



ÉTUDE DE CAS UX

FR EN 中文

Paris Tennis Dispo

À Paris, être le premier averti quand quelqu'un annule

6 jours 22 heures Du premier cahier des charges au dépôt sur l'App Store

En solo — produit, design, front iOS, back Python, publication

Jiayue Liu

Août 2026

Sommaire

FR / EN / 中文

01	Vue d'ensemble		02	Le point de départ	Ce n'est pas un persona inventé. C'est mon mercredi soir.
03	La quatorzième minute	J'ai terminé mon premier plan, et je l'ai enterré quatorze minutes plus tard.	04	Deux champs de bataille	Renoncer à la ruée est exactement ce qui m'a fait voir le combat que personne ne menait.
05	Prouver d'abord, construire ensuite	Avant la première ligne d'interface, j'ai écrit trois scripts.	06	Intention de design	Quatre points de friction, quatre principes qui devaient tenir.
07	Architecture de l'information	Trois onglets pour trois états : je cherche, j'attends, je me gère.	08	Le système de couleurs	Sept critères partagent un écran. Les couleurs n'ont pas été choisies : elles ont été calculées.
09	Trois détails qui n'améliorent rien	L'icône, l'animation de lancement, le bouton de langue. Aucune ne rend la recherche plus rapide - ensemble, elles décident si l'app a l'air faite par quelqu'un.	10	Trois décisions d'interface	Chacune sait répondre à « pourquoi pas la solution évidente ».
11	Trois décisions d'interaction	Une bonne interaction se manifeste souvent par « l'utilisateur n'a pas remarqué que quelque chose avait été traité ».	12	Du design au code	Les décisions de design ne s'arrêtent pas à la maquette. Ci-dessous : l'idée à gauche, ce qu'elle est devenue à droite.
13	La publication	Le dernier jour : faire passer les achats intégrés de « ça a l'air de marcher » à « ça marche ».	14	Sept jours	Le parcours complet, reconstitué à partir des horodatages et de l'historique git.
15	Rétrospective	Ce que je pense avoir bien jugé, et ce que je ferais autrement.			

Vue d'ensemble

Une seule personne, sept jours, pour transformer « je n'arrive pas à réserver un court » en une application iOS déposée sur l'App Store.



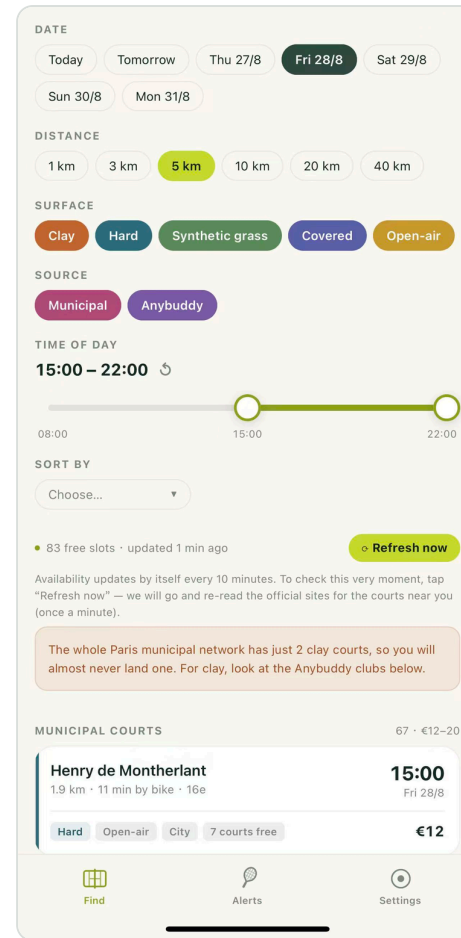
Chercher • 中文

Les six critères de filtre sur le premier écran, rien dans un tiroir



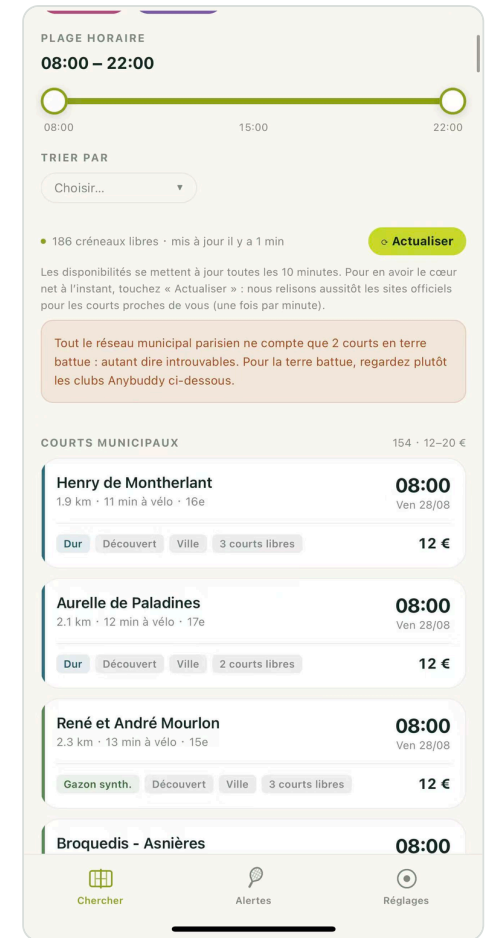
Résultats • 中文

Municipal et Anybuddy : même écran, même tri, mêmes filtres



Find • English

Même structure en trois langues ; la mise en page se réorganise seule



Chercher • Français

Le français est la langue par défaut des utilisateurs, pas une traduction

MON RÔLE	En solo — cadrage, design, front iOS, back Python, publication
PÉRIODE	19 – 25 août 2026, soit 6 jours et 22 heures
RÉSULTAT	iOS 1.0.0 déposé pour validation ; interface trilingue ; abonnement
TECHNIQUE	React Native (Expo) • StoreKit 2 • Python / FastAPI • PostgreSQL • Railway
ÉCHELLE	54 sites surveillés en continu • 202 courts municipaux • un passage toutes les 10 minutes (en août 2026)

6 j 22 h

De l'idée au dépôt

52

Commits (29 front / 23 back)

3

Langues, toutes rédigées par moi

14 min

Temps qu'il m'a fallu pour rejeter mon premier plan

02

Le point de départ

Ce n'est pas un persona inventé. C'est mon mercredi soir.

