

Technical and operator manual

Future Freedom Planner v20.12 · 2 October 2026

This release is an owner preview with payments disabled. It contains the commercial integration and legal/privacy drafts, but it is not a completed production deployment, legal opinion, FCA clearance or actuarial certification.

Architecture

Component	Responsibility
Next.js 16 / React 19	Inputs, charts, cashflow controls, pricing and access interface
FastAPI	Input validation, calculations, reports and paid-feature enforcement
Python financial engine	Annual nominal cashflows, tax, strategies, target solving and simulation
SQLite	Minimal access-pass records and processed webhook IDs
Stripe hosted Checkout	Card payments and payment identity; no pension figures sent

Planning figures stay in React memory. API calculations and reports are processed in memory and returned; they are not written to a customer-plan database or report file store. Save/Load uses a local JSON file with main inputs and goal settings. Files are checked locally before replacement; results and access keys are excluded. Downloads remain on the customer's device.

Source map

- frontend/app/CashflowTable.tsx: expandable cashflow and whole-plan zero filters.
- frontend/app/Upgrade.tsx, LegalInfo.tsx: tiers, access, privacy and terms.
- backend/app/services/engine.py, pension_target.py: financial calculations.
- backend/app/services/privacy.py: input allowlist and controlled pension labels.
- backend/app/services/billing.py: Stripe verification, access and refunds.
- backend/billing_admin.py: private backup, restore, key recovery and preflight tools.
- backend/app/services/purchase_mail.py: durable confirmation via encrypted SMTP.
- backend/billing_privacy_qa.py: real API/SQLite tests with simulated Stripe transport.

The ZIP excludes installed dependencies, real environment files, access databases and credentials. The shipped environment examples are placeholders only.

Residence scope and saved plans

Supported launch scope

The commercial release is scoped to people resident in England, Wales or Northern Ireland throughout their plan, using UK pension and savings assumptions and GBP spending. Foreign public pension sources remain manual gross GBP inputs. Scotland, overseas residence and changing tax systems are excluded from net calculations and new paid purchases. This is a scope control, not a legal determination of residence or automated geographic blocking.

`tax_region` is a controlled option: `unconfirmed`, `england_wales_ni`, `scotland`, `outside_uk` or `changing_or_unsure`. Omission defaults to `unconfirmed`. The engine rejects Net unless the supported region is explicit, including goal calculations, analyses and reports. Gross uses UK-account assumptions and carries a visible limitation. Unknown options and free text are rejected.

The source-country dropdown does not alter pension ages, forecasts or annual increases. The model does not calculate foreign entitlements, local tax, treaty relief or exchange-rate changes. The interface makes this distinction before purchase and reports carry the declared scope.

Saved plan schema 4 adds `transition_mode`, `transition_age_basis` and `transition_end_age` to the pension and optional ISA/GIA overrides. Schemas 1-3 migrate to legacy transition mode, preserving explicit rates, dates and balances. Schema 3 includes residence, optional ISA/GIA investment overrides and a separate Cash ISA balance, deposit and rate. Schema 2 retains residence and receives zero Cash ISA savings with inherited investment settings. Schema 1 is migrated to `unconfirmed` residence while keeping the numbers and goal settings. Unknown fields, oversized files and invalid choices remain rejected. API clients sending a Net request must now supply explicit supported residence, including if their main plan was Gross and they request a Net pension target.

Checkout requires an unticked-by-default scope declaration as well as the existing terms and immediate-access consent. Server checks run before Stripe or order creation. The immutable contract records the accepted scope, without storing customer addresses or plan inputs. This declaration does not replace business tax-location evidence or legal review.

Windows installation and local use

Use a current patched **Python 3.12** and a maintained Node version satisfying **Node 20.9 or later**. Node 22 is a suitable choice. The verified Python environment is 3.12; much older runtimes are not supported by this dependency set.

Extract the ZIP. Open its **backend** folder in File Explorer. Type cmd into the address bar and press Enter. Run one command at a time:

```
py -3.12 -m venv .venv
.venv\Scripts\python -m pip install --upgrade pip
.venv\Scripts\python -m pip install -r requirements.txt
.venv\Scripts\python run_server.py --reload
```

Open a second Command Prompt in **frontend**:

```
npm ci
npm run dev
```

Open <http://localhost:3000>. Keep both terminal windows open. On later runs use only the final backend command and `npm run dev`. Stop with Ctrl+C in each window.

Configuration files

Local preview works without copying environment files. The default API is <http://127.0.0.1:8000/api/v1>. `/health` should return status ok.

To customise settings, copy `backend/.env.example` to `.env` and `frontend/.env.local.example` to `.env.local` in their respective folders. The examples are included in this ZIP. A missing old screenshot path is unrelated to application startup.

Do not blindly run `npm audit fix --force`; major upgrades need review and regression checks. Install from the supplied lock file with `npm ci`.

Production builds and operating modes

Linux or macOS backend

```
cd backend
python3.12 -m venv .venv
.venv/bin/python -m pip install --upgrade pip
.venv/bin/python -m pip install -r requirements-lock.txt
.venv/bin/python run_server.py
```

The requirements lock records the verified production dependencies. Platform-specific wheel availability still needs checking on the deployment host.

Frontend

```
cd frontend
npm ci
npm run build
npm start
```

Set `PLANNER_API_URL` to the hosted API URL ending in `/api/v1` **before building**. Rebuild if it changes; the origin is also used in the Content Security Policy.

Mode	Behaviour
<code>BILLING_MODE=preview</code>	All owner features; checkout disabled
<code>BILLING_MODE=test</code>	Tier enforcement; Stripe test keys/payments
<code>BILLING_MODE=live</code>	Tier enforcement; live checkout only when configured

`APP_ENV=production` requires live billing mode and an HTTPS app URL. A test-mode staging service must be access-controlled. Do not expose the local owner preview publicly.

