



Modernize – deployment and operations guide

English translation of `devops-telepitesi-es-uzemeltetesi-utmutato.md`.

Current application version: 1.1.1.

This short runbook describes a new client's go-live, releases, and day-to-day operations. The command-level reference is `scripts/deploy-client.md`; the full variable sample is `.env.example`.

Architecture and responsibility

Every client gets a separate Vercel project and a separate Supabase project/database. There is no shared `tenant_id`; isolation is at the infrastructure level. In one deployment, one `StorefrontConfig` and one `Provider` record are the base configuration.

Piece	Role
Vercel	Next.js application, environment secrets, domain, and cron
Supabase PostgreSQL	Business data, auth-adapter tables, outbox, and rate limit
Supabase Storage	Public media and private customer attachments
Better Auth + Google/Facebook	Session and sign-in
Stripe	Hosted Checkout, deposit/full payment, and refund
Resend / Twilio	Email and optional SMS
Google Calendar	One-way booking sync
Sentry	Optional error tracking and source-map upload

Privacy notice and provider list

The application's privacy page shows, in HU and EN, the provider list for Vercel, Supabase, Stripe, Resend, Twilio, Google, and Meta. Online payment, SMS, transactional email, Google Calendar, and Facebook sign-in apply only when the matching configuration is present. The current wording for purpose, role, and international transfer is a visible placeholder; before go-live it must be replaced with bilingual text approved by a lawyer. If either Sentry DSN is active, Sentry must also be added to the public bilingual provider list; if the deployment does not use it, the Sentry steps can be skipped.

Before go-live, name an owner for the client, Vercel, Supabase, domain, Google, Stripe, and notification accounts. Do not put a secret in Git, chat, a client configuration JSON, or a log; load it only from an approved secret store.

Prerequisites

- Node.js `^22.12.0` or `≥ 24.0.0`, npm, and Git.
- A Vercel project; the default region is `fra1` unless the database location justifies something else.
- One dedicated Supabase project and two PostgreSQL connections: a transaction pooler for the application, and a direct/session pooler for migrations.



- Your own domain and DNS access.
- Provider accounts created for the integrations you need.

There is no local database stack in the project: development and checks also use the configured PostgreSQL/Supabase database. Do not run a seed, a reset, or tests against an unknown production database.

Supabase

1. Create a client-specific project in a region close to the application.
2. The transaction pooler URL is `DATABASE_URL` ; the direct or session pooler URL that can run migrations is `DIRECT_URL` .
3. Create the `storefront` bucket as **public**, and the `booking-intake` and `data-exports` buckets as **private**. `data-exports` stores the generated GDPR data-export ZIP packages, separate from customer-uploaded intake files. The server-side service-role key manages all three, and on the first upload it also creates a missing private bucket automatically.
4. Set backup/PITR to match the provider plan, then try a restore in an isolated environment before go-live.

Public logo, banner, gallery, and service images are at most 5 MB; a private field file uploaded by the customer is at most 10 MB. The service-role key must never go into a `NEXT_PUBLIC_*` variable or into browser code.

External integrations

Google OAuth and Calendar

Add both production redirect URIs to one Google Cloud OAuth web client:

```
https://booking.example.com/api/auth/callback/google
https://booking.example.com/api/google/calendar/callback
```

`GOOGLE_CLIENT_ID` and `GOOGLE_CLIENT_SECRET` are required in the application. After go-live the admin separately connects their account and selects a writable calendar. Sync works only from the application toward Google.

Stripe

Set a live API key and a webhook to this URL:

```
https://booking.example.com/api/stripe/webhook
```

The webhook secret is that endpoint's `whsec_...` value. The system uses hosted Checkout, so a publishable Stripe key is not required. Before switching to live, separately test a successful payment, the 31-minute expiry, and a refund. A payment that arrives after the hold expires confirms the booking only if capacity is still free; otherwise the booking expires and an automatic refund starts, which the admin refund list and the five-minute retry follow. The admin can also restart a



permanently failed (`FAILED`) refund by hand on the Refunds page; the row shows a Stripe-classified failure reason, the attempt count, and a direct Stripe Dashboard link to the PaymentIntent (the URL differs for test and live mode).