FR / EN / 中文

Je joue au tennis à Paris. Quand je veux un court, il faut aller sur le site de réservation de la Ville et cliquer case par case : choisir un centre, choisir une date, choisir un créneau, valider, attendre le chargement, lire « aucun créneau disponible », revenir en arrière, essayer une autre heure, recommencer.

Au bout d'une vingtaine de tentatives, je finis en général par fermer l'onglet en pensant la phrase que tout produit devrait redouter —

L'instant où cette pensée apparaît, c'est le problème que le produit doit résoudre.

Ce n'est pas qu'un clic soit trop lent : c'est que **l'attention que réclame l'ensemble est assez coûteuse pour faire abandonner.**

Ma motivation de départ était parfaitement égoïste : je voulais réussir à jouer. Je n'ai découvert qu'ensuite à quel point le problème est répandu chez les joueurs parisiens.

« Laisse tomber, je ne joue pas aujourd'hui. »

La quatorzième minute

J'ai terminé mon premier plan, et je l'ai enterré quatorze minutes plus tard.

FR / EN / 中文

Le premier plan s'appelait « réservation automatique » : stocker les identifiants de l'utilisateur et lui attraper un court à la seconde de l'ouverture. **Ça marchait, et ça ne devait pas exister.**

v0.1 • Réservation automatique

- Oblige à stocker les mots de passe d'une plateforme tierce
- Viole clairement les conditions anti-automatisation
- Très probablement refusé par la validation App Store
- Prend la place des autres sur une ressource publique
- Une collecte séparée à faire tourner par utilisateur

Une fois les deux colonnes côte à côte, il n'y avait plus à hésiter. Les horodatages sont sans ambiguïté : v0.1 écrit à **18 h 47**, v0.2 à **19 h 01**.

v0.2 • Agrégation + alerte

- Aucun mot de passe, aucun compte, rien à stocker
- N'utilise que des données publiquement visibles
- Une catégorie établie, publiable
- Ne prend le court de personne : fait seulement arriver l'information plus vite
- Une collecte centralisée, partagée par tous

C'est la page sur laquelle je veux le plus être jugée

Non pas parce que j'ai eu raison, mais parce que **j'ai enterré le plan le plus séduisant alors que personne ne regardait**. La réservation automatique sonne bien plus impressionnant et fait une bien meilleure démonstration. Elle mise aussi, d'un coup, les mots de passe des utilisateurs, les règles des plateformes et l'équité d'une ressource publique.

La valeur d'un chef de produit n'est pas d'imaginer des fonctionnalités. Elle est de savoir lesquelles ne doivent pas exister.

Ce virage a coûté quelque chose de réel : j'ai renoncé à gagner les courts les plus demandés au moment de leur ouverture. Mais c'est en chiffrant cette perte que

j'ai trouvé la vraie opportunité du produit — page suivante.

04

Deux champs de bataille

Renoncer à la ruée est exactement ce qui m'a fait voir le combat que personne ne menait.

FR / EN / 中文

« Obtenir un court demandé » ressemble à un seul problème. En le démontant, on en trouve deux, totalement différents.

TERRAIN A

L'ouverture des créneaux

- Chaque jour à heure fixe
- Des centaines de gens qui rafraîchissent ensemble
- Épuisé en quelques secondes
- Se gagne aux réflexes et à la latence

Ingagnable. Abandonné volontairement.

TERRAIN B

Les courts libérés par une annulation

- Au hasard, toute la journée
- Presque personne ne surveille
- Le premier qui voit réserve
- Se gagne à qui l'apprend en premier

Voilà l'ouverture.

Personne ne rafraîchit un site deux cents fois par jour. Un serveur peut balayer toute la ville toutes les dix minutes.

Une fois cela clair, le centre de gravité est passé de la recherche à la surveillance — autrement dit, **le moment le plus utile de ce produit a lieu pendant que l'utilisateur ne le regarde pas.**

Ce constat a déterminé tout le reste : les alertes sont devenues un onglet principal et non une option secondaire ; la recherche reste gratuite pour toujours et seule l'alerte est payante ; et la qualité des notifications est devenue le problème d'ingénierie le plus important du projet.

Prouver d'abord, construire ensuite

Avant la première ligne d'interface, j'ai écrit trois scripts.

FR / EN / 中文

Tout le plan reposait sur une hypothèse : **les disponibilités des deux plateformes sont lisibles sans connexion**. Si elle tombait, toutes les décisions de design qui suivaient reposaient sur du vide.

Alors, entre 19 h 18 et 20 h 41 le 19 août, j'ai écrit trois scripts jetables qui ne faisaient qu'une chose : récupérer de vraies disponibilités pour voir leur structure, leur fréquence de mise à jour et si les champs suffisaient.

- ① `v.py` — les pages de disponibilité municipales sont-elles seulement récupérables ?
- ② `v2.py` — ce qui revient peut-il être analysé en créneaux structurés ?
- ③ `v3.py` — les deux plateformes peuvent-elles tenir dans un même modèle ?

Pourquoi cette étape a sa place dans un portfolio

À cause de ce que dit **l'ordre**. Le premier réflexe de beaucoup de gens est de dessiner les écrans : c'est joli, ça se montre, ça donne le sentiment d'avancer. Mais si la donnée est inaccessible, le plus bel écran du monde ne vaut rien.

Quatre-vingt-dix minutes à essayer de réfuter l'hypothèse la plus fatale, puis six jours à construire. Cet ordre n'est pas une astuce d'efficacité, c'est un jugement.

Ce n'est qu'une fois les scripts passés que j'ai eu les chiffres : 43 centres de tennis municipaux, 202 courts — et **exactement 2 en terre battue**. Ce nombre est devenu directement une décision d'interface — voir la section 09.

Intention de design

Quatre points de friction, quatre principes qui devaient tenir.

FR / EN / 中文

Le design ne commence pas à « j'aimerais faire un bel écran ». Il commence à « où l'utilisateur bloque-t-il ». J'ai donc posé les frictions observées une par une, et tiré de chacune un principe que le produit n'avait pas le droit d'enfreindre.

