

Quantum Web Platform

Felix Chelu

felix.c@quantumgaming.com

- 1 [Version History](#)
 - 2 [Introduction](#)
 - 3 [System Structure](#)
 - 4 [Backend Structure](#)
 - 4.1 [Data Cluster](#)
 - 4.2 [Micro Services](#)
 - 4.2.1 [Micro Service Types](#)
 - 4.2.1.1 [Web Micro Service](#)
 - 4.2.1.2 [Content Micro Service](#)
 - 4.2.1.3 [Notification Micro Service](#)
 - 4.2.1.4 [Sanity Micro Service](#)
 - 4.2.1.5 [Report Micro Service](#)
 - 4.2.1.6 [Export Micro Service](#)
 - 4.2.1.7 [Casino Game Integration Micro Service](#)
 - 4.2.1.8 [Sportsbook Integration Micro Service](#)
 - 4.2.1.9 [Casual/Social Game Integration Micro Service](#)
 - 4.2.1.10 [Payment Integration Micro Service](#)
 - 4.2.1.11 [Affiliate Integration Micro Service](#)
 - 4.2.1.12 [CRM Integration Micro Service](#)
 - 4.2.1.13 [Segmentation Micro Services](#)
 - 4.2.1.14 [Campaign Micro Service](#)
 - 4.2.1.15 [Infrastructure Micro Service](#)
- 5 [Technology Stack](#)
 - 5.1 [3rd-party Services](#)
 - 5.2 [Frontends](#)
 - 5.3 [Data Consistency](#)
 - 5.4 [Parallel Data Modification](#)
 - 5.5 [Atomic Data Modification](#)
 - 5.6 [Data Sanity](#)
 - 5.7 [Data Reporting](#)
 - 5.8 [Data Export](#)

Version History

Version	Date	Author
v0.3	2024-01-28	Added further details

v0.3	2023-04-10	Update
v0.2	2021-09-03	Update
v0.1	2020-08-28	First draft

Introduction

The following document introduces the key concepts and components of the Quantum Web Platform - in the following abbreviated as **QW**.

 Document is still work-in-progress. It's a draft and content will be added continuously.

System Structure

The whole system - Frontends, Backend micro-services, CDN, Load Balancer, etc. - is designed to run in the *AWS Cloud*. Wherever possible and useful - not to reinvent the wheel - 3rd Party Services are used.

[Figure 1](#) visualizes the top-level structure of the system.

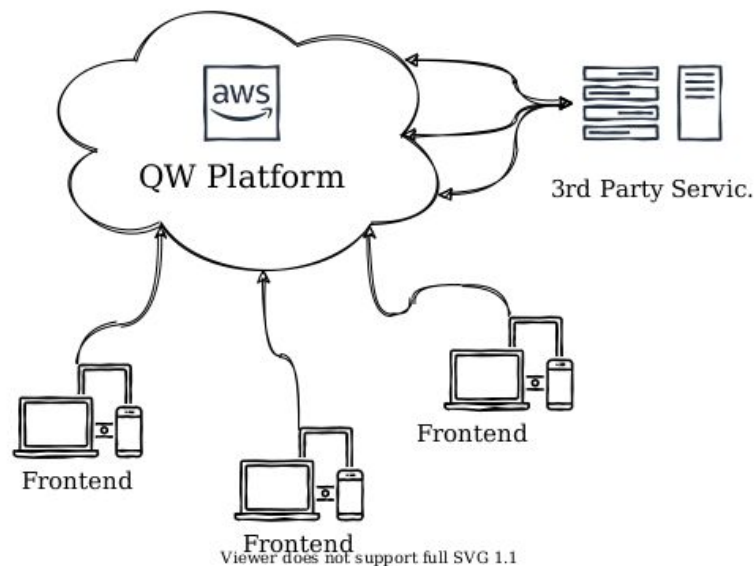


Figure 1. System Structure Overview

The frontend applications, the casino brands, or the Back-Office (Administration/Management) application are so-called SPAs (Single Page Applications) that are loaded via CDN (AWS *CloudFront* → *S3*).

