

# BackendGen.dev

## User Manual

*Generate your own backend for Unity, Godot and Unreal games, apps and small businesses*

Generator version 1.0.1 · November 2026
https://backendgen.dev · hello@backendgen.dev

# Contents


1. Welcome ..... 5
   - What you need ..... 5
   - How this manual is organised ..... 5
2. Your Account ..... 6
   - Creating an account ..... 6
   - Signing in and out ..... 6
   - Your details and password ..... 6
   - Locked out of your email? ..... 6
   - Deleting your account ..... 7
3. Licences and Buying ..... 7
   - Buying ..... 7
   - Refunds ..... 8
   - Bought Universal Backend Generator on the Unity Asset Store? ..... 8
4. The Game Generator ..... 8
   - The Your download panel ..... 9
   - Basics tab ..... 9
   - Players & Security tab ..... 10
   - Ranking (SR) tab ..... 10
   - Add-on tab (Advanced licence) ..... 11
   - Advanced tab ..... 11
5. Running Your Game Backend ..... 11
   - What is in the zip ..... 11
   - Start it on your computer ..... 12
   - Changing settings later ..... 12
   - Check it: self-test and load test ..... 12
6. Unity, Godot, Unreal and Other Clients ..... 13
   - Connecting your Unity game ..... 13
   - Platforms ..... 14
   - Connecting your Godot game ..... 14
   - Connecting your Unreal game ..... 14
   - Web games, apps and other engines ..... 15
7. Player Sign-In and Access Rules ..... 16
   - The safety net ..... 16
   - Switching a live game from report to enforce ..... 16
   - Player data rights ..... 17
8. Skill Rating (SR) ..... 17
   - How a match is scored ..... 17
   - Uploading results ..... 17
   - Staff tools ..... 17

9. Advanced Features..... 18
Database and scaling..... 18
Deployment and hosting..... 18
Admin dashboard..... 18
Live services..... 18
Community safety and sign-in..... 19
Game services..... 19
Multiplayer services..... 20
Operations..... 20
10. Putting a Game Backend Online..... 20
Backups..... 20
Security checklist..... 21
11. Omni Cluster: Running Your Game Servers..... 21
Step 1: Download your backend with game servers switched on..... 21
Step 2: Start your backend..... 21
Step 3: Make a server key..... 22
Step 4: Download and open Omni Cluster..... 22
Step 5: Import your setup file..... 22
Step 6: Fill in CLUSTER SETTINGS..... 22
Step 7: Launch..... 23
Check it from the dashboard..... 23
Everyday use on Windows..... 24
If something goes wrong on Windows..... 24
On a Linux server..... 25
How your game finds the servers..... 28
Voice chat between players (optional)..... 28
Three rules..... 30
12. The Business Generator..... 31
The tabs..... 31
Starting points..... 32
Run it and sign in..... 32
The shop (Advanced)..... 32
Bookings (Advanced)..... 33
Emails..... 33
Your website..... 33
Going live..... 33
13. Saved Projects and Updates..... 34
Saving your settings..... 34
Updating a live server..... 34
14. Troubleshooting..... 35

15. Getting Help..... 36

# 1. Welcome

BackendGen.dev makes the server side of your project for you. You choose your settings in the browser, click **Download**, and get the complete source code of a working backend: a Node.js server you run on your own computer, VPS or cloud host. There are no monthly platform fees and no per-player charges, and the code is yours to keep, change and ship.

There are two generators:

- **Game generator**: backends for multiplayer games, apps and online communities. Player accounts and sign-in, per-player data security, match results and skill rating (SR), backups and a self-test, plus generated clients for Unity (C#), Godot 4 and Unreal Engine 5 (a plugin with Blueprint nodes), and a JavaScript client. With the Advanced licence you can add PostgreSQL and several servers, an admin dashboard, live updates, realtime rooms, push notifications, moderation, platform sign-in, an economy with coupons and probability tables, leaderboards, game-server fleets, parties, guilds and matchmaking, and you get **Omni Cluster**, which runs your dedicated game servers on Windows or Linux.
- **Business generator**: backends for shops, salons, clinics, trades and offices. A staff portal your team signs in to (owner, manager and staff roles) and the record types you choose. With the Advanced licence you can add an online shop (card payments through Stripe and/or PayPal) and online bookings.

BackendGen.dev is the web version of **Universal Backend Generator**, the Unity Editor tool on the Unity Asset Store. Both make the same game backends from the same templates; the Godot and Unreal clients come with BackendGen.dev downloads.

## What you need

| To… | You need |
|---|---|
| Use the generators | A BackendGen.dev account with a licence and a modern web browser. |
| Run a backend | Node.js 22 or newer (the LTS version) on the computer or server that runs it. Windows, macOS and Linux all work. |
| Connect a Unity game | Unity 2022 LTS or Unity 6. The client works in the Editor and in Windows, macOS, Linux, Android, iOS and WebGL builds (Mono and IL2CPP). |
| Connect a Godot game | Godot 4.2 or newer (GDScript). The client works in every export: Windows, macOS, Linux, Android, iOS and the web. |
| Connect an Unreal game | Unreal Engine 5 (built and tested on 5.8) and Visual Studio with the **Game development with C++** workload (Visual Studio 2026 for Unreal 5.8). Blueprint-only projects work too. |
| Put it online | A server that stays on (a small VPS is plenty to start) and, for real players, a domain name and HTTPS. |
| Run Omni Cluster (Advanced) | A 64-bit Windows 10, 11 or Windows Server machine, or a Linux server with systemd (x64 or ARM64). Nothing else to install. |

## How this manual is organised

| Chapter | Covers |
|---|---|
| 2. Your account | Signing up, signing in, your details, passwords, deleting your account. |
| 3. Licences and buying | Base, Advanced and the Add-on upgrade, discount codes, refunds, Asset Store customers. |

| Chapter | Covers |
|---|---|
| 4 to 9 | The game generator, running a game backend, Unity, Godot, Unreal and other clients, sign-in and access rules, skill rating, and the Advanced features. |
| 10. Putting a game backend online | Servers, HTTPS, hosting templates, backups and security. |
| 11. Omni Cluster | The program that runs your dedicated game servers on Windows or Linux (Advanced), step by step. |
| 12. The business generator | Staff portal, records, shop, bookings, emails, your website and going live. |
| 13. Saved projects and updates | Saving settings and updating a live server safely. |
| 14 and 15 | Troubleshooting and getting help. |

> **Every download has its own guides**
> Each backend you download contains `START-HERE.txt` and a `README-SETUP.txt` written for exactly the settings you chose, and every Advanced feature has its own guide (a .txt file in the backend folder). This manual explains the whole picture; those guides go into the details of your backend.

# 2. Your Account

## Creating an account

1. Go to backendgen.dev/account and choose **Create account**.
2. Enter your name, your company or studio (optional), your email address and a password of at least 8 characters.
3. Open the email we send you and click the link to confirm your address. Your account is created when you click it.

Where the sign-in page shows **Sign in with Google**, you can use your Google account instead of a password.

## Signing in and out

Sign in on the account page. The menu at the top of every page then shows the **Game generator**, the **Business generator**, **My projects**, **Help** and **Your account**. **Sign out** is in the same menu.

## Your details and password

- **Your details** (on your account page): change your name and company.
- **Change your password**: under Your details. You need your current password.
- **Forgotten your password?** On the sign-in page click **Forgot your password?** and we email you a link to set a new one.
- **Signed up with Google?** To use a password as well, sign out and use **Forgot your password?** on the sign-in page.

## Locked out of your email?

If you can no longer get into the email address your account uses, email hello@backendgen.dev from your new address with your name, the old address, and your Paddle order number (it is on your receipt) or the

Unity Asset Store invoice number you claimed a licence with. Once we have checked it is you, we move your account and licence to the new address and send you a link to set a new password.

### Deleting your account

At the bottom of your account page, under **Delete your account**, type the confirmation and click the button. This deletes your sign-in, your saved projects and any Asset Store claim straight away and can't be undone. Backends you have already downloaded are yours and keep working. A record of your licence and purchases is kept because UK accounting rules require it; see the privacy policy.

## 3. Licences and Buying

A licence is a **one-time payment**, not a subscription. It covers unlimited projects, and every update in the current major version (1.x) is free. When a new major version comes out, licence holders are offered a discounted upgrade. Backends you have downloaded keep working whatever happens to your licence.

| Licence | What it includes |
|---|---|
| **Base** | The game generator: single-server game backends with SQLite, player sign-in and access rules, match results and SR ranking, backups, self-test and load test, the Unity, Godot, Unreal and JavaScript clients and the OpenAPI description. The business generator: staff portal with roles and your own record types. |
| **Advanced** | Everything in Base, plus the Advanced Add-on: PostgreSQL and several servers, deployment files and hosting templates, admin dashboard, live updates, realtime WebSocket, push notifications, moderation, platform sign-in, game services (remote config, cloud saves, leaderboards, economy, coupons, probability tables, mail, trades, auction house, analytics), multiplayer services (game-server fleet, parties, guilds, matchmaking). In the business generator: the online shop and online bookings. Also includes **Omni Cluster**, the program that runs your dedicated game servers on Windows or Linux (chapter 11). |
| **Advanced Add-on** | An upgrade for Base owners: turns a Base licence into Advanced. |

Current prices are on the pricing section of the website. UK and EU prices include VAT; elsewhere any sales tax is added at checkout, and the checkout always shows the final price before you pay.

### Buying

1. Sign in and open your account page.
2. Under **Buy a licence** (or **Upgrade to Advanced** if you already have Base), click the licence you want.
3. Pay in the checkout. Payments are handled by **Paddle**, our reseller and Merchant of Record, so the receipt and invoice come from Paddle.
4. Your licence appears on your account page a few seconds after the payment, and the download buttons unlock.

#### Try before you buy

With a free account you can use both generators fully: fill in every setting, see the checks and the exact file list, Add-on included. Only downloading needs a licence. Pick **Base** or **Advanced** in the **Your download** panel to see what each one gives you; the button then says **Buy Base to download** or **Buy Advanced to download**, and your settings are kept while you buy. Already own Base? Pick **Advanced** to try every Add-on feature; downloading with it needs the Add-on upgrade, and Base downloads work as always.

#### Discount codes

Got a code? On your account page click **Have a discount code?** under the licences, enter it, then click **Buy**. The checkout applies it and shows the discounted price before you pay. If a code isn't valid for that licence,

the checkout opens at the normal price and the page tells you the code wasn't accepted, so you can close the checkout and check it.

### Refunds

- **14 days, before your first download**: you can get a full refund within 14 days of buying if you haven't downloaded a backend since the purchase. For an Add-on upgrade this counts from the upgrade.
- **If it doesn't work as described** and we can't fix it within a reasonable time, you get a refund even after downloading.
- To ask: email hello@backendgen.dev from your account's address with your Paddle order number, or use the link in your Paddle receipt.

A refunded purchase is removed from your licence. The full wording is in the refund policy.