OÙ L'UTILISATEUR BLOQUE On ne sait qu'en ouvrant : 301 allers-retours de page pour couvrir une seule semaine	DONC CECI DOIT TENIR Mettre la disponibilité avant le choix, pas derrière	CE QUE ÇA DONNE Une liste qui montre d'un coup tous les créneaux correspondants ; filtrer resserre, ça ne sonde pas
OÙ L'UTILISATEUR BLOQUE Deux plateformes, des allers-retours, aucune comparaison possible	DONC CECI DOIT TENIR Comparer dans une même vue, pas dans une même tête	CE QUE ÇA DONNE Un seul modèle de données : municipal et Anybuddy partagent écran, tri et filtres
OÙ L'UTILISATEUR BLOQUE Le tennis se dilue dans une application multisport	DONC CECI DOIT TENIR Remonter au premier écran ce qui compte vraiment pour un joueur de tennis	CE QUE ÇA DONNE Surface, couverture, distance et créneau en interrupteurs ; la collecte ne lit que la sous-page /tennis
OÙ L'UTILISATEUR BLOQUE La vraie occasion est une annulation aléatoire que personne ne peut guetter	DONC CECI DOIT TENIR Le produit doit travailler pendant que l'utilisateur ne le regarde pas	CE QUE ÇA DONNE Surveillance de fond continue et notifications ; les alertes en onglet principal

Derrière chaque principe de design, il doit y avoir une personne précise à un moment précis.

Architecture de l'information

Trois onglets pour trois états : je cherche, j'attends, je me gère.

FR / EN / 中文

Chercher

Je veux jouer maintenant

Six critères de filtre + une liste inter-plateformes + trois tris

Alertes

J'attends que quelqu'un annule

Conditions de veille + « quand je peux jouer » + notifications

Réglages

Je veux changer quelque chose

Adresse de départ, langue, notifications, abonnement

Une vraie session, sans montage



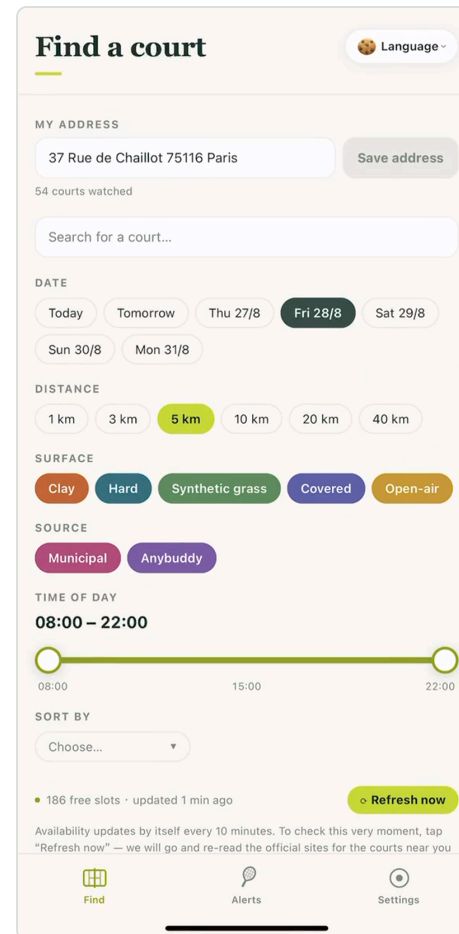
00:00

Le démarrage à froid arrive directement sur « Chercher »



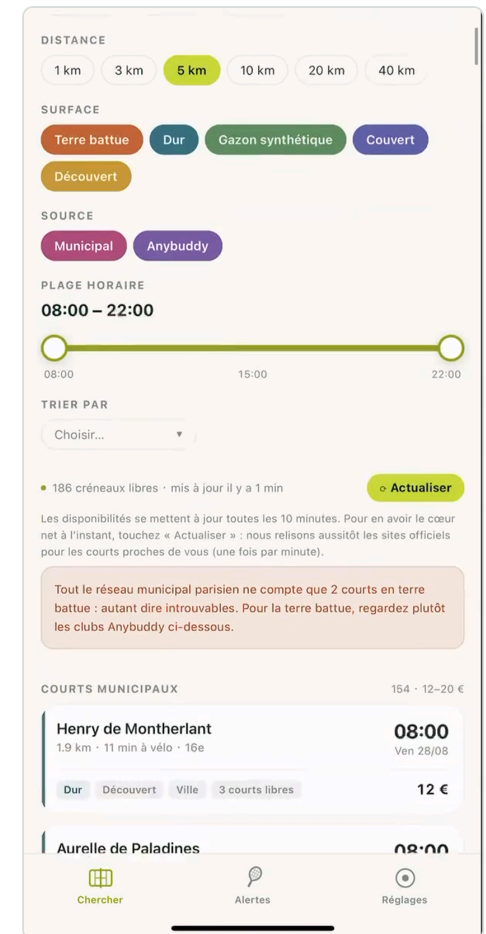
00:06

Filtres trop serrés → état vide, avec une porte de sortie



00:15

Passage à l'anglais : aucun rechargement, filtres conservés



00:21

Passage au français, liste inchangée

Vingt-sept secondes, aucune coupe. Du démarrage à froid aux résultats, **aucune étape ne demande qui vous êtes**. Le changement de langue est instantané — pas de redémarrage, pas de filtres perdus, pas de retour à l'accueil — parce que la langue n'est qu'une couche de textes et ne touche jamais à l'état.

Une duplication volontaire

« Chercher » et « Alertes » emploient le même vocabulaire de filtres, mais chacun garde **son état indépendant**.

Le coût, c'est que l'utilisateur configure deux fois. Je l'accepte, parce que « ce que je cherche maintenant » et « ce dont je veux être prévenu » sont deux intentions

différentes : l'une est passagère, l'autre permanente. Partager l'état laisserait une recherche faite à la volée corrompre en silence une règle d'alerte de longue durée.

En compensation, la configuration des alertes est volontairement plus légère : son créneau est rattaché à un horaire permanent, « quand je peux jouer », réglé une fois par jour de la semaine puis oublié.

08

Le système de couleurs

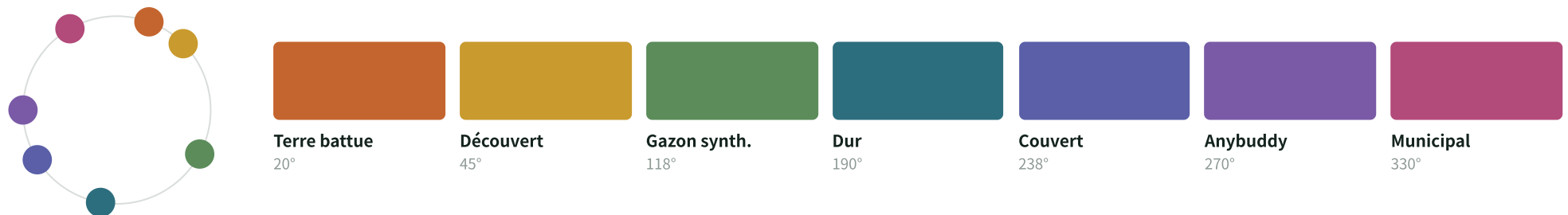
Sept critères partagent un écran. Les couleurs n'ont pas été choisies : elles ont été calculées.