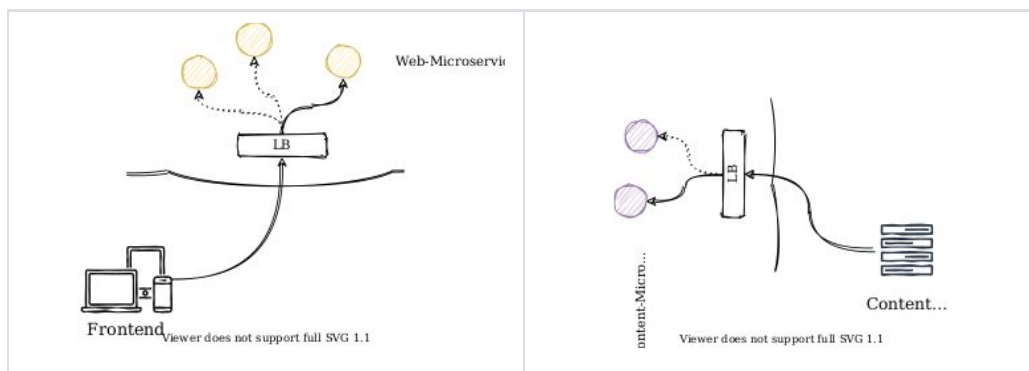
After the SPA is loaded and initialized on the client devices, it connects to the backend via persistent websockets (*STOMP*, see <https://stomp.github.io/stomp-specification-1.2.html>) using

a load balancing layer (AWS Application Load Balancer). The STOMP payload/body are JSON documents.

All static content, like the frontend application (SPA) itself or static images, styles, etc. are delivered from the CDN and only the API calls go directly to the backend services. The load balancing layer is introduced

- due to security reasons,
- and it hides any restarts of the web services in the backend from the frontend.

All microservices, that dispose endpoints to any 3rd-party services, e.g. Game or Payment providers are behind load-balancers due to the same reasons.



Backend Structure

The backend can be split into two main components, as highlighted in [Figure 2](#).

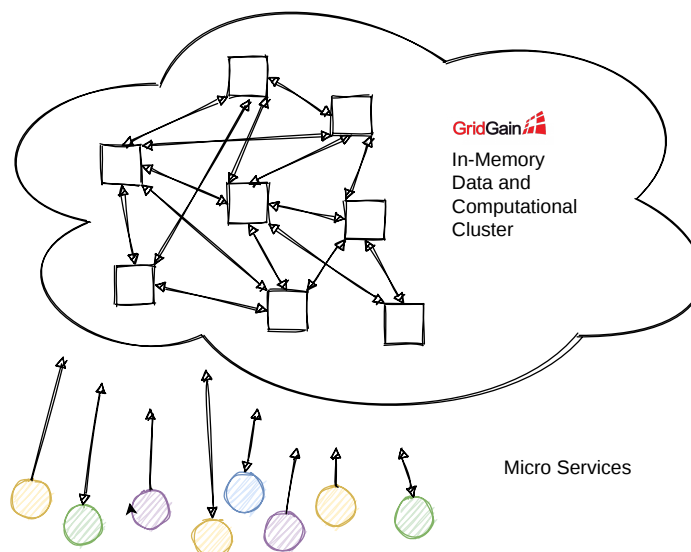


Figure 2. Backend Structure Overview

The backend system is designed to have NO single-point-of-failure. Every service can be scaled horizontally and so every service/server can be stopped/restarted at ANY time and if replaced in time, business continuity is guaranteed. The correct system size (scale) is defined by the system load and reliability requirements vs. operation costs.