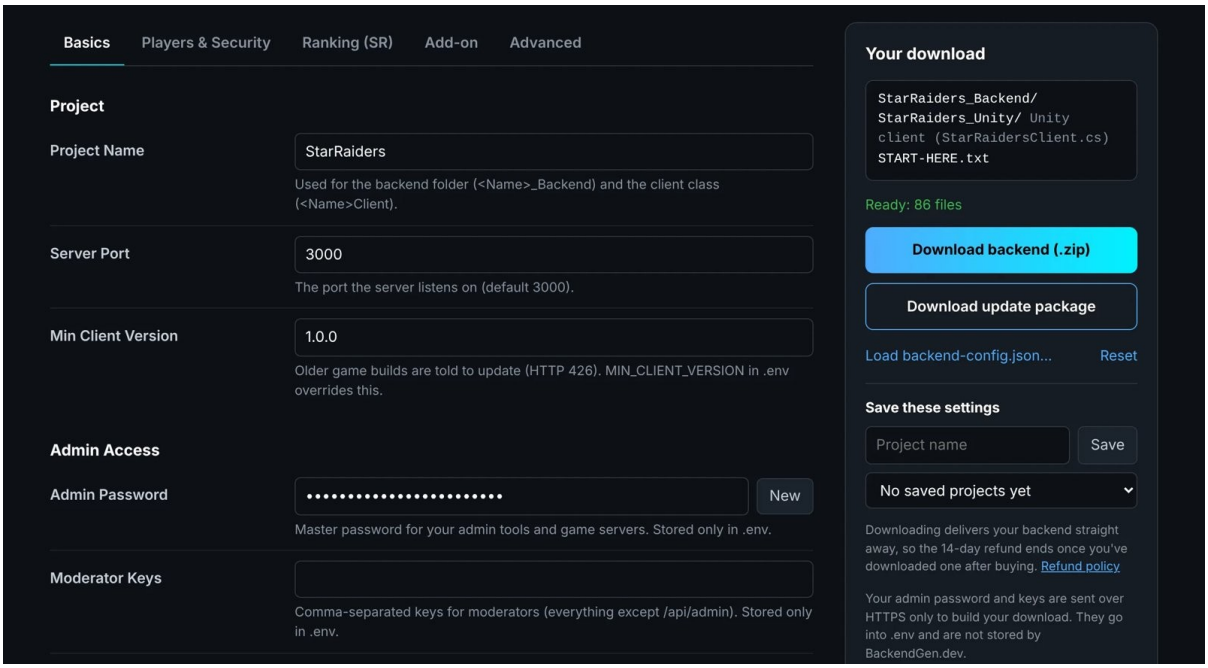
### Bought Universal Backend Generator on the Unity Asset Store?

Asset Store customers get a free web licence (and the Add-on too, if you bought it there).

1. Create your account (or sign in).
2. On your account page, under **Bought it on the Unity Asset Store?**, enter your invoice number, and the Add-on's if you have one. You will find them in Unity's order confirmation email or your order history on the Asset Store.
3. We check every invoice by hand. Once it is approved, the licence appears on your account and we email you.

## 4. The Game Generator

Open it from the menu (**Game generator**). Settings are on tabs on the left; the **Your download** panel on the right shows what your download will contain, checks your settings as you type, and has the download buttons.




*The game generator: settings on the left, Your download on the right.*

## The Your download panel

| Part | What it does |
|---|---|
| File list | The folders your zip will contain: `<Name>_Backend` (the server), `<Name>_Unity` (the Unity client files), `<Name>_Godot` (the Godot 4 client), `<Name>_Unreal` (the Unreal Engine 5 plugin), `START-HERE.txt` and, with Advanced and game servers on, `OmniCluster` (chapter 11). |
| Edition (while trying) | Without a licence, or with Base only: choose **Base** or **Advanced** to see what each one gives you. The download button says what you need to buy (or upgrade) to download that edition. |
| Status | **Ready: N files** when everything is fine. Otherwise it lists the problems to fix first (red) and notes you can read but still download with (yellow). |
| Download backend (.zip) | Builds and downloads the complete backend. |
| Download update package | Builds an update package for a backend that is already running on a server (chapter 13). |
| Load backend-config.json… | Loads the settings of a backend you downloaded earlier (pick its `backend-config.json`, and its `.env` too if you want the keys back). |
| Reset | Back to the default settings (click twice). |
| Save these settings | Saves the settings under a project name, so you can come back later (chapter 13). Passwords and keys are never saved. |

> **Passwords and keys**
> Your admin password, moderator keys, database address and other secrets are sent over HTTPS only to build your download. They go into the backend's `.env` file and are **never stored by BackendGen.dev**. Keep that `.env` file safe: it is the only copy.

## Basics tab

| Setting | What it means |
|---|---|
| Project Name | Names the backend folder (`<Name>_Backend`) and the client class (`<Name>Client`). |
| Server Port | The port the server listens on (default 3000). |
| Min Client Version | Older game builds are told to update (HTTP 426 "Upgrade Required"). `MIN_CLIENT_VERSION` in `.env` overrides it. |
| Admin Password | Master password for your admin tools and game servers. **New** makes a strong random one. Stored only in `.env`. |
| Moderator Keys | Comma-separated keys for moderators (everything except `/api/admin`). Stored only in `.env`. |
| Player Accounts (Identity) | Gamertags linked to a device or platform id, with rename and move-to-new-device support. Choose which identifier your game passes to sign-in (hardware ID, Steam ID64, Epic Account ID, PlayFab ID or custom). |
| Match Results (Harvester) | Accepts finished matches from your game or game servers (`POST /api/matches/upload`). Match Format: native Unity SDK, generic JSON webhook or UE4SS legacy logs. |

| Setting | What it means |
|---|---|
| Skill Rating (SR) | Ranks players per game mode from their match results (set it up on the Ranking tab). |

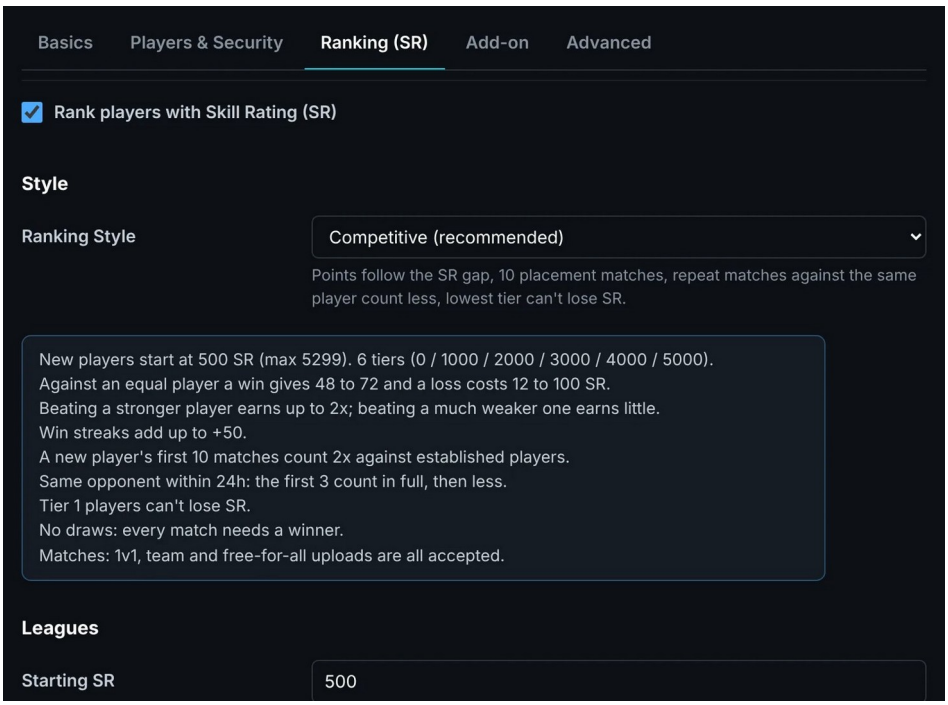
The **Capacity** box tells you what the chosen setup is good for: one SQLite server comfortably serves indie and community games (hundreds to a few thousand players online at once); with Advanced you can switch to PostgreSQL and several servers.

## Players & Security tab

| Setting | What it means |
|---|---|
| Player Access Rules | Whether access-rules.json is off, only reports what it would refuse, or enforces it (chapter 7). |
| Protect Private Data | Keep this on. Off is for development only: every non-protected path becomes publicly readable and writable. |
| Admin API Endpoints | Staff endpoints under `/api/admin` (search, backups, recalculation, raw queries…). |
| Starter Rules | The rule a new backend gets for every public write route that access-rules.json doesn't cover yet, e.g. "only that player may change their own area". You can choose a style per route. |
| Player Data | Which paths belong to a player (exported with "download my data", deleted with "delete my account") and where their content lives elsewhere (anonymised on deletion). |
| Rename settings | Conversation lists, conversation keys and references (friend lists, requests) that follow a player when they change their gamertag. |
| Online Status | Presence timeout, extra presence lists (e.g. a companion app) and the UDP keep-alive ping for game servers. |

## Ranking (SR) tab

Turn on **Rank players with Skill Rating (SR)** to give every player an SR per game mode, worked out by the server from match results. Choose a **Style** (Competitive, Community, Hardcore or Classic), set your **Leagues** (starting SR, tiers, win and loss points), and your **Game Modes** (which modes allow draws, whether the win margin is judged by points or rounds). Everything else, such as gap scaling, streak bonuses, placement matches, repeat-opponent protection and the hidden matchmaking rating, is under the fine-tune settings. Chapter 8 explains how a match is scored.




*The Ranking (SR) tab.*

### Add-on tab (Advanced licence)

Switch **Use the Advanced Add-on for this backend** on to choose Advanced features. Every feature is one plugin with its own settings file and its own guide in the backend folder. The groups are **Core**, **Game Services**, **Multiplayer services** and **Database & Scaling**. Chapter 9 describes each one.