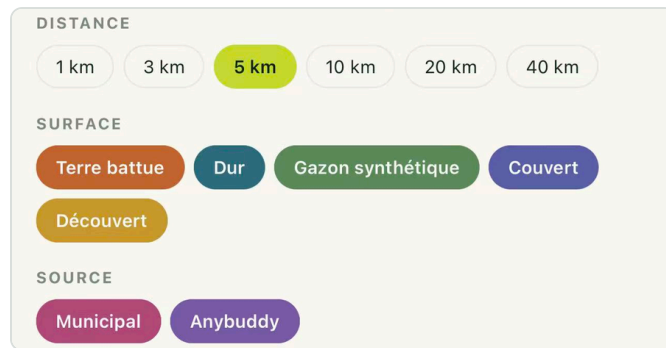
FR / EN / 中文

L'écran « Chercher » doit montrer en même temps : trois surfaces de court, couvert ou découvert, deux sources de données, et la date. Choies à l'œil, ces

couleurs se gênent à l'écran et l'utilisateur n'a plus qu'à lire chaque étiquette mot à mot.

Je les ai donc réparties **à intervalles réguliers sur la roue chromatique** :





Les sept teintes réunies sur un écran, telles que livrées. Cinq couleurs de surface au-dessus, deux couleurs de source en dessous. Deux pastilles voisines sont assez éloignées en teinte pour que l'image, réduite à la taille d'un ongle, reste lisible.

Remarquez la ligne DISTANCE au-dessus : elle emploie le jaune tennis, une famille à part — parce que la distance est un **choix unique**, pas une catégorie.

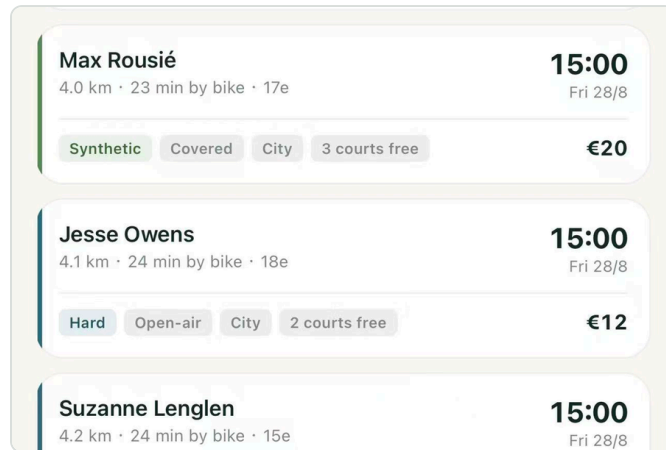
- Les cinq premières teintes sont à peu près équidistantes sur la roue : **deux pastilles voisines diffèrent assez** pour être distinguées sans lire
- Les deux couleurs de source occupent volontairement les creux restants, violet et magenta, pour ne jamais entrer en conflit avec une surface
- **Chaque teinte porte son propre repère de mémoire** : terre cuite pour la terre battue, vert d'herbe pour le gazon synthétique, or ensoleillé pour le découvert, indigo d'ombre pour le couvert

La ligne des dates n'a volontairement pas eu de huitième teinte

Le bandeau de dates emploie un vert de pin proche du texte courant, et non une teinte vive de plus — parce que **la date est un axe, pas une catégorie**. Ce n'est pas la même espèce de chose que les lignes de filtres en dessous.

Donnez-lui une teinte et on la lit comme l'égale de la surface et de la source. Employer « une autre espèce de couleur » pour exprimer « une autre couche de structure » fait plus de travail que n'importe quel filet de séparation supplémentaire.

Le bénéfice : chaque fiche de résultat porte, sur son bord gauche, une barre verticale à la couleur de sa surface. En défilant vite, **la vision périphérique suffit à écarter les surfaces dont on ne veut pas**. Les étiquettes de texte sont toujours là, mais leur rôle passe d'identifier à confirmer.



Cette barre, c'est là que tout le système de couleurs est encaissé. La fiche du haut est en gazon synthétique (vert), celle du bas en dur (bleu) — les fiches sont par ailleurs identiques, et toute la distinction tient dans 3

dp de couleur sur le bord gauche.

Dépenser la couleur sur l'instantané, pas sur le joli.

Trois détails qui n'améliorent rien

L'icône, l'animation de lancement, le bouton de langue. Aucune ne rend la recherche plus rapide - ensemble, elles décident si l'app a l'air faite par quelqu'un.

FR / EN / 中文

Les produits utilitaires glissent tous vers le même visage : utilisable, correct, sans expression. J'ai volontairement passé du temps sur trois points qui ne font bouger aucun indicateur d'efficacité.

I - L'icône n'est pas une balle de tennis



Un moineau parisien en bandeau tricolore, debout sur un court, trois balles à côté.

La réponse par défaut, c'est de dessiner une balle. Une seconde pour y penser, une seconde pour l'oublier. Cherchez « tennis » sur l'App

Store : un écran entier de ronds jaune-vert.

Le moineau est l'oiseau le plus banal des rues de Paris ; le bandeau tricolore, c'est la ville. **Une icône ne doit pas répondre « quel sport » mais « quel sport, dans quelle ville ».**

II - Les 0,86 seconde du lancement

L'écran de lancement natif d'Expo est **une image fixe** : son texte ne peut pas suivre la langue. J'ai donc découpé le lancement en deux couches : la native ne

porte que l'icône, pas un seul mot ; dès qu'elle disparaît, une couche rendue par le code prend le relais - et c'est seulement là que la signature peut suivre la langue du système, en français, anglais ou chinois.



La balle tombe de 160 pt au-dessus de l'écran en 520 ms, sur `cubic-bezier(0.34, 1.56, 0.64, 1)`. **Le deuxième point de contrôle a $y = 1,56$, supérieur à 1 : c'est pour cela que la balle dépasse son point d'arrivée puis revient.** Une fois posée, le texte apparaît en fondu sur 340 ms. 0,86 seconde en tout : pas lent, mais autrement plus vivant qu'une

coupe sèche.