Data Cluster

The key component of the backend is *GridGain: In-Memory Data Grid and Computational Platform*, see <https://www.gridgain.com>. The underlying open-source product is Apache Ignite, see <https://ignite.apache.org>. Given technology is delivering a very performant and scalable data architecture. For a deeper insight I recommend reading the corresponding documentation at Apache and *GridGain*.

GridGain can be compared with other products like Redis, Hazelcast, Geode and Memcached but probably also other ones not listed here. There are massive articles on the net comparing these products, e.g. <https://db-engines.com/en/system/GridGain%3BGeode%3BHazelcast%3BMemcached%3BRedis> and they all have their pros and cons. The final reason for going with *GridGain* is the sum of following features

- ANSI99 SQL query support
- ACID transactions
- Highly scalable
- Distributed in-memory performance
- Subscriptions and notifications on data manipulations
- Data affinity / collocation
- Open source solution by Apache

Like in standard SQL database solutions, a data schema/model is defined. In terms of *GridGain* it's named a cache. Each cache is configured regarding, redundancy, persistence, transactional behaviour, etc.

New nodes (further instances in the cloud) can easily join or leave the cluster, see [Figure 3](#). Each topology change will shortly stop the cluster for the rebalancing process, where each cache can newly redistribute its data to equally distribute load on all nodes.

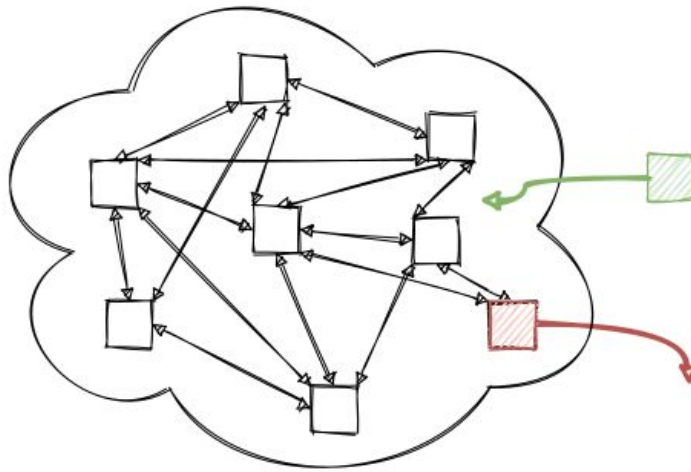


Figure 3. Data Cluster Scaling

Apache Ignite was tested using thousands of nodes and petabytes of data, see [Distributed Database - Apache Ignite®](#), so we expect no practical limit in the up-scaling process.

The whole data cluster is also available as a service (SaaS), e.g. *GridGain* is offering an enterprise solution: [Nebula](#) to fully maintain and guarantee a working cluster.

Micro Services

The microservices are the real workers in the backend that are triggered by incoming web requests, cluster events, scheduled jobs, etc. to generate or manipulate data.

All microservices are completely stateless, as whole state is stored in the data cluster. They are small software units that focus on one specific task in the system, e.g. web-services are handling the API calls from the frontends.

There are many types of microservices, and they all can easily scale up or down, as shown in [Figure 4](#). This rescaling process is absolutely transparent to the frontend, or the 3rd party services, and does not block or interfere with the data cluster.

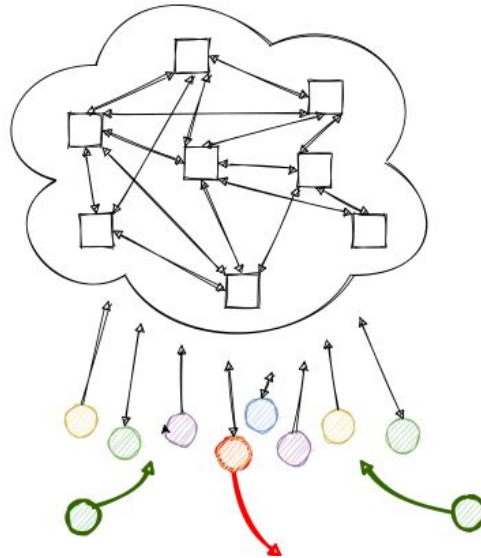


Figure 4. Micro Services Scaling

There is only little direct communication (messaging) between microservices. The brain of the whole system remains the data cluster. Microservices create or modify data in the cluster and are getting notified about data creation or data modification that they have subscribed for. E.g. a microservice needs to execute some tasks in case of a new player login. So it subscribes for the creation of new login records and gets notified when such records get created.