### Advanced tab

| Setting | What it means |
|---|---|
| Storage & Backups | Database file name, backup every N hours (0 = off) and how many backups to keep. |
| Public Routes | Paths anyone may read, paths anyone may write (the access rules then decide per player), paths that are always private, staff-only paths, and the largest public write. |
| Search | Fields that are indexed for the admin search. |
| Scheduled Jobs | Load `jobs/leaderboardCron.js` from the backend folder for your own scheduled jobs. |
| Discord (optional) | A webhook URL and message ID for your own announcements (stored only in `.env`). |

## 5. Running Your Game Backend

### What is in the zip

| File or folder | What it is |
|---|---|
| `START-HERE.txt` | The first steps, written for your project. |

| File or folder | What it is |
|---|---|
| <Name>_Backend/ | Your server: server.js, plugins, backend-config.json (settings), .env (secrets), README-SETUP.txt, the self-test, the load test, the JavaScript client in clients/, and the guides for any Advanced features. |
| <Name>_Unity/ | The Unity client files. Copy its Assets folder into your Unity project (it merges into Assets/Scripts/Network). |
| <Name>_Godot/ | The Godot 4 client: a plugin in addons/ with an example scene, and a README.txt (chapter 6). |
| <Name>_Unreal/ | The Unreal Engine 5 plugin in Plugins/<Name>Backend, and a README.txt (chapter 6). |
| OmniCluster/ | Advanced, with the Add-on's game servers switched on: the settings file for Omni Cluster and a README.txt (chapter 11). |

## Start it on your computer

1. Install Node.js 22 or newer from nodejs.org (the LTS version).
2. Unzip the download and open the <Name>_Backend folder.
3. Double-click start-local.cmd (Windows) or start-local.command (macOS). On Linux, open a terminal in the folder and run npm install, then node server.js.
4. You should see "<Name> backend is LIVE on port 3000" (or your port). Check it in a browser: http://127.0.0.1:3000/healthz.

start-local is for trying it out: the server stops when you close that window. For a server that stays online, run it with PM2 (chapter 10).

> **Keep .env secret**
> .env holds your admin password and keys. Never share it, commit it to git or put it in a game build. BackendGen.dev does not keep a copy.

## Changing settings later

Most settings live in backend-config.json in the backend folder: edit it and restart the server, no new download needed. To put a new version on a live server, use **Download update package** (chapter 13).

## Check it: self-test and load test

node verify-backend.js [server address] runs about 35 checks against a running server in a few seconds: admin password strength, anonymous access and private data, path tricks and outdated versions, that players can sign in, change their own data and not anyone else's, public routes and backups. It cleans up its own test players. Any WARN line tells you what to fix.

load-test.js measures how many players your server can carry. Run it from another machine for real numbers:

```
node load-test.js http://your.server:3000 --players 500 --rate 2
node load-test.js --players 200 --rate 0      (no pause: the maximum)
```

Test players save data, read other players, send presence and use the leaderboards. The report shows requests per second and how fast 50 / 95 / 99 % were answered, and everything is removed afterwards.

**Measured performance**

Our load test (3 to 5 November 2026, Advanced features on, "comfortably" = 98% answered, 95% within 100 ms, under 0.1% errors): on SQLite, one server on a desktop Ryzen 5 comfortably answered 2,200 requests a second (95% within 53 ms) and a small 8 GB Windows VPS 500. In PostgreSQL mode, 1, 2 and 4 servers behind a load balancer answered 1,500, 2,600 and 3,750 (the highest step tried). If each player calls the server once every 30 seconds, 2,200 requests a second is about 66,000 players online and 3,750 over 110,000. Gameplay itself runs on your game servers. Your own numbers depend on your hardware and your game, so run the load test on your server.

Real-time (Advanced, 9 November 2026, on a rented 8-core Linux cloud server with the test players sent from a second server; rooms of 20, so each message goes to 19 players): one SQLite server delivered 1,000 room messages a second (19,000 deliveries, 95% within 3 ms); 4 PostgreSQL servers 2,000; 6 servers sharing rooms through Redis 4,000 (76,000 deliveries, 95% within 32 ms). 50,000 players stayed connected at once on 4 servers + Redis, none dropped. With 3 or more servers and busy rooms, set REDIS_URL so rooms travel through Redis.

# 6. Unity, Godot, Unreal and Other Clients

## Connecting your Unity game

1. Copy the `Assets` folder from `<Name>_Unity` into your Unity project's folder. It merges into `Assets/Scripts/Network`.
2. Add `<Name>Client` to an empty, persistent GameObject in your first scene (e.g. "NetworkManager").
3. Tick **Use Localhost** for local testing. For a live server, untick it and enter your server's address (and tick **Use Https** when it is behind HTTPS).
4. Sign the player in once at start-up, and again when they switch profile:

```
<Name>Client.Instance.SignIn(gamertag);
```

Every request after that carries the player's session automatically; if it expires, the client signs in again by itself. Every callback is `(bool ok, string response)`; on failure, response holds the server's error text.

| Method | What it does |
|---|---|
| `GetDbNode` / `SetDbNode` / `PatchDbNode` / `DeleteDbNode` | Read and write your data |
| `UploadMatch` | Send match results to the SR engine (the answer shows each player's SR before and after) |
| `GetSrHistory` | A player's recent SR changes in a mode |
| `RegisterIdentity` / `RenameIdentity` | Manage gamertags |
| `StartProfileTransfer` / `ClaimProfileTransfer` | Move a profile to a new device |
| `ExportMyData` / `DeleteMyAccount` | Player data rights |
| `SessionExpired` (event) | The session ended and could not be renewed: show your sign-in screen |

With the Advanced features switched on, the client also gets their methods (live updates, platform sign-in, moderation, remote config, cloud saves, leaderboards, economy, coupons, probability tables, parties, guilds, matchmaking, mail, trades, auctions and analytics), and the download includes `<Name>Realtime.cs` for the realtime WebSocket.

## Platforms

- **IL2CPP** (Android, iOS, WebGL): the download includes a `link.xml` that stops code stripping from removing the client's data classes. Keep it next to the client.
- **Plain http:// is blocked by Unity 2022 and newer** unless Player Settings allow it. While testing, allow HTTP in development builds (Project Settings > Player > Other Settings > Allow downloads over HTTP); release your game with HTTPS.
- **WebGL**: the server must allow your game's web address. Leave `CORS_ORIGINS` in `.env` empty (any site) or list your site, e.g. `CORS_ORIGINS=https://yourgame.itch.io`. A game served over https must also talk to an https server.
- **iOS**: Apple requires HTTPS and a domain name (Apple tests on IPv6-only networks, where a bare IPv4 address can fail), and apps that create accounts must let players delete them in the app: add a button that calls `DeleteMyAccount`. On iOS the device id can change when a player reinstalls, so pass a Game Center or Sign in with Apple id to `SignIn`, or offer profile transfer.

## Connecting your Godot game

Godot 4.2 or newer. The client is written in GDScript and works in every export, the web included.

1. Copy the `addons` folder from `<Name>_Godot` into your Godot project's folder (next to `project.godot`).
2. In Godot, open **Project > Project Settings > Plugins** and tick **<Name> Backend**. That adds the autoload `Backend`, so every script can use it.
3. Start your server (`start-local`), open the example scene `addons/<name>_backend/example/backend_example.tscn` and press **F6**. Sign in, save a bio and chat to see it working.

```
func _ready() -> void:
    Backend.url = "http://127.0.0.1:3000"        # your server, without /api
    var r = await Backend.sign_in("PlayerOne")
    if not r.ok:
        print("Sign-in failed: ", r.error)
        return
    await Backend.write("users/PlayerOne/profile/bio", "Hello")
    var bio = await Backend.read("users/PlayerOne/profile/bio")
    print(bio.data)
```

Every call returns a result with `ok`, `status`, `data` (the server's answer, parsed from JSON) and `error`; use `await` so your game keeps running while the request is on its way. Sessions renew by themselves; if one cannot, the `session_expired` signal fires. Your game's version (Project Settings > Application > Config > Version) is sent with every request, so the server can turn away old builds.

The method names follow Godot's style: `read`, `write`, `patch`, `remove`, `upload_match`, `sr_history`, `register_identity`, `start_profile_transfer`, `export_my_data`, `delete_my_account`, and `request()` for any other route. With the Advanced features on, the client also has `subscribe()` for live updates, the game-service methods (`cloud_save`, `leaderboard`, `buy_item`, `party_create`, `guild_create`, `queue_join` and more) and, with realtime on, `Backend.realtime()` for rooms and direct messages. `README.txt` lists them all.

- **Android**: tick the INTERNET permission in your Android export preset.
- **Web**: serve your game over HTTPS and your server too, and allow your game's address in `CORS_ORIGINS` (or leave it empty).

## Connecting your Unreal game

Unreal Engine 5 (built and tested on 5.8). The plugin is C++ with Blueprint nodes, and Blueprint-only projects work too: Unreal builds the plugin the first time you open the project.

1. Install Visual Studio with the **Game development with C++** workload (Visual Studio 2026 for Unreal 5.8; 2022 for Unreal 5.7 and older). Unreal needs it to build the plugin.

2. Close the Unreal Editor. Copy the `Plugins` folder from `<Name>_Unreal` into your project's folder (next to the `.uproject` file).
3. Open the project. When Unreal says the `<Name>Backend` module is missing and offers to rebuild it, click **Yes**.
4. Open **Edit > Project Settings > Plugins > <Name> Backend** and set **Url** to your server, without `/api`. The default is the backend on this PC (`http://127.0.0.1:3000`, or your port).
5. Right-click in any Blueprint graph and type **<Name> Backend** to see the nodes.

Every call is one node with **On Success** and **On Failure**. Both give a Result with `Ok`, `Status` (0 = no connection), `Json` (the server's answer) and `Error` (why it failed). A first test, on Event BeginPlay:

- **Sign In**, Gamertag "PlayerOne". Do this once at start-up: later calls carry the session and renew it by themselves.
- On Success: **Write Data**, Path `users/PlayerOne/profile/bio`, Json Value from **Json Make String** "Hello".
- On Success: **Read Data**, same path, then **Json Get String** (Result Json, Field empty) into **Print String**.

Read fields of an answer with **Json Get String / Number / Int / Bool / Array** (Field is a path such as `player`, `server.port` or `entries.0.score`), and build JSON to send with **Json Make String / Number / Bool / Object / Array**. Anything without its own node works with **Backend Request** (Method, Path, Json Body). **Get <Name> Backend Subsystem** gives you **Is Signed In**, **Get Signed In Player** and the events **On Signed In** and **On Session Expired** (show your sign-in screen again). Your game's version (**Project Settings > Project > Description > Project Version**) is sent with every request, so the server can turn away old builds.

From C++, add `<Name>Backend` to your game module's `PublicDependencyModuleNames`, then:

```
#include "<Name>BackendSubsystem.h"

auto* Backend = U<Name>BackendSubsystem::Get(this);
Backend->SignIn(TEXT("PlayerOne"), FString(),
    [Backend](const F<Name>BackendResult& R)
{
    if (!R.bOk) { UE_LOG(LogTemp, Warning, TEXT("%s"), *R.Error); return; }
    Backend->Request(TEXT("PUT"), TEXT("users/PlayerOne/profile/bio"),
        TEXT("\"Hello\""), [](const F<Name>BackendResult& Saved) {});
});
```

With the Advanced features on, the plugin also has nodes for platform sign-in, remote config, cloud saves, leaderboards, the economy, coupons, probability tables, game servers, matchmaking queues and player reports; parties, guilds, mail, trades and auctions work with **Backend Request**. The realtime WebSocket and live updates have no Unreal nodes yet. With game servers on, it has game-server nodes for your dedicated servers too (chapter 11, **How your game finds the servers**). `README.txt` in `<Name>_Unreal` lists everything.