Ces images ne sont pas des captures. Elles ont été recalculées à partir des paramètres de la courbe et de la géométrie des coutures présentes dans le code : chacune occupe donc la position exacte de cette milliseconde.

III - Aux langues, des viennoiseries plutôt que des drapeaux

Un drapeau n'est pas une langue. Quel drapeau pour l'anglais - États-Unis, Royaume-Uni, Australie ? Et pour le chinois ? Pire : une rangée de drapeaux nationaux dans un produit français se lit davantage comme de la politique que comme de la langue.

Les coutures sont calculées, pas tracées

Ces deux arcs ne sont pas des courbes dessinées : ce sont **les bords de deux cercles bien plus grands**. Si les cercles sont petits, les arcs se croisent près du centre et la balle devient une amande.

Les valeurs qui marchent : rayon de couture 1,62r, centre à 1,96r du centre de la balle - chaque arc ne dépasse alors le centre que de 0,34r, ce qui est l'allure d'une vraie balle. La balle entière est assemblée avec des View plutôt qu'une image : elle reste nette à n'importe quelle densité d'écran.



中文

Raviolis



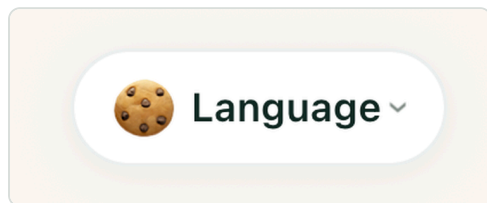
English

Cookie



Français

Croissant



Les trois viennoiseries ne représentent aucun pays. Elles représentent **ce qu'on trouve sur la table le matin là où cette langue se parle**. Aucune ambiguïté, aucune position prise - et toujours reconnaissables réduites à 15 pt.

La pastille occupe le même coin sur les trois onglets. Changer de langue est un geste qu'on fait à tout moment : ça n'a pas sa place au troisième niveau des Réglages.

10

Trois décisions d'interface

Chacune sait répondre à « pourquoi pas la solution évidente ».

FR / EN / 中文

I Le mur payant montre le produit, pas des adjectifs

CE QUE J'AI FAIT

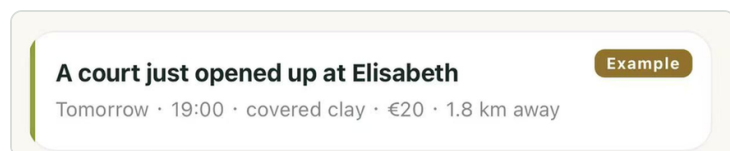
La première chose sur l'écran d'abonnement n'est pas une liste de fonctionnalités : c'est un exemple de notification, en tout point identique à la vraie.

POURQUOI

Avant de décider de payer, il faut voir ce que l'on achète. Ce que trois lignes d'adjectifs n'arrivent pas à dire, une fiche d'exemple le dit en deux secondes.

COÛT / LIMITE

C'est justement parce qu'elle a l'air vraie que la pastille « Exemple » dans son coin ne peut pas sauter : sans elle, quelqu'un croira qu'un court s'est réellement libéré et se déplacera pour rien. La confiance gagnée par le réalisme ne doit pas être encaissée par la tromperie.

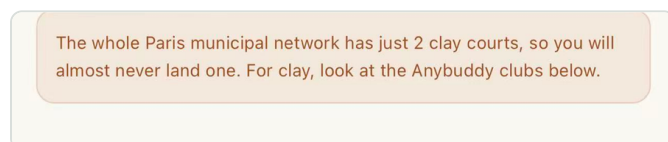


Voici la fiche d'exemple qui ouvre l'écran d'abonnement. Elle utilise le même composant et la même typographie qu'une vraie alerte : nom du site, demain 19 h 00, terre battue couverte, 20 EUR, à 1,8 km.

La pastille « Exemple » dans le coin est le trait dont cette décision ne peut pas se passer : plus la fiche a l'air vraie, plus il coûte cher de ne pas la signaler.

II L'avertissement terre battue n'apparaît que quand il sert

CE QUE J'AI FAIT	Cocher « Terre battue » fait apparaître une explication — mais seulement lorsque Municipal et Terre battue sont cochés ensemble .
POURQUOI	Sur 202 courts municipaux à Paris, exactement 2 sont en terre battue. Sans explication, l'utilisateur en conclut que l'application est cassée.
COÛT / LIMITE	Mais si l'utilisateur ne regarde pas du tout les courts municipaux, le même message n'est que du bruit. Plus la condition est serrée, plus le message vaut cher.

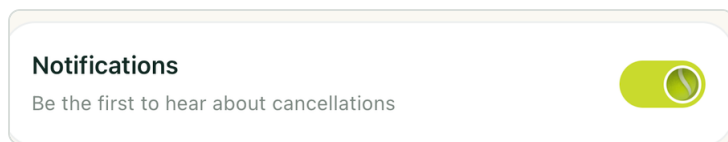


Le message ne se contente pas d'annoncer la mauvaise nouvelle : il donne **le coup d'après**, aller voir les clubs

Anybuddy. Un message qui ne dit que « non » est une plainte. Celui qui propose une route est du design.

III Le curseur de l'interrupteur est une balle de tennis

CE QUE J'AI FAIT	Tout le monde a déjà vu l'interrupteur par défaut du système. J'ai remplacé son curseur par une balle de tennis.
POURQUOI	C'est la seule modification purement décorative de toute l'application, et elle n'améliore aucun indicateur d'efficacité.
COÛT / LIMITE	Mais elle fait lire l'application comme quelque chose dont quelqu'un s'est soucié , plutôt qu'un assemblage de bibliothèque de composants. Je la garde dans le portfolio parce que je considère ce jugement comme faisant aussi partie du design.



Le curseur est une balle de tennis - coutures comprises. Tout le monde a vu le point blanc par défaut ; remplacé par une balle, cet interrupteur n'appartient qu'à cette application.

Il n'améliore aucun indicateur. Mais on le touche à chaque fois qu'on active ou coupe les alertes, et à cet instant le produit a un visage.

11

Trois décisions d'interaction

Une bonne interaction se manifeste souvent par « l'utilisateur n'a pas remarqué que quelque chose avait été traité ».

