Distributed Applications Development

Computer Engineering	3rd Year	1st Semester	2025-26	Periodic Evaluation
Project	Limit date for results divulgation: 16 Jan 2026			
Date: 31 October 2025	**Delivery Date: 3 Jan 2026**			

Project – *Bisca*

1. OBJECTIVE

This project's objective is to implement a single-page application (SPA) for the *Bisca* Game platform, using Vue.js framework for the development of the web client, a Restful API server and a WebSocket server.

2. BISCA GAME

Bisca is a traditional Portuguese card game for two players, with the following set of rules.

1. Overview

Bisca is played with a 40-card deck (*baralho*) with the objective to win the most points by capturing valuable cards during tricks (*vazas* ou *jogadas*). The game includes a trump suit (*naipe de trunfo*) that outranks all others.

2. Deck and Setup

The game uses a 40-card deck formed by removing the eights, nines, and tens from a standard deck. The suits (*naipes*) are hearts (*copas*), diamonds (*ouros*), clubs (*paus*), and spades (*espadas*). Each player is dealt nine cards (their hand, or *mão*). In an alternative variant - 'Bisca de 3' - each player is dealt three cards; all other rules remain the same.

The last card of the deck is turned face up to reveal the trump suit. The remaining cards are placed face down on top of that card, leaving part of the trump visible beneath the deck throughout the game.

The non-dealer leads the first trick. During play, it is not permitted to look at cards that have already been played or that are in the piles of won tricks.

3. Card Values

The cards have the following point values: the Ace (*Ás*) is worth 11 points; the Seven (*Sete*, also called *Bisca* or *Manilha*) is worth 10; the King (*Rei*) is worth 4; the Jack (*Valete*) 3; and the Queen

(*Dama*) 2. All other cards (2 to 6) have no point value. There are 120 points in total in the deck, and a player must reach 61 or more points to win a game.

4. Gameplay

The game proceeds through a series of tricks, in which each player plays one card per trick.

- First trick: the non-dealer leads. The opponent may play any card – there is no obligation to follow suit (*assistir*) while cards remain in the stock (*monte*).
- Determining the winner of a trick: if both cards are of the same suit, the higher ranked one wins. If they are of different suits and one is of the trump suit, the trump wins. Otherwise, the first card played wins the trick.
- Drawing cards: After each trick, the winner draws the top card from the stock, followed by the loser. This continues until the stock is empty.
- Final phase: once there are no more cards to draw, players must follow suit (*assistir*) if they can. If unable to do so, they may play a trump or any other card. The winner of each trick collects the cards into their pile of won cards.
- Winning the game: when all cards have been played, both players total their points. The player with 61 or more points wins the game.

5. Match Scoring

Each game (*jogo*) contributes marks on a match (*partida*); the first player to reach four marks wins the match. A score between 61 and 90 points gives one mark (*risca* ou *moca*); 91 to 119 points gives two marks (*capote*); and 120 points counts as a clean match win (*bandeira*). A draw gives zero marks to each player.

3. BISCA GAME PLATFORM

The Bisca platform allows users to play the *Bisca* card game, either as a standalone game or full matches (set of games with four marks by the winner). Games and matches can be played in single-player mode against a bot, or in multiplayer mode against human opponents (limited to two players per game or match). The platform should support both “Bisca de 3” (three-card hands) and “Bisca de 9” (nine-card hands). The bot is intentionally simple: it must follow the game rules and, when playing second in a trick, attempt to win; if it cannot, it plays its lowest-value card.

Anonymous users can only play single-player games or matches. Registered users may play both single-player and multiplayer games and matches. In multiplayer games, if a player resigns or exceeds a 20-second move timer, the opponent is awarded all remaining cards (both hands and the

stock/draw pile), and the game ends. Final scores are then computed accordingly. If a player resigns any game within a match, that resignation applies to the entire match—the player forfeits all remaining games in the match.