- **Android and iOS**: use HTTPS for a live server; phones refuse plain http:// by default.
- **Testing from a phone or a second PC**: 127.0.0.1 means "this device". Use your computer's network address instead (for example `http://192.168.1.20:3000`) and allow the port in the Windows firewall.

## Web games, apps and other engines

The backend is not tied to one engine. Every backend also contains:

| What | Use it for |
|---|---|
| `clients/<Name>Client.mjs` (+ `.d.mts` types) | A JavaScript / TypeScript client for browser games, websites, Capacitor or Electron apps, admin tools and Node.js 18+. Same features as the Unity client, with promises. |

| What | Use it for |
| --- | --- |
| `GET /api/openapi.json` | An OpenAPI 3 description of every endpoint of your server (your plugins included). Import it into Postman or Swagger UI, or generate a client for Python and more with openapi-generator. |

```
import { MyGameClient } from './MyGameClient.mjs';
const api = new MyGameClient({ url: 'https://api.mygame.com', appVersion:
'1.0.0' });
await api.signIn('PlayerOne');
await api.set('users/PlayerOne/profile/bio', 'Hello');
const bio = await api.get('users/PlayerOne/profile/bio');
```

# 7. Player Sign-In and Access Rules

Players sign in (`SignIn` on the client, `POST /api/identity/session` on the server) and send their session token with every request. `access-rules.json` in the backend folder then decides what each player may do: for example `users/{player}` can only be changed by that player, direct messages only by the people in the conversation, and chat messages only with your own name as the sender. Admins and moderators are never restricted by these rules.

| Mode (accessRulesMode) | What happens |
| --- | --- |
| enforce | Requests the rules forbid are refused (403). |
| report | Nothing is refused, but every request that would be refused is logged ("Would refuse ..." in the server log, and `GET /api/admin/rule-violations`). |
| off | No per-player rules. |

## The safety net

- **Starter rules**: a new backend gets a rule for every public write route (by default "only that player may change their own area").
- **Unprotected routes**: `node access-rules.js check` lists every write route no rule protects, and the server warns about them at start-up. `node access-rules.js add-starter-rules` adds rules for them (and backs up your file first).
- **Test a request**: `node verify-backend.js --test write users/Alice/profile --as Bob` answers "allowed" or "refused" and names the rule that decided.

## Switching a live game from report to enforce

1. Make sure every app that talks to the server signs in and sends its session.
2. Raise the minimum client version so older builds that don't sign in are told to update (in `backend-config.json` and in `.env`; if they differ, `.env` wins).
3. Release the new builds and leave the server in report mode for a day or two.
4. Run the self-test. It should say no real player would be refused by enforce, or list who would be and why.
5. Set `"accessRulesMode": "enforce"` and restart. If something goes wrong, set "report" again and restart.

`access-rules.json` is yours: updates never overwrite it. `access-rules.example.json` always shows the current defaults.

### Player data rights

Signed-in players can call `ExportMyData` (all their data as JSON) and `DeleteMyAccount` (confirmed with their gamertag). Deletion removes the paths listed as player data and the device link, and anonymises what the player wrote elsewhere. Admins have `GET /api/admin/players/<name>/export` and `POST /api/admin/players/<name>/delete`, and every deletion is logged. Renaming (`RenameIdentity`) moves the player's stats, data, SR history and matches to the new gamertag and updates the references you listed.

> **Never put a password or key in a game build**
>
> Players only ever need `SignIn`. Never put the admin password, a moderator key or a server key into a public build: anyone could extract it. Game servers use a Game Server Key with only the permissions they need.

## 8. Skill Rating (SR)

Every match uploaded to the server changes each player's Skill Rating in that game mode; each mode has its own SR (`player_stats/<player>/sr_<mode>`). All SR settings are saved in `backend-config.json`, so you can tweak them there and restart, no new download needed.

### How a match is scored

- **Base points**: each tier in your Leagues has its own win and loss points. Higher tiers usually win less and lose more.
- **Rating gap**: with smooth gap scaling (the default) points follow the exact SR difference. Equal SR gives exactly the base points; beating a much stronger player gives up to 2x, beating a much weaker one very little, and losses work the other way round.
- **Win margin**: bonuses for a shut-out, dominant or clear win (and matching penalties for the loser), judged by points or by rounds won per mode.
- **Win streaks**, **placement matches** (a player's first matches move SR faster), **repeat-opponent protection** (stops win trading), **leaver penalties** and **Tier 1 protection** (or Hardcore, where SR can drop to 0) are all optional.
- **Draws** are allowed only in the modes you list.

### Uploading results

Matches come from your game server (with a Game Server Key that has the "matches" scope) or from staff tools, never from a game build players can download. In Unity: one `AddPlayer` per player when the match ends, `SetWinningTeam` or `SetTeamScore` for team modes, then `UploadMatch`. The answer says whether the match counted and each player's SR before and after.

If a match was not counted, the answer's `sr` says why: "not_finished" (the match status is still in progress), "no_winner" (no winner in a mode without draws), "practice" (marked as no stats) or "duplicate" (that match id was already counted).

### Staff tools

`GET /api/sr/history/<player>` shows a player's recent SR changes. Changed the rules on a live game? `POST /api/admin/recalculate-history` rebuilds every SR from the match history, with a preview that shows who would change before anything is saved.

# 9. Advanced Features

With the Advanced licence, the Add-on tab adds these features. Each one is a plugin with its own settings file (`*.json`), explained in the backend's `README-SETUP.txt` or in its own guide (`*.txt`) in the backend folder. Your settings files are never overwritten by an update.

## Database and scaling

- **SQLite** (default): one server process, fastest per server, nothing to install.
- **PostgreSQL**: several server replicas behind a load balancer share one database. One replica takes charge of scheduled jobs and another takes over automatically if it stops; changes reach players on every replica. Use the bundled database (Docker Compose / Kubernetes) or any managed PostgreSQL 13+ (Neon, Supabase, AWS RDS, Azure, Google Cloud SQL, DigitalOcean) with `DATABASE_URL` in `.env`.
- `migrate-sqlite-to-postgres.js` copies an existing SQLite database across and checks every row. Guide: `POSTGRES-SETUP.txt`.

## Deployment and hosting

Docker Compose (multi-stage Node.js image), Kubernetes manifests, PM2 cluster mode, an nginx load balancer and Redis caching, plus hosting templates for your own VPS with automatic HTTPS (Caddy), Render, Fly.io and Railway.

## Admin dashboard

A web page at `/admin` for admins and moderators: players, who is online, the data browser, moderation queue, leaderboards, economy, coupons, probability tables, remote config, push, game servers, matchmaking, guilds, analytics, market, server keys, access-rule reports, backups, data requests and a staff audit log of every change.

On a server with HTTPS the dashboard can be **installed like an app** on a phone or PC: open `/admin` in Chrome or Edge and choose **Install** (on a phone: **Add to Home screen**), or click **Install as an app** at the bottom of the dashboard's menu. It only ever talks to your own server, and nothing from your players' data is kept on the device.

## Live services

| Feature | What it does | Guide |
|---|---|---|
| Live updates | Clients subscribe to data paths and get changes instantly instead of polling, filtered by the access rules (`Subscribe()` in Unity). | README-SETUP.txt |
| Realtime WebSocket | Party, guild, match and lobby rooms, direct messages to online players, and data reads, writes and subscriptions over one connection. The Unity script reconnects and re-joins rooms by itself, WebGL included. With several servers, rooms reach players on every server (through Redis when REDIS_URL is set; lobby names can end in * for families like region-*). | PLAYER-SERVICES.txt |
| Push notifications | Firebase Cloud Messaging for direct messages, @mentions, friend requests, invites and friends coming online, controlled by a rules file. Includes a 6-digit phone pairing flow for companion apps. | PUSH-NOTIFICATIONS-SETUP.txt |

## Community safety and sign-in

| Feature | What it does | Guide |
|---|---|---|
| Moderation | A word filter that catches l33t speak and spaced letters, chat spam limits, timed mutes and bans, a player report queue. | README-MODERATION.txt |
| Chat blocks | The server refuses 1-on-1 messages between players who blocked each other. | README-SETUP.txt |
| Platform sign-in | Steam, Epic, Google, Apple, Game Center, Discord or any OpenID Connect provider, verified by the server with the platform itself. Players can link existing accounts. | PLATFORM-SIGN-IN.txt |

## Game services

| Feature | What it does | Guide |
|---|---|---|
| Remote config + maintenance | Change values without a new build, A/B test with percentages, target platforms and versions, and take the game offline with a message and an automatic end time. | GAME-SERVICES.txt |
| Cloud saves | Per-player save slots with conflict detection and version restore. | GAME-SERVICES.txt |
| Leaderboards | Daily, weekly, monthly and all-time boards, "around me", and boards that only accept scores from your game server. | GAME-SERVICES.txt |
| Economy | Server-side currencies and inventory, safe retries (players are never charged twice), a ledger, and Google Play / App Store receipt checking. | ECONOMY-SETUP.txt |
| Coupons | Redeem codes that give currencies and items: one code for everyone (WELCOME) or up to 10,000 one-off codes at a time, downloadable as a CSV. Start and end dates, a total limit, a web page where players redeem with their name, and limits on wrong guesses. Needs the Economy. | COUPONS-AND-LOOT.txt |
| Probability tables | Loot boxes, chests, gacha, a free daily spin and enemy drops, rolled by the server. Weights become exact percentages published at `/api/loot/tables` (the App Store and Google Play require the odds for paid random items), with pity, duplicates, daily limits and safe retries. The dashboard shows published odds next to real results. Needs the Economy. | COUPONS-AND-LOOT.txt |
| Mail, trades, auction house | Mail with attachments, two-player trades where both sides move at once, and an auction house with buyout, bids and a fee. | PLAYER-SERVICES.txt |
| Analytics | Daily active players, new players, retention (day 1/3/7/14/30), your own events with breakdowns, CSV export. | PLAYER-SERVICES.txt |

### Multiplayer services

The backend runs everything **around** your game servers; your dedicated servers (Unity headless with Mirror, Netcode for GameObjects or FishNet, or any engine) run the gameplay itself.

| Feature | What it does |
|---|---|
| Game Server Keys | Scoped keys for your servers: fleet, sessions, matches, matchmaking, leaderboards, economy, saves, groups, moderation or a data area. Shown once, revocable any time. |
| Game server fleet | Servers send heartbeats with their region, mode, zone, address and capacity. Players ask the backend for a server and get a one-use join ticket; the server redeems it and learns who the player really is. Zone handoff moves a player (and their state) between servers. |
| Parties and guilds | Roles, invitations and guild chat. |
| Matchmaking queues | Skill-based queues that keep parties together and reserve a server. |

Guide: `MULTIPLAYER-SERVICES.txt`. A C# SDK for your dedicated servers is in `game-server-sdk/UBGGameServer.cs`, and the Unreal plugin has game-server nodes (chapter 6). **Omni Cluster** (chapter 11) starts your dedicated servers on Windows or Linux, keeps them running and registers them in the fleet for you.

