SAPIENZA
UNIVERSITÀ DI ROMA

Design and Implementation of a Sponsored CAPTCHA Platform: Real Time Bidding based on Click-Through Rate Prediction

Faculty of Information Engineering, Computer Science and Statistics
Master Course in Computer Science

Dennis Cimorosi
1646604

Advisor
Prof. Gabriele Tolomei

Co-Advisor
Prof. Mauro Conti

A/Y 2021/2022

Abstract

Numerous websites have employed CAPTCHAs in recent years to safeguard their security by denying access to automated bots. This project proposes a platform that enables website owners to employ CAPTCHAs that generate passive income through an integrated advertisement. Online advertising enables web content providers, or publishers, to monetize their websites by reserving ad slots. Advertisers are those who compete for and reward publishers for such slots. The proposed platform combines both paradigms by utilizing CAPTCHAs as ad slots to enable website owners to operate as publishers.

This premise is accomplished via two parallel theses works. This thesis' aim is to design the system's overall architecture and implement the Real-Time Bidding mechanism for online advertising. One of the design requirements is to create the platform as a cloud application that takes full advantage of the major benefits of the latest web technologies and paradigms.

The design of the platform was developed in conjunction with the complementary thesis, which was also responsible for handling all the aspects related to the specific CAPTCHA employed.

Real-time bidding (RTB) is the method by which companies purchase and sell online advertisements through automated auctions. RTB simplifies advertising by enabling advertisers to place hundreds of thousands of ads online, often in less than a second, without contacting individual online publishers. This process involves a number of participants, generally independent entities, that manage advertising for advertisers and/or publishers. A Sell-Side Platform, or SSP, allows publishers to fill ad slots and make a profit. Ad Exchanges are online markets where advertisers, publishers, and others can participate in auctions to acquire and sell content. A DSP, or Demand-Side Platform, assists advertisers in identifying and purchasing marketplace slots by providing a bidding strategy to be used during auctions and tailored to the advertiser's objectives.

The thesis is divided into five main parts, the first three of which were elaborated in conjunction with the author of the complementary thesis: an introduction to the use of **CAPTCHAs in Online Advertising**; a clear definition of the goals and **scope of the project**; a detailed description of the **application's design architecture**; an in-depth analysis of the practical **implementation**; an examination of the **bidding strategy** employed during RTB deployment and evaluation.

Contents

Contents **v**

1 CAPTCHaStar! in Online Advertising **1**
1.1 Online Advertising . . . 1
1.1.1 Entities . . . 1
1.1.2 Real Time Bidding . . . 2
1.2 CAPTCHaStar! . . . 4

2 Project Scope **7**
2.1 Problem Definition . . . 7
2.2 Users and Operations . . . 8
2.2.1 Publisher . . . 8
2.2.2 Advertiser . . . 8
2.2.3 Visitor . . . 8

3 Architecture **9**
3.1 Design Choices . . . 9
3.1.1 Functionalities . . . 10
3.1.2 Auction . . . 11
3.2 Main Components . . . 12
3.2.1 SSP Website . . . 12
3.2.2 Ad Exchange . . . 13
3.2.3 DSP Bidding . . . 13
3.2.4 DSP campaign . . . 13
3.2.5 CAPTCHA . . . 14
3.2.6 Log . . . 14
3.2.7 APIs . . . 14
3.3 Additional Considerations . . . 15

4 Implementation **17**
4.1 Technical Choices . . . 17

4.1.1 AWS Services . . . 17
4.2 Services . . . 19
4.2.1 SSP Website . . . 20
4.2.2 Ad Exchange . . . 22
4.2.3 DSP Bidding . . . 25
4.2.4 DSP Campaign . . . 27
4.2.5 Log . . . 29
4.3 APIs . . . 30
4.3.1 Web Access . . . 30
4.3.2 Get Ad . . . 31
4.4 Front-End . . . 32
4.4.1 Static-Content S3 Bucket . . . 32
4.4.2 Cloudfront . . . 33
4.5 RTB Performance . . . 33

5 RTB: Bidding Strategy & Evaluation 35
5.1 Bidding Strategy . . . 35
5.1.1 Second-Price Auction & Truthful Bidding . . . 35
5.1.2 Bidding Below Max eCPC . . . 37
5.2 CTR Estimation . . . 38
5.2.1 iPinYou Dataset . . . 38
5.2.2 Dataset Transformation . . . 40
5.2.3 Data Preprocess . . . 41
5.2.4 Training the Model . . . 42
5.2.5 Evaluation . . . 42

6 Conclusions 51
6.1 Obtained Results . . . 51
6.2 Future Work . . . 52

A Code 55

Bibliography 59

1. CAPTCHaStar! in Online Advertising

A rising number of websites are ensuring that users are not automated bots and, as a result, improving their security by employing CAPTCHA services. Google's services now dominate the market for CAPTCHAs since they are simple to use and free. This thesis aims to develop an alternative platform that enables website owners to adopt a CAPTCHA that provides passive revenue from an integrated advertisement in addition to the protection that any CAPTCHA offers.

1.1 Online Advertising

Online advertising is a kind of business marketing that involves delivering online advertisements to potential customers over the Internet. As the number of Internet users has expanded and Internet technology has advanced, more businesses have begun marketing their products and services online.

1.1.1 Entities

The fundamental concept underlying online advertising is to enable online content providers (*publishers*) to profit by reserving certain fixed slots on their websites for the display of advertisements.

Advertisers (companies seeking to advertise themselves) compete for these spots and are willing to pay publishers for them. This system includes a variety of participants, the majority of which are independent companies that handle various advertising components on behalf of advertisers and/or publishers:

- A Sell-Side Platform, or **SSP**, is a platform that allows online publishers to manage their *ad slots*, fill them with advertising, and receive revenue.

- **Ad Exchanges** are digital marketplaces where advertisers, publishers, and others may purchase and sell inventory by auctioning slots.

- Demand-Side Platform, also known as **DSP**, is a system that assists marketers in finding and purchasing slots from the marketplace. DSPs are also in charge of handling advertisers' real-time bidding.

Figure 1.1: Online Advertising Entities

1.1.2 Real Time Bidding

Real-time bidding (RTB) is an online auctioning system that enables companies to buy and sell advertisements in real time. Real-time bidding removes the need for advertisers to contact each web publisher individually, allowing them to simultaneously post hundreds of thousands of advertisements online, usually in less than a second. The following figure illustrates the general operation of RTB systems.

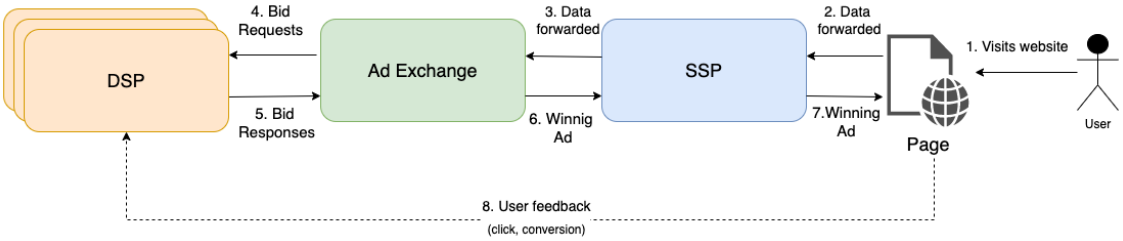

Figure 1.2: RTB Pipeline Overview

1. When a customer visits a website, an ad slot is produced on the publisher's website.

2. The information regarding the slot, along with the user's data, is sent to the publisher's SSP.

3. The SSP transmits this data to an Ad Exchange.

4. The slot auction is broadcast to all accessible DSPs by the Ad Exchange.

5. If the DSP wishes to participate, it provides a bid.

6. The Ad Exchange determines the winner based on the DSP bids and sends the winning ad to the SSP.

7. The SSP sends the winning advertising to the publisher's website, where it is displayed.

8. The user input is collected to determine if the user clicked the advertisement and whether the advertisement led to a *conversion* (e.g., the user bought something from a website store).

Pricing models

The most prominent pricing models in online advertising include:

- **CPM** (Cost Per Mille): it represents the cost of 1,000 impressions of a particular advertisement. This is the most natural approach for paying for exposure and brand awareness. For instance, the majority of Facebook advertisements, including mobile video ads, are priced using CPM.
- **CPC** (Cost Per Click): an advertiser only pays when an individual engages with the advertisement by clicking on it. Typically, search engine result page advertisements are paid in this way.
- **CPA** (Cost Per Action): before an advertiser gets paid, a user must not only click on an advertisement but also execute a certain action, such as filling out a form or installing an app. For instance, several campaigns operate on a CPI (cost per install) basis.

Furthermore, the distinction between **second** and **first-price** auctions may be drawn. The winner of a first-price auction pays the exact amount bid, whereas the winner of a second-price auction pays the amount bid by the second highest bidder.

1.2 CAPTCHaStar!

CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) is a challenge-response authentication technique for performing security assessments. By requiring users to complete a simple test, CAPTCHA protects them against spam and password theft. This test is used to determine if the user is a person or a bot attempting to access a password-protected account.

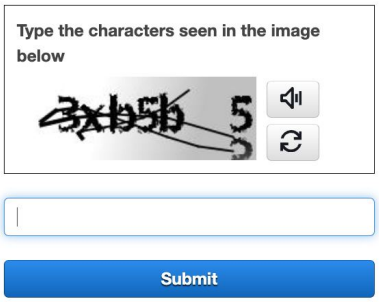

Figure 1.3: A Text-based CAPTCHA

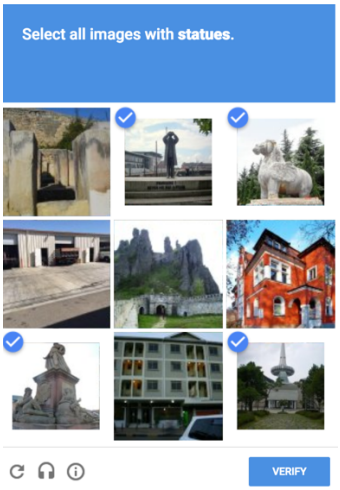

Figure 1.4: An image-based CAPTCHA

We specifically adopt a recent CAPTCHA innovation described by Conti et al. [1] called **CAPTCHaStar**. It is ideal for displaying ads through images since it concentrates the user's attention on pictures and shapes.

Depending on the location of the cursor, CAPTCHaStar displays *stars.* within a square in different locations. The user must move the cursor until the stars form a recognizable shape.

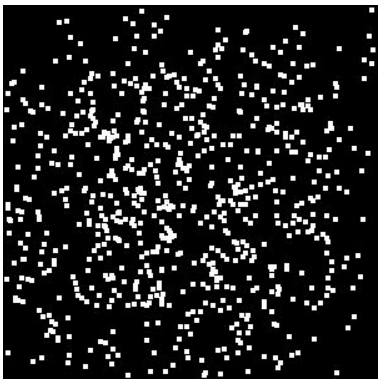

Figure 1.5: An incorrect configuration

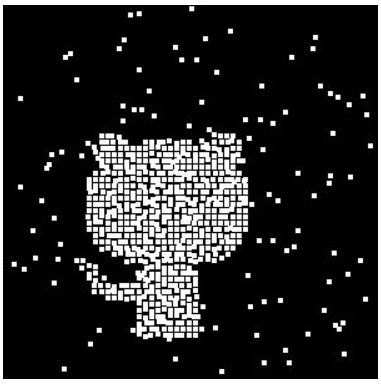

Figure 1.6: A correct configuration

It is essential to stress one feature of any CAPTCHA system: both requirements of usability for users, and resilience to automated attacks must be satisfied simultaneously. A more complex and secure problem would be more difficult for both a machine and a person to solve. This may make customers uncomfortable and discourage them from accessing the website. Therefore, it is essential to evaluate the trade-off between **security** and **usability**. Regarding CaptchaStar, Conti et al. [1] provided a detailed analysis of this problem in their original work.