FR / EN / 中文

IV Zéro inscription — le premier écran est le produit

CE QUE J'AI FAIT

L'ouverture de l'application donne directement sur la recherche, utilisable immédiatement. Pas d'inscription, pas de connexion, pas de carrousel d'accueil. L'identifiant est une chaîne aléatoire générée localement au premier lancement.

POURQUOI

Ce produit démontre sa valeur en trente secondes — « tiens, il y a vraiment un court à cette heure-là ». Tout ce qui se dresse devant ces trente secondes prélève un coût avant que l'utilisateur ait la moindre raison de me faire confiance.

COÛT / LIMITE

Le prix à payer : changer de téléphone fait perdre ses réglages. Je l'accepte — la friction d'inscription, la récupération de mot de passe et la responsabilité RGPD qu'apporte un système de comptes pèsent bien plus lourd que ce qu'il résoudrait.

V La fraîcheur des données est annoncée à voix haute

CE QUE J'AI FAIT

Chaque résultat porte « mis à jour il y a X minutes », avec une actualisation manuelle, un délai de garde sur celle-ci, et la mention que les pages officielles font foi.

POURQUOI

Le plus grand risque de confiance d'un agrégateur, c'est « ce que vous me montrez est-il périmé ? ».

COÛT / LIMITE

Écrire « cela peut avoir quelques minutes de retard » dans l'interface vaut mieux que feindre le temps réel. Un utilisateur qui tombe sur un décalage venant d'un produit qui ne l'a jamais prévenu perd sa confiance d'un coup ; prévenu à l'avance, le même décalage se lit comme normal.

VI Échouer ouvert — plutôt laisser passer que verrouiller à tort

CE QUE J'AI FAIT	Quand l'état de l'abonnement est illisible (coupure réseau, anomalie système), le produit laisse passer au lieu de bloquer.
POURQUOI	Un hoquet réseau qui met dehors quelqu'un qui a payé — et qui lui laisse croire que son abonnement est cassé — est un dommage inacceptable.
COÛT / LIMITE	Le prix : très rarement, un utilisateur non abonné aperçoit brièvement du contenu payant. Des deux maux, celui-ci est nettement le moindre.

12

Du design au code

Les décisions de design ne s'arrêtent pas à la maquette. Ci-dessous : l'idée à gauche, ce qu'elle est devenue à droite.

FR / EN / 中文

Il n'y avait personne à qui passer le relais — j'ai fait le design et l'implémentation. Cela a un vrai avantage : **chaque décision de design devait être constructible, sinon elle mourait sur place.**

« Tennis uniquement » n'est pas un slogan

L'INTENTION

« Extraire la couche verticale d'une plateforme horizontale et la refaire. »
Laissée dans un document, cette phrase est vide.

```
app/sources/anybuddy.py
```

```
CLUB_PAGE = ORIGIN + "/fr/club/{slug}/tennis"
```

L'URL se termine littéralement par `/tennis`. Le padel, le badminton et le squash n'entrent jamais dans nos données, dès la première étape. Le troisième jour porte un commit nommé, en toutes lettres, « exclure les courts de padel ».

Pas un mot du prix ni de l'essai ne doit être écrit en dur

L'INTENTION

Le prix affiché sur le mur payant doit correspondre au débit réel — sinon c'est un motif de refus documenté. Les tarifs Apple, les taux de change et le traitement des taxes diffèrent d'un pays à l'autre.

```
src/iap.js

// Les prix viennent de StoreKit à l'exécution – jamais écrits en dur
export async function loadPrices() {
  const list = await fetchProducts({ skus: SKUS, type: 'subs' })
  return Object.fromEntries(list.map(p => [p.id, {
    price: p.displayPrice,      // le prix localisé donné par Apple
    currency: p.currency,
    trial: trialOf(p),          // l'essai est lu en direct lui aussi
  }]))
}
```

Si aucun essai n'est configuré côté App Store Connect, le mot « gratuit » n'apparaît nulle part dans l'interface : le bouton devient simplement « S'abonner » et la mention de reconduction bascule sur la version sans essai. L'interface n'a aucun moyen de mentir.

La porte d'accès est un fait physique, pas une condition

L'INTENTION

Quelqu'un sans abonnement ne doit pas recevoir de notification. L'approche habituelle : le serveur valide le reçu, décide du droit, décide s'il envoie.

```
le droit est déterminé sur l'appareil

# Pas de droit → on retire le jeton de notification du serveur.
# Dès lors, le serveur ne peut physiquement plus joindre l'appareil.
if not entitled:
    await registerPushToken(user_id, token=None)
```

Le serveur n'a jamais besoin de savoir qui a payé. **Aucune validation de reçu, aucun service d'abonnement tiers, aucune donnée de paiement stockée.** On révoque le jeton et la notification s'arrête — ce n'est pas un `if`, c'est un fait physique.

Une majuscule a rendu les abonnements immortels

L'INTENTION

En testant le parcours d'abonnement avant la publication, le seul cas que je n'arrivais jamais à reproduire était l'expiration.

```
src/iap.js

// Le champ d'Apple est expirationDateIOS – les trois lettres en capitales.
// Le expirationDateIos que j'avais écrit ne renvoyait jamais rien.
const v = p.expirationDateIOS ?? p.expirationDate
```

Les noms de propriétés JS sont sensibles à la casse : la valeur était toujours indéfinie, et le repli « pas de champ d'expiration donc toujours valable » s'appliquait alors à chaque transaction. **Tout compte ayant acheté une seule fois restait déverrouillé à vie.** Ce genre de bogue ne plante jamais et ne journalise rien. Il se contente de vous prendre votre chiffre d'affaires en silence.

Pourquoi ce chapitre existe

Le défaut habituel d'un portfolio de design : de beaux écrans ne montrent pas si la réflexion derrière était réelle, et montrent encore moins si elle était constructible.

Ce chapitre est là pour démontrer deux choses : **que chacune de mes décisions de design a un chemin d'implémentation concret**, et **que je sais parcourir ce chemin moi-même**.

13

La publication

Le dernier jour : faire passer les achats intégrés de « ça a l'air de marcher » à « ça marche ».

FR / EN / 中文

Le 25 août, j'ai parcouru cinq scénarios d'achat dans l'ordre. Les trois premiers sont passés vite. Les deux derniers ont fait sortir quatre vrais bogues — chacun touchant directement la capacité des utilisateurs à payer.