Deploy behind HTTPS, restrict CORS to the actual frontend, and keep SQLite on private durable storage. Multiple replicas need a shared entitlement database and distributed rate limiting. Independent SQLite copies and ephemeral serverless disks are unsuitable for paid access.

Financial calculation contract

Periods run from one birthday to the next. Horizon 100 ends at the 100th birthday; the last cashflow is age 99-100. Contributions stop at retirement. Public-pension entitlement is entered, not prefilled.

Balances and spending are calculated in future pounds. A real value is the nominal value divided by $(1 + \text{inflation})^{\text{years}}$. Deflation is a display/comparison conversion, not another deduction from assets.

Returns are total nominal returns, including reinvested dividends and interest. Separate cash interest is distinct from cash within the investment allocation. Do not add another dividend yield to the assumptions.

Weighted portfolio gross return is the sum of asset weights times assumed returns. Net annual rate is $(1 + \text{gross return}) \times (1 - \text{fee}) - 1$. At 6.5% gross return and a 0.5% fee, net return is 5.9675%. See MODEL_ASSUMPTIONS.md for annual withdrawal ordering and period-end contribution instalments.

Net mode solves the pension withdrawal needed after modelled tax. Gross mode is before tax. Phased tax-free withdrawals use the remaining lump-sum allowance. The interface converts displayed fee 0.5 to API decimal 0.005 once.

The tax scope is one-person England/Wales/Northern Ireland pension-income tax. Scottish and overseas tax, capital gains, dividends, cash-interest tax, contribution allowance charges and household coordination are excluded. Future tax-band growth is an explicit assumption, not a legislative forecast.

Bridge and spending

An unfunded bridge before pension withdrawals remains a separate savings requirement; it never becomes pension depletion or debt.

Inflation spending rises with prices. Fixed spending inflates the target to income start, then holds pounds flat. In the main plan, that date is the latest current/retirement/pension withdrawal age; in Pension target it is the goal start. Contribution and pension indexation remain independent. Age-stage multipliers still apply; extra care remains indexed.

Pension targets, strategy comparisons and limits

The inverse solver searches for the pension required at the goal start age, using the same annual engine, tax and public/defined benefit income. Other assets may be included or excluded after the bridge.

Gold compares six starting mixes with four retirement choices: keep the starting mix, Conservative 25/65/10, Balanced 50/45/5 and Growth 70/25/5 (shares/bonds/cash). Previous Lifestyle 45/50/5 appears as a fifth choice only if already selected. Transition mode and date are independent. A gradual transition linearly blends each annual allocation towards the target at its chosen date; switch mode changes at that date. The inverse cache includes every annual mix during drawdown for every invested account, plus account fees and residual balances, so two plans with different ongoing transitions cannot share an incorrect target.

Targets assume steady returns, zero legacy and no adverse-market reserve. Simulations use assumed lognormal annual returns and correlations; they are not historical backtests. Higher/lower chart paths are deterministic return sensitivities. Neither is a guarantee or personalised investment recommendation.

Preserved previous-assumption regression example

This reference deliberately retains the previous 5.5% share return and legacy switch at retirement, rather than using new-plan defaults.

Age 44, retire 53, draw 57, horizon 100; £245,000 current pension, £1,000 monthly gross contributions rising with inflation, £2,500 net monthly target, 2.5% inflation, 0.5% fees, High growth then Balanced income, no tax-free cash, fixed tax bands, no other savings and £12,547.60 annual public pension today from 68.

Result	Verified reference
Projected pension at 57	£629,609.21
Required: inflation-linked spending	£1,289,916.08
Required: fixed pounds	£612,771.01

The separate goal does not silently update the main plan or its PDF reports. The goal CSV records its own settings and annual calculation. The main-plan extra care budget is not added to the separate goal.

Cashflow filtering is display-only. Zero detection scans all years, not just visible key ages. Five balance totals remain visible. Key ages are individual age-years, not multi-year totals.

Account-specific investments

The main strategy, `post_retirement_strategy`, `fees`, `glide_path_years`, `transition_mode`, `transition_age_basis` and `transition_end_age` belong to the pension. `account_investments` accepts optional `isa` and `gia` objects with these seven fields. Missing fields inherit pension inputs dynamically. When the UI creates custom settings, it copies the mix, fee and transition mode, then defaults the account date basis to stop-work age for bridge planning. Following the pension again removes overrides.

Cash ISA inputs are `cash_isa_balance`, `monthly_cash_isa_contribution` and `cash_isa_return`. The API uses decimal rates, including nested fees; explicit `rate_unit=percent` converts them once. Frontend saved JSON uses display percentages. No personal account names or unrestricted metadata are accepted.

Each account has its own annual mix, fee, growth and crash attribution. Equity/bond shocks use that account's allocation; whole-portfolio shocks affect the three investment accounts only. Cash ISA and ordinary cash have no portfolio fee or explicit market crash. The cash asset's volatility applies to both interest-rate paths, with shared shocks; Cash ISA has its own compound mean.

Different investment mixes use common correlated asset normal shocks, combined by each account's volatility weights, before the compound-calibrated lognormal transformation. Identical mixes retain the earlier two-normal simulation path so seeded legacy results stay unchanged. Annual shocks remain independent over time. This is not a historical market calibration.

Withdrawal order is cash, Cash ISA, investment ISA, GIA, then eligible pension. Bridge cash top-ups and the inverse pension target use the same account-specific rates and order. Inverse caching includes residual balances, all retirement mixes and account fees. Excluding other assets excludes all four accessible accounts. Strategy comparisons change the pension and accounts following it; custom account settings stay fixed.

`account_rates` describes each account's realised annual gross/net/real rate, fee and mix. `growth_by_pot`, `fees_by_pot`, `crash_loss_by_pot`, contributions and withdrawals include `cash_isa`. `opening_allocation` weights each account's mix by its opening balance and includes both cash accounts once. `combined_asset_mix` is zero if there are no assets. Legacy `investment_return`, `gross_investment_return`, `real_investment_return` and `asset_mix` now explicitly describe the pension; customer labels say so.