2. Project Scope

2.1 Problem Definition

The objective of the system is to provide an integrated platform with CAPTCHA services and a mechanism for online advertising. When a publisher website intends to provide a user with a CAPTCHA challenge, an ad slot is created for it. The slot is then auctioned off on the platform. Instead of a conventional ad, the winning advertisement is a challenge based on the image chosen by the winning advertiser. Since CAPTHCHaStar may theoretically display any kind of image, we aim to utilize it to display images that can be considered advertising, making profits for a publisher who chooses to employ this type of CAPTCHA.

Aside from developing the platform, the two primary tasks are to build an embedded RTB system (to give a mechanism to choose which ad to present as a challenge) and to convert the CAPTCHaStar tool to operate for this purpose.

This thesis' work focuses on the RTB implementation, while the work on CAPTCHaStar is discussed in the complementary thesis [9]. Note that the overall design and main architectural choices were made in collaboration between the two theses since they would highly impact the work of each individual project. For this reasons, this chapter and chapter 3 are shared between the two theses.

Figure 2.1: Base logo

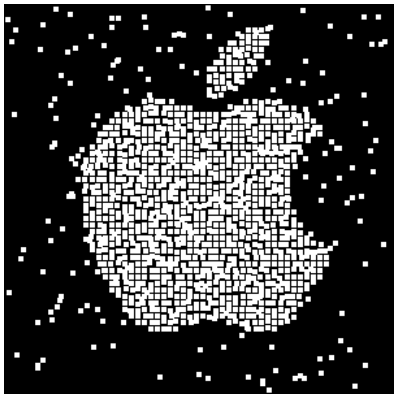

Figure 2.2: Sponsored CaptchaStar

2.2 Users and Operations

This section will describe the various identified actors and the corresponding operations they must be able to perform.

2.2.1 Publisher

The publisher is the owner of the website that wants to show sponsored CAPTCHAs. The operations that he can perform on the platform are:

- create his own publisher profile, which grants him access to the protected publisher area;
- register his own websites in the publisher's restricted area using the relative domain;
- generate a sponsored CAPTCHA slot, by using the appropriate APIs, whenever a visitor requests it.

2.2.2 Advertiser

The advertiser represents a company that wishes to advertise itself and is willing to pay for the slots held by the publishers. The operations that he can perform on the platform are:

- create his own advertiser profile, which grants him access to the protected advertiser area;
- create a CaptchaStar challenge using an image of his advertisement in the advertiser's restricted area;
- create an ad campaign in the advertiser's restricted including the budget and CAPTCHA to be shown when a slot is won using the RTB system.

2.2.3 Visitor

The visitor to the publisher's website is the actor who, by requesting a CAPTCHA, initiates the RTB auction and then receives a CAPTCHA challenge. Finally, he sends back to the platform his own solution to be evaluated. If it is acceptable, the system shows the original ad image from which the challenge was originally generated and grants the visitor access to the protected area.

3. Architecture

This chapter will describe the architecture of the system and the main choices behind it. However, technical and implementation details, which are part of this thesis responsibilities, will be analyzed in chapter 4.

3.1 Design Choices

The system is designed to be structured in *microservices*, an architectural style that seeks to organize an application into independent services. As a result, our modules are loosely coupled and independently deployable, making them easy to manage, scale, and test. As a result, each service may be constructed using a range of technologies without regard to the ones employed by other services.

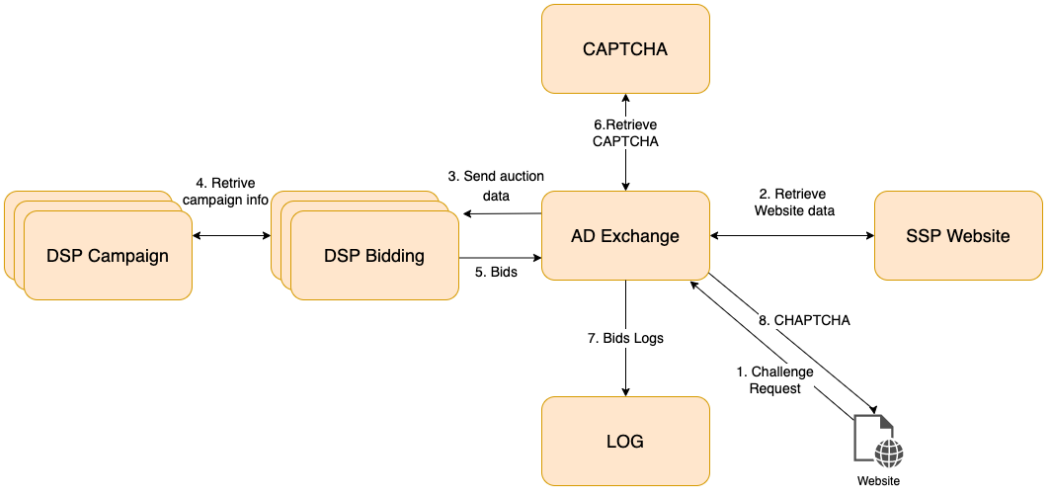

Figure 3.1: High-level Architecture

Figure 3.1 shows the main components and how they interact with each other during the main flow of the application. In designing the microservices, we start from the usual RTB model: DSP, Ad Exchange, and SSP. However, the DSP functionalities are split into two distinct services: DSP Campaign and DSP Bidding, whose responsibilities are respectively to handle the data of the ad campaigns, and to bid on behalf of the advertisers. One major change from the original model is the fact that the request for a challenge is directly handled by the Ad Exchange instead of the SSP. Since we can handle all the steps of RTB in the

same system, we do not need to get the SSP Service involved in the auction itself: it is only responsible for handling the publishers' websites data. Finally, we include the CAPTCHA and Log modules: the first service generates, saves and manages CAPTCHA challenges, while the second keeps track of all auctions.

3.1.1 Functionalities

Diving into some details, the functionalities currently implemented are:

- **RTB Flow**: when a user visits a publisher's website, an ad slot is automatically created. Its data is sent to the *Ad Exchange* service through the corresponding API. After retrieving the website's data from the the *SSP Website* service, the Ad Exchange invokes a *DSP Bidding* instance for each available ad campaign. Each instance of the DSP Bidding retrieves the data for the campaign from the *DSP Campaign* service. Once all the necessary data is gathered for the corresponding DSP Bidding instance, it provides an adequate bid to the Ad Exchange service. The Ad Exchange establishes the winning campaign and retrieves the corresponding challenge from the CAPTCHA service. Finally, it sends it back to the webpage to be displayed. At this point, there are two optional operations that can be performed:
  - **Check Solution**: when the visitor selects his proposed solution, he sends it to the CAPTCHA service through the corresponding API. The CAPTCHA service checks if it is acceptable or not and forwards the answer back to the browser. If the solution is correct, it also sends back the original ad image from which the challenge was generated so that it can be shown to the user.
  - **Log Click**: if the user successfully completes the CAPTCHA, the original image is displayed. If the user clicks on the ad image, he triggers the *Log* service that logs this click event.
- **Log Auction**: once the Ad Exchange service has selected the winning campaign, it contacts the Log service to record auction-related data.
- **Add CAPTCHA**: an advertiser using the CAPTCHA service may submit an image to be converted into a CAPTCHaStar challenge.
- **Add Campaign**: an advertiser may build an ad campaign through the DSP Campaign service by submitting the necessary data, such as a budget and the sponsored CAPTCHA to use.

- **Add Website**: a publisher may register a website using the SSP Website service by supplying the data needed.

3.1.2 Auction

When deciding the type of auction and bidding strategy to implement, we tried to make a decision that would minimize the coupling between the **DSP Bidding** and **Ad Exchange** services. We opted to charge the advertiser for each impression instead of implementing, for instance, a CPC paying method (i.e., the DSP is charged only if a click occurs).

If we had chosen the latter, the Ad Exchange's logic would be drastically different: it would receive the maximum amount the DSP would be willing to pay for a click (*max CPC*) and would rank the bids based on some criteria (e.g., CTR). In this way, if a new type of campaign or bidding strategy were created (one that does not have as an output a max CPC), a new, ad-hoc kind of auction would need to be created as well. So, this decision was made in order to create an interface between DSP and Ad Exchange: the auction receives as input a bid (the amount that a DSP is willing to pay for a specific slot) regardless of the strategy or goal of the DSP (maximizing clicks, impressions, and so on). On the other hand, each kind of DSP is responsible for implementing its own logic in order to present the Ad Exchange with the amount it is willing to pay for a specific slot. Thus, in the future, different kinds of campaigns (and associated DSPs) might be created without the need to change the logic of the auction itself.

3.2 Main Components

This section will analyze each component of the system, its responsibility, and the functionalities it encompasses.

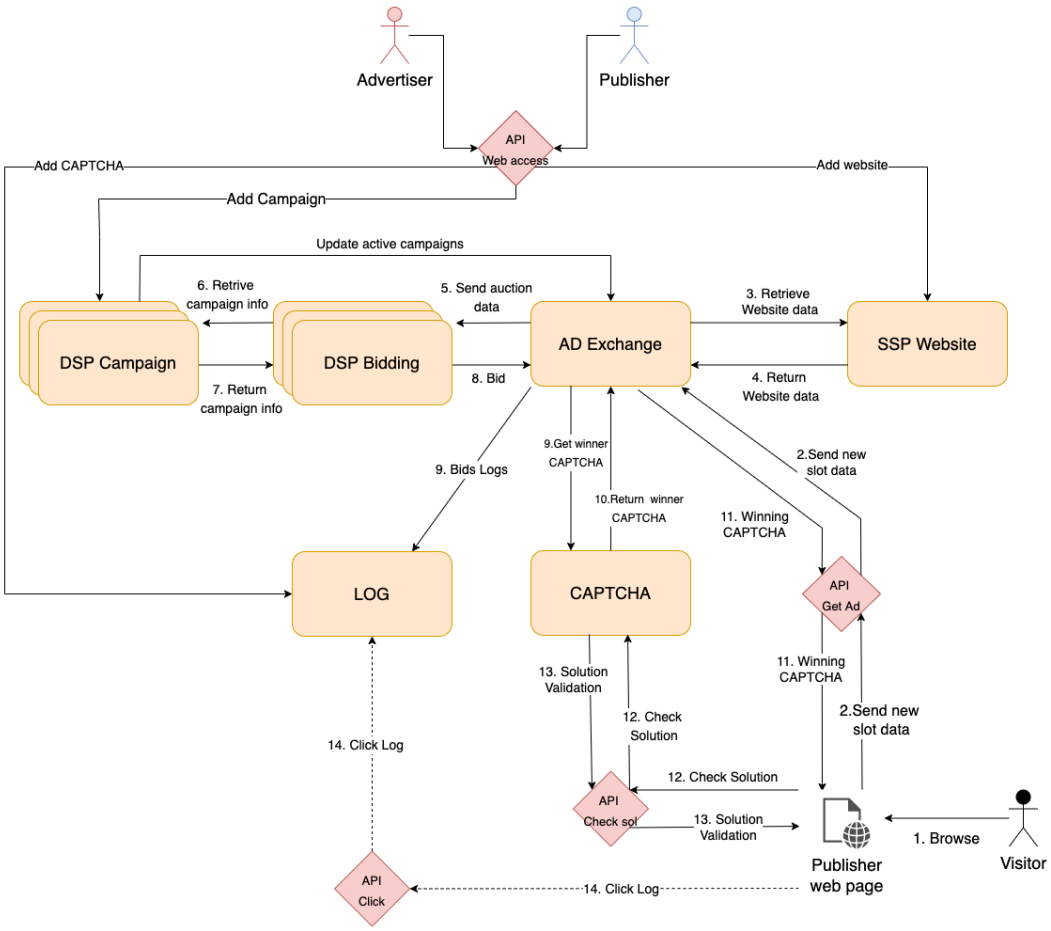

Figure 3.2: **Architecture Overview**

3.2.1 SSP Website

This module is responsible for managing the website's data. Current functionalities include:

- **Add Website**: it enables the publisher to register a new website by supplying the required information. This information is stored in a dedicated storage.

