resource management package/service** is a cloud-based task management and data distribution service. It provides basic CRUD operations for resources, uses Amazon S3 object storage service for storing and retrieving data from Amazon DynamoDB.

## 2.4. Core Computation Package




**Core Computation Package** manages long-running tasks related to training machine learning models for *UserTrust (*similar to *TrueSkill)*[1] and doing computations based on our paper-referenced crowdsourcing and consensus algorithms [2].

## 3. Class(Service) Interfaces

### 3.1. User Package

| AuthenticationService | |
|---|---|
| **Description** | Signs and verifies secure *JWT*s. [3] Uses signed JWTs for user authentication. It defines multiple login/signup **strategies** such as OAuth 2.0 methods that enables Social Identity Providers(e.g. Google, Github) to be used in the application backend. |
| **Attributes** | • isAuthenticated: boolean<br>• authToken: string<br>• strategies: PassportStrategies[] |
| **Functions** | • signJWT(id, role): string<br>• retrieveTokenFromHeader(): string<br>• comparePassword(p1, p2): bool<br>• verifyJWT(token): bool<br>• useStrategy(strategy): void<br>• loginWith(): Function<br>• signupWith(): Function<br>• invalidateTokens(string[]): void |

| UserService | |
|---|---|
| **Description** | UserService handles CRUD operations related to the “User” entity. |
| **Attributes** | • authToken: string |
| **Functions** | • findByEmail(email): User<br>• findById(id): User<br>• findByActivityStatus(status): User |

| | |
|---|---|
| | • create(User): User<br>• remove(User): User<br>• update(User): User<br>• getToken(): string |

| Player | |
|---|---|
| **Description** | Player entity object that defines its attributes and maps them to the related database Table. |
| **Attributes** | • accuracyRatio: double<br>• isAnonymous: bool<br>• totalSessions: long<br>• wins: long<br>• loses: long<br>• abandonedSessions: long<br>• trustRatio: double<br>• lastUpdated: DateTime |
| **Functions** | • computeAccuracyRatio(): double |

| User | |
|---|---|
| **Description** | In other words, “Customer” entity. It extends “AbstractUser”. |
| **Attributes** | • companyName: string<br>• accountPlanType: Enum<AccountPlanType> |
| **Functions** | • Getters and Setters |

| AbstractUser | |
|---|---|
| **Description** | Abstract class that enforces the usage of the following attributes in inherited classes (e.g. User, Player). |
| **Attributes** | • id: uuid4<br>• email: string<br>• verified: bool |
| **Functions** | • Getters and Setters |

| Middleware | |
|---|---|
| **Description** | Applies multiple middlewares to the whole or part of the pipeline. |
| **Attributes** | • authToken: string |
| **Functions** | • applyCombinedMiddlewares(mw):void<br>• combineMiddlewares(mw): void |

## 3.2. Game Network Package

| Hub | |
|---|---|
| **Description** | Hub maintains the set of active connections and broadcasts messages to the connections. |
| **Attributes** | • register: Channel<Subscription><br>• unregister: Channel<Subscription><br>• broadcast:  Channel<Message><br>• rooms: Map<Room, Connection[]><br>• subscriptions: Map<Room, Subscriber[]><br>• roomEvents: Map<Room, Event[]> |
| **Functions** | • subscribeRoom(id): Room<br>• run()<br>• matchPairs() |

| Connection | |
|---|---|
| **Description** | Connection is a middleman between the websocket connection and the hub subscription. |
| **Attributes** | • send: Channel<Message><br>• ws: WebsocketConnection<br>• hub: Hub |
| **Functions** | |

| Message | |
|---|---|
| **Description** | Bidirectional websocket messages that are transferred over TCP protocol. |
| **Attributes** | • messageType: string<br>• messagePayload: Object |
| **Functions** | • handleMessage() |

| Subscription | |
|---|---|
| **Description** | Subscription maintains connection, room and subscriber information. It provides listen and read channel for handling event messages asynchronously. |
| **Attributes** | • connection: Connection<br>• room: Room<br>• subscriber: Subscriber |
| **Functions** | • listenWrite()<br>• listenRead() |

| Subscriber | |
|---|---|
| **Description** | Subscriber is the person connected to the hub. |
| **Attributes** | • id: uuid<br>• info: Object |
| **Functions** | |

| Room | |
|---|---|
| **Description** | Room is the active node on the hub for registering peers into game session. |
| **Attributes** | • id: uid<br>• capacity: int<br>• numOfPeers: int |
| **Functions** | |