Combined ISA deposits are flagged above £20,000, with a further prompt for Cash ISA deposits above £12,000 below age 65. The engine does not cap inputs, align whole age-years to tax years, certify eligibility or implement future ISA reforms. Instrument-specific cash charges and Lifetime/Junior ISA rules are excluded. See the customer FAQ's HMRC link and review tax rules before commercial use.

Run `python account_investments_qa.py` for independent growth, timing, mixed-income tax, bridge, target, covariance, API privacy and legacy regression controls. `tools/plan-file-check.cjs` and `tools/browser-check.cjs` cover saved files and the customer journey.

Browser access and data minimisation

`browser_access.py` registers at most two browsers per original purchase, shared with its upgrade. Only the selected active browser may use Silver or Gold tools. Status polling never switches it; the customer must choose Use this browser. Each verification creates a new 256-bit random cookie; SQLite stores its SHA-256 hash only. The cookie lasts at most 30 days, capped by pass expiry. A switch increments a session version; long calculations recheck it before returning results. Existing downloaded or displayed results cannot be recalled.

New browsers need an 8-digit code, sent only to `customer_details.email` retrieved from the original Stripe Checkout session. The app does not persist that email. Challenges bind to a separate random `HttpOnly` cookie, expire after 10 minutes and allow five wrong entries. Codes are salted with the random challenge ID and hashed using PBKDF2-SHA256 (150,000 iterations). Resends have a 60-second cooldown and a five-per-hour order limit. Successful verification cancels all outstanding challenges for that order.

Registration/replacement and active-browser updates use SQLite transactions. A third browser must pass email verification before seeing the two existing registrations and choosing a replacement. No user-agent fingerprint, device name, browser history, customer name/address or plan data is stored. Access controls discourage ordinary sharing; they cannot guarantee prevention of deliberate code/cookie sharing or compromised email.

Record	Stored fields and expiry
browsers	Random ID, order reference, cookie hash, creation/verification/expiry dates. Maximum 30 days, capped by pass expiry
browser_auth	Order reference, active browser ID, session version, resend-window dates and counts. Removed on pass expiry/revocation
access_challenges	Random ID, order reference, binding/code hashes, dates, wrong-attempt count and verified flag. 10 minutes; consumed on success

Cleanup runs on database access and the scheduled operator cleanup command. Refund/dispute/expiry rejects access immediately and deletes auth records on the next database access. This is logical deletion; backups and provider retention are separate. Purchase records retain the existing expiry-plus-180-day policy. These linked identifiers are pseudonymous personal data, not an anonymous service.

Hosting and migration

The frontend must run as a Next.js Node service, not a static export. Set its server-side `PLANNER_API_URL` to the backend `/api/v1` URL. The older `NEXT_PUBLIC_API_URL` is a temporary fallback. Browser fetches always use `/api/v1` on the frontend origin. Send Stripe webhooks directly to the backend. Set backend `PUBLIC_APP_URL` and `CORS_ORIGINS` to the real HTTPS frontend origin. Do not cache the API or log Cookie, Set-Cookie, Authorization, request/response bodies or code emails. Cookies cannot be shared across different frontend hostnames.

Before updating, back up SQLite. New tables are added automatically; existing purchase periods and frozen contracts stay unchanged. Existing recovery keys become email-verified references. Confirm original paid Checkout sessions have retrievable emails. A missing or inaccessible email needs ownership-verified support, not a bypass. Refund/dispute events still require working signed Stripe webhooks.

The application limiter is process-local and sees the proxy connection address; configure production ingress limits and capacity for the expected traffic. It is not a distributed denial-of-service defence. Run one API instance with durable SQLite; shared access across replicas requires a shared database and distributed limits.

Required hosted acceptance

Test actual checkout and code delivery with your Stripe/SMTP accounts, original-tab recovery, two browser profiles, a third-browser replacement, explicit switching, expiry/refund, HTTPS cookies, deployed origins and webhook delivery. Local automated tests simulate providers; they do not prove live inbox delivery. Keep public sales disabled until hosted acceptance and the existing launch requirements are completed.

References:

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html and https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html

API access and security

Free permits basic inputs and returns only overview figures and the central projection. Cashflows, simulations and comparisons are restricted by tier. Selected assumptions and warnings remain public.

Paid tools use a random HttpOnly cookie through the same-origin proxy. The backend verifies the purchased tier and active browser, and rechecks access before returning long calculations. Recovery keys cannot unlock planner/report APIs. Do not log credentials or figures.

Route under /api/v1	Access
/billing/catalog, /billing/access	Public catalog / current browser status
/planner/run	Free overview; paid detailed result
/planner/pension-target	Plan; comparison datasets require Full
/planner/analysis, /planner/compare, /validation/run	Gold (full)
/reports/pdf	Silver: summary and Essential; Gold: all reports/appendices
/billing/checkout	Public; consent, scope and price checks
/billing/complete, /billing/purchase	Matching pending recovery reference
/billing/refund, /billing/confirmation	Verified purchase browser
/access/request-code, /access/verify	Email code bound to requesting browser
/access/browsers, /switch, /forget	Verified browser; all under /access
/billing/webhook	Signed Stripe request, direct to backend

The plan allowlist rejects personal/extra fields without echoing them. Report options accept only include_appendix and money_basis (today/future). The summary report rejects an appendix. Money basis controls the one-page report; longer reports label their own bases. Bodies are limited to 64 KiB (webhooks: 256 KiB); simulations to 2,000 paths. Process-local rate limits are 90 general requests/minute and 10 checkouts/10 minutes per salted connection-address hash. The proxy aggregates clients; add production ingress limits and capacity controls.

Responses are no-store with no-referrer, nosniff and frame denial. CSP restricts sources; inline scripts/styles remain needed. Secure, HttpOnly, SameSite=Strict cookies and POST origin checks protect access, but cannot defeat all script injection or session theft. Keep dependencies updated. No formal penetration test or security certification is claimed.

