Women's Pool Festival 2027

We celebrate Women's pool

AUSSCHREIBUNG

Arbeitsregeln für Codex

Diese Regeln gelten für das gesamte Repository. Ziel ist, mehrere Aufträge parallel bearbeiten zu können, ohne dass sich Chats gegenseitig den Git-Index, Dateien, Builds oder Commits verändern.

Parallele Entwicklung

- Entwickle nicht direkt im gemeinsam genutzten main-Worktree. Isoliere jeden Auftrag in einem eigenen Feature-Worktree; die fertige Änderung darf anschließend automatisch nach main integriert werden.
- Prüfe vor jeder Änderung mindestens:
- aktuellen Branch,
- git status --short,
- vorhandene Worktrees,
- aktuellen Stand von origin/main, soweit ein Fetch möglich und erforderlich ist.
- Wenn der aktuelle Arbeitsbereich auf main liegt, erstelle für den Auftrag einen eigenen Git-Worktree mit eigenem Feature-Branch.
- Verwende für neue Branches das Schema codex/<YYYYMMDD-HHMM>-<kurzer-auftragsname>.
- Lege lokale Worktrees unter .worktrees/<branch-ohne-codex-prefix> ab. .worktrees/ ist nicht Teil des Repositories.
- Wenn der Chat bereits in einem eindeutig zugewiesenen Feature-Worktree arbeitet, verwende diesen und erstelle keinen weiteren.
- Ein Worktree und Branch gehört genau einem Auftrag beziehungsweise einem koordinierenden Chat. Bearbeite nicht gleichzeitig denselben Branch aus mehreren Worktrees.

Beispiel:

``powershell git fetch origin git worktree add ".worktrees/20260724-standings" -b "codex/20260724-standings" origin/main ``

Falls das Anlegen des Worktrees wegen fehlender Berechtigung nicht möglich ist, frage nach Freigabe. Weiche nicht stillschweigend auf direkte Entwicklung in main aus.

Umgang mit fremden Änderungen

- Vorhandene Änderungen, die nicht eindeutig zum aktuellen Auftrag gehören, stammen vom Benutzer oder einem anderen parallelen Auftrag.
- Verändere, stage, committe, verschiebe oder verwerfe diese Änderungen nicht.
- Verwende kein git reset --hard, git checkout --, pauschales Stashing oder andere Befehle, die fremde Arbeit überschreiben könnten.
- Wenn ein Auftrag zwingend dieselben Dateien wie ein anderer laufender Auftrag betrifft, arbeite trotzdem im eigenen Worktree. Konflikte werden erst bei der Integration gelöst.
- Übernimm Änderungen aus einem anderen Feature-Branch nur, wenn sie eine ausdrücklich

genannte Abhängigkeit sind. Dokumentiere diese Abhängigkeit im Abschluss.

Commits und Pushes

- Erstelle kleine, thematisch geschlossene Commits auf dem Feature-Branch.
- Stage nur Dateien und Hunks des aktuellen Auftrags.
- Pushe den Feature-Branch als Sicherung und integriere einen abgeschlossenen, geprüften Auftrag anschließend automatisch nach main.
- Aktualisiere unmittelbar vor der Integration origin/main und rebase den Feature-Branch darauf.
- Pushe den rebasierten Feature-Branch ausschließlich per Fast-Forward nach main, zum Beispiel mit git push origin HEAD:main.
- Aktualisiere nach dem erfolgreichen Push auch den primären lokalen main-Worktree per Fast-Forward, damit dort kein manueller „Sync Changes“-Schritt nötig bleibt. Prüfe zuvor, dass dieser Worktree auf main steht und sauber ist; verwende anschließend dort git pull --ff-only. Wenn der Worktree nicht sauber ist oder kein Fast-Forward möglich ist, verändere ihn nicht und dokumentiere den Grund im Abschluss.
- Räume nach der erfolgreichen Integration den abgeschlossenen eigenen Feature-Worktree sowie den lokalen und den gleichnamigen Remote-Feature-Branch auf. Prüfe zuvor, dass der Feature-Worktree sauber ist, seine Branch-Spitze vollständig in main enthalten ist und der primäre main-Worktree synchronisiert wurde. Entferne niemals Worktrees oder Branches anderer laufender Aufträge. Kann die Zugehörigkeit oder der Integrationsstand nicht eindeutig bestätigt werden, behalte sie und dokumentiere den Grund.
- Verwende niemals Force-Push für main.
- Wenn ein anderer Auftrag zwischenzeitlich main aktualisiert hat und der Push abgelehnt wird: erneut fetchen, rebasen, Konflikte inhaltlich lösen, relevante Prüfungen wiederholen und den Fast-Forward-Push erneut versuchen.
- Nenne im Abschluss:
- Branchname,
- Commit-Hash oder Commits,
- ausgeführte Prüfungen,
- offene Abhängigkeiten oder erwartete Integrationskonflikte.