Coins are used to enter multiplayer games and matches and—if implemented by the student team—to purchase extra platform or in-game items and features, with pricing set by the team. Newly registered users receive a welcome bonus of 10 coins. Coin purchases use a fixed conversion rate of €1 = 10 coins (purchases are made in €1 increments). Each multiplayer standalone game costs 2 coins; in the event of a draw, each player is refunded 1 coin. Winner payouts: 3 coins for a basic win (≥61 points), 4 for a *capote* (≥91), and 6 for a *bandeira* (120). A match (a set of games with four marks by the winner) requires a minimum stake of 3 coins per player. By mutual consent, players may raise the stake before play begins, up to 100 coins. The match winner receives the combined stake from both players, minus 1 coin retained by the platform as a commission.

The platform should log all transactions (welcome and other bonuses, coin purchases, coins spent and earned in games and matches, and any platform or in-game items and features) as well as all multiplayer games and matches played by registered users. Users should be able to access their complete transaction and multiplayer game history. The platform should also provide per-user statistics and global leaderboards visible to all users, including anonymous visitors.

Platform administration is handled by administrators, who can view all users (players and administrators), block or unblock players, create additional administrator accounts, and remove any account except their own. Player accounts may be removed by any administrator or by the player himself. If a player has made at least one transaction or played at least one multiplayer game or match, the account must be *soft-deleted*, preserving transaction and game records, which cannot be deleted. Administrators should also have read-only access to all transactions and games across the platform, including usage summaries and statistics.

4. DAD PROJECT

The DAD project aims to develop and deploy the Bisca Game Platform as a web application. The web client must be a single-page application (SPA) built with Vue.js (mandatory) and hosted on the designated internal school server. The system must also include a backend with, at minimum, a database, a RESTful API server and a WebSocket server. Students may choose the backend technologies, provided they are open-source or free external services and can be installed on—or accessed from—the designated server.

5. FEATURES

The web application must implement a set of groups of features described in this section. The implementation must align with the previously outlined business model, comply with the constraints (Section 6) and non-functional requirements (Section 7), and adhere to all applicable standards and best practices for the selected technologies and architectural patterns. Students are free to design the user interface and user experience, and may define reasonable business details that are not explicitly specified.

G1. USER REGISTRATION, AUTHENTICATION, PROFILE, END SESSION, ACCOUNT REMOVAL

Any user may register using a unique email address and nickname, their name, and a password (minimum 3 characters); a photo or avatar is optional. Newly registered users receive a welcome bonus of 10 coins credited to their account. Authentication uses the email address and password. After signing in, users can view and update their profile (email, nickname, name, photo/avatar, password), sign out, or delete their account.

Account deletion requires explicit confirmation (e.g., re-entering the current password or nickname, or entering a system-provided confirmation phrase) and permanently forfeits all coins associated with the account. Administrators have the same profile capabilities as other users; except they cannot delete their own accounts.

G2. COINS AND TRANSACTIONS

Registered users should be able to view their coin balance and purchase coins within the platform. Purchases should be handled through an external API that simulates payment transactions. Coin purchases use a fixed conversion rate of €1 = 10 coins (purchases are made in €1 increments). Users spend coins within the platform and may earn them through gameplay and bonuses.

All coin-related transactions—bonuses, purchases, spending, and earnings—should be logged and shown as a transaction history. Players can view only their own history. Administrators may view all players' histories but cannot create, modify, or delete transactions; access is strictly read-only.

G3. GAMES AND MATCHES

Users can play standalone games or full matches (set of games with four marks by the winner). Games and matches are limited to two players and may be played in single-player mode against a bot, or in multiplayer against human opponents. Anonymous users can only play single-player;

registered users may play both single-player and multiplayer. The platform must support both “Bisca de 3” (three-card hands) and “Bisca de 9” (nine-card hands) variants in all modes, and enforce the official rules defined in the "2 Bisca Game" section. The bot is intentionally simple: it must follow the game rules and, when playing second in a trick, attempt to win; if it cannot, it plays its lowest-value card.