1 La feuille de paiement n'apparaissait jamais

CAUSE	<code>expo-iap</code> <code>5.x</code> <code>request.apple.sk</code> ; le code <code>request.ios.sk</code> attend passait la forme 3.x,
SI NON CORRIGÉ	L'utilisateur appuie sur Acheter, rien ne se passe, et l'interface affiche un vague « échec de l'achat »

2 Les abonnements n'expiraient jamais

CAUSE	<code>expirationDateIos</code> mal capitalisé : l'expiration n'était jamais lue
SI NON CORRIGÉ	Tout compte ayant acheté une fois restait déverrouillé à vie

3 « Restaurer les achats » ne servait pas à ceux qui en avaient besoin

CAUSE	La fonction ne lisait que le cache local, sans jamais se synchroniser avec l'App Store
SI NON CORRIGÉ	Nouveau téléphone ou réinstallation — précisément ceux qui ont besoin de restaurer — ne pouvaient jamais restaurer

4 La durée de l'essai était écrite en dur

CAUSE	« 7 jours gratuits » était une constante dans le code, pas une valeur lue depuis StoreKit
SI NON CORRIGÉ	Sans essai configuré, c'est de la publicité mensongère et un motif de refus documenté

Les trois premiers bogues ne plantent jamais et ne journalisent rien. Ils ne font que coûter du chiffre d'affaires en silence, ou faire refuser l'application.

Une fois corrigés, les cinq scénarios sont passés : achat, déverrouillage du droit, arrêt forcé et relance, restauration, résiliation en cours, et retour à l'état gratuit à l'expiration. Déposé pour validation App Store à 17 h 12 le jour même.

14

Sept jours

Le parcours complet, reconstitué à partir des horodatages et de l'historique git.

FR / EN / 中文

19/08	18:47	●	Cahier des charges v0.1 — la réservation automatique
19/08	19:01	●	Cahier des charges v0.2 — je me contredis : agrégation + alerte
19/08	19:18	●	Scripts de validation : confirmer que la donnée est récupérable
20/08	06:41	●	Prototype d'interface + trois variantes de fond

20/08	11:18	● Premier commit back-end
20/08	14:09	● Premier commit front-end
21/08	11:53	● Anybuddy : tennis uniquement, vraies surfaces détectées
22/08	01:04	● Bandeau de créneaux + nouvelles icônes + prix complet
22/08	08:24	● Exclusion des courts de padel
23/08	19:55	● Anybuddy résolu jusqu'au court individuel
24/08	07:28	● Notifications fonctionnelles de bout en bout
25/08	08:00	● Achats intégrés branchés
25/08	13:20	● Correction des quatre vrais bogues d'achat
25/08	17:12	● Dépôt pour validation App Store

6 j 22 h

Durée totale

52

Commits

14 min

De la v0.1 à la v0.2

1

Personne

Rétrospective

Ce que je pense avoir bien jugé, et ce que je ferais autrement.

FR / EN / 中文

Ce que je pense avoir bien jugé

- **Réfuter l'hypothèse la plus fatale avant de construire.** Quatre-vingt-dix minutes de scripts ont écarté le risque de six jours perdus.
- **Enterrer le plan le plus séduisant sans témoin.** Personne pour surveiller, personne pour le savoir — et c'était pourtant le seul choix juste.
- **Traiter la couleur comme une structure d'information et non comme un ornement.** La déduction par la roue chromatique est la partie du projet dont je suis la plus satisfaite.
- **Implémenter moi-même chaque décision de design.** Cela a fait du « ce n'est pas faisable » une étape du design, et non une mauvaise surprise découverte après le passage de relais.

Ce que je ferais autrement

- **Brancher les achats intégrés bien plus tôt.** Les avoir laissés au dernier jour a fait sortir quatre bogues d'un coup. C'est la partie la moins tolérante à l'erreur et la plus difficile à tester : sa place était au troisième jour.
- **Faire tourner un build de production sur un vrai appareil plus tôt.** Un build de développement et un build signé de production ne sont pas la même chose ; certains problèmes n'apparaissent que sous une vraie signature.

- **Sept jours pour finir un produit, cela se paie en utilisateurs réels.** Toutes les frictions décrites ici viennent de moi et des joueurs que je connais. L'étape suivante doit faire entrer de vrais retours — ce que je veux le plus mesurer, c'est le taux de réception des alertes, et la conversion d'une notification en réservation effective.

Ce que ce projet représente pour moi

Je suis encore étudiante et je veux devenir chef de produit. Je sais qu'écrire « sens du produit » sur un CV ne convainc personne ; j'ai donc choisi de le dire avec quelque chose de réellement construit, réellement publié, et que des inconnus téléchargeront vraiment.

Dans un domaine où l'information était déjà publique, la réorganiser suffit à créer une valeur réelle.

Les utilisateurs paient toujours sur les plateformes officielles. Les courts appartiennent toujours à la ville. La seule chose qui a changé, c'est qu'on sait enfin quand l'un d'eux est libre.



UX CASE STUDY

FR **EN** 中文

Paris Tennis Dispo

Be the first to know when someone cancels

6 days 22 hours From the first requirements doc to App Store submission

Solo — product, design, iOS front end, Python back end, release

Jiayue Liu

August 2026

Contents

FR / EN / 中文

01	At a glance		02	Where it started	This is not a hypothetical user story. This is my own Wednesday evening.
03	Minute fourteen	I finished my first plan and killed it fourteen minutes later.	04	Two battlefields	Giving up the rush is exactly what let me see the fight nobody was having.
05	Prove it, then build it	Before the first line of interface code, I wrote three scripts.	06	Design intent	Four pain points, four principles that had to hold.
07	Information architecture	Three tabs for three states: actively looking, passively waiting, managing myself.	08	The colour system	Seven filter dimensions share one screen. The colours were not picked. They were computed.
09	Three details that improve nothing	The icon, the launch animation, the language button. None of them makes search faster - and together they decide whether the app reads as something a person made.	10	Three interface decisions	Each one can answer "why not the obvious alternative".
11	Three interaction decisions	Good interaction usually shows up as "the user never noticed something was handled".	12	From design to code	Design decisions do not stop at the mockup. Below, the idea is on the left and what it became is on the right.
13	Shipping	The last day: taking in-app purchase from "looks like it works" to "actually works".	14	Seven days	The full run, reconstructed from file timestamps and git history.
15	Looking back	What I think I got right, and what I would do differently.			

At a glance

FR / **EN** / 中文

One person, seven days, turning "I can't get a tennis court" into a shipped iOS product.