- **Get Site Data**: it enables any other service to obtain all the information regarding a specific website.

3.2.2 Ad Exchange

This module is responsible for initiating and managing the RTB auction. This module provides two functionalities:

- **RTB Auction**: it receives a challenge request from a website, collects the website's data from the SSP website service, retrieves all available campaigns from its storage, and launches a DSP Bidding instance for each. After receiving the bids and CAPTCHA challenge identifier from each DSP, the system determines the winner (the DSP with the highest bid), fetches the associated CAPTCHA from the CAPTCHA service, and delivers it back to the web page. In parallel, after determining the winner, it calls the Log service to save the auction's data (participants, corresponding bids, and winner).
- **Update Active Campaigns**: it replaces current active campaigns in the storage with received campaigns (see section 3.3).

3.2.3 DSP Bidding

This module manages the bidding strategy for each auction from the perspective of a single advertising campaign. It provides the following functionality:

- **Get Bid**: it determines the appropriate bid value and delivers it back to the Ad Exchange after receiving campaign-specific data.

3.2.4 DSP campaign

This module is responsible for managing an advertising campaign's data. Its functionalities consist of:

- **Add Campaign**: it creates an ad campaign given various campaign data such as budget, CAPTCHA, and so on.
- **Get Campaign Data**: it supplies all information pertaining to a campaign.

3.2.5 CAPTCHA

This module is responsible for creating, managing, and storing all CAPTCHaStar challenges. It provides three main functionalities:

- **Add Captcha**: given an image, it generates a new CAPTCHaStar challenge and stores it together with the corresponding solution in a designated storage location. Refer to the complementary thesis [9] for details on the generation process.
- **Get Captcha**: it retrieves a specific challenge from the storage.
- **Get Default Captcha**: it returns one of the default non-sponsored CAPTCHaStar challenges (see section 3.3 for details).
- **Check Solution**: given a suggested solution and a CAPTCHA challenge identifier, it verifies that the solution is valid and, if so, returns the advertisement image from whence the challenge was formed.

3.2.6 Log

The purpose of this module is to manage the auction logs. It provides two primary functionalities:

- **Log Auction**: given all of an auction's data, it records this information in its own storage.
- **Update On Click**: given an auction identifier, it modifies the record of the auction's winner to *clicked*.

3.2.7 APIs

Four APIs were designed to serve diverse purposes:

- **Web Access**: it enables advertisers and publishers to manage their private areas through a website.
- **Get Ad**: it enables a CAPTCHaStar challenge to be requested on a publisher's website using an RTB auction process.
- **Check Solution**: it enables a publisher's website to determine if a user's solution to a given CAPTCHaStar challenge is valid.
- **Click**: it enables a publisher's website to register a user's click on a CAPTCHaStar challenge after its resolution.

3.3 Additional Considerations

For the sake of the thesis work, several simplifications were made when modelling the problem. First of all, we chose to link only one CAPTCHA with each campaign. However, it is very simple to update the storage of each campaign to associate it with a list of CAPTCHAs instead of just one. For simplicity, the campaign can only be considered to be active or not. However, this mechanism should be expanded to include other features like duration time, start date, or end date. In regard to the data used, we lack any data on the visitor itself, and we do not actually have some information on the website, like the category. This is because of the shape of the dataset used to train the machine learning model for the bidding strategy (more details on the dataset in subsection 5.2.1). However, the logs are purposely structured in a way that would allow them to be directly used for training a model that suits our needs and data.
Furthermore, certain decisions were taken for more technical reasons. Such as redundantly storing the active campaigns in a dedicated storage of the Ad Exchange module (rather than keeping them only in the DSP Campaign storage) to serve as a sort of cache. This decision was made in order to minimize the number of interactions between components and thus make the whole auction process as fast as possible. Finally, we designed a fallback system to serve a default non-sponsored CAPTCHA in different scenarios when it is not possible to provide a sponsored challenge, for example when there are no campaigns willing to participate in the auction or when some errors occur during the auction process.

4. Implementation

This chapter will discuss the technical choices and implementation details regarding the development of the architecture discussed in the previous chapter.

4.1 Technical Choices

Instead of a simple prototype, the goal of the implementation process was to create a platform that could serve as a solid foundation for future improvements and developments. As a result, we designed the system to work in an environment as realistic as possible while keeping the constraints of a thesis project into account.

In recent years, businesses of all sizes have been considering a shift to cloud computing. There are many reasons why this is taking place. Features like scalability, elasticity, less entry cost, ease to access, pay per use etc. compel the businesses to migrate themselves from the traditional platforms to the **Cloud** based platform (see [2] for additional insight). As such, we employ the modern cloud paradigm to leverage its benefits.

In particular, we work on *AWS* (Amazon Web Services), one of the most popular cloud service providers. Due to the budget limitations of an academic project, we have to stick to the free-tier plan offered by the AWS platform. This plan allows us to build the entire system but under some constraints, such as: limited amount of traffic, storage capabilities, available services, etc.

According to the cloud design principles, a modern application should generally be decomposed into loosely coupled services (see section 3.1) and be as serverless as possible. A serverless architecture is a way to build and run applications without having to manage infrastructure. This means that, in our scenario, all the server management, provisioning, and scaling is done by AWS. This allows us to focus on the functionalities and organization of the system.

4.1.1 AWS Services

This section will describe the main services employed while developing the application among those offered by AWS.

- **Cloudfront**: a CDN service that mostly serves static content efficiently.
- **S3**: an object storage service often used to store static website content.

- **API Gateway**: a service that enables developers to design, publish, and manage endpoint APIs in order to facilitate communication between the front and back ends of any application.
- **Cognito**: a service that synchronizes the identities of users and the data associated with them.
- **Lambda**: a service that enables developers to deploy event-driven functions without having to provision the environment in which those functions operate.
- **Dynamo DB**: a fully-managed, key-value NoSQL database developed for use with high-performance applications.
- **RDS**: a suite of cloud-based managed services that simplify the deployment, administration, and scalability of relational databases.
- **Cloudwatch**: a monitoring service that delivers data to assess your applications, react to changes in performance, and optimize resource consumption. In addition, it allows for the scheduling of events and triggers.

4.2 Services

This section will elaborate on each of the services mentioned in section 3.2. The closely related functions to the complimentary thesis will not be examined here (see [9]).

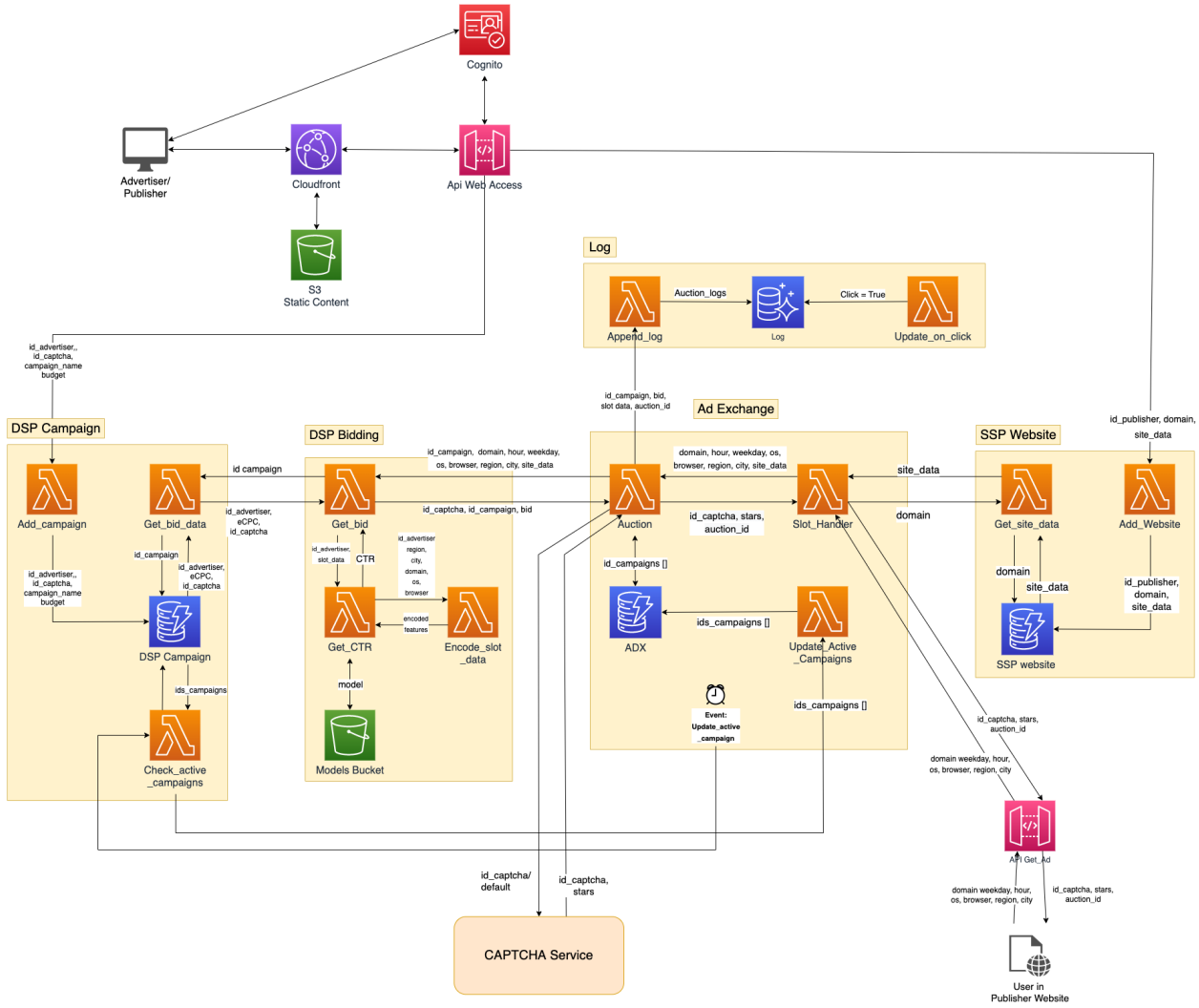

Figure 4.1: System Overview

4.2.1 SSP Website

This module has the responsibility of handling the data of the publishers' websites. All of the functionalities of this module are implemented through Lambda Functions written in Python, which use a DynamoDB table to store and retrieve all the data.

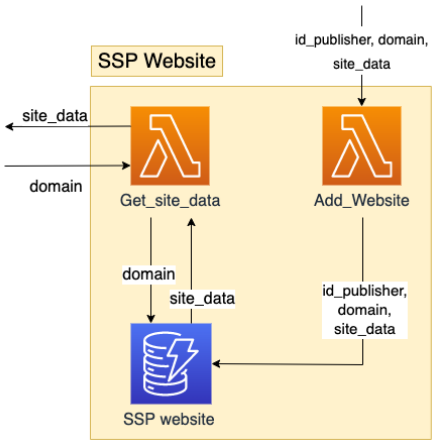

Figure 4.2: SSP Website Service

Add Website

Inputs: id publisher (string), domain (string), site data (list)

Description: it stores the input data in the DynamoDB table.

Outputs: None

Site data is an empty placeholder object for future information that could be added to a website.

Get Site Data

Inputs: domain (string)

Description: it retrieves all the information stored regarding the specific website.