### Operations

Prometheus metrics (`/metrics`), a browsable API reference at `/docs` with the OpenAPI JSON, and LiveOps season rollover, which archives SR and starts a new season without taking servers offline.

## 10. Putting a Game Backend Online

1. Get a server that stays on. A small VPS (for example from Hetzner, DigitalOcean, OVHcloud, Linode or AWS Lightsail) is plenty to start; Windows servers work too.
2. Copy the backend folder to it (with `.env`, without `node_modules`), install Node.js 22 or newer, run `npm install`.
3. Keep it running with PM2 (not `start-local`, which stops when you close its window): `npm install -g pm2`, then `pm2 start server.js --name mygame` and `pm2 save`. (Or a systemd service; the Advanced hosting templates include one.)
4. Make it start again after the server reboots. Linux: `pm2 startup` (run the command it prints), then `pm2 save` again. Windows: `npm install -g pm2-windows-startup`, then `pm2-startup install` (this starts PM2 when you sign in; use a Windows service such as NSSM or WinSW if it must start before anyone signs in).
5. Point a domain name at the server and put HTTPS in front of it. Caddy is the simplest: a two-line Caddyfile `api.mygame.com { reverse_proxy localhost:3000 }` gets a certificate by itself.
6. In your game, untick Use Localhost, enter the domain and tick Use Https.
7. Run the self-test against the live address: `node verify-backend.js https://api.mygame.com`.

### Backups

The server backs up its database every 6 hours (configurable) into the `backups` folder and keeps the newest 28. `node restore-backup.js` lists them, `node restore-backup.js <name>` restores one (stop the server first; the current database is kept as a "before-restore" copy). Copy backups to another machine regularly: a backup on the same disk doesn't help if that disk dies.

### Security checklist

- A strong admin password (the generator's **New** button makes one), and `.env` kept off git and out of builds.
- HTTPS for every live server.
- A firewall that only lets in what you use: 80 and 443 for Caddy, your game servers' ports, and remote access (SSH or Remote Desktop), from your own address only if your host allows that. When everything reaches the backend through Caddy, keep its own port (3000) closed.
- Backups copied off the server regularly (see **Backups** above).
- Access rules in **enforce** mode once your game signs players in.
- Game Server Keys with only the scopes each server needs, instead of the admin password.
- The self-test shows no WARN lines.

## 11. Omni Cluster: Running Your Game Servers

**Omni Cluster** looks after your dedicated game servers, on Windows or on Linux. You tell it where your game's server build is; it starts the servers, restarts one if it crashes, starts another when they fill up, closes spare empty ones, and lists them all in your backend so players can find them. It works with Unity and Unreal server builds and comes with the **Advanced** licence.

Follow the seven steps below in order. Each one says exactly what to click and what you should see. It takes about 15 minutes the first time.

> **Before you start, you need**
>
> An **Advanced** licence (Base + Add-on).
>
> A **Windows 10 or 11** PC, or a Windows server, to run the game servers on. It can be the same machine as your backend or a different one. (On a Linux server, see **On a Linux server** further on.)
>
> Your game's **dedicated server build**: a folder with the `.exe` that runs a game server.
>
> Your backend's **admin password**. It is the `ADMIN_PASSWORD` line in the `.env` file in your backend folder.

### Step 1: Download your backend with game servers switched on

1. Sign in on backendgen.dev and open the **Game backend generator**.
2. On the **Add-on** tab, make sure **Use the Advanced Add-on for this backend** is on and **Game Servers** is ticked. (Both are on unless you switched them off.)
3. Click **Download backend (.zip)** and unzip the file.
4. Look inside the unzipped folder. You should see a folder called **OmniCluster** next to your `<Name>_Backend` folder. If it is not there, go back to point 2.

Already have a backend running from an earlier download? You do not need to replace it. Download again with the same settings, then copy just one file, `omni-licence.json`, from the new `<Name>_Backend` folder into your existing backend folder and restart the backend.

### Step 2: Start your backend

Start the backend the way you normally do (chapter 5 on your own PC, chapter 10 on a server). Two things matter here:

- The file `omni-licence.json` must stay in the backend folder. Do not delete, rename or edit it. It tells Omni Cluster that this backend belongs to your licence.
- Write down the backend's address. On the same PC it is `http://127.0.0.1:3000`. On a server it is the server's address, for example `http://203.0.113.5:3000` or `https://api.mygame.com`. (Use your own port if you changed it from 3000.)

## Step 3: Make a server key

A server key is a password made just for your game servers. Omni Cluster uses it instead of your admin password, so your admin password never leaves the backend.

1. Open a browser and go to your backend's address followed by `/admin`, for example `http://127.0.0.1:3000/admin`.
2. Log in with your admin password.
3. In the menu on the left, click **Server keys**.
4. In the **New server key** box, type a **Name**, for example `Omni Cluster`.
5. Tick two boxes: **fleet** and **matches**. Leave the others unticked.
6. Click **Create key**. The key appears in a yellow box. It is shown only once and starts with `ubgs1.`
7. Copy the whole key and paste it into Notepad for the moment. You will need it in Step 6. If you lose it, just make another one.

## Step 4: Download and open Omni Cluster

1. On backendgen.dev open **Your account**. In the **Omni Cluster** card, click **Download Omni Cluster**. (The card only shows for Advanced accounts.)
2. Unzip `OmniCluster.zip` somewhere you will find it again, for example `C:\OmniCluster`. Keep all the files together.
3. Double-click `OmniCluster.exe`.
4. Windows shows a blue box saying **Windows protected your PC**. This is normal for a new program. Click **More info**, then click **Run anyway**. There is nothing else to install.

You should now see the Omni Cluster window with a **Getting started** card.

## Step 5: Import your setup file

1. On the Getting started card, click **IMPORT SETUP…** (it is also on the **APP SETTINGS** tab).
2. Go to the folder you unzipped in Step 1, open the **OmniCluster** folder and pick the file ending in `.omnisetup`.
3. A box lists what will change. Click **Yes**.

Two new tabs appear at the top: **CLUSTER** and **CLUSTER SETTINGS**. The setup file has filled in your licence and set Omni Cluster up for this kind of backend.

## Step 6: Fill in CLUSTER SETTINGS

Click the **CLUSTER SETTINGS** tab and fill in the boxes in this table. Leave every other box as it is.

| Box | What to put in it |
|---|---|
| **Backend API URL** | Your backend's address from Step 2 with `/api/` on the end. Examples: `http://127.0.0.1:3000/api/` or `https://api.mygame.com/api/`. The ending `/api/` is needed. |
| **Backend key** | The server key from Step 3 (it starts with `ubgs1.`). |
| **Host name** | A short name for this machine, for example `EU-1`. If you run Omni Cluster on two machines, give each a different name. |
| **Region** | Where this machine is, for example EU or US. Players can filter servers by it. |
| **Game modes** | One row per game mode you want servers for. Click **+ Add**, type the **Mode name** (for example `Deathmatch`) and, for Unreal, the **Map**. Leave **Host it** ticked. |
| **Engine** | `Unity` or `Unreal`. |

| Box | What to put in it |
|---|---|
| **Game folder** | Click **Browse...** and pick the folder that holds your server build. |
| **Game exe** | The name of the server `.exe` inside that folder, for example `MyGameServer.exe`. For Unreal it is usually in a sub-folder: `MyGame\Binaries\Win64\MyGameServer.exe`. |
| **Launch command line** | What the game needs to start as a server. Omni Cluster replaces `{port}`, `{map}`, `{mode}` and `{maxPlayers}` with the right values for each server. Unity example: `-port {port} -mode {mode}`. Unreal example: `{map}?listen?Port={port}?MaxPlayers={maxPlayers} -log`. |
| **Headless switch** | Runs the server without drawing graphics. Unity: `-batchmode -nographics`. Unreal: `-NullRHI`. |
| **Max players** | How many players fit on one server. |
| **First game port** | The port of the first server, for example 7777. The next servers use the ports after it. |

Then click **SAVE CLUSTER SETTINGS** at the top.

> **Let players in**
> Players connect straight to your game servers, so the game ports must be open. On a home network, forward the ports on your router to this PC (or tick **Open ports with UPnP** in CLUSTER SETTINGS and let Omni Cluster ask the router). On a server, allow them in the firewall. When Windows asks whether to allow the game or Omni Cluster through the firewall, click **Allow**.

## Step 7: Launch

1. Click the **CLUSTER** tab.
2. Click the big **LAUNCH CLUSTER** button.

Within a few seconds you should see:

- **Cluster RUNNING** in green next to the button.
- One row per game mode under **CLUSTER SERVERS**, each with the state **Running**.
- The same servers under **ACTIVE SERVERS (BACKEND)** on the right. That list comes from your backend, so it proves the two are talking.

If a box with a message appears instead, look it up under **If something goes wrong on Windows** below.

## Check it from the dashboard

Open your backend's dashboard (`/admin`) and click **Game servers** in the menu. Your servers are listed with their players and status. From there you can:

| Button | What it does |
|---|---|
| **Restart** | Restarts that server as soon as it is empty. |
| **Stop** | Stops that server now, even with players on it. If it was the only server of its mode, Omni Cluster starts a fresh one. |
| **Restart all** | Restarts every server, each one when it is empty. |
| **Start private server** | Starts an extra server with a password. It is not shown to other players. |

Omni Cluster carries these out within a few seconds. It has to be running with at least one server for the buttons to do anything.