Release Notes

- Die Release-Notes-Seite besteht aus zwei Ebenen:
- Die obere, kuratierte Übersicht zeigt ausschließlich die wichtigsten Änderungen der jeweils letzten sieben Tage.
- Das darunterliegende Änderungsarchiv beginnt am 21. Juli 2026 und wird dauerhaft fortgeführt. Einträge im Archiv werden nicht entfernt, nur weil sie älter als sieben Tage sind.
- Jeder Feature-Branch muss vor seinem Abschluss mindestens einen neuen Archiveintrag in lib/i18n/releaseNotes.ts ergänzen. Das gilt auch für Bugfixes, technische Verbesserungen, Sicherheitsänderungen und interne Betriebsanpassungen.
- Ergänze neue Archiveinträge am passenden Datum und innerhalb des Tages in absteigender Reihenfolge, sodass die neuesten Änderungen oben stehen. Verwende eine dauerhafte, eindeutige id, einen passenden Themenbereich und den zutreffenden Änderungstyp.
- Formuliere Änderungen aus Sicht der Benutzer:innen beziehungsweise des Betriebs verständlich. Verwende keine reinen Commit-Titel, internen Ticketnummern, Commit-Hashes oder unnötigen Implementierungsdetails.
- Pflege jeden Archiveintrag inhaltlich gleichwertig auf Deutsch und Englisch.
- Prüfe zusätzlich, ob die Änderung in der kuratierten Sieben-Tage-Übersicht ergänzt oder dort in einen bestehenden Punkt eingearbeitet werden muss. Aktualisiere für diese obere Übersicht

Zeitraum, Aktualisierungsdatum und Kennzahlen. Entferne dort veraltete Inhalte, aber niemals die entsprechenden Einträge aus dem dauerhaften Archiv.

- Wenn parallele Feature-Branches dieselben Release-Notes-Bereiche ändern, müssen beim Rebase alle Archiveinträge und alle noch relevanten Sieben-Tage-Inhalte zusammengeführt werden. Bevorzuge keine Konfliktseite ungeprüft und verliere keine Änderungen anderer Aufträge.
- Ein Entwicklungsauftrag ist erst vollständig abgeschlossen, wenn Anwendungscode, zugehöriger Release-Notes-Eintrag und gegebenenfalls notwendige Übersetzungen gemeinsam geprüft und committed wurden.

Automatische Integration

Der Abschluss eines normalen Entwicklungsauftrags umfasst:

1. Änderungen im Feature-Worktree prüfen und committen.
2. Feature-Branch pushen.
3. Den neuesten Stand von origin/main fetchen.
4. Feature-Branch auf origin/main rebasen.
5. Konflikte inhaltlich lösen; keine Variante ungeprüft bevorzugen.
6. Nach dem Rebase mindestens TypeScript und die relevanten Tests ausführen, wenn sich durch die Integration ein Risiko ergeben hat.
7. Den geprüften Stand ohne Force-Push nach main pushen.
8. Den sauberen primären main-Worktree mit git pull --ff-only auf den integrierten Stand bringen. Bei lokalen Änderungen oder einem nicht möglichen Fast-Forward abbrechen und den Zustand dokumentieren.
9. Vom primären Worktree aus den sauberen, vollständig integrierten eigenen Feature-Worktree entfernen, danach den lokalen Feature-Branch mit git branch -d und den gleichnamigen Remote-Branch mit git push origin --delete löschen.