Stripe setup and payment lifecycle

Internal tier IDs remain free, plan and full. Frozen contracts keep their original names and terms.

Create three **one-time GBP prices**: 2900 pence for Silver / 90 days, 4900 pence for Gold / 365 days and 2000 pence for the Silver-to-Gold upgrade. There is no subscription or automatic renewal. The backend verifies the price and completed session's amount, currency, line item and order reference.

After changing an offer, configure matching Stripe prices. Existing orders retain their saved price, terms and duration. New offers use terms version 2026-09-28-v8.

Set STRIPE_SECRET_KEY, STRIPE_WEBHOOK_SECRET, STRIPE_PLAN_PRICE_ID, STRIPE_FULL_PRICE_ID and STRIPE_UPGRADE_PRICE_ID in private backend configuration. Staging uses test keys/prices. Live mode requires a live secret key and HTTPS app URL.

Checkout also requires SELLER_NAME, SELLER_ADDRESS, SUPPORT_EMAIL, PRIVACY_OPERATIONS_TEXT, and LEGAL_REVIEW_CONFIRMED=true and PAYMENTS_ENABLED=true. The separate payment switch defaults to false. The acknowledgement does not perform a legal review. Complete actual business, privacy and payment setup first.

Configure Stripe business details, support, privacy/terms URLs, receipts and tax treatment. The frontend terms page is /legal. Prices are advertised as the total charged in GBP; VAT and cross-border obligations need business-specific review. No automatic tax configuration is supplied.

Flow

1. Customer accepts terms and requests immediate access, with statutory cancellation rights and the current versioned service-deduction notice.
2. Backend creates a 256-bit random access key and stores its hash with a pending order.
3. Stripe receives the offer and opaque order reference, never pension figures.
4. Browser stores a pending key and opens checkout in a separate tab.
5. A verified callback or signed webhook activates the matching paid order.
6. Accepted terms and purchase reference are emailed to the checkout address. The customer returns to the original tab, requests a code and verifies that browser. Payment confirmation itself never grants a browser credential.

Register /api/v1/billing/webhook for checkout.session.completed, checkout.session.async_payment_succeeded, charge.refunded and charge.dispute.created. Verification uses raw-body HMAC and a five-minute timestamp window. Keep the server clock synchronised.

Real merchant checkout, receipts, public webhook delivery and provider refunds still require a Stripe test-mode acceptance run before launch.

Silver to Gold upgrades

`upgrades.py` links one completed Gold upgrade to an original £29 Silver order. The original price, contract, activation and expiry are never rewritten. Older £29/30-day purchases qualify while active; other historical prices do not.

Record	Purpose
<code>order_upgrades</code>	Original order ID and upgrade order ID; no customer identity
<code>entitlement_limits</code>	Database view deriving the effective tier and expiry
Frozen upgrade offer	£20 charge, original purchase reference and anchored Gold expiry

Gold ends at original Silver activation plus 365 days. It shares the original browser records, verification email and two-browser limit. Each browser still verifies for at most 30 days. An expired Silver order remains recoverable while its linked Gold access is active.

Purchase and failure handling

The upgrade route requires the active verified browser and current terms/scope. The server chooses the original order, price and expiry. One pending checkout is reused; concurrent requests share its idempotency key. A provider-confirmed expired checkout can be replaced. Valid payment is checked against the frozen offer. Delayed callbacks never add time. A payment after original revocation is refunded without granting Gold.

Refunds and recovery

Refunding the upgrade restores only the original Silver remainder. Refunding or disputing Silver revokes both and refunds the linked upgrade. Each payment has its own cancellation period (including applicable end-of-day and working-day extensions). Failed dependent refunds remain revoked with `refund_pending`; signed-webhook retries or this operator command finish them:

```
python billing_admin.py retry-upgrade-refunds
```

Monitor webhook failures and unresolved refunds. The payment provider's idempotency key prevents a duplicate refund. Confirmation retry remains available. Recovery checks both active payments and revokes existing browser credentials before returning a replacement recovery reference.

Keep the original order through the linked upgrade's access and retention period. Unresolved refunds are retained until resolved. Back up the database before updating; never restore only the original order without its upgrade/link.

New routes: `POST /billing/upgrade`, `GET /billing/upgrade/confirmation`. `POST /billing/cancellation-quote` accepts `{purchase: original}` or `{purchase: upgrade}`. `POST /billing/refund` requires that choice plus the returned `quote_token`; amounts and dates

cannot be supplied by the browser. Request bodies never accept a parent order ID, price or expiry. New purchases require all three configured price IDs before checkout opens.

Access records, refunds and privacy operations

The database stores opaque order ID, hashed key, tier, Checkout/payment-intent references, status, created/activated/expiry dates, terms version, frozen offer/terms and confirmation-sent timestamp. Events store ID and processed time only. No customer planning figures, names, emails, addresses, card data or raw webhook bodies are stored there. The contract copy contains supplier details and the accepted fixed planning scope. Checkout requires `tax_region=england_wales_ni` and `accepted_scope=true`, in addition to existing consent and the current terms version. Missing, unsupported or false declarations fail before Stripe or order creation. Changing residence in the interface resets scope consent and clears old calculations. No geolocation or identity verification is claimed. References can still be personal data when linked to Stripe.

Repeated confirmation/webhooks do not extend expiry. Full refunds and disputes revoke access, including when the event arrives before activation. Partial external refunds do not automatically revoke access. A won dispute requires controlled operator review to restore the original entitlement; no admin console is included.

No added satisfaction guarantee is offered. The statutory-aware cancellation flow has a server-generated quote, accepted policy evidence, immediate revocation and a refund-retry ledger. Self-service uses a verified browser and provider idempotency keys. Statutory claims, provider failures, disputes and lost keys can still require support. An independent product cannot eliminate these supplier duties.