## Everyday use on Windows

| You want to | Do this |
|---|---|
| Stop all servers | CLUSTER tab > **STOP CLUSTER**. |
| Start them when Windows starts | CLUSTER SETTINGS > tick **Auto-launch the cluster when Omni Cluster opens**, and put a shortcut to `OmniCluster.exe` in the Windows Startup folder. |
| Add or change a game mode | CLUSTER SETTINGS > **Game modes** > **+ Add** or edit a row, **SAVE CLUSTER SETTINGS**, then stop and launch the cluster again. |
| See what a server is printing | **SERVERS** tab > click the server. Its log is underneath. |
| Run servers on a second machine | Install Omni Cluster there, import the same `.omnisetup` file, use the same server key (or make a second one), and give it a different **Host name**. |
| Move to a new PC | APP SETTINGS > **EXPORT SETUP…** on the old PC, **IMPORT SETUP…** on the new one. |

## If something goes wrong on Windows

| What you see | What to do |
|---|---|
| There is no **OmniCluster** folder in the download | You need the Advanced licence, and on the Add-on tab both **Use the Advanced Add-on for this backend** and **Game Servers** must be on. You also have to be signed in to your own account when you download. |
| There is no **Omni Cluster** card on the account page | The card only shows for accounts with Advanced. Check **Your licence** at the top of the page. |
| "Windows protected your PC" | Click **More info**, then **Run anyway**. |
| "Cluster mode needs a licence token" | Import the `.omnisetup` file from the **OmniCluster** folder of your download (Step 5). |
| "Enter a server key (it starts with ubgs1.)" | The **Backend key** box is empty or holds something else. Paste the server key from Step 3 and save. |
| "Could not reach the backend's game-server list" | Read the end of the message. **401** means the server key is wrong or was deleted: make a new one. **403** means the key has no **fleet** scope: make a new key with **fleet** and **matches**. Anything else: check **Backend API URL** (it must end in `/api/`) and that the backend is running. |
| "This backend has no licence file" | `omni-licence.json` is missing from the backend folder. Download the backend again, copy that one file into the backend folder and restart the backend. |
| "This licence token belongs to a different backend" | The backend was downloaded from another account. Use the `.omnisetup` file and the backend from the same download. |
| "Game exe not found" | Check **Game folder** and **Game exe** in CLUSTER SETTINGS. |
| A server starts and stops again straight away | Click it on the **SERVERS** tab and read its log. Usually the **Launch command line** is not what your game expects. |
| Servers are running but players cannot join | The game ports are not open. See **Let players in** under Step 6. |

| What you see | What to do |
|---|---|
| The dashboard buttons do nothing | Omni Cluster must be open with the cluster running. Wait about ten seconds and click **Refresh**. |

## On a Linux server

Omni Cluster also comes for Linux. The Linux edition runs the same cluster engine as the Windows program, as a background service: it starts your servers, restarts them, adds and closes them, runs the voice relay and obeys your dashboard. It has no window: you answer a few questions once, then use your dashboard. Linux servers usually cost less to rent than Windows ones.

You need:

- A **Linux server** (VPS) with systemd. When you rent one, pick **Ubuntu** (22.04 or newer); Debian 12 works too. Keep the email from your provider: it has the server's **IP address**, the **user name** (often `root`) and the **password**.
- Your game's **Linux server build** (Linux step 1). The Windows `.exe` does not run on Linux.
- The `.omnisetup` file from the `OmniCluster` folder of your backend download, and a **server key** (Step 3 above).
- A Windows PC to do the setup from. Everything below is done from it.

### Linux step 1: Make a Linux server build of your game

**Unity:**

1. Open **Unity Hub** > **Installs**. Click the gear next to your Unity version > **Add modules**, tick **Linux Dedicated Server Build Support** and click **Install**. Restart Unity when it is done.
2. Unity 6: **File** > **Build Profiles** > **Add Build Profile** > **Linux Server** > **Add Build Profile**, then **Switch Profile**. Unity 2021 or 2022: **File** > **Build Settings** > **Dedicated Server**, set **Target Platform** to **Linux**, then **Switch Platform**.
3. Click **Build** and pick a new, empty folder, for example `MyGameServer` (no spaces in the name).
4. The folder now holds a file ending in `.x86_64` (that is the server), a `MyGame_Data` folder and `UnityPlayer.so`. You copy the **whole folder** to the server in Linux step 3.

**Unreal:** package your game for the **LinuxServer** target. This needs Epic's Linux cross-compile toolchain on your PC (see Epic's guide to cross-compiling for Linux). The packaged folder has a `MyGameServer.sh` file: that is what Omni Cluster runs.

### Linux step 2: Download Omni Cluster for Linux

On your account page, click **Download for Linux (x64)**. That is right for almost every server. Only if your provider says the server is **ARM** (for example Ampere or Graviton), use **Linux (ARM64)** instead.

### Linux step 3: Copy the files to the server

WinSCP is a free program that copies files to a server, like a second File Explorer.

1. Download WinSCP from **winscp.net** and install it.
2. Open it. In the **Login** window set **File protocol** to **SFTP**, **Host name** to your server's IP address, **Port number** to **22**, and type the **User name** and **Password** from your provider. Click **Login**.
3. The first time, it asks whether you trust this server: click **Accept** (or **Yes**).
4. The left side is your PC, the right side is the server. Drag these from left to right: the Omni Cluster download (`OmniCluster-linux-x64.tar.gz`), your game's server folder from Linux step 1, and your `.omnisetup` file.
5. Look at the folder name at the top of the right side, for example `/root`. Your game folder on the server is that plus the folder's name, for example `/root/MyGameServer`. Write it down: you need it in Linux step 5.

### Linux step 4: Connect to the server and install

1. On your PC, open the **Start** menu, type **Terminal** (or **Command Prompt**) and open it.
2. Type `ssh root@203.0.113.5`, with your user name and your server's IP address, and press **Enter**.
3. The first time, it asks "Are you sure you want to continue connecting?": type `yes` and press **Enter**.
4. Type the password and press **Enter**. Nothing appears while you type it: that is normal.
5. You are now typing on the server. Type these three lines, pressing **Enter** after each:

```
tar -xzf OmniCluster-linux-x64.tar.gz
cd omni-cluster
sudo bash install.sh
```

It ends with "Omni Cluster ... is installed" and the next steps. To paste into the Terminal, **right-click**.

> **Logged in as root?**
> If a command answers `sudo: command not found`, you are logged in as `root` on a server without sudo. Then leave out the word `sudo` in every command in this section, for example `bash install.sh`.

### Linux step 5: Answer the setup questions

Type `sudo omni-cluster setup` and press **Enter**. It asks one thing at a time. Press **Enter** on its own to keep the value shown in [brackets].

| It asks | What to type |
|---|---|
| **Path to the .omnisetup file** | Where you copied it, for example `/root/MyGame.omnisetup`. |
| **Backend API URL** | Your backend's address, for example `http://203.0.113.5:3000/api/`. |
| **Server key** | Right-click to paste the key from Step 3 (it starts with `ubgs1.`). Stars appear instead of the key. |
| **Host name**, **Region** | A name for this machine, for example `EU-1` (each machine needs its own), and its region, for example EU. |
| **Engine** | `unity` or `unreal`. |
| **Game folder** | The folder from Linux step 3, for example `/root/MyGameServer`. |
| **Game file to run** | It lists the Linux programs it found and suggests one, for example `MyGame.x86_64` (Unity) or `MyGameServer.sh` (Unreal). Press **Enter** if that is right. |
| **Launch command line** | Press **Enter** to keep the example, unless your server needs other switches (as on Windows, Step 6). |
| **Players per server**, **Mode names**, **Most servers**, **First game port** | As on Windows. Separate mode names with commas, for example `Duel, Team`. |
| **Run the voice relay?** | `yes` or `no`. With yes, type the connection key your game uses (see **Voice chat between players**). |

At the end it checks everything. Each line marked **[!!]** says what is wrong and the command that fixes it. Fix them, then type `sudo omni-cluster check` to check again.

### Linux step 6: Open the ports

Players connect to your game servers over **UDP**. The check printed the ports, for example 7777 to 7794. Two things can block them:

1. Your provider's firewall, if it has one: in their web control panel, allow those **UDP** ports.

2. The server's own firewall: type `sudo ufw status`. If it says **inactive** (or "command not found"), there is nothing to do. If it says **active**, type the line the check printed, for example `sudo ufw allow 7777:7794/udp`.

> **Don't lock yourself out**
> Don't turn ufw on just for this. If you ever do, allow SSH first (`sudo ufw allow OpenSSH`), or you can no longer connect to your server.

### Linux step 7: Start it

```
sudo systemctl start omni-cluster
omni-cluster status
```

`status` lists your servers, their ports and players, and they appear in your dashboard under **Game servers** within seconds. You can close the Terminal now: Omni Cluster keeps running, and it starts by itself whenever the server restarts.

### Every day on Linux

| Command | What it does |
|---|---|
| `omni-cluster status` | The servers, their ports and players. |
| `sudo omni-cluster check` | Checks everything and says what to fix, with the command that fixes it. |
| `journalctl -u omni-cluster -f` | What it is doing, live. Ctrl+C stops watching. |
| `sudo systemctl restart omni-cluster` | Restarts it, for example after changing a setting. |
| `sudo omni-cluster set MaxServers 6` | Changes one setting. `omni-cluster settings` lists them all with their names. |
| `sudo omni-cluster modes add Ranked` | Adds a game mode (also: remove, on, off, map). |

Connect with `ssh` as in Linux step 4 to type these. To **update your game**, type `sudo systemctl stop omni-cluster`, copy the new build over the old one with WinSCP, then `sudo systemctl start omni-cluster`. To **update Omni Cluster**, copy the new download over, then type the three lines from Linux step 4 again: your settings are kept. Your game's own logs are in `/opt/omni-cluster/data/Logs` (you can open them in WinSCP).

### If something goes wrong on Linux

Start with `sudo omni-cluster check`: it names the problem and the command that fixes it.

| What you see | What to do |
|---|---|
| `Permission denied (publickey)` when you connect | Your provider uses a key file instead of a password. Use the key they gave you, or the web console in their control panel. |
| `sudo: command not found` | You are logged in as root: leave out the word `sudo`. |
| `cannot execute binary file`, or "This is the arm64 download" | Wrong download. Type `uname -m`: `x86_64` needs **Linux (x64)**, `aarch64` needs **Linux (ARM64)**. |
| "Run it with sudo" or "can't read Omni Cluster's settings" | Put `sudo` in front of the command. |
| **[!!] Licence** | Import the `.omnisetup` file that came with **this** backend: `sudo omni-cluster import /root/MyGame.omnisetup`. |
| "still says YOUR-SERVER-ADDRESS" | Type the command it shows, with your backend's real address. |

| What you see | What to do |
|---|---|
| "is a Windows program" | Copy your game's **Linux** server build (Linux step 1), not the Windows one. |
| A server starts and stops again straight away | Read its log in `/opt/omni-cluster/data/Logs`. Usually the launch command line is not what your game expects: run `sudo omni-cluster setup` again. |
| `omni-cluster status` says it is not running | Type `journalctl -u omni-cluster -n 50` to see why, fix it, then `sudo systemctl start omni-cluster`. |
| Servers run, but players can't connect | The UDP ports are closed: Linux step 6 (and your provider's control panel). |
| Servers don't appear in the dashboard | Check the server key has the **fleet** and **matches** scopes, and that the backend address is right: `sudo omni-cluster check`. |

To remove Omni Cluster: `sudo bash /opt/omni-cluster/install.sh --remove` (your settings stay in `/opt/omni-cluster/data`). UE4SS, UPnP and typing into a game window are Windows-only.

## How your game finds the servers

Your game asks the backend for a server and connects to the address it gets back, with a one-use join ticket. In Unity that is `FleetServers` (the list) and `FleetJoin` (pick one and get a join ticket); in Godot `fleet_servers` and `fleet_join`; in Unreal the **List Game Servers** and **Join Game Server** nodes. Other engines call `GET /api/fleet/servers` and `POST /api/fleet/join`. Your game server then checks the ticket: `RedeemTicket` in the C# SDK, the **Redeem Join Ticket** node in Unreal. The guide `MULTIPLAYER-SERVICES.txt` in your backend folder explains join tickets.