Notifications and monitoring

- Resend: a verified domain/sender, `RESEND_API_KEY`, and `EMAIL_FROM`; `deploy:build` blocks if either is missing or the sender format is invalid.
- The destination for admin booking notifications is the separate `StorefrontConfig.bookingNotificationEmail` setting. The public `contactEmail` is not an operational mailbox; in admin Settings, save the real inbox, then send a test email.
- Twilio: SMS works only if `FEATURE_SMS` is not turned off, all three `TWILIO_*` variables are present, and the admin enables it. A partial configuration is invalid.
- Facebook: optional; `FEATURE_FACEBOOK_LOGIN` and both `FACEBOOK_*` variables are required.
- Sentry: the two DSNs are optional; source-map upload also needs the org, project, and auth token.

Environment variables

Group	Variables	Note
Database	<code>DATABASE_URL</code> , <code>DIRECT_URL</code>	Required; pooled runtime and migration connection
Auth	<code>BETTER_AUTH_SECRET</code> , <code>BETTER_AUTH_URL</code> , <code>GOOGLE_CLIENT_ID</code> , <code>GOOGLE_CLIENT_SECRET</code>	In production the app origin is explicitly required; the secret is at least 32 characters
Base settings	<code>DEFAULT_LOCALE</code> , <code>ADMIN_EMAILS</code>	Default route locale; comma-separated admin allowlist
Cron	<code>CRON_SECRET</code>	Required in production; at least 32 characters recommended
Storage	<code>NEXT_PUBLIC_SUPABASE_URL</code> , <code>SUPABASE_SERVICE_ROLE_KEY</code>	Required for media/file features
SEO	<code>NEXT_PUBLIC_SITE_URL</code> , <code>SEO_CONFIG_PATH</code>	The origin is optional; only the developer <code>db:seed</code> reads the file, the production client seed does not
Stripe	<code>STRIPE_SECRET_KEY</code> , <code>STRIPE_WEBHOOK_SECRET</code>	Both are required for online payment/refund
Email	<code>RESEND_API_KEY</code> , <code>EMAIL_FROM</code>	Both are required to send
Email test	<code>EMAIL_RESERVED_RECIPIENT_POLICY</code>	Default <code>deliver</code> ; set <code>suppress</code> only on a demo/QA deployment so RFC-reserved recipients (<code>example.com</code> , <code>.test</code> , <code>.invalid</code> , <code>localhost</code> , and similar) are not sent to Resend and stay <code>SUPPRESSED</code>
SMS	<code>TWILIO_ACCOUNT_SID</code> , <code>TWILIO_AUTH_TOKEN</code> , <code>TWILIO_FROM_NUMBER</code>	All three together, or none



Group	Variables	Note
Optional auth	FACEBOOK_CLIENT_ID , FACEBOOK_CLIENT_SECRET	Together, and only if FEATURE_FACEBOOK_LOGIN is not turned off
Feature flags	FEATURE_ONLINE_PAYMENT , FEATURE_SMS , FEATURE_EMAIL_NOTIFICATIONS , FEATURE_GOOGLE_CALENDAR , FEATURE_FACEBOOK_LOGIN , FEATURE_GOOGLE_MAPS_EMBED , FEATURE_WAITLIST , FEATURE_PROMO_CODES , FEATURE_ON_PREMISE_APPROVAL	Optional true / false . When empty, the default below applies. There is no NEXT_PUBLIC_FEATURE_*
Smoke	PLAYWRIGHT_BASE_URL , SMOKE_ADMIN_EMAIL , SMOKE_ADMIN_PASSWORD	The two credentials together; only for a temporary check
Sentry	SENTRY_DSN , NEXT_PUBLIC_SENTRY_DSN , SENTRY_ORG , SENTRY_PROJECT , SENTRY_AUTH_TOKEN	Optional