Outputs: site data (list)

SSP Website DynamoDB Table

The storage of this service was implemented through the **SSP-Website DynamoDB** table. This choice was made for two main reasons: we wanted to increase the speed of the system (a critical factor in RTB) and because we wanted a database which could easily handle changes in the data structure (e.g., new field for the site data). Even if the structure is not fixed, we saved our records as follows:

Key	**Value**	**Value**
Domain	Publisher id	Site Data

The value used as the key is the website domain because, in our use-case, when querying this database, we expect to search based on the website and not the publisher id.

4.2.2 Ad Exchange

This module has the responsibility of running the RTB auction and deciding the winner among the campaigns willing to participate. All of the functionalities of this module are implemented through Lambda Functions written in Python, some of which use a DynamoDB table to store and retrieve the data.

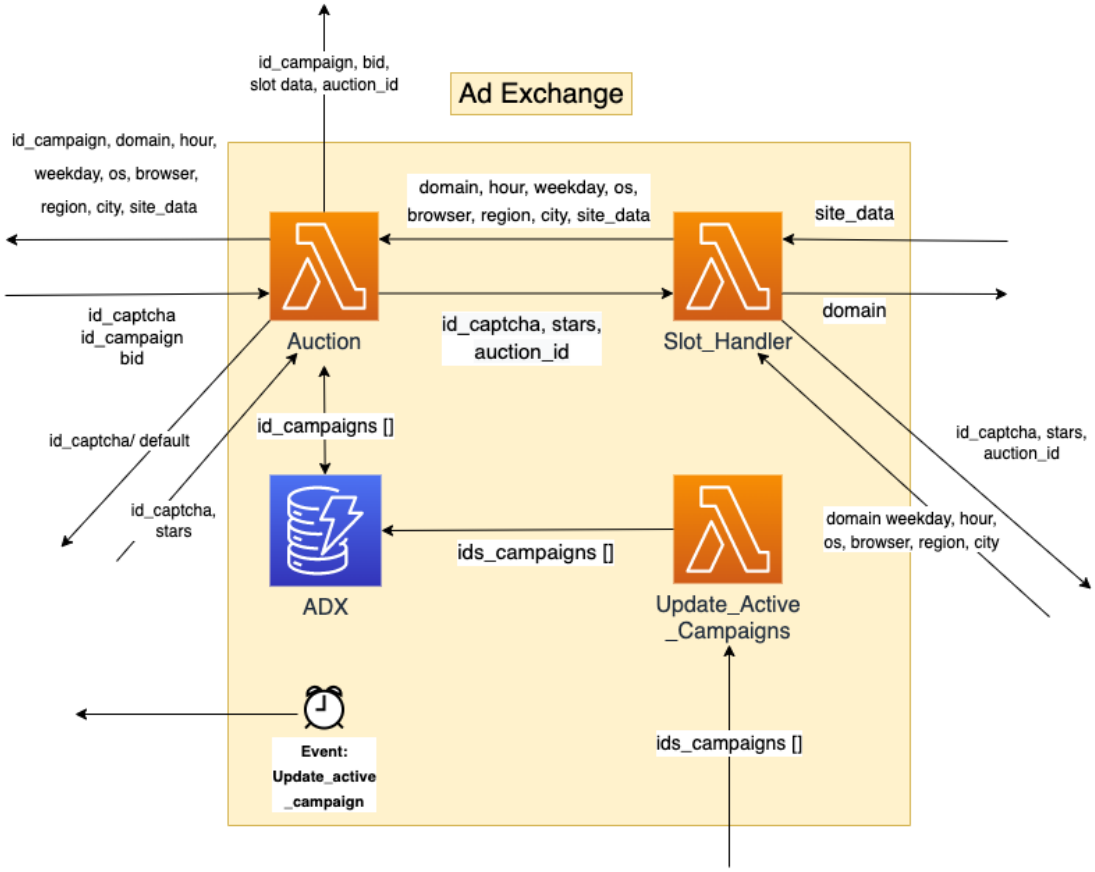

Figure 4.3: Ad Exchange Service

Slot handler

Inputs: domain (string), weekday (int), hour (int), os (string), browser (string), region (string), city (string)

Description: it retrieves all the data from the website by invoking the SSP Website - Get Site Data function using the domain as input. After that, it forwards the input and the website's data to the Ad Exchange - Auction function. Once the winner is decided, this function receives back: id captcha, stars[1], auction id.

Outputs: id captcha (string), stars (string), auction id (string)

Auction

Inputs: domain (string), weekday (int), hour (int), os (string), browser (string), region (string), city (string), site data (list)

Description: it collects from its storage all active campaigns and, for each, invokes the DSP Bidding - Get Bid function with the campaign id as input. After collecting all of the data, the following information regarding each campaign is obtained: bid, id campaign, and id captcha. Now, it may choose the winner by selecting the campaign with the highest bid. Then, two parallel operations are performed simultaneously: using the Log - Append Log function with all of the auction's data; and getting the winning campaign's CAPTCHA using the id captcha value by executing Captcha - Get Captcha. Moreover, if any errors occur or no campaigns are willing to participate, this function retrieves a default, non-sponsored CAPTCHA from Captcha - Get Default.

Outputs: id captcha (string), stars(string), auction id (string).

[1] it refers to the string which encodes the CAPTCHA challenge, see [9]

Update Active Campaigns

Inputs: active campaigns (list)

Description: it updates the list of active campaigns in the storage.

Outputs: None

Event: Update Active Campaigns

This event, implemented as a *Cloudwatch Event*, triggers the Check DSP Campaign - Check Active Campaigns function that, in turn, invokes the Ad Exchange - Update Active Campaigns function. Currently, this event starts every day at midnight.

Ad Exchange DynamoDB Table

The storage of this service was implemented through the **Ad-Exchange DynamoDB** table. This choice was made for two main reasons: we wanted to increase the speed of the system (a critical factor in RTB) and because we wanted a database that could easily handle changes in the data structure.

This storage is used as a sort of cache: it contains a copy of all the campaigns (normally stored in the DSP Campaign storage) flagged as active. This allows the Ad Exchange - Auction function to save several interactions with the DSP Campaign service. The structure is as follows:

Key	Value
Campaigns	ids

4.2.3 DSP Bidding

This module has the responsibility of deciding the bidding values on behalf of the advertisers. All of the functionalities of this module are implemented through Lambda Functions, some of which use S3 bucket storage to retrieve the machine learning model (see Get Bid function below for details).

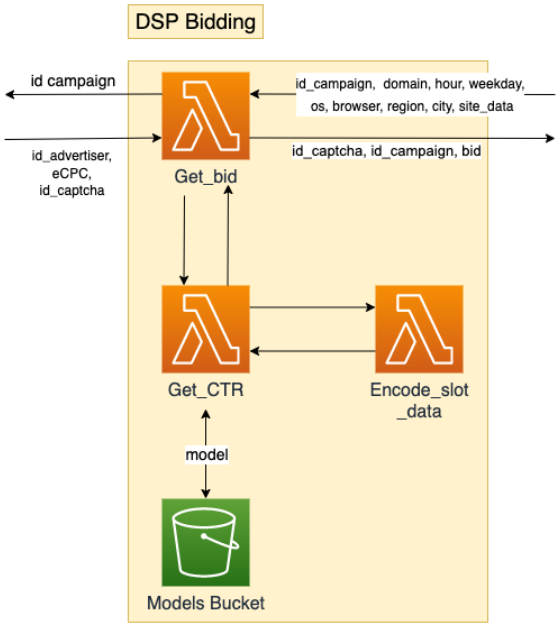

Figure 4.4: DSP Bidding Service

Get Bid

Inputs: campaign id (string), domain (string), weekday (int), hour (int), os (string), browser (string), region (string), city (string), site data (list)

Description: after retrieving the data about the campaign from DSP Campaign - Get Bid Data, it invokes DSP Bidding - Get CTR with the data regarding the slot (domain, weekday, etc.), site data, and id advertiser as inputs in order to retrieve the estimated CTR. Then, the bid value is calculated as follows (see chapter 5 for details):

$$Bid = eCTR * maxCPC$$

Outputs: id captcha (string), id campaign (string), bid (float)

Get CTR

Inputs: domain (string), weekday (int), hour (int), os (string), browser (string), region (string), city (string), site data (list), id advertiser (string)

Description: first, it retrieves features suitable for prediction from Encode Slot Data using as inputs the following features: domain, os, browser, city, and region. Then, after retrieving the model from the storage, it predicts the estimated CTR using the features (see section 5.2 for details).

Outputs: CTR (float)

Encode Slot Data

Inputs: id advertiser (string), domain (string), os (string), browser (string), region (string), city (string)

Description: it encodes the features in one-hot format.

Outputs: domain (list), os (list), browser (list), region (list), city (list)

The system might have a common model for all the advertisers or a specific model for each one (see section 5.2.5 for details). In the latter case, we would need to keep track of the various encodings used at train time for each single model. Right now, since there is no correspondence between the values of the features used to train the model and the real-life input, we simply return random values.

Models S3 Bucket

The storage for this function was implemented through an S3 Bucket. It is used to store the machine learning model(s) that Get CTR employs.

4.2.4 DSP Campaign

This module has the responsibility of handling the data of the advertisers' campaigns. All of the functionalities of this module are implemented through Lambda Functions written in Python, which use a DynamoDB table to store and retrieve all the data.

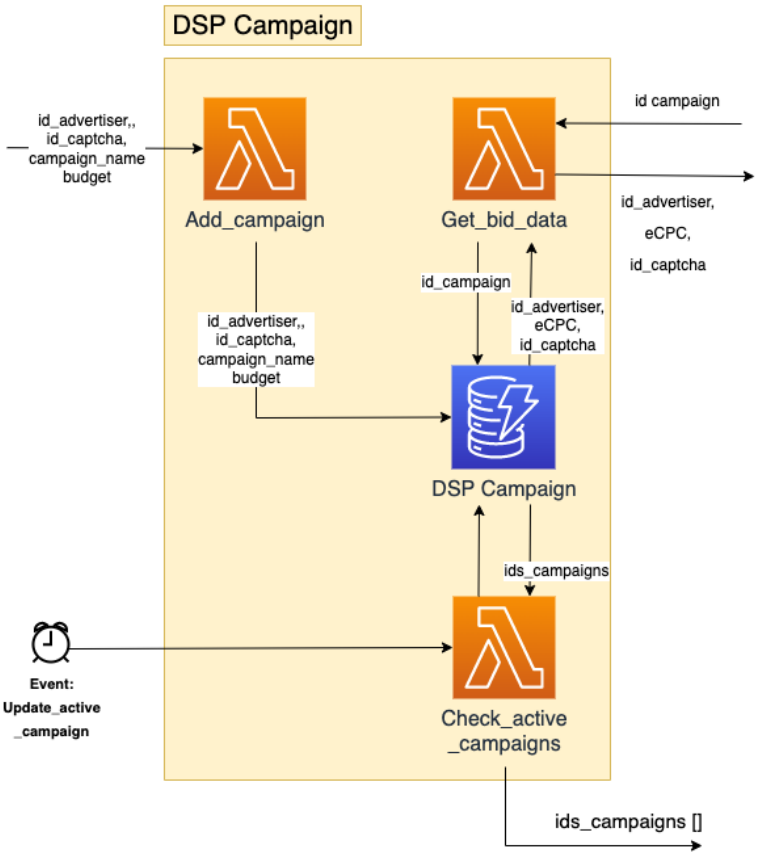

Figure 4.5: DSP Bidding Service

Add Campaign

Inputs: id advertiser (string), id captcha (string), campaign name (string), budget (int), ecpc (float)

Description: it stores the input data in the DynamoDB table.

Outputs: None

Get Bid Data

Inputs: id campaign (string)

Description: it retrieves all the data regarding the campaign from the storage.

Outputs: id captcha (string), maxCPC (float), id advertiser (string)

Check Active Campaign

Inputs: None

Description: it gets all the campaigns from the storage and filters out those that don't have the *active* flag set to True.