Cleanup occurs on database access: pending orders after 30 days, event IDs after 30 days, expired passes after the default 180-day retention. Never-activated refunded/disputed records use creation date plus retention. Schedule daily cleanup if the service can remain idle; define separate backup expiry.

Actual data flow

Figures/results live in tab and request memory. Browser storage holds theme, purchase reference and a temporary pending checkout key. Essential HttpOnly cookies hold a random browser credential and a short-lived verification binding. Legacy local recovery keys are cleared after successful verification. Downloads stay on the customer's device. Stripe may process billing identity. The confirmation and verification email is read from Stripe and passed to the SMTP provider without being saved in the app database. Provider logs and support may contain personal data.

`run_server.py` disables Uvicorn access logs. Configure proxy/platform logs separately. Never capture bodies, auth headers, keys or report bytes in monitoring or session replay. Identify actual controller, providers, lawful bases, retention, transfer safeguards and rights contact in the deployment notice. App cleanup does not erase provider or statutory accounting records. Do not advertise zero risk or complete anonymity.

Legal and operational launch checklist

The release supplies disclosures, transparent tiers, terms acceptance, immediate-access consent, privacy wording, a cancellation template, refund controls and downloadable confirmation. These are implementation measures, not legal certification.

Before sales, obtain review for the actual business location and customer market:

1. **Supplier:** real legal/trading name, contact address and monitored support email, consistent across terms, checkout and receipts.
2. **FCA perimeter:** review outputs, strategy comparisons, action wording and marketing. Disclaimers alone do not determine whether activity is regulated. Do not imply FCA approval or protection.
3. **Consumer contracts:** confirm service/digital-content classification, pre-contract information, cancellation, statutory remedies and durable confirmation delivery. Test receipt and download availability.
4. **Privacy:** complete actual hosting/payment/support roles, retention, safeguards, rights process, ICO fee assessment, processor contracts and incident response.
5. **Tax/territory:** review total-price VAT and cross-border selling obligations. The code does not determine them.
6. **Operations:** verify real Stripe test payments, receipt, signed webhooks/retries, refund, expiry, backups, recovery, accessibility and hosting security.

Store secrets in the host's secret manager. Restrict SQLite/backups to the service account. Keep staging separate. Monitor aggregate failures without recording financial inputs or keys.

Lost-key recovery uses a private operator command after ownership verification through the original payment contact. A reference alone is insufficient. Original price, duration, features and terms are now frozen at checkout; older missing snapshots are not invented. Do not change purchased periods retrospectively.

Official sources: <https://www.gov.uk/online-and-distance-selling-for-businesses> ;
<https://handbook.fca.org.uk/handbook/perg8/perg8s41> ;
<https://ico.org.uk/for-organisations/data-protection-fee/data-protection-fee-self-assessment/> .

Reproduce checks and maintain the release

From backend, in an isolated environment:

```
python -m pip install -r requirements-dev.txt
python v19_qa.py ../audit/financial-current.json
python pension_target_qa.py ../audit/target-current.json
python spending_options_qa.py ../audit/spending-current.json
python report_qa.py report-qa-output
python billing_privacy_qa.py ../audit/billing-current.json
python launch_operations_qa.py ../audit/operations-current.json
python scope_qa.py ../audit/scope-current.json
```

Core controls reconcile balances, contributions, fees, withdrawals, tax, timing and bridge treatment. Target controls independently check inverse results and strategy combinations. Spending checks cover both inflation choices. Report checks exercise every type, appendix, edge cases, no-name fields and API downloads.

Billing/privacy checks use real FastAPI and temporary SQLite with **simulated Stripe transport**. They test tier gates, prices, consent, signatures, duplicates, refunds, expiry, retention, validation, CORS, request bounds and headers. They do not prove the real merchant integration works.

The final audit/v20_4-verification.json records exact check counts and limits. Browser checks use the production frontend and a real local API, with preview and seeded test-mode access. Screenshots show desktop and phone layouts. No live payment is taken.

Review dependency advisories regularly and rerun relevant checks after changes. Tax/pension rules and assumptions need review at each material change. No live financial data feed is included. A dated audit with no known vulnerabilities is not a permanent security guarantee.

Supporting references

- Model details: MODEL_ASSUMPTIONS.md, PENSION_TARGET_GUIDE.md and SPENDING_OPTIONS_GUIDE.md.
- Next.js support: <https://nextjs.org/support-policy>
- Stripe Checkout: <https://docs.stripe.com/api/checkout/sessions/create>
- Stripe signatures: <https://docs.stripe.com/webhooks/signature>
- ICO identifiers: <https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/personal-information-what-is-it/what-is-personal-data/what-are-identifiers-and-related-factors/>

The customer guide provides self-service help. Exceptional billing, consumer-rights and privacy issues still require a responsible supplier contact.

Hosted deployment and sales controls

The complete walkthrough is `docs/deployment.md`; the prepared Render template is `render.yaml`. It creates two paid web services only when an operator submits it. No hosting account, domain, DNS or paid resource has been created by this release.

Deploy the free service first

1. Place the extracted source in a private repository.
2. Create the Python backend with a private persistent disk at `/var/data`, one instance, and `BILLING_DB_PATH=/var/data/access.sqlite3`.
3. Use `APP_ENV=production`, `BILLING_MODE=live` and `PAYMENTS_ENABLED=false`. This keeps tier enforcement and closes sales.
4. Build the Node frontend with its actual `PLANNER_API_URL` ending `/api/v1`.
5. Set backend `PUBLIC_APP_URL` and `CORS_ORIGINS` to the actual frontend HTTPS origin. Connect the domain and verify HTTPS.

The marketing page is `/welcome`; `/` opens the calculator. The fictional sample report is `/samples/essential-sample.pdf`. The sample has no customer identity or real entitlement forecast.