Do not set `NODE_ENV` in an env file. Put every production variable in the Vercel Production scope; use a separate database and test keys for Preview. Except for `NEXT_PUBLIC_*` , every value is a server secret. Keep the `DEFAULT_LOCALE` value that drives routing aligned with the client seed `locale` field. `PLAYWRIGHT_USE_PRODUCTION_SERVER` and `NEXT_DIST_DIR` are internal rehearsal variables; a normal client deployment does not need them set by hand.

Feature flags

The `FEATURE_*` variables are the SKU layers of a single deployment. A feature is live when the flag (or its default), the vendor keys, and — where one exists — the admin database setting all allow it. Do not put a `NEXT_PUBLIC_FEATURE_*` variable in the browser.

Flag	Default when empty
FEATURE_ONLINE_PAYMENT	on, if both <code>STRIPE_SECRET_KEY</code> and <code>STRIPE_WEBHOOK_SECRET</code> exist
FEATURE_SMS	on, if the three <code>TWILIO_*</code> values are present together
FEATURE_FACEBOOK_LOGIN	on, if both <code>FACEBOOK_*</code> values exist
FEATURE_EMAIL_NOTIFICATIONS , FEATURE_GOOGLE_CALENDAR , FEATURE_GOOGLE_MAPS_EMBED , FEATURE_WAITLIST , FEATURE_PROMO_CODES , FEATURE_ON_PREMISE_APPROVAL	on

With `false` , the matching checkout, menu item, notification channel, or sync stops. A saved admin setting remains, and applies again when the flag is restored. Account-lifecycle emails (verification, temporary password, data export) still go out when `FEATURE_EMAIL_NOTIFICATIONS=false` .

First client deployment

- Copy `scripts/new-client.example.json` to the ignored name `scripts/new-client.json` , then set the business name, IANA timezone, `hu|en` language, ISO 4217 currency, and owner email.
- The same email must be on the `ADMIN_EMAILS` allowlist. It receives the admin role on the first sign-in.



3. On a trusted workstation, with the client's variables:

```
`` bash npm ci npm run db:deploy npm run seed:new-client -- scripts/new-client.json npm run db:check npm run build ``
```

The production seed is idempotent; it creates or updates the singletons, but it does not manufacture a service catalog. `db:seed` is a CI/development fixture, and `db:seed:demo` belongs only in a staging/demo environment.

4. After the first admin sign-in, upload the four client-specific legal documents on `/hu/admin/legal` or `/en/admin/legal` (Hungarian and English privacy notice, and terms). Saving publishes the selected document immediately and creates a revision history; empty content shows a “not configured” fallback on the public page. The legal Markdown must be approved by a lawyer, and it must stay consistent with the Google and optional Facebook sign-in provider notice the application displays. The optional `seo` JSON must still be set in a separate operator process if the SEO fallback built from brand data is not enough.
5. The new client configuration's required `impressum` block contains the controller/legal entity's name and legal form, registered office, registering authority and company registration number, tax number, representative, legal contact email, and the hosting provider's name, address, email, and website. The EU VAT number and phone numbers are optional. `seed:new-client` stops on an incomplete block; the development seed deliberately does not set Impressum identifiers. Before go-live, check the `{/locale}/impressum` page in both languages.
6. Connect the Git repository to the Vercel project. The `vercel.json` build command is `npm run deploy:build`, which migrates, checks the Impressum identifiers, then builds; running it again is safe, because Prisma applies only the missing migrations. `db:seed:demo` fills a staging/demo environment with complete but explicitly fictional QA data; that data cannot be used as a live legal identifier.
7. Attach the domain, check TLS, then update the app origin, OAuth redirects, and the Stripe webhook to the final hostname.
8. Sign in as the owner, set branding, media, the map, contact, language, currency, timezone, business rules, the catalog, opening hours, and blackouts; then connect Google Calendar.