Multiplayer games use real-time communication (WebSockets) with the server acting as the source of truth for dealing, turn order, valid moves, trick resolution, drawing from stock (when applicable), scoring, and end-of-game/match conditions. If a player resigns or exceeds a 20-second move timer, the opponent is awarded all remaining cards (both hands and the stock/draw pile), the game ends, and scores are computed accordingly. If resignation occurs within a match, the player forfeits all remaining games in that match. Entry fees, stakes, payouts, and any platform commission must be applied automatically according to the business rules defined earlier. The system must persist game and match results and relevant metadata for later display in user histories and administrative read-only views.

G4. GAME HISTORY AND LEADERBOARDS

The platform should track multiplayer games and matches (single-player games and matches are not tracked), storing minimal metadata such as start/end timestamps, variant (“Bisca de 3” or “Bisca de 9”), outcome (win/loss/draw/forfeit), final points and marks (e.g., *capote*, *bandeira*), and basic duration. Each player can view only their own multiplayer history; administrators can view all players’ histories. For any multiplayer game or match, detailed records should be visible to the administrator and participating players only.

The application should provide multiplayer-only personal and global leaderboards. Personal leaderboards (visible only to the owner) should at least summarize totals for game wins, match wins, *capotes*, and *bandeiras*, optionally segmented by variant. Global leaderboards (visible to all users, including anonymous visitors) should highlight the top players (nicknames) by game and match wins, and may also include *capotes* and *bandeiras*. In the event of ties, the earlier achiever ranks higher.

G5. ADMINISTRATION

Platform administrators will be responsible for monitoring and maintaining the bisca game platform. Administrators cannot play games or hold coins. They have profiles like regular users but cannot delete their own accounts, and new administrator accounts can only be created by an

existing administrator (not via public registration). Administrators can view all users (players and administrators), block or unblock players, create additional administrator accounts, and remove any account except their own; when removing a player with prior activity, the account must be *soft-deleted* per platform rules. Administrators also have read-only access to all transactions and to all multiplayer games and matches, plus platform usage summaries and statistics.

G6. STATISTICS

The platform should provide statistics in both visual (charts) and textual/tabular forms. Anonymous users and players see only generic, anonymized aggregates (e.g., total registered players, games and matches played, recent activity, etc.). Administrators have full, non-anonymized access, including breakdowns and time-series (e.g., purchases by period, purchases by player, game/match volumes). The exact set of metrics and visualizations is up to the student team, provided privacy for non-admin viewers is preserved.

G7. EXPLORATION AND IMPLEMENTATION

To distinguish between projects, students should either add custom features not specified in this statement, deliver exceptionally high-quality implementations of the features explicitly specified in this statement, or include implementation details that meet one or more of the following criteria: use technologies or techniques not covered in classes; optimize performance; improve architecture; or enhance implementation structure or quality. Priority is given to distributed capabilities that improve architecture, scalability, or resilience. Credit is proportional to demonstrated impact and evidence—empirical where feasible (e.g., benchmarks, load tests, or failure-injection tests); when such testing is impractical on the provided infrastructure, rigorous documentation and discussion may be used instead.

Examples (non-exhaustive) of custom features include: supporting three or more players per game (as an optional extension while preserving the required two-player mode); adding a spectator mode for multiplayer games and matches; enabling full playback of past games; adding custom card sets; extending bot capabilities and integrate them into multiplayer games.

Examples (non-exhaustive) of implementation details include: adding a queue server or worker for asynchronous, long-running operations; horizontally scaling the WebSocket server; implementing session recovery after WebSocket (or other types of nodes) failure; and, where appropriate (avoid adding components without a clear, justified purpose), incorporating a cache server, a NoSQL

database, a search engine, an asynchronous messaging broker (e.g., Kafka or RabbitMQ), or other types of distributed components.

Evaluation is cumulative and based on the teacher's qualitative judgment. Each feature or implementation detail will be assessed for quality, complexity, and results, and weighted accordingly. Earning full credit may require several additions or a single exceptional contribution. Technologies or features applied in the context of other disciplines (e.g., TAES) are permitted but will receive reduced weight relative to novel, project-specific work.