Outputs: active campaigns (list)

DSP Campaign DynamoDB Table

The storage of this function was implemented through the **DSP-Campaign DynamoDB** table. This choice was made for two main reasons: we wanted to increase the speed of the system (a critical factor in RTB) and because we wanted a database that could easily handle changes in the data structure (e.g., new field for the site data). Even if the structure is not fixed, we store our records as follows:

Key	Value	Value	Value	Value	Value
id Campaign	Budget	Deleted	eCPC	id Advertiser	id Captcha

The value used as the key is the id campaign because, in our use-case, when querying this database, we expect to search based on the id campaign and not the id advertiser. The id campaign is composed via a string concatenation:

$$id\_campaign = campaign\_name + " - " + id\_advertiser$$

4.2.5 Log

This module has the responsibility of handling the data from the logs. All of the functionalities of this module are implemented through Lambda Functions written in Python, which use an RDS database to store and retrieve all the data. Only the functionality related to this thesis work are going to be discussed here; for the rest, see the complementary thesis [9].

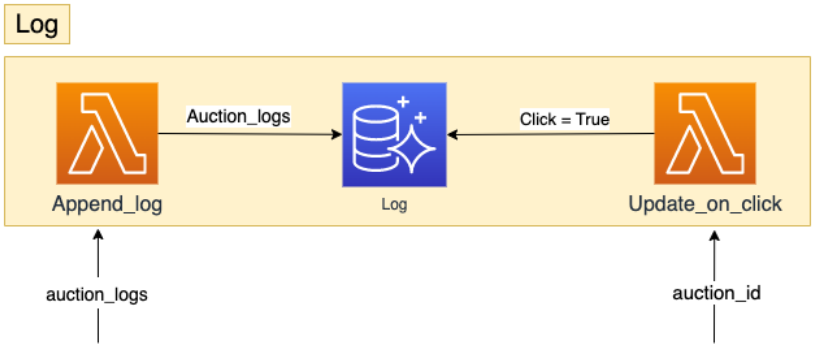

Figure 4.6: Log Service

Append Log

Inputs: auction logs (list)

Description: it stores the records in the dedicated Database.

Outputs: None

The *auction logs* parameters contain a list of data associated with a specific auction shaped as follows: id campaign, bid, auction id, domain, hour, weekday, os, browser, region, city, site data, win (True only for the winning bid), click (always False at this phase). This shape was chosen so that the logs could eventually be used for training ad-hoc CTR models.

Log RDS Database

We took the decision to implement this database as a relational one (instead of a DynamoDB as the others discussed so far) because, during the Update On Click function, we need to perform a slightly complex query (search for the winner of a

specific auction) which cannot be performed with a simple key-query. Implementing this with a key-query would be cumbersome and would not take advantage of the strength of NoSQL databases. The storage is organized as follows:

Primary Key	Attributes	Type
Id (auto increment)	id campaign	Varchar
	bid	Float
	auction id	Varchar
	domain	Varchar
	hour	Integer
	weekday	Integer
	os	Varchar
	browser	Varchar
	region	Varchar
	city	Varchar
	site data	List
	win	Boolean
	click	Boolean

4.3 APIs

API Gateway are used as an entry point to invoke the different micro-services. The API Gateway is implemented as a RESTful API to exploit the common HTTP methods to seamlessly invoke our endpoints and keep the whole system as lightweight as possible.

4.3.1 Web Access

This set of endpoints is used by advertisers and publishers to access different functionalities from the dedicated website of the platform (eventually, this API could be split into two separate APIs, each dedicated to a type of user). This thesis encompasses the following endpoints:

- **Add Website**: this resource, through a POST method, allows a publisher to add a website to the system by calling the **SSP - Add Website** function (see section 4.2.1).
- **Add Campaign**: this resource, through a POST method, allows an advertiser to add an ad campaign in the system by calling the **DSP - Add Campaign** function (section 4.2.4).

All of the resources under this API can be accessed only by authenticated users (being publisher or advertiser); this is accomplished by the **AWS Cognito** service.

AWS Cognito

AWS Cognito is a service that provides authentication, authorization, and user management capabilities to the web app. First, two user pools were created, one for advertisers and another for publishers. Through these, Cognito provides the hosted authentication pages for our application where users can sign up, sign in, and retrieve their password.

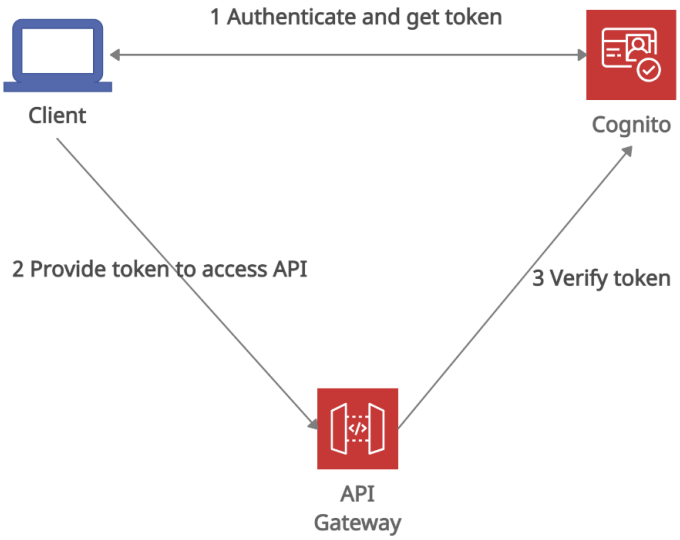

Figure 4.7: Cognito's Implicit Grant Flow

We employ the OAuth 2.0 *Implicit Grant Flow* (supported by Cognito), which means that, once a user is signed in, it receives a token and a cookie for authorization and session management.

When the application invokes a protected API that requires the user to be authenticated, it also provides the token previously received in the Authorization header of the request. In this way, Cognito can verify the identity of the user who is trying to access the API. Lastly, when a user tries to log out, the application deletes the session token and cookies.

4.3.2 Get Ad

This set of endpoints is used by the publisher's website to perform various actions regarding each ad slot. This thesis encompasses the following endpoint:

- **Get Ad**: it initiates an auction for the ad slot by calling the **Ad Exchange - Slot Handler** (section 4.2.2) function whenever the publisher's website requires a sponsored CAPTCHA over the GET method.

4.4 Front-End

This section will discuss the front-end part of the application dedicated to advertisers and publishers to manage their respective campaigns and websites (see section 2.2). Note that all the front-end operations, such as: monitoring ad campaigns, checking all active campaigns, adding CAPTCHA, etc., are not implemented since we thought that they would not add any value to our thesis work. However, the following structure is organized in such a way to allow such functionalities to be easily integrated in the future. The following image shows how the user can communicate with the front-end application.

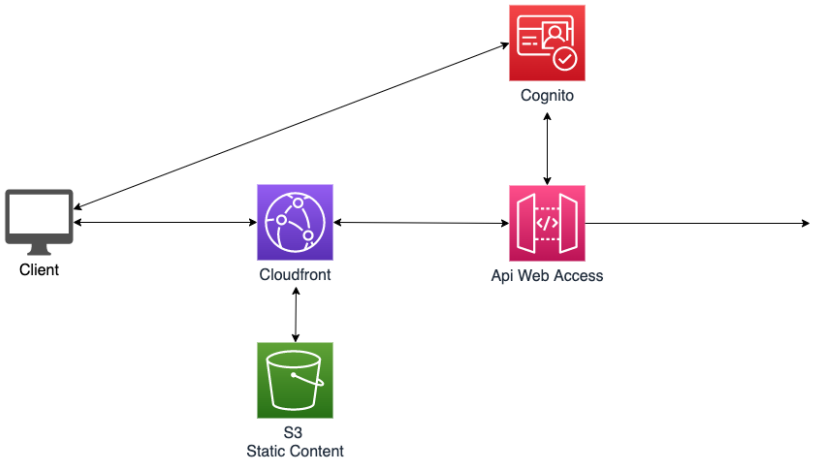

Figure 4.8: Front-end

4.4.1 Static-Content S3 Bucket

An S3 Bucket is used to store all of the static elements of the application: images, JavaScript files, HTML files, and CSS files. These static elements are retrieved by the Cloudfront distribution and served to the client whenever requested. In this way, it can increase the scalability, reliability, and speed of our static storage.

4.4.2 Cloudfront

Cloudfront is the CDN service of AWS. This service is used in order to minimize the latency between requests that can be cached by the Cloudfront edge locations. While using the application, the client makes requests for static or dynamic content. When requesting static content, the Cloudfront distribution fetches it from the Static-Content S3 Bucket. While, when requesting dynamic content, the Cloudfront distribution invokes the corresponding API from the API Gateway (see section 4.3) and responds with the data received from the business logic.

4.5 RTB Performance

As stated before, performance in terms of time is a crucial parameter while evaluating any RTB system. In our specific scenario, however, we can be more permissive since our ad is not of main importance to the user experience since it is shown in very specific situations, being a CAPTCHA. For instance, a common use case would be to show it during a sign-up operation. In such a scenario, the CAPTCHA challenge could easily be loaded while the user is compiling the associated form.

Nevertheless, I still give much weight to the performance of the RTB auction. From the AWS Cloudwatch console, I can easily monitor the performance during a specific time frame.

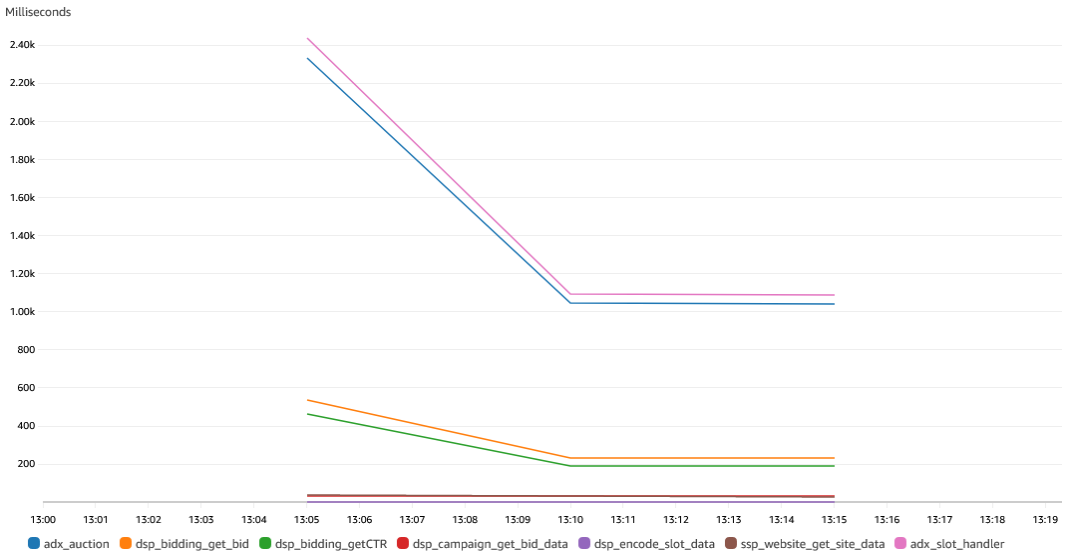

Figure 4.9: RTB Performance Test

From the graphic above, we can see that the average execution time is around 1.5s. This test was performed with the following parameters: AWS free tier plan, 10

minute duration, 10 active campaigns, and with a download connection speed of roughly 68 Mbps. Note that the number of active campaigns should not impact on the performance in any serious way, since all the operations on the DSP Bidding are performed in parallel for each campaign. The initial peak is caused by the *cold start* problem of AWS Lambda: when a function is not invoked for a long period, it will take some time to "warm up".