### Adding servers when they fill up

Omni Cluster starts another server of a mode when every server of that mode is full, and closes a spare one after it has been empty for a while. To know how full a server is, the game server has to report its player count to the backend. With the C# SDK (`game-server-sdk/UBGGameServer.cs`) that is one call to `StartHeartbeat`; in Unreal it is the **Start Heartbeat** node, which counts the players for you (Port 0 uses the server's port). Use the server id Omni Cluster gives each server, in the environment variable `OMNI_SERVER_ID` (the port is in `OMNI_PORT`; the Unreal plugin reads both itself), and the **same server key** as Omni Cluster. Set the key as an environment variable on the machine (`UBG_SERVER_KEY`, with `UBG_BACKEND_URL` for the address) and Omni Cluster passes it on to every server it starts. If your game does not report players, everything else still works; Omni Cluster simply keeps one server per mode.

## Voice chat between players (optional)

Omni Cluster can also carry voice chat. Next to every game server it runs a small **voice relay**: each player's game sends its microphone to the relay, and the relay passes it on to everyone else on that server. So every game server is its own voice channel, with no voice service or monthly fee.

Your game needs a voice chat that talks to the relay. For **Unity** one is included: the **Unity voice chat** folder in the Omni Cluster download (Windows and Linux downloads alike). It works in Windows, Mac, Linux, Android (including Meta Quest) and iOS builds, but not in WebGL, because browsers cannot use UDP. For other engines, see **Other engines** at the end of this section.

### Voice step 1: Choose a connection key

Make up a key, for example `mygame-voice-1`. It is not a password; it just stops stray traffic from joining your voice channels. You will type it in two places (voice steps 2 and 3), and it must be exactly the same in both, including capital letters.

### Voice step 2: Switch the relay on in Omni Cluster

1. Click the **CLUSTER SETTINGS** tab and scroll down to **Voice relay**.
2. Tick **Run the voice relay**.
3. In **Connection key**, type your key from voice step 1.
4. Click **SAVE CLUSTER SETTINGS**. If the cluster is running, click **STOP CLUSTER** on the **CLUSTER** tab and then **LAUNCH CLUSTER** again.

On a Linux server, answer `yes` to **Run the voice relay?** in `sudo omni-cluster setup` and type the key, or set it later with `sudo omni-cluster set VoiceEnabled yes` and `sudo omni-cluster set VoiceConnectionKey mygame-voice-1`, then `sudo systemctl restart omni-cluster`. Open the voice ports with ufw as in Linux step 6.

Each server's voice port is its game port + 1: a server on port 7777 has its voice on 7778. (That is the **Voice port offset** box under **Ports**; leave it at 1.)

> **Open the voice ports too**
>
> The voice ports must be open for **UDP**, just like the game ports. On a server, allow them in the firewall: if your game servers use ports 7777 to 7799, allow 7777 to 7800 so every voice port is covered. On a home network, forward them on the router too, or tick **Open ports with UPnP**, which opens the game port and the voice port of every server.

### Voice step 3: Add the voice chat to your Unity project

1. Open the folder where you unzipped Omni Cluster (Step 4, or Linux step 2), then the **Unity voice chat** folder inside it.
2. In your Unity project, make a folder `Assets/Plugins/UBGVoice`. Drag `UBGVoiceChat.cs`, `LiteNetLib.dll` and `LiteNetLib-LICENSE.txt` into it.
3. Pick an object that stays for the whole match, for example your network manager. Click **Add Component** and choose **UBG > UBG Voice Chat**.
4. In the component's **Connection Key** box, type the same key as in Omni Cluster.
5. iOS only: fill in **Player Settings > Other Settings > Microphone Usage Description**. On Android and Quest there is nothing to set: the game asks the player for the microphone the first time.

- If your project already has LiteNetLib, keep only one copy, and it must be version **1.2.0**: that is the version Omni Cluster uses.

### Voice step 4: Connect when the player joins a server

When your game joins a game server, call `ConnectToGameServer` with the same address and port your game connects to. When the player leaves the match, call `Disconnect`. For example:

```
using UBG.Voice;
public UBGVoiceChat voice;  // drag the object with the component here

void OnJoinedServer(string address, int gamePort) {
    voice.playerName = gamertag;  // optional: shown to the others
    voice.ConnectToGameServer(address, gamePort);  // voice = port + 1
}
void OnLeftServer() { voice.Disconnect(); }
```

If your game finds its server with `FleetServers` or `FleetJoin`, the reply also has each server's voice port, in `meta.voicePort`. You can pass it to `voice.Connect(address, voicePort)` instead.

### Voice step 5: Test it

1. Test your microphone on its own first: on the component, tick **Hear Myself**, press **Play** in Unity and talk. You should hear yourself. Untick it afterwards.
2. Launch the cluster in Omni Cluster.

3. Join the same game server from two copies of your game, for example two PCs, or a build and the Unity Editor.
4. Omni Cluster's log shows `[voice 7778] client connected` for each player.
5. Talk. The other copy should hear you.

### Settings on the component

| Setting | What it does |
|---|---|
| **Talk Mode** | **Voice Activated** sends when the player speaks. **Push To Talk** sends while a key is held. **Always On** sends all the time. |
| **Push To Talk Key** | V unless you change it. With the new Input System or a VR controller button, set `voice.TalkButtonHeld` to true or false from your own input code instead. |
| **Voice Activation Level** | Raise it if background noise is sent. Lower it if the start of words is cut off. |
| **Volume**, **Mute Everyone** | What the player hears. |
| **Spatial Blend** | 0 = everyone sounds the same. 1 = 3D voice: move each speaker's sound to their player with `voice.GetSpeakerSource(id)`. |

From your own code you can also show a speaker icon (`voice.IsSpeaking(id)`), mute one player for yourself (`voice.SetSpeakerMuted(id, true)`) and learn who is who (`voice.OnSpeakerNamed`). The `README.txt` in the **Unity voice chat** folder lists them all.

### If voice chat does not work

| What you see | What to do |
|---|---|
| "the relay refused the connection key" in the Unity Console | The key on the component is not the same as **Connection key** in Omni Cluster. Check for spaces and capital letters. |
| "no answer (is the voice relay on, and is UDP port … open?)" | **Run the voice relay** is not ticked, the cluster is not running, or the firewall or router blocks that UDP port. Omni Cluster's log should say `voice hub started` for each server. |
| "could not open the voice port (already in use?)" in Omni Cluster's log | Another program uses that port. Change **First game port** in CLUSTER SETTINGS. |
| Connected, but nobody is heard | Both players must be on the **same** game server. Test the microphone with **Hear Myself**, and check that **Voice Activation Level** is not too high. |

### Other engines

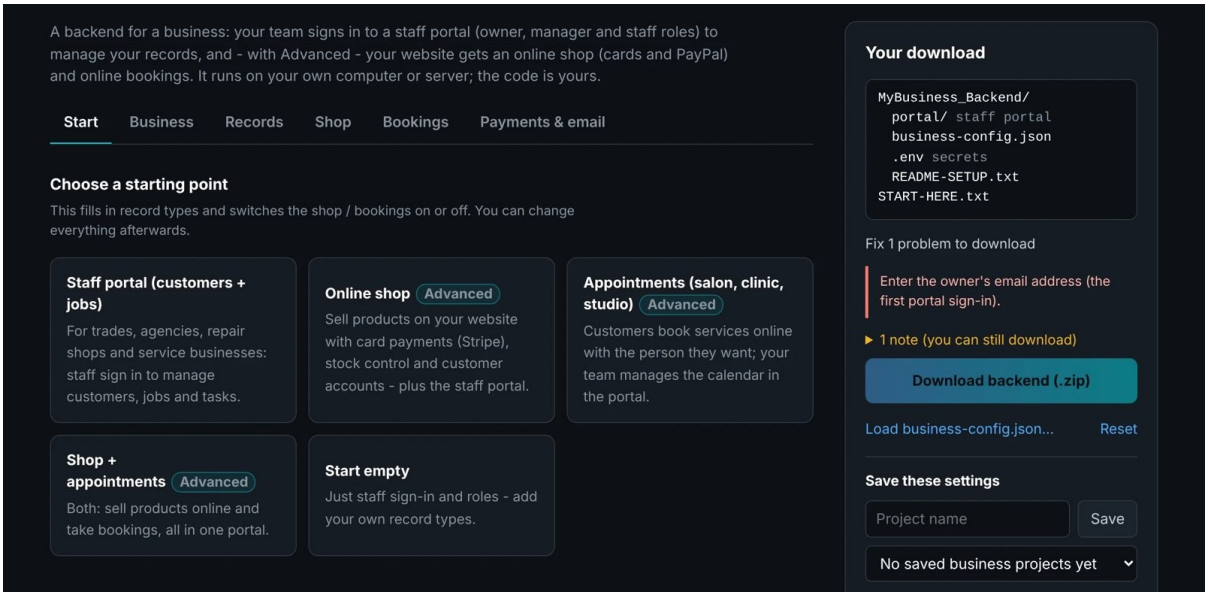
The relay itself works with any engine: it accepts LiteNetLib 1.2.0 connections with your connection key and passes every packet a player sends to everyone else on that server. The ready-made voice chat is for Unity only; Godot and Unreal have none yet. To build your own, the packet format is described at the top of `UBGVoiceChat.cs`.

## Three rules

- **One Omni Cluster per backend.** One installation looks after the servers of one backend. For a second game, use another PC, another Windows user account or another Linux server.
- **Keep the setup file to yourself.** It holds your licence token, and the token only works with backends downloaded from your account.
- **Never put the server key or the admin password in a game build.** They belong on your servers only.

# 12. The Business Generator

Open it from the menu (**Business generator**). It makes a backend for a business: a staff portal your team signs in to, with the records you choose, and with Advanced an online shop and online bookings for your website. It runs on your own computer or server and keeps its data in one database file.