6. CONSTRAINTS

The project's **mandatory constraints** are as follows::

C1. The application must be implemented exclusively as a Single-Page Application (SPA).

C2. The client application must be built with Vue.js.

C3. All external coin purchases must be processed via the provided RESTful API, the "Payment Gateway Service."

C4. The application must be deployed on the designated internal school server and be accessible via a desktop web browser

C5. Backend technologies for the database, RESTful API, and WebSocket server must be open-source or available as free external services

C6. Backend choices are constrained by the designated server's capabilities: technologies must be installable on, or accessible from, that server. If the server cannot support a chosen technology, students must select a compliant alternative.

C7. Backend technologies are constrained by the technical characteristics of the designated server as they must be installed on or accessible from the designated server. If the server does not support a specific technology, students must replace it with a valid alternative.

C8. The deployed application's database must be seeded with realistic data—in both scale and content—to simulate typical usage patterns.

7. NON-FUNCTIONAL REQUIREMENTS

The project's non-functional requirements are as follows:

NF1. The client application's code and structure must follow Vue.js principles and best practices.

NF2. The REST API server must follow RESTful service principles and best practices.

NF3. All implemented features must be correctly integrated and behave consistently throughout the application.

NF4. Visual design and layout must be consistent across the application and appropriate to its purpose and usage context.

NF5. The application should provide good usability: information and operations must be clear, interactions straightforward, and common patterns used to minimize user effort

NF6. User input must be validated on both client and server according to rules derived from the business model and feature groups

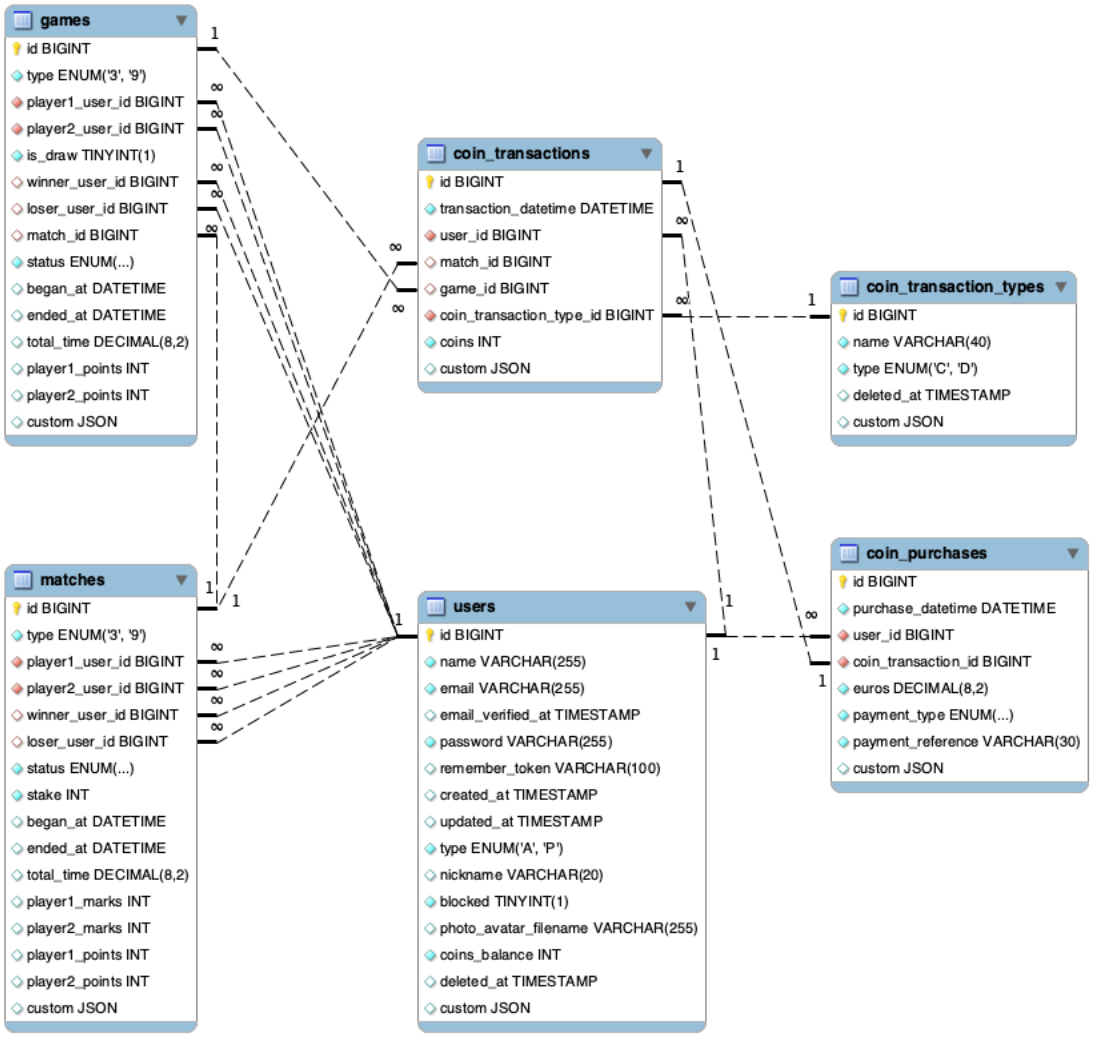
NF7. Security must be prioritized: protect user data and privacy, and enforce authentication and authorization using established best practices