Supabase-Migrationen

- Verwende für neue parallele Migrationen einen UTC-Zeitstempel statt der bisherigen fortlaufenden vierstelligen Nummer:

YYYYMMDDHHMMSSkurzebeschreibung.sql.

- Prüfe vor dem Anlegen, ob der Zeitstempel beziehungsweise Dateiname bereits existiert.
- Ändere eine bereits auf einer gemeinsamen oder produktiven Datenbank ausgeführte Migration nicht nachträglich. Erstelle dafür eine neue Korrekturmigration.
- In der aktuellen Produktentwicklungsphase dürfen geprüfte Migrationen als normaler Teil des Auftrags sofort auf die konfigurierte gehostete Supabase-Datenbank angewendet werden.
- Prüfe vor der Ausführung das konkrete Zielprojekt und teste die Migration, soweit praktikabel, zunächst in einer Transaktion mit Rollback.
- Migrationen, die Daten löschen, Spalten oder Tabellen entfernen, bestehende Werte irreversibel umschreiben oder längere Sperren verursachen können, benötigen weiterhin eine ausdrückliche Freigabe.
- Gestalte sofort angewendete Migrationen abwärtskompatibel zum aktuellen main, bis der zugehörige Anwendungscode integriert ist.
- Dokumentiere im Abschluss, ob eine Migration nur erstellt, lokal geprüft oder auf eine Datenbank angewendet wurde.

Supabase-CLI, Anmeldung und Zielprojekt

- Das konfigurierte Produktionsprojekt steht in .supabase/config.prod.toml. Verwende für den

vorgesehenen Ablauf zuerst npm run db:doctor, dann npm run db:prod, anschließend npm run db:push:dry-run und erst nach erfolgreicher Prüfung npm run db:push.

- Alle Datenbankskripte verwenden scripts/supabase-common.ps1. Der Wrapper erzwingt exakt die in package.json deklarierte Supabase-CLI-Version; rufe für diese Abläufe nicht ersatzweise eine zufällig aus einem übergeordneten node_modules aufgelöste CLI auf.
- db:doctor prüft CLI-Version, Codex-Proxy, Zielprojekt, Anmeldung, Datenbankpasswort, Worktree-Verknüpfung und API-Erreichbarkeit, ohne Geheimnisse auszugeben. Führe diese Diagnose aus, bevor du einen Supabase-Blocker meldest.
- Scheitert db:doctor in einer eingeschränkten Sandbox an npm-Cache, DNS oder Transport, wiederhole genau den Diagnoselauf mit der vorgesehenen Netzwerkfreigabe. Erst das Ergebnis dieses freigegebenen Laufs darf als externer Supabase-Blocker gewertet werden.
- Verwechsle die folgenden Zustände nicht:
- CLI deklariert: supabase ist in package.json vorhanden.
- CLI installiert: npm ls supabase --depth=0 beziehungsweise npx --no-install supabase --version funktioniert.
- Ziel konfiguriert: .supabase/config.prod.toml enthält die erwartete project_ref.
- CLI angemeldet: Die Anmeldung kann unter Windows im Credential Manager als Supabase CLI:supabase liegen, auch wenn SUPABASEACCESSTOKEN nicht als Umgebungsvariable gesetzt ist.
- Worktree verknüpft: supabase/.temp/project-ref existiert und stimmt mit .supabase/config.prod.toml überein. Diese ignorierte Runtime-Datei wird nicht automatisch in neue Worktrees übernommen.
- Schließe aus fehlenden Umgebungsvariablen oder einer fehlenden supabase/.temp/project-ref niemals allein auf eine fehlende Supabase-Anmeldung. Prüfe unter Windows ohne Ausgabe von Geheimnissen zusätzlich:

``powershell cmdkey /list | Select-String -Pattern "Supabase CLI" -Context 0,2 ``