Cron and production checks

Vercel sends `CRON_SECRET` as a Bearer token. The schedule in `vercel.json`:

Frequency	Job
5 minutes	Payment-hold expiry (<code>expire-holds</code>)
5 minutes	Waitlist promotion (<code>advance-waitlist</code>)
5 minutes	Notification, Google Calendar, and GDPR data-export processing (<code>dispatch-notifications</code>)
5 minutes	Retry of automatic Stripe refunds (<code>retry-automatic-refunds</code>)
15 minutes	Reminder scheduling
Daily at 03:00 UTC	Deletion of old rate-limit counters (<code>prune-rate-limits</code>)



Frequency	Job
Daily at 04:00 UTC	GDPR retention: expired exports, 90-day outbox and StripeEvent, 24-month booking PII anonymization (<code>retention-cleanup</code>)

The endpoints return `401` for a bad token and `500` for a missing server secret. In the Vercel cron and function logs, check for `2xx` responses and the processing counters. The application also records every cron start and successful finish in the database. The admin **Operations health** page (`/{{locale}}/admin/ops`) flags crons that are stale or have never finished successfully, plus stalled notifications, paid but unconfirmed bookings, and refund errors. The view is a diagnostic aid; Vercel, Stripe, and provider logs remain the source of external system state.

Before a release, on a clean temporary schema:

```
npm run verify:deployment
```

This migrates, seeds twice, checks drift, runs a production build and the `@smoke` test, then cleans up. **A release is closed by the full Playwright suite** (not only `@smoke`): the weekday scheduled CI run, a manual `workflow_dispatch` of the `CI` workflow, or:

```
npm run verify:deployment:full
```

A pull request opened against `main` also runs the `@critical` browser paths alongside `@smoke` (`npm run test:e2e:pr`: booking management, customer profile, refund, waitlist, catalog editing, analytics, responsive admin). This PR gate does not replace the full release suite. Details: `README.md`, “CI and release gates”.

For a production smoke test, use a temporary admin credential that has been added to the allowlist:

```
PLAYWRIGHT_BASE_URL=https://booking.example.com \  
SMOKE_ADMIN_EMAIL=deployment-smoke@example.com \  
SMOKE_ADMIN_PASSWORD=<generated-password> \  
npm run test:e2e:smoke
```

The test creates a public on-premise booking, signs in as admin and checks the booking, then deletes its own fixtures. Immediately afterwards, remove the temporary email from the allowlist and rotate or delete the password.

Also check by hand, in both languages, the landing page, catalog, legal pages, on-premise and Stripe booking, the email, the management link, the portal, the admin list, a refund, the waitlist, the crons, and a Google Calendar event. Look at the public pages and the important admin pages on mobile as well.

Release and migrating several clients

Before a release, `typecheck`, `lint`, `test`, the **full** `test:e2e` (scheduled/ `workflow_dispatch` CI or `verify:deployment:full`), `build`, `db:validate`, a clean-schema check, and the database backup



must be green. The `@smoke` + `@critical` gate of a PR merged to `main` is not enough on its own. Do not edit a migration afterwards, and do not use `prisma migrate dev` in production.

To prevent referrer leakage of tokenized booking and Stripe return URLs, the application sends a `Referrer-Policy: no-referrer` header on every route, together with `X-Content-Type-Options: nosniff`, `X-Frame-Options: DENY`, and a Content-Security-Policy that forbids framing. After deployment, check the booking, management, and public routes as well:

```
curl -sSI 'https://booking.example.com/en/booking/<id>?token=<token>' \
  | grep -iE '^(referrer-policy|x-content-type-options|x-frame-options|content-security-policy):'
curl -sSI 'https://booking.example.com/en/manage/<token>' \
  | grep -iE '^(referrer-policy|x-content-type-options|x-frame-options|content-security-policy):'
```

Expected values: `no-referrer`, `nosniff`, `DENY`, and a CSP containing `frame-ancestors 'none'`. Sentry error and transaction events mask booking/management tokens and sensitive query parameters the same way before they are sent; the named regression tests in `tests/unit/sentry-scrub.test.ts` check that.