NF8. Performance, reliability, and distributed characteristics must be considered: optimize latency/throughput and resource use, avoid unnecessary network traffic, handle data efficiently, and design for scalability and fault tolerance appropriate to the provided infrastructure

NF9. Optimize WebSocket full-duplex communication for performance and security: minimize message sizes and target only relevant clients (avoid unnecessary broadcasts and data exposure).

8. DATABASE

As noted earlier, the project will include a database technology chosen by students, provided it is an open-source database system or a free external service. A fully functional reference database will also be provided for students to use as-is or adapt as needed. The schema is defined via Laravel migrations and populated with sample data using a provided Laravel seeder. The schema is as follows:



*Note: All users have the same password: "**123**"*

TABLES AND COLUMNS

In this section we will describe all tables and the most relevant columns of the database.

USERS

Table with all users (players and administrators).

- **name** - User's name.
- **email** - User's email (credentials = email + password)
- **password** – Hashed value for the user's password.
- **type** – User type ('A' for administrator; 'P' for player),
- **nickname** - User's nickname – used multiplayer games/matches and scoreboards.
- **blocked** – Boolean indicating whether the user is blocked.

- **photo_avatar_filename** – Filename of the user's photo or avatar.
- **coins_balance** – Current balance of " coins" for the player.
- **deleted_at** – Soft-delete timestamp (record is logically removed but retained for history/integrity).
- **custom** – JSON field for any additional information students choose to store.

GAMES

Table of multiplayer games played by registered users (standalone or within matches):

- **type** – Type of game ('3' for 'Bisca de 3'; '9' for 'Bisca de 9').
- **player1_user_id** – User ID of the player 1 – typically the user that created the game.
- **player2_user_id** – User ID of the player 2 – typically the user that joins the game.
- **is_draw** – Whether the game ended in a draw (default: false)
- **winner_user_id** – User ID of the winner.
- **loser_user_id** – User ID of the loser.
- **match_id** – Match ID if the game is part of a match; null for standalone games.
- **status** – Game status: 'Pending', 'Playing', 'Ended', or 'Interrupted'. Note: depending on the project's implementation, some statuses may never be used.
    - Pending: waiting for players
    - Playing: in progress
    - Ended: finished
    - Interrupted: stopped due to technical issues (unfinished; no winner)
- **began_at** – Date and time when the game started.
- **ended_at** – Date and time when the game ended.
- **total_time** – Total duration of the game (in seconds).
- **player1_points** – Total points for player 1 (player1_points + player2_points == 120).
- **player2_points** – Total points for player 2 (player1_points + player2_points == 120).
- **custom –** JSON field for any additional information students choose to store.

MATCHES

Table of multiplayer matches played by registered users. Each match includes a set of games until one of the players reaches 4 marks.