Durable confirmation email

Set `RECEIPT_SMTP_HOST`, `RECEIPT_SMTP_PORT` (465), `RECEIPT_SMTP_USERNAME`, `RECEIPT_SMTP_PASSWORD` and `RECEIPT_FROM_EMAIL` privately on the backend. Use a verified sender/domain. The email includes the accepted contract as an attachment; no planning figures or access key are sent.

Checkout requires email configuration as well as Stripe, supplier/privacy details and the review acknowledgement. Test actual inbox delivery. A failure leaves paid access active and requests a retry. After successful SMTP acceptance, the app records delivery time; a crash between sending and recording can result in a duplicate email.

Keep Stripe test mode on a private, separate staging service. Test payments, receipts, signed webhooks, refunds, expiry and recovery there. Run configuration preflight before enabling live sales:

```
python billing_admin.py preflight
```

This prints boolean checks, never secrets. Presence checks are not proof of legal compliance, actual disk persistence, email delivery or payment-provider acceptance.

Backup, recovery and support tools

Read docs/operations.md before using these operator-only commands. They are not public API endpoints. Run them from backend with private paths outside the web root.

```
python billing_admin.py cleanup
python billing_admin.py backup private-backups/access-DATE.sqlite3
python billing_admin.py check-backup private-backups/access-DATE.sqlite3
```

Backup uses SQLite's consistent API and integrity check. It creates a private file and refuses to overwrite one. Encrypt the off-host copy and verify retrieval. Schedule the job in the actual hosting account; the release does not run a scheduled backup service.

For restoration, stop the backend, move the existing database and journal/WAL files into a private rollback folder, then use the restore command with --service-stopped. It refuses to overwrite an existing target. Reconcile Stripe purchases, refunds and disputes since the backup before reopening.

Recovery without a customer appointment

A customer normally verifies a browser using their purchase reference and an email code. If every copy is lost, verify ownership through the original Stripe payment contact. Use recover-key with the original Checkout reference, a private output path and --ownership-verified. The command checks paid/refund/dispute status, replaces the key and preserves expiry. The old recovery key, all browser credentials and outstanding verification challenges stop working; a replacement key still requires the original checkout email code. Deliver the new file only through the verified contact channel.

To retry a missing confirmation, use retry-confirmation with the original Checkout reference. Delivery uses the checkout email from Stripe; it cannot redirect to an unverified support address.

The full syntax and support reply templates are in docs/operations.md. The source does not send any support messages by itself.

Maintenance to configure

- Daily: uptime, payment/webhook/mail failures, support, cleanup and off-host backup.
- Each release: relevant tests, rollback backup, hosted check and release record.
- Monthly: restore drill, dependency advisories and support/retention review.
- On tax/pension changes: review assumptions and affected financial controls.

Set PAYMENTS_ENABLED=false to stop new checkouts while preserving existing passes. Do not expose owner preview on the public service.

Customer pilot and launch evidence

Use docs/launch-acceptance.md as the working acceptance record. It includes the business-details worksheet, hosted test matrix, independent model/legal review briefs and a pilot with 5-10 non-financial users.

Pilot tasks cover entering a fictional case, explaining fees and inflation, identifying the bridge, opening cashflow sections, saving/reloading a plan and understanding the offer. Record actual task completion and interpretation errors using anonymous participant labels. Automated browser tests are not a human pilot.

Local verification

audit/v20_4-verification.json distinguishes tests performed locally from external work. Financial tests, browser checks, payment simulations, backup restoration and document review do not certify the hosting account or legal position. No real customer payment or message was sent during local testing.

The public smoke-check tool checks the actual website and API, manual/sample links, headers, sales status and a small concurrent free projection. Use it after deployment:

```
python tools/hosted_check.py --web-url https://WEB-HOST \  
--api-url https://API-HOST --expect-sales closed
```

Run this command from the repository root, not backend. Replace the two origins. It uses fictional figures, without keys or payments. The small check is not a capacity guarantee or penetration test.

Remaining owner/account decisions

Supplier identity, territory, domain, hosting/Stripe/email accounts, actual provider privacy terms, independent reviews, human pilot results and final hosted acceptance remain outside the prepared source. Keep sales closed until those items have evidence and a responsible owner has approved launch.

Summary, display and decision tools (v20.10)

The financial formulas, inverse pension target, default assumptions and Save/Load schema are unchanged. Sensitivity results now include their actual current age and inflation. The additional services interpret or rerun the existing deterministic model. They do not apply another tax, inflation or fee deduction.

Money display

`frontend/lib/moneyDisplay.ts` converts each nominal amount at its own age using $\text{nominal} / (1 + \text{inflation})^{(\text{age} - \text{current_age})}$, with elapsed years floored at zero. UI rates are percentages; API/result rates are decimals. Null/out-of-horizon values stay unavailable. Display follows the last calculated inputs, not stale edits. Each scenario uses its own returned inflation, including the higher-inflation case. Overview and scenario charts use opening birthday balances; the horizon point is the previous year's closing balance. Cashflow and investment illustrations retain their labelled bases.

Service and access boundaries

`decision_tools.decision_summary` reads the first retirement cashflow, later-income gap and separate bridge. Incomplete bridge funding is null, never a covered zero. Free receives no new paid datasets. Silver and Gold receive this interpretation; only Gold receives three practical option summaries. The options are independent: extra monthly pension contribution +100, retirement age +1, or base monthly spending -100. Unavailable ages and small targets are explained.

POST `/planner/compare` requires active Gold access and exactly original/alternative numerical plans. Both pass the existing allowlist, validation and origin checks. Current age, horizon, calculation basis and tax residence must match. Each response includes its own inflation, central opening-balance path and key choices. Access is rechecked after computation. No database or report-file writes are added.

Option A is in React memory only. Refresh, Load plan, residence changes and access changes clear it. Requests are invalidated when inputs, access or the saved option change. The comparison result and pinned option are not added to plan files or browser storage.