The last thing to highlight is that, when integrated with the CAPTCHA service, the RTB performance deteriorates quite significantly. This issue is tackled by the complementary thesis [9]; in any case, this problem can be easily solved by switching from the free tier plan on AWS to an adequate one.

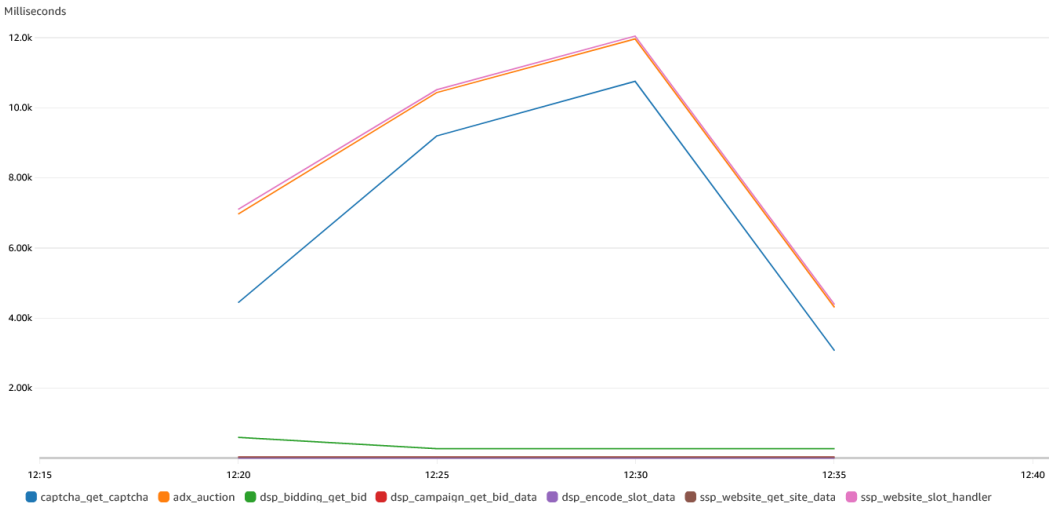

Figure 4.10: RTB Test after CAPTCHA Integration

5. RTB: Bidding Strategy & Evaluation

This chapter will cover the development and assessment of a bidding strategy used by the DSP Bidding service to produce a bid for the RTB auction, as stated in chapter 4.

5.1 Bidding Strategy

As described in subsection 3.1.2, the output of the bidding strategy must comply to a design constraint: the bid must refer to a specific impression, given that the advertiser is charged for each impression. Typically, a bidding strategy seeks to optimize a key performance indicator (KPI), such as the number of impressions, clicks, etc. In particular, I decided to employ a strategy whose objective is to get as many clicks as possible for a given budget.

$$\begin{aligned} &\max_{\text{bidding strategy}} \#click \\ &\text{subject to cost} \leq \text{budget} \end{aligned} \tag{5.1}$$

As described in section 1.1.2, there are several auction types in online advertising; among those, a *second-price auction* is employed by the Ad Exchange service during the RTB. On the other hand, the DSP Bidding service follows a *truthful bidding strategy* while producing the bids. The next sections will further elaborate these concepts, and discuss the reasoning behind these decisions.

5.1.1 Second-Price Auction & Truthful Bidding

In a second-price auction, the winner pays the amount of the second-highest bid rather than its own value (as it would in a first-price auction). While, truthful bidding refers to a strategy in which each bidder that employs it offers their private value (i.e., how much they are willing to pay for this particular slot). It is important to highlight that, in the context of a second price auction, truthful bidding is a dominating strategy[1].

[1]The dominant strategy in game theory refers to a situation where one player has superior tactics regardless of how their opponent may play.

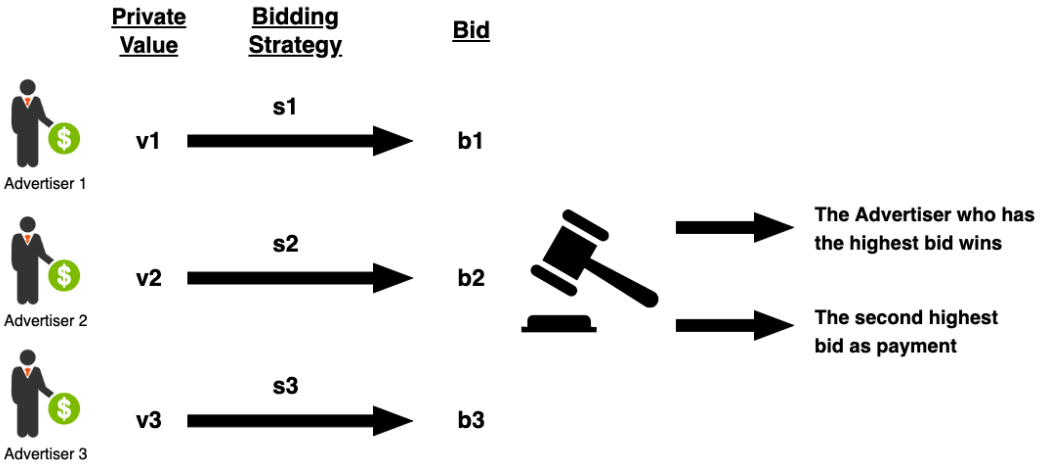

Figure 5.1: Second-price Auction Example

v_i = bidder i's private value for the object
b_i = bidder i's bid for the object
s_i = bidder i's strategy
$b_i = s(v_i)$

The payoff to bidder i with value v_i and bid b_i is:

$$\text{Payoff} = \begin{cases} v_i - \max_{j \neq i} b_j & \text{if } b_i > \max_{j \neq i} b_j \\ 0 & \text{otherwise} \end{cases}$$

Theorem: Bidding $b_i = v_i$ is a dominant strategy.

- If $b_i > v_i$ bidder i could get object and pay more than what she values it for (and thus go negative).
- If $b_i < v_i$ bidder i could fail to obtain the object; obtaining the object can get her positive payoff.

To recap, in second-price auctions, since this is the dominant strategy, each bidder's true value is known. Each participant bids the exact amount they believe the item is worth, with the winner paying the second highest bid, thus receiving a discount.

The issue arises if the advertiser's information about their private values is inaccurate. In addition, second-price auctions do not consider budget restrictions.

5.1.2 Bidding Below Max eCPC

The truth-telling strategy adopted is denoted as *Bidding Below Max eCPC* (Mcpc). The bidder's true value is expressed as:

$$bid = max\ eCPC * CTR$$

Where **max eCPC** (max expected Cost Per Click) is the upper bound of expected cost per click, and **CTR** (Click-through Rate) denotes the probability that the ad will be clicked.

I opted to proceed in this setting because it seemed to be a reasonable trade-off between effectiveness (truthful bidding being a dominating strategy) and simplicity (easy to interpret). In addition, this setting, which is rather frequent in online advertising, is extensively documented and straightforward to evaluate. It is not my intention to declare this configuration to be the most suitable for this scenario. The choice seems to be quite complex (as an example, some companies have shifted from second-price auctions to first-price auctions in recent years) and affected by factors outside the scope of this thesis. However, this decision is not binding; the system has been designed such that any modifications to the auction process and bidding strategies may be simply implemented.

5.2 CTR Estimation

This section will focus on the CTR estimation task, which is an essential element for the optimal functioning of the DSP bidding strategy.

In online advertising, the click-through rate (CTR) is the proportion of page visitors which see and then click on a particular advertisement. As previously noted, an inaccurate CTR would have a substantial impact on the bidding strategy. As a result, relying just on the average CTR would be too simplistic; it would not account for the individual ad, and the viewer who may click. For these reasons, I decided to employ a machine learning approach: to predict CTR from historical data using a model, and to get as close to the most accurate estimate as possible.

5.2.1 iPinYou Dataset

To develop the CTR estimation model, a dataset including information on advertisements and consumers who may or may not have clicked on them is needed. The choice fell on the iPinYou dataset[3], published via the iPinYou DSP production platform, and containing historical data about RTB auctions gathered by the platform during several weeks.

Several criteria related to our context and the dataset's suitability influenced this choice. iPinYou is the most used dataset for CTR estimation, therefore the findings of this work can be compared to those of other studies. In contrast to similar datasets, its column names are not encoded, which is very valuable for understanding certain features that may be relevant in a real-world scenario, and beginning to collect them in the implemented system's logs. Lastly, it is the only publicly available dataset that includes bid and prices, which are crucial for replicating a genuine RTB scenario in order to test the model.

The dataset is divided into three seasons and in each one there are four different types of log:

- **bids**: contains all the bids proposed by DSP in the recorded auctions;
- **impression**: contains all the winning bids proposed by DSP in the recorded auctions;
- **click**: contains all the impression log that resulted in a click;
- **conversion**: contains all the impression log that resulted in a conversion.

As the objective is to estimate the CTR, the bids that did not result in an impression do not carry any interesting information, thus the bid log files can be ignored. The structure of the remaining three log types can be summarize as follows:

Col #	Description	Example
0	Bid id	0153000083f
1	Timestamp	2013021800120368
2	Log Type	1
3	iPinYou id	4856431685112
4	User-Agent	Mozilla/5.0 (compatible; \MSIE 9.0; Windows NT \6.1; WOW64; Trident/5.0)
5	IP	118.81.189.*
6	Region	15
7	City	16
8	ad Exchange	2
9	Domain	1856esf1...ew6s8f1
10	URL	1yu8k1y5...6yu84k
11	Anonymous URL id	Null
12	Ad Slot id	186468468431561
13	Ad Slot Width	300
14	Ad Slot Height	250
15	Ad Slot Visibility	SecondView
16	Ad Slot Format	Fixed
17	Ad Slot Floor Price	0
18	Creative id	1g8k3...381vum8u
19	Bidding Price	753
20	Paying Price	15
21	Key Page URL	1y68ten...1g7tr35d
22	Advertiser id	2345
23	User Tags	123,5678,3456

Table 5.1: iPinYou record example

- **Timestamp**: the column uses the format of yyyyMMddHHmmssSSS;
- **Log Type**: the possible values include: 1 (impression), 2 (click) and 3 (conversion);
- **User-Agent**: the column describes the device, operation system and browser of the user;
- **Domain**: the domain of the hosting web page of the ad slot. The values are hashed;
- **Ad slot features**: like width, height, etc.

- **Ad slot floor price**: floor (or reserve) price of the ad slot. No bid lower than the floor price could win auctions;
- **Bidding price**: the bid price from iPinYou for this bid request;
- **Paying price**: the paying price is the highest bid from competitors, also called market price and auction winning price;
- **User Tags**: encoded information related to users.

According to the CPM pricing model, all monetary values (e.g., bid price, pay price, and floor price) are expressed in Chinese fen x1000.

Considerations

It is crucial to highlight that the same features might have different effects on the CTR for different advertisers. For instance, according to the data statistics evaluated by Zhang et al.[5], different advertisers obtained the greatest CTR on different days of the week; this may imply that the models must be developed individually for each advertiser.

However, in a real-life scenario, it would be more difficult to manage one model for each advertiser compared with the general one. In the following sections, the performances of the two alternatives will be analyzed.

5.2.2 Dataset Transformation

In order to formalize the iPinYou RTB data into a standard format for training a CTR model and evaluating it, I utilized Weinan Zhang's script[4], as illustrated in [5]. After executing the script, I obtained 10 distinct datasets, 9 of which were associated with a single advertiser and one comprising data from all advertisers.

Each record in these datasets maintains the same structure of the original one, as seen in Table 5.1, with the following exceptions:

- **Timestamp** column has been split into "weekday" and "hour" columns (the timestamp column is still present);
- **Log Type** column contains now only one's (i.e., impression);
- **Click** column was added. The value of this column contains one if this impression was also present in the click log file, else zero.