- **type** – Type of games ('3' for ’Bisca de 3’; '9' for ’Bisca de 9’) used in the match.
- **player1_user_id** – User ID of the player 1 – typically the user that created the match.
- **player2_user_id** – User ID of the player 2 – typically the user that joins the match.
- **winner_user_id** – User ID of the winner.
- **loser_user_id** – User ID of the loser.
- **status** – Match status: 'Pending', 'Playing', 'Ended', or 'Interrupted'. Note: depending on the project’s implementation, some statuses may never be used.
    - Pending: waiting for players
    - Playing: in progress
    - Ended: finished
    - Interrupted: stopped due to technical issues (unfinished; no winner)
- **stake** – Amount each player wagers for the match (mutually agreed).
- **began_at** – Date and time when the match started.
- **ended_at** – Date and time when the match ended.
- **total_time** – Total duration of the match (in seconds).
- **player1_marks** – Total marks for player 1.
- **player2_marks** – Total marks for player 2.
- **player1_points** – Cumulative points scored by player 1 across all games in the match.
- **player2_points** – Cumulative points scored by player 2 across all games in the match.
- **custom –** JSON field for any additional information students choose to store.

COIN_TRANSACTION_TYPES

Table with the types of coin transactions.

- **name** - Human-readable coin transaction type, e.g., "Bonus", "Coin purchase", "Game fee", "Match stake", "Game payout", "Match payout".
- **type** - 'C' = Credit (increases coins), 'D' = Debit (decreases coins).
- **deleted_at** - Soft-delete timestamp (record is logically removed but retained for history/integrity).
- **custom –** JSON field for any additional information students choose to store.

COIN_TRANSACTIONS

Table with all coin transactions (movements)

- **transaction_datetime** - date and time when the transaction occurred.
- **user_id** - User ID of the owner of the transaction.
- **match_id** - Match associated to the transaction (if applicable).
- **game_id** - Game associated to the transaction (if applicable).
- **coin_transaction_type_id** - Identifies the transaction type
- **coins** – total coins of the transaction. Positive value = credit (increases balance); negative = debit (decreases balance).
- **custom –** JSON field for any additional information students choose to store.

COIN_PURCHASES

Table with the details of external coin purchases

- **purchase_datetime** - date and time when the purchase occurred.
- **user_id** - User ID of the purchaser (the same as the associated coin transaction).
- **coin_transaction_id** – The coin transaction associated to the purchase.
- **euros** – Amount charged in euros (real money).
- **payment_type** – Type of payment for the purchase: MBWAY, IBAN (bank transfer), MB (Multibanco payment) and VISA.
- **payment_reference** – Reference for the payment.
    - MBWAY: 9 digits starting with 9 (e.g., "915785345")
    - PAYPAL: A valid email (e.g., "john.doe@gmail.com")
    - IBAN: 2 letters followed by 23 digits (e.g., "PT50123456781234567812349")
    - MB: 5 digits (entity number), a hyphen ("-"), and 9 digits (e.g., "45634 123456789").
    - VISA: 16 digits starting with 4 (e.g., "4321567812345678").
- **custom –** JSON field for any additional information students choose to store.

9. PAYMENT GATEWAY SERVICE

The platform uses an external payment gateway to handle coin purchases, available at: **`https://dad-payments-api.vercel.app`**

This is a simulated ("fake") service that mimics external financial transactions—no real payments occur. The service exposes a single endpoint:

`post`	`<service URI>/api/debit`	Creates a new debit on the external entity

When a user purchases coins, the platform must create a debit with the external Payment Gateway Service. The endpoint accepts a JSON payload in the following format:

```
{
     "type": "MB",
     "reference": "45634-123456789",
     "value": 3
 }
```

On success, the service returns the status code `201 Created`. If the request is invalid (e.g., bad payload, unrecognized reference, value over the limit), it returns the status code `422 Unprocessable Entity` with error details. Other HTTP error codes may occur.

VALIDATION

The properties "type", "reference", and "value" are mandatory and validated by the Payment Gateway Service; they must also be validated by the game platform.