- Gib niemals Credential-Inhalte, Authorization-Header, Tokens oder Datenbankpasswörter aus. Das Vorhandensein eines Credential-Manager-Eintrags darf dokumentiert werden.
- Der gemeinsame Wrapper darf fehlende SUPABASEACCESSTOKEN- und SUPABASEDBPASSWORD-Werte ausschließlich für den laufenden Prozess aus der bereits freigegebenen .env.local des primären Worktrees laden. Er kopiert keine .env-Datei in Feature-Worktrees und gibt weder Werte noch Header aus.
- Proxywerte, die exakt auf den bekannten Codex-Sentinel 127.0.0.1:9 zeigen, werden nur im Supabase-Unterprozess ignoriert. Andere, absichtlich konfigurierte Proxys dürfen nicht verändert werden.
- Wenn npx --no-install supabase nicht verfügbar ist, prüfe zuerst npm ls supabase --depth=0. Eine fehlende lokale Installation ist ein Abhängigkeitszustand und kein Anmeldefehler. Verwende bei freigegebenem Netzwerk ersatzweise npx supabase ... und dokumentiere die tatsächlich verwendete CLI-Version; ändere dabei nicht vorsorglich Lockfiles.
- Ein neuer Feature-Worktree darf für den Datenbanklauf erneut mit dem erwarteten Projekt verknüpft werden. Prüfe danach den Inhalt von supabase/.temp/project-ref, bevor eine Migration ausgeführt wird.
- Unterscheide Netzwerk-, Authentifizierungs- und Berechtigungsfehler anhand der konkreten CLI-Ausgabe:
- 401 oder eine Meldung zu einem ungültigen Access Token weist auf die Anmeldung hin.
- 403 weist typischerweise auf fehlende Projektberechtigungen hin.
- Transport error, DNS-, TLS-, Timeout- oder Verbindungsfehler beim Aufruf von api.supabase.com sind Netzwerkprobleme und dürfen nicht als fehlende Anmeldung beschrieben werden.

- Wiederhole einen Transportfehler höchstens einmal und prüfe anschließend die neuesten Einträge in ~/.supabase/traces, ohne Header oder Geheimnisse auszugeben. Bleibt der Fehler bestehen, wende keine Migration unbestätigt an und dokumentiere den exakten Endpunkt sowie die Fehlerklasse.
- Das Fehlen von Docker oder Podman verhindert nur die lokale Supabase-Prüfung. Es sagt nichts über CLI-Anmeldung, Remote-Verknüpfung oder Erreichbarkeit des gehosteten Projekts aus.

Projektarchitektur und geschützte Bereiche

- Das klassische Turniermodul und das Event-Modul besitzen teilweise unterschiedliche Datenmodelle. Vermische Tabellen, Statusmodelle oder Server Actions nicht ohne eine bewusst geplante Migration.
- Behandle das Scoreboard als besonders kritisch. Verändere Scoreboard-Routen, Scoreboard-Komponenten oder dafür relevante globale Styles nur mit Vorsicht und nach Rücksprache.
- Prüfe bei Änderungen an gemeinsam genutzten Layout-, Theme-, Realtime- oder Match-Komponenten, ob das Tablet-Scoreboard betroffen sein kann.
- Führe kein Next.js-, React-, Supabase- oder anderes grundlegendes Framework-Upgrade ohne ausdrücklichen Auftrag durch.