After this first preprocessing step, I proceeded by deleting some columns for different reasons:

- **Ad Slot features** (slot id, slot width, slot height, slot visibility, slot format, creative): in our scenario we only have one type of ad with a fixed size (CAPTCHaStar).
- **Advertiser id**: only present with the file with all advertisers.
- **User tags**: we don't have any data regarding visitors in our scenario.
- **Other columns** (bid id, log type, ipinyou id, IP, Ad Exchange, url id, url, key page): they are not useful for our purpose.

The final change I made was to split **user-agent** into os and browser.

After all of these steps, the shape of each record in the dataset is the following:

Col #	Description	Example
0	Weekday	3
1	Hour	18
2	Timestamp	20131023183901686
3	Region	40
4	City	53
5	OS	Android
6	Browser	Chrome
7	Domain	6e9c0e0bc0e584838ad187cad
8	Floor Price	5
9	Bid Price	277
10	Pay Price	222
11	Click	0

Table 5.2: Dataset record example after transformations

5.2.3 Data Preprocess

Before training the model, many preprocessing operations needed to be performed. First, I had to address the columns with missing values: *Domain* was the only column affected. I opted to replace the *null* values with *unknown* as they make up a significant portion of some datasets (both those pertaining to individual advertisers and the one that covers all of them). Regarding each of the categorical columns (os, browser, domain, city, and region), I experimented with both label and one-hot encoding. As expected, given the huge number of values for each

feature, one-hot encoding resulted in higher performance; hence, it was used to obtain all the results presented in subsection 5.2.5.

5.2.4 Training the Model

For the reasons outlined in section 5.2.1, the following approach was employed to train both the models of each individual advertiser and the one that encompasses all advertisers (so-called *overall* model for simplicity).

Some features of the dataset were maintained only to be used in the evaluation protocol (see section 5.2.5), hence I discarded them when performing the CTR estimation training (said features are: timestamp, bid price, floor price, pay price). Regarding the *advertiser id* feature, it was kept just for training the overall model.

As suggested by [6, 7], I employed a *Logistic Regression* model to estimate *CTR* in online advertising. According to [8], the LR model is suited for dealing with large-scale sparse data utilizing regularization for training and is capable of accomplishing the most of click rate prediction tasks.

The models were implemented through the use of the *Scikit-learn* library. Since the goal of this assessment is not to produce the best possible model for the task, but rather to test the approach itself (and speculate on how well it would work with the eventual data gathered by the platform), I decided to use roughly the same hyper-parameters for every model. However, I still dedicated quite an effort into the tuning process in order to obtain reasonable results. After the tuning process, I decided to use the following parameters as a starting point while training each single model:

- **maximum number of iteration** = 100
- **regularization** = L2
- **solver** = sag (Stochastic Average Gradient Descent)

5.2.5 Evaluation

Evaluation of model performance is one of the most challenging aspects of CTR estimation. In our scenario, where CTR is a critical factor of the bidding strategy and affects how much the winning advertiser is charged, traditional metrics are inadequately exhaustive and difficult to interpret. This is because CTR estimation is an imbalanced problem (the click ratio is about 0.08 in the iPinYou dataset), and we may not want to balance it since it could lead to an overestimation.

Using a traditional classification metric such as accuracy, the model may get a score of 99.9% by constantly predicting "*no click*" and missing each and every click opportunity. Even when using a more sophisticated metric such as AUC, similar issues might arise: attempting to improve the AUC (e.g., by balancing the dataset) would cause the model to overestimate the CTR; in our scenario, this would result in the advertiser paying more for no clicks.

As shown in [5], it is preferable to evaluate an aggregate of several kinds of measures, such as AUC and RMSE. In this way, the RMSE measure enables us to determine whether our estimate deviates too much from the average CTR (i.e., we are overestimating it). I actually used this approach on a dataset of a single advertiser (advertiser 1458, the one with the largest number of records); after balancing the data, although AUC score rose (+ 0.06), the RMSE also increased (from 0.029 to 0.45).

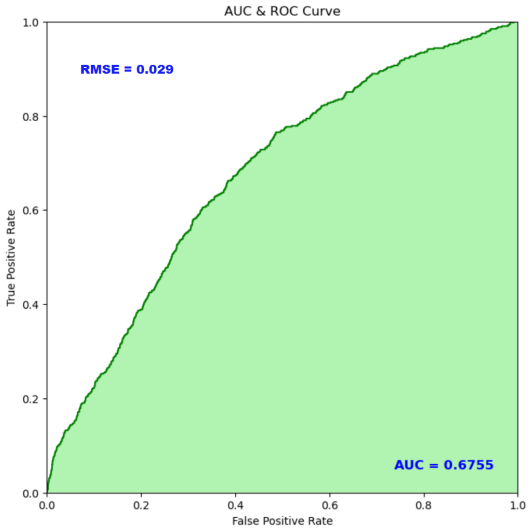

Figure 5.2: Without oversampling

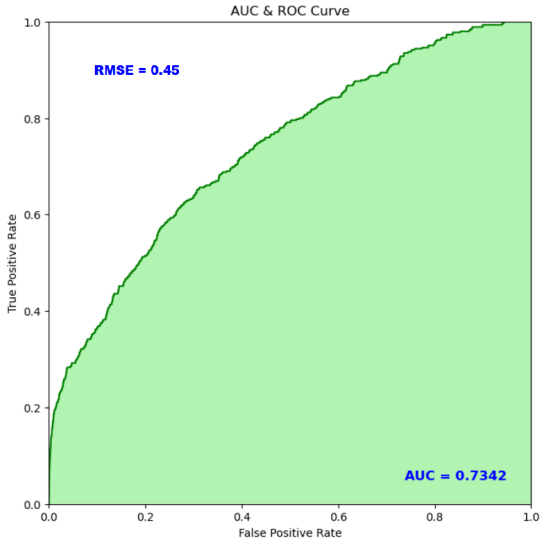

Figure 5.3: With oversampling

This indicates that the model's estimate deviates significantly from the real mean (the click probability gets overestimated). This result is due to the fact that balancing the dataset affects the data distribution (we have more clicks than in the real world). This has an influence on the collected clicks: because the likelihood of a click is considerably greater, the advertiser bids high values, exhausting its budget before to earning a click.

Nonetheless, this technique is insufficient to offer the interpretability necessary to understand how the model would perform in a real-world scenario: the number of clicks gained given a predetermined budget.

In light of these considerations, I use an evaluation framework based on [5], which gives us the real number of clicks collected from the test set.

Evaluation Protocol

Given a bidding strategy and a budget, this protocol enables us to perform a simulation by going through test bid logs and producing a KPI-based measure (i.e., number of clicks). The following is a detailed description of each step:

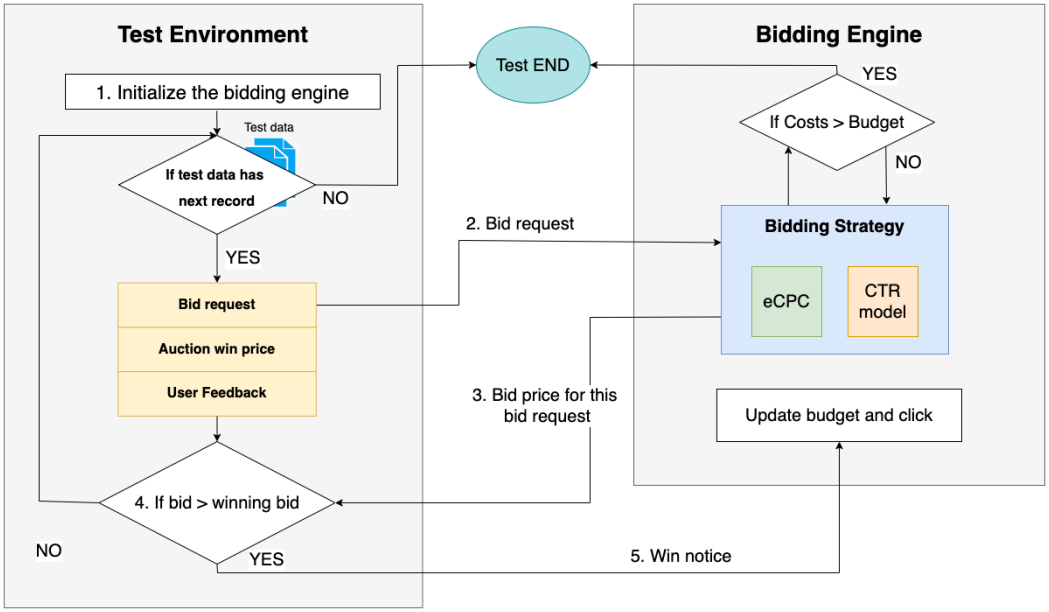

Figure 5.4: Evaluation protocol

1. The first step involves initializing variables:
   - **Budget**: determined as the total of all the charged prices in the test set (pay price) multiplied by a factor (I chose 1/2, 1/8, and 1/32 in accordance with the original work).
   - **eCPC**: derived from the train set by dividing the total pay price by the number of clicks obtained.
2. Next, process the bid requests in ascending temporal order in the bidding engine; each bid request conforms to the format outlined in Table 5.2.
3. The bidding strategy generates a bid for this item (in our scenario, it employs the estimated CTR from the model); if the cost exceeds the budget, the protocol ceases execution.
4. Simulate the auction by analyzing the ad impression logs. If the bid produced by the bidding engine is greater than the price registered as the auction winner, the advertiser wins the auction and receives the ad impression.

5. Finally, update the impressions and clicks gathered (if the impression record has the click columns set to 1). The amount paid also decreases the budget (since we are dealing with a second price auction, it would be the second highest bid).

Results

To properly analyze the performance of the models, all of the adopted metrics are displayed simultaneously (AUC, RMSE, and clicks gained). First, a comparison between a bidding strategy based on a trained model and one that employs the CTR mean (0.0008) as an estimate of the click probability.

Advertiser	Model	AUC	RMSE	Budget	Click
1458	Baseline	-	-	1/32	20
1458	Baseline	-	-	1/8	78
1458	Baseline	-	-	1/2	146
1458	Trained	0.66	0.029	1/32	26
1458	Trained	0.66	0.029	1/8	118
1458	Trained	0.66	0.029	1/2	257

Table 5.3: Baseline Comparison

This test was conducted on a particular advertiser (the one with the largest number of records) trained without oversampling (as in subsection 5.2.4) and one-hot encoding for categorical features. The results show that the performance difference widens as the budget grows.

Now that we have established that the models perform better than just bidding using the mean CTR, we can compare the performance of each advertiser's model with that of [5]. Clearly, our results do not stack up to those of the paper; this is partly attributable to the fact that we do not employ all of the iPinYou features, but just those that are suitable for our scenario (see Table 5.2).

Table 5.4: Single Advertiser Results

Advertiser	AUC	*AUC	RMSE	*RMSE	Budget	Click	*Click
1458	0.67	0.98	0.028	0.019	1/32	26	261
1458	0.67	0.98	0.028	0.019	1/8	118	502
1458	0.67	0.98	0.028	0.019	1/2	257	502
2259	0.67	0.68	0.017	0.017	1/32	7	7
2259	0.67	0.68	0.017	0.017	1/8	31	25
2259	0.67	0.68	0.017	0.017	1/2	66	86
2261	0.62	0.62	0.016	0.016	1/32	8	3
2261	0.62	0.62	0.016	0.016	1/8	28	23
2261	0.62	0.62	0.016	0.016	1/2	41	70
2821	0.64	0.64	0.023	0.023	1/32	23	16
2821	0.64	0.64	0.023	0.023	1/8	106	61
2821	0.64	0.64	0.023	0.023	1/2	112	239
2997	0.55	0.60	0.061	0.058	1/32	31	22
2997	0.55	0.60	0.061	0.058	1/8	90	78
2997	0.55	0.60	0.061	0.058	1/2	334	329
3358	0.78	0.97	0.029	0.024	1/32	29	85
3358	0.78	0.97	0.029	0.024	1/8	84	310
3358	0.78	0.97	0.029	0.024	1/2	160	310
3386	0.75	0.79	0.028	0.028	1/32	29	39
3386	0.75	0.79	0.028	0.028	1/8	96	128
3386	0.75	0.79	0.028	0.028	1/2	258	336
3427	0.70	0.97	0.026	0.021	1/32	25	67
3427	0.70	0.97	0.026	0.021	1/8	83	361
3427	0.70	0.97	0.026	0.021	1/2	207	361
3476	0.66	0.96	0.023	0.023	1/32	18	33
3476	0.66	0.96	0.023	0.023	1/8	64	148
3476	0.66	0.96	0.023	0.023	1/2	187	290

* denotes that the column refers to the results of [5].