- **type**: Accepts one of these values: "MBWAY", "PAYPAL", "IBAN", "MB", and "VISA".
- **reference**: Validation rules depend on the type:
    - MBWAY: 9 digits starting with 9 (e.g., "915785345")
    - PAYPAL: A valid email (e.g., "john.doe@gmail.com")
    - IBAN: 2 letters followed by 23 digits (e.g., "PT50123456781234567812349")
    - MB: 5 digits (entity number), a hyphen ("-"), and 9 digits (e.g., "45634-123456789").
    - VISA: 16 digits starting with 4 (e.g., "4321567812345678").

- **value**: Accepts any positive integer from 1 to 99 – (e.g., 3 is valid; 3.01 is invalid).

SIMULATION RULES

To simulate successful and failed debits, the fake service applies test-only rules that the platform cannot pre-validate by design. Their sole purpose is to trigger error responses and validate the platform's error handling.

The following reference patterns are treated as invalid by the service, simulating non-existent MBWAY, PayPal, IBAN, MB, or Visa accounts:

- MBWAY: phone numbers starting with "90" (e.g., 901645932)
- PAYPAL: emails starting with "xx" (e.g., xx.john.doe@gmail.com)
- IBAN: references starting with "XX" (e.g., XX5012345678123456781234 9)
- MB: entity numbers starting with 9 (e.g., 95634-123456789)
- VISA: references starting with "40" (e.g., 4021567812345678)

To simulate insufficient funds, the service applies per-method limits; debits above these limits fail:

- MBWAY: €5
- PAYPAL: €10
- IBAN: €50
- MB: €20
- VISA: €30

10. DEPLOYMENT AND DELIVERABLES

The application must be deployed on the designated internal school server and be accessible on the ESTG intranet via a desktop web browser (testing will be conducted in Google Chrome). Functional evaluation will occur only on the deployed application, so **deployment is mandatory**. The deployed environment must mirror the intended production configuration as closely as possible, including performance optimizations and realistic, production-like data.

Project delivery must include three files ("project," "report," and "report complement") per group, plus one individual report per student. For example, a four-student group must submit seven files in total (three group files and four individual reports). Exactly one student should upload the three group files (and their own individual report); the remaining students upload only their individual reports.

Files to be submitted:

- **prj_GG.zip** — A zip archive containing all source code and project assets for all platform services/projects, excluding the vendor, node_modules, and .git folders (note: .git may be hidden)
- **grp_report_GG.xlsx** — The group report with group identification and high-level implementation information. A template will be provided
- **grp_report_complement_GG.zip** (optional) — A ZIP archive with documents that complement the report, including at least one image of the database diagram used in the project (if different from the provided one) and any additional artifacts that clarify the internal structure and implementation details.
- **individual_report_NNNNNNN.xlsx** — The individual report including self-assessment and peer evaluation. A template will be provided

Note: Replace GG with your group number and NNNNNNN with your student number.

11. EVALUATION

The evaluation of the project will consider the compliance with the mandatory constraints (C1… C8), Groups of features (G1 … G7) and Non-Functional Requirements (NF1 … NF9).

Mandatory Constraints (C1…C8)

All constraints are mandatory (including deployment). Non-compliance → grade 0.

Groups of features (G1…G7)

Scored on correctness/completeness against the business model, integration and reliability, usability, and code quality/structure. Evidence from the deployed application prevails.

Non-Functional Requirements (NFR1…NFR10)

Assessed as a dedicated 10% criterion and also factored into all feature-group criteria. Grading is holistic (no per-NF scores), based on code/structure review and functional testing.

Weight of all evaluation criteria:

	Groups of features	**Weight**
G1	User registration, authentication, profile, end session, account removal	**10%**
G2	Coins and transactions	**10%**
G3	Games and matches	**30%**
G4	Game history and leaderboards	**10%**
G5	Administration	**10%**
G6	Statistics	**10%**
G7	Exploration and implementation	**10%**
NFR	All Non-Functional Requirements	**10%**