For several existing clients, copy `scripts/clients.example.json` to the ignored name `scripts/clients.json`, then:

```
npm run migrate:all -- scripts/clients.json
```

The runner proceeds in order, does not log URLs, and stops at the first error. It can be run again after a fix; Prisma skips migrations that are already applied. Never commit the production manifest. After every client, at least the smoke check should run.

Rollback and incidents

- **Application code:** use the previous known-good Vercel deployment. If the new database schema is additive and backward compatible, rolling back the application alone can be enough.
- **Database:** do not edit an applied migration and do not “roll it back” by deleting it by hand. Prepare a forward-fix migration that has been checked in advance; if data is lost, use the approved backup/PITR procedure.
- **Secret leak:** revoke and rotate the affected key, update it in Vercel, redeploy, check the provider audit logs, and document the time window and the impact.
- **Payment or notification failure:** do not replay a webhook or a refund by hand without checking state. Inspect the Stripe event, the idempotency record, and the `NotificationLog` outbox state. Also check the five-minute retry cron for automatic refunds.
- **Booking-ownership mismatch:** after a support investigation, `npm run repair:booking-ownership` can be used; before a production run, always check the target database and take a backup.
- **Stalled GDPR data export:** `npm run repair:data-exports` safely sets failed `DataExportJob` rows back to `PENDING`, brings forward exports that are still waiting and have not expired, and re-



queues failed “export ready” notifications if the related export is still `READY` and has not expired. Running it again is safe.

- **Wrong delivery of fixture recipients in a demo/QA environment:** if `EMAIL_RESERVED_RECIPIENT_POLICY` is later switched to `suppress`, `npm run repair:suppressed-notifications -- --apply` retrospectively sets still-pending or failed email outbox rows addressed to RFC-reserved recipients to `SUPPRESSED`; without `--apply` it only does a dry run. The script stops with an error in production (under the `deliver` policy).

The incident log contains the times, the affected client, the version, the symptom, the impact, the steps taken, the recovery, and the follow-up task — without secrets and without full personal data.

Regular operations

Daily: Vercel/Sentry errors, crons, Stripe webhooks, failed outbox and Google syncs, database and Storage capacity.

Weekly: the refund and waitlist queues, repeating rate-limit or delivery errors, expiring integration states, backup success.

Monthly/quarterly: dependency and security updates, permissions and the admin allowlist, secret rotation, a restore drill, domain/TLS/sender-domain state, and a cookie audit. `retention-cleanup` performs data retention: GDPR export ZIPs and failed exports for 7 days, delivered/failed outbox and StripeEvent for 90 days, and personal data of ended bookings anonymized after 24 months (the row remains for analytics; the account is not deleted). `AdminPiiAccessLog` (Finding 24) is not part of this sweep: it is admin PII-access metadata, not customer PII, and keeping it is an operator decision. These are operational defaults, not legal advice. The cron can be called again at any time to drain a backlog.

Audience / age (Finding 22)

The booking information society service (website, checkout, account) is **not** aimed at users under 16. The booker or account holder is at least 16. There is no Article 8 age gate (date of birth, parental consent, blocked checkout): audience exclusion is the documented branch of the Finding 22 acceptance criterion.

A children's haircut or a paediatric visit in the demo catalog means an **adult** can book that service, not that the website is offered to minors. If a deployment deliberately targeted independent bookers under 16, that would need a separate age gate outside the product's current scope — this release does not support that audience.

This is an operational rule, not legal advice.