Hosting und Free-Tier-Grenzen

- Das Produktionssystem verwendet Vercel Hobby und Supabase Free. Plane und implementiere Funktionen so, dass sie ohne kostenpflichtiges Upgrade in diesen Tarifen funktionieren.
- Setze keine Funktion, Kapazität oder Ausführungsfrequenz voraus, die nur in einem kostenpflichtigen Vercel- oder Supabase-Tarif verfügbar ist. Ist eine Anforderung im Free-Tier nicht sinnvoll umsetzbar, dokumentiere die konkrete Grenze, schlage eine Free-Tier-kompatible Alternative vor und hole vor einem Tarifwechsel eine ausdrückliche Entscheidung ein.
- Prüfe bei zeitgesteuerten Jobs, Hintergrundverarbeitung, Functions, Realtime, Datenbankgröße, Storage, Bandbreite und anderen tarifabhängigen Ressourcen die aktuell geltenden offiziellen Limits. Verlasse dich nicht auf frühere oder nur aus anderen Tarifen bekannte Grenzwerte.
- Vercel Hobby erlaubt derzeit keine hochfrequenten Cronjobs; ein Cron-Ausdruck, der häufiger als einmal täglich läuft, darf nicht in vercel.json aufgenommen werden. Verwende für zeitkritische Abläufe eine innerhalb des Free-Tiers tragfähige, authentifizierte und idempotente Alternative.
- Berücksichtige Supabase-Free-Limits bereits im Datenmodell und in Abfragen: vermeide unnötiges Polling, unbegrenztes Datenwachstum und Designs, die kostenpflichtige Datenbank-, Backup-, Netzwerk- oder Function-Funktionen voraussetzen.
- Nenne im Abschluss, wenn eine Änderung relevante Free-Tier-Ressourcen verbraucht oder nur unter einer bestimmten Betriebsbedingung zuverlässig arbeitet, zum Beispiel solange eine Verwaltungsansicht geöffnet ist.

Supabase und serverseitige Sicherheit

- Verwende createAdminSupabaseClient() ausschließlich serverseitig und erst nach einer passenden Authentifizierungs- und Berechtigungsprüfung.
- Neue Tabellen mit Benutzer-, Vereins- oder Turnierdaten benötigen geeignete RLS-Regeln und eine Prüfung der relevanten Indizes und Foreign Keys.
- Sicherheits-, Kapazitäts-, Teamzugehörigkeits-, Wartelisten- und Statusregeln dürfen nicht nur im UI geprüft werden. Sichere sie zusätzlich atomar in Serverlogik, Datenbankfunktionen oder Constraints ab.
- Übertrage über Realtime keine personenbezogenen Daten. Erweitere bevorzugt die

vorhandenen Event-Realtime-Versionen und Refresh-Komponenten.

Benutzeroberfläche und Übersetzungen

- Alles, was normale Benutzerinnen und Benutzer sehen, muss Deutsch und Englisch unterstützen. Ergänze neue Texte über lib/i18n in beiden Sprachen.
- Verwende die vorhandenen Dialog-, Bestätigungs-, Toast-, SafeActionForm- und SafeActionButton-Komponenten, statt parallele Einzellösungen einzuführen.
- Dialoge dürfen bei einem Klick außerhalb nicht unbeabsichtigt schließen und benötigen eine sichtbare Abbrechen-Aktion.
- Prüfe neue oder wesentlich geänderte Oberflächen auf Mobilgeräten und in Dark-, Light- und Pink-Theme.

Abhängigkeiten und generierte Dateien

- Ändere package.json und package-lock.json nur, wenn der Auftrag tatsächlich eine Abhängigkeitsänderung benötigt.
- Führe keine Paketinstallation nur zur vorsorglichen Aktualisierung des Lockfiles aus.
- Committe keine .env*-Dateien, Tokens, Secrets, Supabase-Zugangsdaten oder lokale Runtime-Artefakte.
- Kopiere Secrets nicht automatisch in neue Worktrees. Verwende nur bereits freigegebene Konfigurationen.

Prüfstrategie

- Führe Prüfungen proportional zum Risiko aus.
- Verwende für die Typprüfung npx tsc --noEmit.
- Verwende für gezielte Tests npx vitest run tests/unit/<datei>.test.ts und für die vollständige Suite npm test.
- Für kleine UI- oder Textänderungen reichen normalerweise die Typprüfung und gezielte Tests.
- Führe npm run build aus, wenn Routing, Server-/Client-Grenzen, Konfiguration, Abhängigkeiten oder andere buildkritische Bereiche verändert wurden.
- Ein Build ist nicht nach jeder kleinen Änderung erforderlich.
- Behebe Fehler des eigenen Auftrags. Verändere unabhängige fehlschlagende Bereiche nicht ohne Auftrag und dokumentiere solche Blocker.

Kontakt: bla@bla.com

Bankverbindung:

DE12340ß3253250928Đ

BIC: WWI1231