Micro Service Types

The following chapter list all available microservice types with short description of their responsibilities.

Web Micro Service

The *Web Micro Service* deals with all casino frontend communication. Every *Web Micro Service* that joins the backend will after its initialization register at the corresponding load balancer and starts receiving incoming web socket connections. Every incoming websocket connection starts with a handshake request, where the *Web Micro Service* executes some pre-checks and provides the casino frontend a basic configuration and environment information. An incoming session is identified using

- a `gsid` (global session ID)
stored as cookie in the browser
- and a `connNumber` (connection number)
generated by the server at every connection handshake.

Multiple tabs within the same browser, will share the `gsid`, but will have an increased connection number.

When processing the connection handshake, a geo-ip and a user-agent resolution will be triggered to query basic client information. Also - in case of an affiliate link - an affiliate partner is available. These resolutions are already used for basic session segmentation, and can already provide a customized player experience, e.g. customized content and translations.

The *Web Micro Services* subscribe for many different player related data in the data-cluster and will be notified in case of relevant changes. Such notifications, e.g. changing of the language/locale, mostly trigger updates to the connected frontends and provide real-time player experience, e.g. for jackpots and other campaign progress.

In case of a reconnection, the connection and the handshake can occur with another *Web Micro Service*, but due to stateless nature of all microservices, no session stickiness is required, and player will not even notice the reconnection, that was transparently handled in the background.

Content Micro Service

Content will be managed and provided by *Contentful*, an enterprise content management system, see [🌈 Why Contentful?](#) .

Every casino has its own *Contentful* space, where a content team - split into different roles with different permission - can easily manage their content visible on the frontends in different languages and for different environment, like staging or production.

Different content can be created for different device types or ip resolved country codes.

Introducing a new language can be fully done by the content team, and no further development related requirements are needed by Quantum Gaming team.

The content structure, so called content model/types, is provided for every *Contentful* space by us, and only the content entries themselves are managed by the content team.

📅 Soon - with one of the upcoming releases - the full session segmentation features of the backend system, can be used to provide different content per session segment.

The *Content Micro Service*

1. fetches the content from *Contentful*
2. prepares content bundles for the casino frontends
3. and uploads well-structured and packaged content files to CDN, from where the casino frontends will fetch them.

Contentful informs via webhooks in case of content changes, and depending on what data was changed, the *Content Micro Service* will re-fetch the changed content data and update the

corresponding CDN files.

Notification Micro Service

Any of the microservice can at any time request to send a notification to a player or a system operator, e.g. successful or failed player deposit. The notification request will be stored and asynchronously processed by the *Notification Micro Service*.

A notification request will be retried to process till it succeeds.

Currently, mail and online-popup notifications are supported.

Processing a notification by a *Notification Micro Service* consists of following two steps:

1. produce recipients list

e.g. in case of a successful payment notification, following recipients are generated

- online-popup f. the player
- email f. the player
- email f. casino operators

2. consume previously generated recipients list

by generating the message body (using correct language and locale)

and then

- sending a mail
- or messaging corresponding *Web Micro Service* to send *STOMP* message to the corresponding player.

Sanity Micro Service

The *Sanity Micro Service* is listening for all player related data changes, especially monitoring the player transaction related data. Using an appropriate debounce mechanism to not overload the system, data sanity will be continuously monitored and in case of any detected data inconsistency reported.

Only sane data will be approved for the reporting layer.