## 3.3. Resource Management Package

| TaskService | |
|---|---|
| **Description** | TaskService provides CRUD operations for updating task state by sending HTTP requests. |
| **Attributes** | • tasks: Task[] |
| **Functions** | • findTasksByStatus(status): Task<br>• findTaskById(id): Task<br>• create(Task): Task<br>• delete(Task): Task<br>• update(Task): Task<br>• cancelTask(id): Task |

| Task | |
|---|---|
| **Description** | Task is an entity for managing the details of the task. |
| **Attributes** | • taskID: int<br>• createdAt: timestamp<br>• deadline: timestamp<br>• annotationType: Enum<Annotation><br>• taskStatus: Enum<pending, running, waiting><br>• attachments: Object[] |
| **Functions** | |

| BucketService | |
|---|---|
| **Description** | BucketService is an interface for uploading, retrieving and deleting bucket objects on the cloud. It uses Amazon S3 SDK for operating these requests. |
| **Attributes** | • bucketObjects: Object[] |
| **Functions** | • signURL(bucketName: string, keyName: string): string<br>• listObjects(bucket: string, prefix: string): Object[]<br>• uploadObject(signedURL: string): bool<br>• deleteObject(bucketName: string, keyName: string): bool<br>• getObject(bucketName: string, keyName: string): Object |

| ResourceDistributionService | |
|---|---|
| **Description** | ResourceDistributionService uses BucketService and TaskManager for creating annotations, processing raw data and distributing generated objects into the game session. |
| **Attributes** | • bucketManager: BucketManager<br>• taskService: TaskService |
| **Functions** | • createLabelingTask(taskID, attachments: Object[])<br>• cancelLabelingTask(taskID)<br>• distributeObjects(): bool |

## 3.4. Core Computation Package

| ComputationEngine | |
|---|---|
| **Description** | ComputationEngine schedules/manages long running tasks such as machine learning computations and stores the result of computations on object storage services. |
| **Attributes** | • taskQueue: Queue<Task> |
| **Functions** | • run() |

| ImageSegmentationService | |
|---|---|
| **Description** | ImageSegmentationService is the service to segmentify and label images according to the segmentation type. |
| **Attributes** | • datasets: Object[] |
| **Functions** | • findGroundTruth(Object): Object<br>• prepareAndNormalize(Object): Object |

| SemanticSegmentation | |
|---|---|
| **Description** | Service to associate each pixels of an image with the class label. |
| **Attributes** | • pixelDots: Array<Point> |
| **Functions** | • maskPixels(image): Array<Point> |

| 2DBoxDetection | |
|---|---|
| **Description** | Service to classify and recognize 2D objects. |
| **Attributes** | • vertices:Object |
| **Functions** | • compute2DBoxes(Object): Object |

| Configuration | |
|---|---|
| **Description** | Configuration of segmented image. |
| **Attributes** | • maskTransparency: double<br>• imageFormat: String |
| **Functions** | • pixelsToMaskImage(long[][]): byte[] |

| Utils | |
|---|---|
| **Description** | Utils consist of utility functions to process and manipulate image data. |
| **Attributes** | |
| **Functions** | • createMaskImage(maskPixels, width, height, transparency): BufferedImage<br>• serializeToJSON(pixels): JSON<br>• deserializeToMask(json): byte[] |

| UserSegmentationService | |
|---|---|
| **Description** | Service to rank user trust and segmentify users according to the predefined rules and feature set. |
| **Attributes** | • users: User[] |
| **Functions** | • updateUserTrust(User, UserLabels): double |

| UserTrust | |
|---|---|
| **Description** | UserTrust is the service to measure the accuracy of user labelings and rank them according to the correctness of the results. |
| **Attributes** | • featureSet: Set<string><br>• userTrustRatios: Map<User, double> |
| **Functions** | • measureTrust(user): double |

## 4. References

[1] Minka, T., Cleven, R., & Zaykov, Y. (2018). “TrueSkill 2: An improved Bayesian skill rating system”. Microsoft Research. *Tech. Rep.*

[2] Law E., Ahn L.. “Input-Agreement: A New Mechanism for Collecting Data Using Human Computation Games”. *ACM Conf. on Human Factors in Computing Systems*, CHI 2009. pp 1197-1206.

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

[4] IEEE DataPort “How to Cite References: IEEE Documentation Style” Available: http://www.etsi.org.