Admin PII access log (Finding 24)

Admin views of bookings, guests, the waitlist, the calendar, refunds, and the dashboard, and GDPR erasure on Settings, write an `AdminPiiAccessLog` row (actor `User.id`, action, resource type, opaque/hashed identifier, `ALLOWED / FORBIDDEN`, UTC `createdAt`). Raw customer identifiers are not



stored. Query it with SQL, for example `SELECT * FROM "AdminPiiAccessLog" ORDER BY "createdAt" DESC LIMIT 100`. There is no in-application delete. GDPR 1.1.0 #346.

GDPR erasure at the processors (Finding 14)

The current subprocessor inventory and the internal DPA package (Finding 25 / GDPR 1.1.0 #346): `app-documents/gdpr-internal-dpa-subprocessor-package.md`.

The application's primary erasure (`eraseCustomerData` / `eraseOwnAccount`) covers its own database, private storage, and the Better Auth account. Processor copies are **not** deleted by an automatic Stripe or Sentry API; the operator runs the steps below, for the same email address, after the Settings GDPR form or after profile account deletion. An admin account cannot be deleted.

For a future booking, the name, email, and phone needed for fulfilment stay on the booking row until `endAt`; private files and the waitlist are deleted immediately. If the guest also has the booking cancelled, cancel it first on the admin or management interface, then run erasure again.

Stripe

Checkout sends `customer_email` and `metadata.bookingId`; `Booking` stores only `stripeCheckoutSessionId` / `stripePaymentIntentId` identifiers, not a Stripe Customer identifier. The application does not call a Stripe deletion API.

1. In the Stripe Dashboard, search for the customer's email address.
2. List the Checkout Sessions, PaymentIntents, Charges, and refunds.
3. If a Stripe Customer object was also created, delete or anonymize it from the Dashboard.
4. Do not destroy a payment record that must be kept for accounting, chargeback, or tax retention.

Google Calendar

Sync is one-way; the event title is `{service} – {customer name}`.

1. For an ended booking (`endAt` is already in the past), erasure re-queues sync, and the title is corrected to `{service} – [REDACTED]`.
2. On a future event the name stays until the booking ends (a repeated erasure then corrects it) or until it is cancelled (sync deletes the event).
3. If the guest also wants the calendar entry removed, cancel the booking, wait for a successful Google sync, then run erasure.

Sentry

Sentry is an optional operations tool. If the server or browser DSN is active, it must appear on the public privacy provider list and in the internal subprocessor inventory. `beforeSend` masks booking/management tokens and sensitive query values; it does not mask email addresses.

1. If `SENTRY_DSN` / `NEXT_PUBLIC_SENTRY_DSN` is not set, skip this step.
2. If it is on, search Issues for the email address, and delete the personal data / events in the Sentry interface.
3. Check that the active Sentry appears in the lawyer-approved public privacy text; when it is turned off, update the deployment's actual provider list to match.



Further processors

- **Resend / Twilio:** search the provider logs by the recipient email address or phone number, and delete or wait out the retention period according to the retention rule.
- **Vercel:** runtime logs expire on their own over time; do not copy full personal data into a ticket or a chat.
- **Supabase:** erasure affects the production database; backup/PITR can contain the earlier state until the end of the retention window.
- **Google / Meta OAuth:** Better Auth `Account` rows disappear together with the user deletion; we do not manage the Google or Facebook account itself.

Handover checklist

- Final domain, TLS, env, and an isolated database are in order.
- Migration, production seed, legal content, and the client catalog are done.
- Storage, Stripe, email, optional SMS, and Google Calendar have been checked.
- All seven crons run successfully.
- Clean-schema rehearsal, production smoke, and manual E2E succeeded.
- Owner admin access works; temporary access has been removed.
- Backup/restore, incident, release, and secret-rotation owners are documented.
- The internal DPA/subprocessor package (`gdpr-internal-dpa-subprocessor-package.md`) is up to date; signed DPAs are in the operator's archive.