Report Micro Service

The *Report Micro Service* listens for data-sane approvals from the *Sanity Micro Services* that trigger the data processing and preparation for the reporting tools.

The microservice will take the sane player and player transaction data and provide it via a report optimized data structure into decoupled and report optimized data regions.

This is the data, that can be accessed via the BackOffice reports or will be exported to external BI or CRM systems.

The *Export Micro Service* uses the previously generated report data from the *Report Micro Service* to

- accumulate per
 - casino
 - player
 - date period
- or export to external systems
 - Curacao Gaming Association
 - Affiliates Provider
 - MyAffiliates (Affiliates Provider)
 [MyAffiliates: Affiliate Marketing Software](#)
 - Affilka
 [iGaming Affiliate Software & Tracking Platform | Affilka](#)
 - Wynta
 [Wynta - Scale up with Wynta!](#)
 - ReferOn
 [ReferOn - Revolutionizing Affiliate Marketing & Reporting](#)
 - CRM
 - FLOWS
 [Home](#)
 - InTarget
 [intarget: flowing digital](#)
 - CustomerIO
 [Customer.io Platform: Automation, Data, Content Creation](#)

In progress ...

Currently, following game providers are integrated

- Tom Horn
- Relax Gaming
- Skywind
- Pragmatic
- Evoplay

- Tiger Games

Complete communication between backend system and external Game Provider services, are handled by the corresponding *Game Micro Service*.

E.g. following [Figure 5](#) illustrates the game play flow:

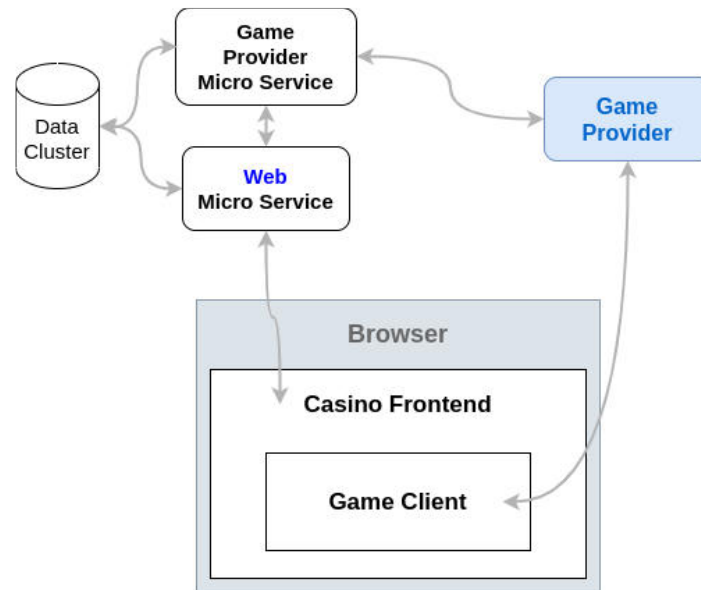


Figure 5. Game Provider Communication

Only the *Game Micro Service* is involved in any communication with the 3rd-party game provider and only the *Web Micro Service* communicates with the casino frontend.

The following table, lists all communication, when starting a new game session.

From		To
Casino Frontend	→	<i>Web Micro Service</i>
A new game session is requested. + The <i>Web Micro Service</i> does some basic checks regarding the player.		
<i>Web Micro Service</i>	→	<i>Game Micro Service</i>
An interservice request is sent to corresponding <i>Game Micro Service</i> to start a new game session. + The <i>Game Micro Service</i> does some basic checks regarding the requested game and its configuration.		
<i>Game Micro Service</i>	→	Game Provider
A new game session is registered at the game provider.		

If all succeeds, the corresponding game client with correct parameters will be started inside the casino frontend.

Sportsbook Integration Micro Service

Currently, following sportsbook providers are integrated

- Atlas

The process of placing sportsbook bets, is similar to the casino games.

Casual/Social Game Integration Micro Service

 Social/casual games are not part of this doc.