One-page reports and verification

`summary_report.py` renders `report_type` summary from existing results. Silver and Gold may download it. `money_basis` defaults to today and accepts only today/future. It does not run simulation or strategy comparisons. `include_appendix=true` is rejected. A layout guard rejects overflow rather than silently issuing a two-page document. No customer title, name or address is accepted. The existing report types remain available.

Repeatable verification

Run `python decision_tools_qa.py <output-directory>` with development dependencies. It checks mixed accounts, separate fees, inflation/fixed spending, net/gross income, already-retired and short-horizon cases; direct-engine equality; server tier gates; origin and input rejection; access changes; and actual one-page PDF counts. Run `tools/browser-decision-check.cjs` using the same environment variables as the design check. It checks displays against API values, stale inputs, comparisons, clearing, downloads, tiers and mobile themes. Current release evidence is in `audit/v20_10-verification.json`.

Offer prices and periods stay unchanged. New contracts use terms version 2026-09-28-v8 to include the additional report/comparison features. Existing frozen contracts and expiries are not rewritten. No new dependencies, database migration or live payment/email operation is required by these improvements.

Investment choices update (v20.11)

New UI and Free-profile defaults: global-share total return 6.5%, bond return 5%, cash return 3.5%; annual volatilities remain 17.8%, 6%, 1.2%. Fees remain 0.5%, inflation 2.5%. Starting Balanced is 55/40/5 and retirement Balanced is 50/45/5, with ten gradual years ending at pension withdrawals. Model rates are compound planning assumptions, not arithmetic means or guaranteed forecasts. See <docs/investment-assumptions-review.md> for dated sources and risk limits.

The pension-withdrawal transition date is the later of retirement, access and chosen drawdown ages; it is not reset as current age advances. A custom transition age must be a whole age 18-120. Zero transition years behaves as a switch. A past or future transition date is allowed. Keep same preserves its starting mix. The default for API requests that omit transition mode is legacy, so existing integrations retain their old timing; new API clients should send the new profile explicitly.

The defined market-fall comparison applies a 40% share and 10% bond fall at $\max(\text{current age}, \text{retirement age}, \text{pension drawdown age})$, before withdrawals. Separate cash and Cash ISA are excluded, as is cash held inside an investment mix. Each invested account is shocked according to its own age-specific mix. This replaces the optional selected crash in the comparison only, and normal assumed returns then continue. Outside-horizon results are marked unavailable. The pension loss is separate from the total loss across accounts. It is not a calibrated tail probability or a worst-case bound.

No automatic cash reserve replenishment or trade-by-trade rebalancing has been introduced. The annual fixed-weight return model approximates maintained allocations. Fees and returns apply within each account; there is no extra fee or dividend stream on the combined balance.

Run `investment_choices_qa.py` for independent weighted-growth, annual transition, changing-mix present-value, cashflow, access and privacy checks. An optional path to the v20.10 engine checks all numeric cashflow fields across 24 legacy combinations. Saved-plan tests cover schemas 1-4. Report and browser checks cover the customer controls and exported values. See audit/v20_11-verification.json for the completed release evidence.

Free access for the owner and invited testers

Future Freedom Planner v20.12 - 2 October 2026

What has changed

This release supports individual complimentary passes. Your owner pass unlocks Gold without payment or a scheduled expiry. Tester passes can provide Free, Silver or Gold. They default to 30 days, starting when you issue them. They never automatically charge anyone. You can extend or revoke a tester pass.

An owner pass unlocks planner features only. It is not an administrator account. Creating and managing passes requires your hosting account's private backend console (or a local terminal for a local installation). No public admin page, shared master code or payment bypass has been added.

After deployment: create your owner pass once

1. Open the private console for the BACKEND service in your hosting account.

2. Work in the backend directory, using the same environment and persistent

access database as the running website. On the supplied Render template, BILLING_DB_PATH is /var/data/access.sqlite3.

3. Run the following command. The new private file must not already exist.

```
python billing_admin.py issue-pass owner --output /var/data/invites/owner.txt
```

4. Read/download the file through your private hosting console. It contains

your complete code beginning FFP-. Keep it in a password manager or a private folder. Do not put the file in your website, repository, public downloads, screenshots or support messages. The command prints only the pass reference and the private file location, never the sign-in code.

5. Open the normal customer website. Choose Plans & access, then Use invitation.

Paste the complete code and select Activate invitation.

6. The page shows Owner access - Gold, no scheduled expiry and no charge.

If you need to read the file in the private Linux hosting console:

```
cat /var/data/invites/owner.txt
```

This intentionally displays a secret in your authenticated console. Do not share the output. No production pass is pre-created in the release ZIP.

Give a tester access

In the same backend console, create a different file and pass for each person:

```
python billing_admin.py issue-pass tester --output /var/data/invites/t01.txt
```

The default is Gold for 30 days from issue. Send that person's private file/code directly to them. They follow the same Plans & access > Use invitation steps. No payment, card or email is required by the invitation system.

To test Silver for 7 days:

```
python billing_admin.py issue-pass tester --tier silver --days 7 \
--output /var/data/invites/t02.txt
```

Use --tier free to check the Free experience. Tester durations accept whole numbers from 1 to 365. Each pass has its own reference, code, expiry and browser allowance. Do not distribute your owner code to testers.

Returning to the planner

The invitation code is a reusable, personal sign-in secret, not a one-time email code. Keep it. The browser stays signed in for at most 30 days and never beyond the tester pass expiry. Signing out or clearing cookies means entering the code again. A session ending does not end your owner entitlement.

Up to two browsers can be registered, with one active at a time. Choose Use this browser to switch. A third browser must replace one of the first two. Anyone who has the private code can use the pass: these controls reduce casual sharing but do not verify the human holding it or prevent deliberate sharing.

Save plan downloads financial inputs separately. Invitation activation does not restore a plan, and the invitation file contains no pension figures.