*The business generator: choose a starting point, then adjust the tabs.*

## The tabs

| Tab | What you set |
|---|---|
| Start | A starting point (below). It fills in record types and switches the shop and bookings on or off; you can change everything afterwards. |
| Business | Business name, the owner's email (the first person to sign in to the portal), time zone, language / region (how dates and prices are written), currency, the folder name, the port, the public address (used for links in emails), the websites allowed to use the shop and booking API, how long staff stay signed in, and backups. |
| Records | Your record types (e.g. Customers, Jobs) and their fields: text, long text, number, money, date, date and time, email, phone, link, choice list or checkbox. Choose which fields show as columns and whether staff may delete. |
| Shop (Advanced) | Card payments (Stripe), PayPal or both, Stripe Tax, delivery address, flat delivery charge and free delivery threshold, customer accounts, low-stock emails, order number prefix and the thank-you page on your website. |
| Bookings (Advanced) | Opening hours, start times every N minutes, minimum notice, how far ahead people can book, the online cancellation cut-off, reminder emails, and whether customers choose who they see. |
| Payments & email | The owner's first password (optional), Stripe and PayPal keys, and your email provider's SMTP settings (sender and where new orders and bookings are announced). |

## Starting points

| Starting point | For | Licence |
|---|---|---|
| Staff portal (customers + jobs) | Trades, agencies, repair shops and service businesses: staff manage customers, jobs and tasks. | Base |
| Online shop | Sell products on your website with card payments, stock control and customer accounts, plus the staff portal. | Advanced |
| Appointments (salon, clinic, studio) | Customers book services online with the person they want; your team manages the calendar. | Advanced |
| Shop + appointments | Both, in one portal. | Advanced |
| Start empty | Just staff sign-in and roles; add your own record types. | Base |

## Run it and sign in

1. Install Node.js 22 or newer. Unzip the download, open the backend folder and double-click `start-local.cmd` (Windows) or `start-local.command` (Mac). Or run `npm install`, then `npm start`.
2. Open `http://localhost:3000/portal` (or your port).
3. Sign in with the owner's email. The first password is the `OWNER_PASSWORD` line in `.env` (only used for this first sign-in); you choose your own straight away and can then delete that line.
4. Open **Team** to add the people who work with you. Each gets a temporary password (shown once) and chooses their own at their first sign-in.

| Role | Can do |
|---|---|
| Owner | Everything, including adding and removing managers and owners. |
| Manager | Everything except changing owners and other managers. |
| Staff | Day-to-day work: view, add and edit records, process orders and update stock, manage bookings. They can't delete (unless a record type allows it), refund, change prices, export or see the activity log. |

Locked out as the owner? Run `node reset-owner.js` on the server for a new temporary password. Anyone else: the owner or a manager resets their password under Team. The **Activity log** shows who did what (which fields changed, never their contents).

## The shop (Advanced)

1. Portal > **Products**: add what you sell (price, stock, picture link). Leave stock empty for unlimited.
2. **Stripe** (cards): create an account at stripe.com in Test mode, copy the secret key into `.env` (`STRIPE_SECRET_KEY`), add a webhook to `https://<your server>/api/stripe/webhook` and copy its signing secret (`STRIPE_WEBHOOK_SECRET`). Test with card 4242 4242 4242 4242.
3. **PayPal** (optional, on its own or next to Stripe): create an app at developer.paypal.com (Sandbox first), copy its Client ID and Secret into `.env`, and add a webhook once your server is live.
4. Portal > **Orders**: mark orders sent or collected (the customer gets an email). Managers can refund card and PayPal payments, in full or in part.
5. Going live: switch Stripe and PayPal to live mode and use the live keys.

Money goes straight to your own Stripe or PayPal account; BackendGen.dev never sees it, and card details never touch your server. Without payment keys the shop still takes orders, shown as "Awaiting payment", and you mark them paid. Stock is held while a customer pays and given back automatically if they don't finish.

## Bookings (Advanced)

1. Set your opening hours (e.g. `09:00-17:00`, or `09:00-12:00, 13:00-17:00` for a lunch break; empty = closed).
2. Portal > Team: tick **Takes bookings** for everyone customers can book, and give them their own hours if they differ.
3. Portal > Bookings > **Services**: what can be booked, how long it takes, a gap afterwards and the price shown.
4. Customers book on your website. They get an email with a link to cancel (up to your cut-off) and a reminder.
5. Portal > Bookings > **Calendar**: today's and this week's bookings; add phone bookings, move or cancel them, mark them done or no-show, and add time off.

Online bookings don't take payment: prices are shown and paid on the day.

## Emails

Without email settings everything works, but nothing is emailed. Add your email provider's SMTP details (`SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASS`) to `.env`, or on the Payments & email tab before downloading, and restart. Most providers work: your web host, Microsoft 365, Google Workspace (with an app password), Brevo, Postmark, Amazon SES, Mailgun.

## Your website

Your website talks to the backend through its public API. Copy `clients/<Name>Client.mjs` into your site and use it like the example pages do (`examples/shop.js`, `examples/booking.js`), add your website's address to the allowed websites (`allowedOrigins`), and set the shop's thank-you page. The full list of endpoints is at `/api/openapi.json` on your server.

Showing your products and taking an order (here the business is called "My Business", so the client is `MyBusinessClient`):

```
import { MyBusinessClient } from './MyBusinessClient.mjs';
const api = new MyBusinessClient({ url: 'https://api.yourbusiness.com' });
const { products } = await api.products();  // name, priceText, inStock...
const r = await api.checkout({ email: 'ann@example.com', name: 'Ann',
  items: [{ productId: products[0].id, quantity: 1 }] });
if (r.checkoutUrl) location.href = r.checkoutUrl;  // pay on Stripe or PayPal
```

Booking an appointment:

```
const { services } = await api.services();
const { slots } = await api.availability(services[0].id, '2026-12-01');
await api.book({ serviceId: services[0].id, start: slots[0].start,
  name: 'Ann', email: 'ann@example.com' });
```

Errors (a time that has just been taken, a sold-out product) arrive as a `BackendError` whose message you can show to the customer as it is.

## Going live

1. Run it on a server that stays on, with PM2 (`npm install -g pm2`, `pm2 start server.js --name mybusiness`, then `pm2 save`) or a systemd service. `start-local` is only for trying it: it stops when you close its window. To start PM2 after a reboot: `pm2 startup` on Linux, or `pm2-windows-startup` on Windows (see chapter 10).
2. Put HTTPS in front of it (Caddy: `api.yourbusiness.com { reverse_proxy localhost:3000 }`) and point a domain name at the server.
3. In `business-config.json` set `publicUrl` to that address and `serveExamples` to false. Restart.
4. Run `npm run self-test` on the server.

5. Copy the backups off the server now and then. Restore one with `node restore-backup.js backups/<file>.sqlite` (stop the server first).

Then go through the **Security checklist** at the end of chapter 10: most of it applies to a business backend too (a strong owner password, `.env` kept private, HTTPS, a firewall that keeps port 3000 closed, backups copied off the server). The self-test for a business backend is `npm run self-test`.

> **Your customers' data**
>
> The backend stores personal data (names, emails, addresses, orders, bookings) on your server, so you are responsible for it (in the UK and EU you are the "data controller"). Tell customers in your website's privacy policy what you keep and why. Customer passwords are stored scrambled, and card and PayPal details never touch your server.

# 13. Saved Projects and Updates

## Saving your settings

In either generator, type a name under **Save these settings** and click **Save**. Saved projects are listed under **My projects** on your account page, where you can **Open** or **Delete** them; you can also open one from the list in the generator. Passwords and keys are never saved: you keep those in your own `.env`.

Lost a project? **Load backend-config.json...** (game) or **Load business-config.json...** (business) reads the settings of any backend you downloaded, and its `.env` if you pick that too.

## Updating a live server

When a new version comes out, or you have changed settings, don't copy files by hand. Open your saved project and click **Download update package**. You get a folder `<Name>_Backend_Update` with the new program files and an updater.

1. Unzip it and copy the whole folder onto the server (for example onto its desktop). If PM2 runs your server it can go anywhere; otherwise put it next to the live backend folder.
2. Optional: double-click **1 - PREVIEW (changes nothing).cmd** to see what would change.
3. Double-click **2 - UPDATE.cmd**. It finds the live backend (and its PM2 process) by itself, lists the changes and asks before it uses new settings.
4. Wait for **ALL DONE - the server is updated and running**.

On Linux or macOS, open a terminal in the update folder and run `node update.js --dry-run` (preview), `node update.js` (update) or `node update.js --rollback latest` (undo). `HOW-TO-UPDATE.txt` in the folder explains every step, question and message.

| The updater does | It never touches |
|---|---|
| Copies only the program files that changed | `.env` (your secrets) |
| Adds your settings files only if they are missing, and asks before replacing the main settings file | The database and its backups |
| Backs up every file it replaces, plus a copy of the database | `node_modules` and keys |
| Runs `npm install` when needed, restarts with PM2, checks the server and runs the self-test | Your own plugins and jobs (game), example pages you changed (business) |
| Puts the old version back by itself if the server doesn't come up | |

Undo the last update with **3 - UNDO LAST UPDATE.cmd**; the newest 10 updates can be undone. Not using PM2? Stop the server first, run the update, then start it again.

## 14. Troubleshooting

| Problem | What to do |
|---|---|
| The generator says you don't have a licence | Buy one, or claim your free one if you bought on the Unity Asset Store, both on your account page. Bought a moment ago? Refresh the page. |
| A Download button is greyed out | The Your download panel lists the problems to fix first (in red). |
| "Node.js was not found" | Install Node.js 22 or newer (LTS) and try again. |
| "Could not locate the bindings file" (better-sqlite3) | Run `npm rebuild better-sqlite3` in the backend folder (`start-local` does this for you). |
| "Upgrade Required" (HTTP 426) | The game's version is lower than the minimum client version. Update the game, or lower it in both `backend-config.json` and `.env`. |
| "Sign in first" / 403 | The request had no valid player session. Call SignIn before other requests. |
| "Permission denied by access rules" | Run `node verify-backend.js --test …` to see which rule decided. |
| "Insecure connection not allowed" | Unity 2022+ blocks plain http://. Allow HTTP in development builds while testing; use HTTPS for your released game. |
| WebGL requests fail with a CORS error | Add your game's web address to `CORS_ORIGINS` in `.env` (or leave it empty), restart, and use https on both sides. |
| Changes not showing on a live server | Restart it (`pm2 restart <name>`). |
| A match was not counted | Check the upload answer's `sr` value (chapter 8). |
| Shop orders stay "Awaiting payment" | Check `STRIPE_WEBHOOK_SECRET` and the webhook address in Stripe; testing on your own PC needs the Stripe CLI (`stripe listen`). |
| No emails arrive | Check the SMTP settings in `.env` and restart; run `npm run self-test` (business) to see what is missing. |
| Omni Cluster will not start my servers | See **If something goes wrong on Windows** or **If something goes wrong on Linux** in chapter 11. |
| Unreal: the plugin "could not be compiled" | Install Visual Studio with the **Game development with C++** workload (Visual Studio Installer), then open the project again. |
| Unreal: Result Status 0 | No connection: the server is not running, the Url in Project Settings > Plugins is wrong, or (on a phone) it is not HTTPS. |
| Something looks wrong after an update | Run the self-test, or undo the update with 3 - UNDO LAST UPDATE.cmd. |

## 15. Getting Help

- **Help page**: backendgen.dev/help answers the common questions.
- **How it works**: backendgen.dev/how-it-works is a walkthrough with screenshots.
- **Email**: hello@backendgen.dev. Write from your account's email address and tell us what you were doing, what you expected and what happened (a copy of the error text or the self-test output helps).

Legal: Terms · Refund policy · Privacy policy. BackendGen.dev is run by Michael Rowley, a sole trader in the United Kingdom (ICO registration ZC286075).

Unity, Unreal, Godot, Steam, Epic Games, Google Play, Apple, Game Center, Discord, Stripe, PayPal and other names are trademarks of their respective owners. BackendGen.dev is not affiliated with or endorsed by them.