Payment Integration Micro Service

Currently, following payment providers are integrated

- Payment IQ
- Praxis

Affiliate Integration Micro Service

Currently, following affiliate providers are integrated

- MyAffiliates
- Affilka
- Wynta
- ReferOn

CRM Integration Micro Service

Currently, following CRM services are integrated

- Flows
- InTarget
- Customer IO

Segmentation Micro Services

Player and games can be segmented and the resulted player or game segments can be used throughout the entire system. The corresponding *Segmentation Micro Service* periodically check the segmentation rules and calculates what players and games will result in which segments.

Campaign Micro Service

The *Campaign Micro Service* deals with all date and period specific tasks related to campaigns, e.g. campaign expirations.

Not yet implemented!

The infrastructure microservice is monitoring the system load, and following the configured rules it handles the up- and downscaling of the system components. E.g. A high load on a specific game provider, triggers the instantiation of a further game provider integration microservice to handle the increased load.

Technology Stack

All code is done using

- Community version
 - Java 11
 - Spring 5
 - Jackson
 - Slf4J / Logback
 - Mapstruct
 - Neovisionaries (ISO Country, Currency, Language, etc. codes)
 - Lombok

and following development tools used

- Maven
- GitLab
 - Continuous Integration
 - Artifactory
- JetBrains IntelliJ
- Docker
- Lets Encrypt

The system is hosted at AWS, using following AWS services

- EC2
 - Instances
 - using AL2023
 - New Relic monitoring service
 - Load Balancers / Target Groups
- Lambda

- S3
- CloudFront
- CloudFormation
- Amplify
- SES
- Certificates

The whole AWS configuration was reviewed and approved by an external audit company as part of a so-called WAR (well-architected-review).

3rd-party Services

Many of the tasks of an online gaming platform are outsourced to existing providers that provide such services. The following list, gives an overview of such 3rd-party services that are integrated into the QW Backend System from one or multiple service providers

- GeoIP Resolution
 - [📍 IP Geolocation API with Threat Intelligence](#)
[📍 IP Geolocation API with Threat Intelligence](#)
 - MaxMind
[📍 Industry leading IP Geolocation and Online Fraud Prevention | MaxMind](#)
- User-Agent Resolution
 - Udger
[📍 User Agents Analysis :: udger.com](#)
- Content Management
 - Contentful
[🌈 Content that scales. Experiences that convert.](#)
- Social Logins
 - Google
 - Facebook
 - Apple
 - Microsoft
- Payment Services / Aggregators
 - PaymentIQ
[🔗 Payments to grow your world | Worldline Global](#)
 - Praxis
[🏦 Praxis Tech | Payment Orchestration Platform](#)

- Game Providers / Aggregators

- Tom Horn

 [Tom Horn Gaming | Unique Online Casino Game Provider](#)

- Relax Gaming

 [Relax Gaming | Casino Supplier of Slots, Bingo, and Table Games](#)

- Skywind

 [Best Gambling Software Provider | SkywindGroup Holdings LTD](#)

- Evoplay

 [Evoplay - Leading Slots and Casino Games Provider](#)

- Pragmatic

 [Bester Casino-Software- und Slot-Anbieter | Pragmatic Play](#)

- PoggiPlay

 [PoggiPlay welcomes you!](#)

- Spribe

 [SPRIBE • INNOVATIVE CASINO GAMES](#)

- Tiger Games

<https://play.betrnk.games/>

- Messaging Services (Mail, SMS, etc.)