Manage passes

List references and dates without exposing any private codes:

```
python billing_admin.py list-passes
```

Replace PASS_REFERENCE below with the 32-character reference from the list or private file. It is not the FFP- sign-in code.

```
python billing_admin.py extend-pass PASS_REFERENCE --days 30
python billing_admin.py revoke-pass PASS_REFERENCE
python billing_admin.py rotate-pass PASS_REFERENCE --output /var/data/invites/new.txt
```

Extension adds days to the existing expiry, or to now if it already expired. It never prolongs a browser's existing 30-day session. A revoked pass cannot be extended. Create a new pass if access should be reinstated.

Rotation replaces a lost/exposed code and immediately ends every old browser session. The expiry and tier stay unchanged. The old code stops working. Only current, non-revoked passes can be rotated. Revocation ends access at once.

Keep your own private mapping of tester names to pass references if needed; the planner deliberately has no name/email fields for these records.

Local Windows testing

For your normal local use, owner preview already unlocks everything. To test invitations and customer tiers locally, set `BILLING_MODE=test` and `PAYMENTS_ENABLED=false` in `backend/.env`, keep `APP_ENV=local`, then restart the backend. No Stripe or email credentials are needed for complimentary passes.

In a Command Prompt opened in backend, use `py` instead of `python` if required:

```
py billing_admin.py issue-pass owner --output C:\PlannerPrivate\owner.txt
py billing_admin.py issue-pass tester --output C:\PlannerPrivate\tester-01.txt
```

Open the private text file, then activate its code at <http://localhost:3000>. Local passes do not work on the hosted website: each uses its own database. On Windows, ensure only you can access the private folder; POSIX file-mode settings alone do not configure Windows folder permissions.

Deployment and privacy notes

Keep production `APP_ENV=production` and `BILLING_MODE=live`. `PAYMENTS_ENABLED` may remain false while invited people test. Do not put the public website in owner preview, and do not distribute the source ZIP as the customer product.

Issue live passes only after deployment, against the live persistent database. The first database access creates two additional tables without rewriting existing purchases. Back up the database before updating. The existing SQLite backup/restore commands include the new tables. A rollback to an older database can restore a revoked pass or old code; reconcile revocations after a restore.

The app stores random references, hashed private codes and browser credentials, permissions, dates and status. It stores no invitation names, addresses, emails or planning figures. A reusable code is sent only in a same-origin HTTPS POST, not a URL, browser storage or analytics. Browser cookies are `HttpOnly` and `SameSite Strict`, and `Secure` in production. Active owner records remain until revoked. Old expired/revoked records are normally removed after the configured 180-day retention period; browser credentials are removed earlier.

Private code files, backups, hosting logs and correspondence require separate handling. They do not become anonymous simply because the calculator asks only for numbers. Protect your hosting account and review infrastructure logging.

No free pass creates a Stripe order, receipt, refund or paid-upgrade credit. Paid purchases retain their existing email verification and refund rules.

Cancellation implementation - 3

October 2026 update

This is a pre-launch service-based implementation, not a legal classification or approval. Have a UK consumer-law adviser review fixed-term access, consent wording, deduction methodology and statutory remedies. No digital-content waiver is implemented.

New checkout requires the exact accepted terms version and independent review of this exact cancellation-policy version. Defaults remain `PAYMENTS_ENABLED=false`, `LEGAL_REVIEW_CONFIRMED=false`, `CANCELLATION_POLICY_REVIEWED_VERSION` blank, `SERVICE_DEDUCTIONS_ENABLED=false`. Existing `.env` files are not overwritten. An old legal-review flag alone cannot open sales.

When enabled after review, service deduction is the purchase price times the seconds represented by complete 24-hour periods after the first successful paid-feature response, divided by purchased access seconds. Round deductions down to whole pence; part days are not charged. Upgrade duration runs only to the anchored Gold expiry, not a new year. No download, administration or processor fee is deducted. Absent consent, policy evidence, successful confirmation email before first supplied service, or enabled/reviewed policy means zero automatic deduction.

`POST /billing/cancellation-quote` returns a quote tied to the verified purchase, valid for ten minutes. `POST /billing/refund` requires its token. Prices, dates and deductions are never accepted from the client. The actual cancellation quote ledger is stored before calling Stripe. Cancelled paid access ends immediately; network failures do not restore it. Retrying uses a stable idempotency key and the provider refund ID. No customer financial data is in the cancellation ledger.

A durable cancellation acknowledgement is returned immediately as a downloadable text file and queued to the original payment email. Monitor delivery and provider failures. Pending is not paid; failed/canceled/needs_action needs operator attention rather than a blind second refund. Monitor and resolve pending items daily and within statutory deadlines.

Operator commands (private shell only):

```
python billing_admin.py cancellation-queue
python billing_admin.py retry-cancellations
```

For an email notice sent in time, verify ownership and the notice timestamp, then use `cancel-purchase REFERENCE --notice-sent-at ISO_DATE_WITH_TIMEZONE --ownership-verified`. For an assessed full statutory remedy after the ordinary period use `--full-statutory-remedy`. These commands issue real refunds when configured with live credentials; never run on an unverified request. An open dispute requires manual provider review. Other remedies or refund amounts may need the Stripe dashboard and support review; this tool does not adjudicate them.

The packaged calendar covers 2026-2028 using England/Wales plus Northern Ireland bank holidays. Refresh `backend/app/data/cancellation-holidays.json` before coverage expires and

after exceptional bank holidays. Checkout fails closed outside its covered horizon; an uncertain deadline is not used to deny an existing customer's claim. Source: GOV.UK bank-holiday calendar.

Register refund.created, refund.updated and refund.failed in addition to the existing payment/dispute events. Pending refunds and unsent acknowledgements must be monitored. A fixed-price test suite does not prove real Stripe/email/hosted delivery, nor legal compliance. Before sales, perform test-mode checkout, confirmation, quote, cancellation, provider failure/retry, cross-browser revocation, support cancellation and original/upgrade scenarios on the hosted service.