The obtained results on a limited budget are comparable to, and often even surpass, those of the original work. Nevertheless, the performance disparity rises as

the budget grows. Obviously, the difference is more evident when the AUC gap between the models is greater.

On the basis of these outcomes, we can establish the usefulness of the evaluation framework: for certain entries, the basic metrics obtained (AUC and RMSE) are equal to those of the original paper, but still result in different effective clicks. Taking advertiser 2259 as an example, the basic metrics are same, but the performance are much different. A more extreme instance is advertiser 2997, for whom the received outcomes are higher despite having poorer AUC and RMSE. This behavior is difficult to explain in a straightforward manner due to the fact that each advertiser has unique distributions and is more impacted by different features. However, we may highlight that our performances are typically close to those of the original paper on advertisers with fewer records: as the number of records rises, our performances are outmatched.

As seen in the table below, after evaluating the bidding strategies using the overall model (the one trained on all of the advertisers), the previously described behaviors are attenuated but still present.

Table 5.5: All Advertisers Model Results

Advertiser	AUC	*AUC	RMSE	*RMSE	Budget	Click	*Click
1458	0.70	0.70	0.027	0.028	1/32	26	26
1458	0.70	0.70	0.027	0.028	1/8	116	118
1458	0.70	0.70	0.027	0.028	1/2	243	257
2259	0.70	0.67	0.027	0.017	1/32	4	7
2259	0.70	0.67	0.027	0.017	1/8	26	31
2259	0.70	0.67	0.027	0.017	1/2	49	66
2261	0.70	0.62	0.027	0.016	1/32	5	8
2261	0.70	0.62	0.027	0.016	1/8	20	28
2261	0.70	0.62	0.027	0.016	1/2	48	41
2821	0.70	0.64	0.027	0.023	1/32	21	23
2821	0.70	0.64	0.027	0.023	1/8	87	106
2821	0.70	0.64	0.027	0.023	1/2	119	112
2997	0.70	0.55	0.027	0.061	1/32	21	31
2997	0.70	0.55	0.027	0.061	1/8	95	90
2997	0.70	0.55	0.027	0.061	1/2	274	334
3358	0.70	0.78	0.027	0.029	1/32	27	29
3358	0.70	0.78	0.027	0.029	1/8	83	84
3358	0.70	0.78	0.027	0.029	1/2	155	160
3386	0.70	0.75	0.027	0.028	1/32	21	29
3386	0.70	0.75	0.027	0.028	1/8	86	96
3386	0.70	0.75	0.027	0.028	1/2	233	258
3427	0.70	0.70	0.027	0.026	1/32	20	25
3427	0.70	0.70	0.027	0.026	1/8	73	83
3427	0.70	0.70	0.027	0.026	1/2	184	207
3476	0.70	0.66	0.027	0.023	1/32	13	18
3476	0.70	0.66	0.027	0.023	1/8	54	64
3476	0.70	0.66	0.027	0.023	1/2	158	187

* results obtained with the model trained on the datasets of **single advertisers**.

As can be seen, the model performs marginally worse than all the others trained on a single advertiser. This creates a trade-off between the performance of the models and the system's manageability. This is due to the fact that additional models would need separate training, storage, selection during the RTB, etc. While

having a single model may be more scalable in terms of the quantity of advertisers, its performance is still worse and this gap may grow in the future as more data on each advertiser is collected. Using a single model may even provide a solution to some issues highlighted in [5], such as advertisers not wanting their data to be used to train a model that could benefit their potential competitors.

6. Conclusions

6.1 Obtained Results

I was able to reach many milestones upon completion of this thesis. In conjunction with the complementary thesis [9], I designed a cloud architecture (section 3.1) that offers the following main functionalities: it provides a sponsored CAPTCHA via an RTB system and offers several utility functions to advertisers and publishers. Moreover, the architecture, which is based on a microservices framework, is able to meet a number of requirements that we regarded critical, such as scalability, ease of management, and adaptability to future developments, particularly to the bidding strategy and auction.

I was responsible for developing the APIs (section 4.3) and Real Time Bidding-related services (section 4.2) for this system, implemented through AWS: SSP Website, Ad Exchange, DSP Bidding, DSP Campaign, and Log. The primary objective of these components is to enable the RTB system's functionality. Ad Exchange manages RTB auctions and determines the winning advertisers. DSP Campaign and SSP Website are responsible for processing advertiser and publisher data respectively, while DSP Bidding is responsible for managing the bidding strategy using a CTR estimation model.

The work involved in developing the actual bidding strategy included determining the strategy to be applied and the kind of auction (chapter 5). I choose to adopt a *Bidding Below Max eCPC* strategy in conjunction with a *second-price auction*. This choice necessitated the development and assessment of a Click-Through Rate prediction model. The aforementioned model was developed using machine learning, trained on the iPinYou dataset (subsection 5.2.1), and evaluated using an ad-hoc evaluation framework (section 5.2.5) that gives helpful insights into the model's practical performance. In addition, the acquired data presented an intriguing trade-off that need a future decision to be made: a universal model that can forecast the CTR for every advertiser is simpler to manage but performs worse than a set of advertiser-specific models.

6.2 Future Work

During the development of this thesis, a number of possible future works have emerged; some are functionalities that I was unable to integrate for different reasons (e.g., lack of data), while others are improvements to the current project.

- Currently, I do not possess or utilize any information about the website itself. Obviously, in a real-world scenario, the most essential website features (e.g., category, language) must be gathered and included into the CTR estimation model (I was limited by the data provided by the iPinYou dataset, as shown in Table 5.1). In anticipation of this, the system was designed so that data can be added to websites at any moment and data about any advertiser can be modified quickly.

- Using the same logic as the website, information about the visitors of any publisher's website might be included into the CTR model to improve its performance. This might be accomplished by integrating a Data Management Platform[1] into our system.

- Given the peculiar nature of our system, adopting a customized pricing strategy may be intriguing (i.e. CPM, CPC). In our instance, obtaining the CAPTCHA alone should not be considered an impression, since the stars do not immediately create a significant shape. A potential solution would be to monitor the percentage of completion of a CAPTCHA (i.e., how near the user is to the solution) and only charge the advertiser for each impression if the user achieves a certain completion threshold. So as not to waste the impression, a visitor who refreshes the CAPTCHA (i.e., requests another test) might be shown the original ad image.

- Currently, the advertiser may only create campaigns with the goal of maximizing clicks. However, it is quite typical for advertisers to seek to enhance the *awareness* of their business, and as a result, they aim to optimize the amount of impressions. In the future, it will be necessary to implement this kind of campaign and corresponding bidding strategy.

- The subject of CTR estimation has made significant progress in recent years, with several research focusing on deep neural networks for this task. For instance, [8] propose a solution based on DNNs that significantly outperforms simpler approaches such as the one used in this thesis. Integrating such

[1] a platform that collects, organizes, and analyzes third-party data in one location

sophisticated solutions into the system might result in the accomplishment of substantial benefits.

- RTB-based display advertising may have a wide range of market prices. This thesis focuses mostly on estimation-based bidding strategies. In contrast, the market price is a significant factor in bidding decisions. In repeated auction games with a budget constraint, the best bidding strategy is not truth-telling, but it does rely on the market price distribution. Therefore, the research challenge of modeling the market price distribution is crucial for establishing a better bidding strategy and, if eventually integrated into the system, might result in improvement.

A. Code

```python
def lambda_handler(event, context):
    auction_id = context.aws_request_id
    slot_data = event['slot_data']
    campaign_ids = get_active_campaigns()
    # check if there is at least one active campaign
    if len(campaign_ids) > 0:
        # get data relative to the bid from each campaign
        bids_data = asyncio.run(get_bids_async_wrapper(campaign_ids, slot_data))
        # exclude DSPs which bidded -1*CTR -> do not partecipate in the auction
        bids_data = [x for x in bids_data if x['bid'] > 0]
        # retrieve bids from every participant
        bid_values = [x['bid'] for x in bids_data]
        n_bids = len(bid_values)
        # check if there are participants
        if n_bids > 0:
            # retrieve the index of the max bid
            winner_index = max(range(n_bids), key=lambda i: bid_values[i])
            # retrieve data of the winner
            winner = bids_data[winner_index]
            winning_id_captcha = winner['id_captcha']
            winning_stars = get_captcha_stars(winning_id_captcha)
            # retrieve second price
            bid_values.sort(reverse=True)
            second_bid = bid_values[1] if n_bids>1 else winner['bid']
            log(auction_id, slot_data, bids_data, winner_index)
        # if there are no bids
        else:
            # return default captcha
            winning_id_captcha, winning_stars = get_default_captcha()
    # if there is no active campaign
    else:
        # return default captcha
        winning_id_captcha, winning_stars = get_default_captcha()
  return (auction_id, winning_id_captcha, winning_stars)
```

Listing A.1: Main Code Ad Exchange-Auction Lambda Function

7. Acknowledgements

Numerous persons have contributed, directly or indirectly, to the achievement of this objective.

I want to start by expressing my gratitude to **Prof. Gabriele Tolomei**, the thesis advisor, and **Prof. Mauro Conti**, the co-advisor, for their involvement in this project and for closely monitoring me throughout the entire time of the work.

I should also like to thank my colleague and friend Bryan Ferracuti, with whom I essentially spent every day of this master's degree journey. In addition to obviously supporting me, Bryan made the intense pace of study both more engaging and significantly more enjoyable.

Then, I'd like to express my sincere gratitude to my family for continuing to support and inspire me so that I could fulfill my ambition of obtaining a master's degree.

Finally, I'd like to thank my friends for supporting me in achieving this important goal by reducing my stress through their constant and thoughtful presence.

8. Bibliography

[1] Mauro Conti, Claudio Guarisco, Riccardo Spolaor. *CAPTCHaStar A Novel CAPTCHA Based on Interactive Shape Discovery.*

[2] Mohit Agarwal, Dr. Gur Mauj Saran Srivastava. *Cloud Computing: A Paradigm Shift in the Way of Computing.*

[3] iPinYou Demand-side Platform. https://contest.ipinyou.com

[4] Weinan Zhang. https://github.com/wnzhang/make-ipinyou-data

[5] Weinan Zhang, Shuai Yuan, Jun Wang, Xuehua Shen. *Real-Time Bidding Benchmarking with iPinYou Dataset.*

[6] Oliver Chapelle, Eren Manavoglu, Romer Rosales. *Simple and scalable response prediction for display advertising.*

[7] Gouthami Kondakindi, Satakshi Rana, Aswin Rajkumar, Sai Kaushik Ponnekanti, Vinit Parakh. *A Logistic Regression Approach to ad Click Prediction.*

[8] Guojing Huang, Qingliang Chen, Congjian Deng. *A New Click-Through Rates Prediction Model Based on Deep&Cross Network.*

[9] Bryan Ferracuti. *Design and Implementation of a Sponsored CAPTCHA Platform: Integration of CAPTCHaStar for Display Advertising and Resiliency Assessment of Adversarial Attacks.*