- AWS SES

 [Cloud Email Sending Service - Amazon Simple Email Service - AWS](#)

- SendGrid

 [SendGrid Email API and Email Marketing Campaigns](#)

- Twillio

 [Conversational AI and APIs for SMS, Email, Voice](#)

- Support Ticket Services

- ZenDesk

 [Zendesk: Transform Customer & Employee Service with AI Agents](#)

- CRM

- FLOWS

 [Home](#)

- InTarget

 [intarget: flowing digital](#)

- CustomerIO

 [Customer.io Platform: Automation, Data, Content Creation](#)

- Sportsbook Providers / Aggregators
 - Atlas
 - [Atlaslive iGaming Platform | Sportsbook & Casino Solutions](#)
 - First
 - [FIRST: Tier-1 Sports Betting Platform & Technology Provider](#)
- Currency Conversion Services
 - Fixer
 - [Fixer API - Foreign Exchange Rates & Currency Conversion API](#)
- Certificate Authority Services
 - Letsencrypt
 - [Let's Encrypt](#)
- Domains Registry Services
 - GoDaddy
 - [Domain Names, Websites, Hosting & Online Marketing Tools - GoDaddy.](#)
- Monitoring Services
 - New Relic
 - [Observability built to understand AI](#)
- Hosting Services
 - AWS
 - [Cloud Computing Services - Amazon Web Services \(AWS\)](#)
 - Vultr
 - [SSD VPS Servers, Cloud Servers and Cloud Hosting](#)
- Affiliate Services
 - MyAffiliates
 - [MyAffiliates: Affiliate Marketing Software](#)
 - Affilka
 - [iGaming Affiliate Software & Tracking Platform | Affilka](#)
 - Wynta
 - [Wynta - Scale up with Wynta!](#)
 - ReferOn
 - [ReferOn - Revolutionizing Affiliate Marketing & Reporting](#)

All 3rd-party integrations are encapsulated into a Spring service layer.

A generic interface declares required methods for an integration of given service type, e.g. geo-resolution and only this interface is used by the microservices. All API specifics are hidden

from the microservices and 3rd-party services can be in most of the cases transparently replaced by another service by only replacing the corresponding service type API integration.

Frontends

Frontend applications are not bound to any technology, but can be freely chosen. They are very loosely coupled to the backend, implementing a strictly defined websocket (WS) API.

ALL content related tasks are totally decoupled from the backend. Managing translations, banners, terms & conditions, different player policies, blogs, howtos, game descriptions, etc. can be FULLY done without any backend interference and so without backend developers.

A content management system (*Contentful*, [Content that scales. Experiences that convert.](#)]) deals with ALL content related tasks.

We offer libraries for *Angular* and *React* that handle following required tasks when implementing a new casino platform:

- Communication Client — Providing high-level methods to deal with request-responses, notification subscriptions or communication error handling. — Providing connection and re-connection handling with clear events to support a fast frontend integration.
- Session Handling — Providing high-level methods to deal with sessions, timeouts, languages, logins, etc.
- Content Handling — Helper methods to faster integrate external content and provider content updates on-the-fly.
- Notification Handling — Providing high-level methods to deal with notification pushes from the Backend System.

Data Consistency

To keep a consistent data set in the data cluster following two main common topics needs to be handled.

1. Parallel data modification
2. Atomic data modification

GridGain offers - similar to relational databases - multiple synchronization methods to handle this. For more details I recommend reading [Concurrency Modes and Isolation Levels](#) .

Parallel Data Modification

Multiple parallel requests raised from external or internal sources can trigger at least partially data modification of the same data records and this needs to be synchronized properly.

For the online gambling market, this challenge is a little simplified as players not really interact with each other. There is no difference from the player perspective if he/she plays alone on a gambling system, or he/she shares it with thousands of other people. This might change with social gaming - but is currently no requirement. The synchronization of data accesses and modification can get solved on a per-player basis. We ensure that no jobs/tasks that access or modify one player's data will run in parallel. This will be guaranteed by a simple locking system shown in [Figure 6](#).

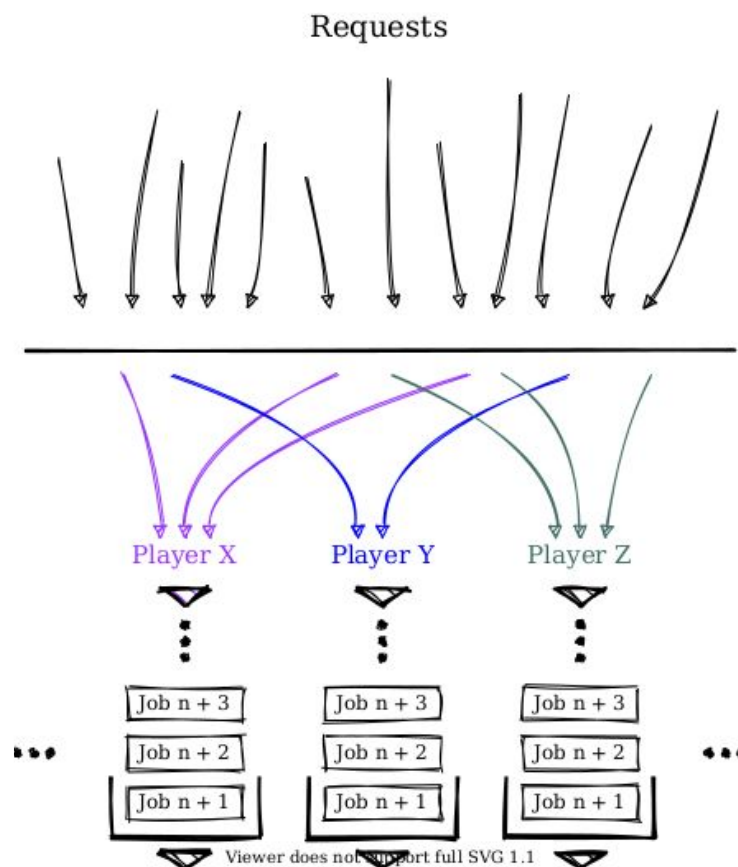


Figure 6. Synchronized Data Access and Modification

The requests that will end in jobs/tasks that need to be executed, have all different sources, but are synchronized on a per-player basis. They must not take too long, as any other parallel jobs might be blocked. For long-running jobs, they will be broken up into multiple smaller jobs, and the implementation of such a job processor will handle the possibility that between first and second part of the job, data might have changed.

In database transaction terminology this can be called a pessimistic locking mode.

This is an example for player data related jobs, but there are many others, that require also synchronization using this principle.

Atomic Data Modification

When data in different caches is being modified, it must be guaranteed that either all succeed or all fail, to keep the data consistent. A very simple example is a game transaction, where the game result itself is inserted, but also the player's wallet updated. It would result in inconsistent data to insert the game result, but not to update the player's wallet.

Data is modified when processing a job, that is already protected vs. concurrent data modification as shown in [Figure 6](#).

Nevertheless, processing such a job can successfully start modifying some data but can fail before it can successfully finish, and so leave an inconsistent data behind. In such a case the job fails to succeed and will get re-scheduled and repeated by the cluster using another microservice of the same type till it succeeds.

So, any job/task can be triggered multiple times, but will only be handled once. This system property is called *idempotence* such that no matter how many times an operation is executed, the same result is achieved. This is a core principle in all microservices.

In case a job keeps failing for a given period of time or a given number of retries, an alerting is implemented.

Data Sanity

Data sanity checks are fundamental part of the system. The system constantly runs sanity operations to ensure data consistency and discover data issues.

Data that has not passed these checks, will not be available for reporting or other export channels, e.g. export to game authorities or BI tools, like *Tableau*. Sanity issues can - in some cases - be reconciled automatically or can require manual interventions.

Data Reporting

After data passes the sanity checks it will be written to "flat" report tables optimized for report queries and that can be viewed/queried in the *BackOffice* application.

Data Export

After data is available for reporting it will be exported to all required services like gaming authorities, affiliate systems or BI tools.

Part of the export layer is also a so-called "accumulation" service, that groups the "flat" report tables into more useful reports, e.g. *Daily Player NGR* or *Monthly Game Plays*.