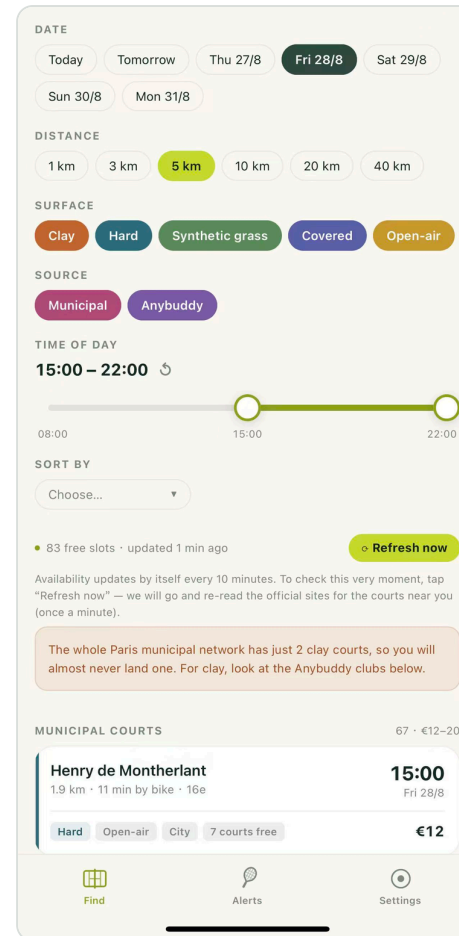
Find • 中文

All six filter dimensions on the first screen — nothing hidden in a drawer



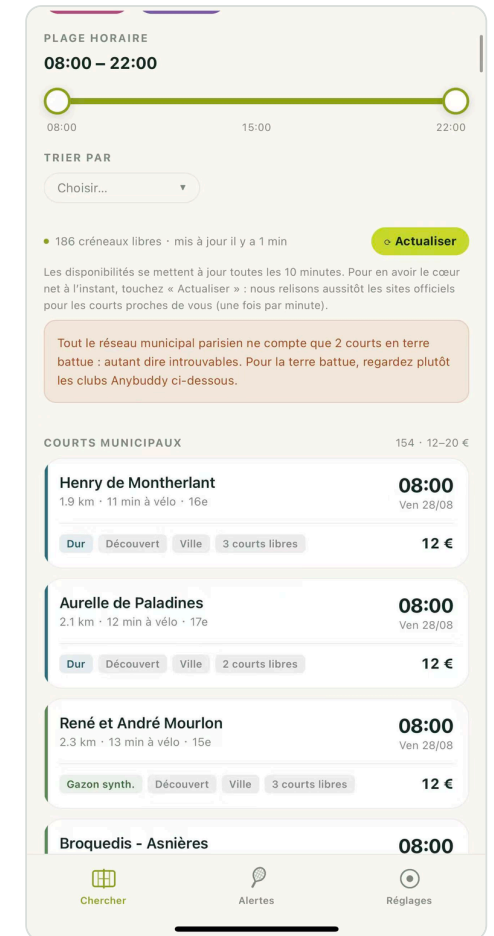
Results • 中文

Municipal and Anybuddy in one list, one sort, one filter set



Find • English

Same structure in three languages; the layout reflows itself



Chercher • Français

French is the local default, not a translated afterthought

MY ROLE	Solo — requirements, design, iOS front end, Python back end, release
TIMEFRAME	19–25 August 2026 — 6 days 22 hours
OUTCOME	iOS 1.0.0 submitted for App Store review; trilingual UI; subscription
STACK	React Native (Expo) • StoreKit 2 • Python / FastAPI • PostgreSQL • Railway
SCALE	54 venues monitored continuously • 202 municipal courts • one sweep every 10 minutes (as of August 2026)

6d 22h

From idea to submission

52

Commits (29 front end / 23 back end)

3

Languages, all written by me

14 min

Time it took to reject my own first plan

02

Where it started

This is not a hypothetical user story. This is my own Wednesday evening.

FR / EN / 中文

I play tennis in Paris. When I want a court, I have to go to the city's booking site and click through it cell by cell: pick a centre, pick a date, pick a time, submit, wait for the page, read "no slots available", go back, try another time, and do it all again.

After about twenty of those, I usually close the tab thinking the sentence every product should fear most —

"Forget it. Not playing today."

The moment that thought appears is the problem the product has to solve. It is not that any single click is slow. It is that **the attention the whole thing costs is high enough to make people quit.**

My first motive was entirely selfish: I wanted to get on a court. Only later did I find out how common the problem is among tennis players here.

Minute fourteen

I finished my first plan and killed it fourteen minutes later.

FR / EN / 中文

The first plan was an auto-booking tool: store the user's credentials, and grab a court for them the instant slots open. **It would have worked, and it should not have been built.**

v0.1 • Auto-booking

- Requires storing user passwords for a third-party platform
- Plainly violates the platforms' anti-automation terms
- Very likely rejected by App Review
- Crowds other people out of a public resource
- One scraping run per user, forever

v0.2 • Aggregate + alert

- No passwords, no accounts, nothing to store
- Uses only publicly visible data
- An established category — shippable
- Takes no one's court; only makes the information arrive sooner
- Scrape once centrally, everyone shares it

Once the two columns were side by side there was nothing left to think about. The file timestamps are unambiguous: v0.1 written at **18:47**, v0.2 at **19:01**.

This is the page I most want to be judged on

Not because I got it right, but because **I killed the sexier plan while nobody was watching**. Auto-booking sounds far more impressive and would have made a better demo. It also bet the user's passwords, the platforms' rules and the fairness of a public resource all at once.

A product manager's value is not in thinking up features. It is in knowing which features should not exist.

The pivot cost something real: I gave up any chance of winning the popular courts at the moment they are released. But it was while pricing that loss that I found the

product's actual opportunity — next page.

04

Two battlefields

Giving up the rush is exactly what let me see the fight nobody was having.

FR / EN / 中文

"Getting a popular court" looks like one problem. Pull it apart and it is two completely different ones.

BATTLEFIELD A

The release rush

- Same moment every day
- Hundreds of people refreshing at once
- Gone in seconds
- Won on reflexes and latency

Unwinnable. Deliberately abandoned.

BATTLEFIELD B

Courts freed by cancellations

- Random, all day long
- Almost nobody watching
- First to see it books it
- Won on who finds out first

This is the opening.

No human refreshes a website two hundred times a day. A server can sweep the whole city every ten minutes.

Once that was clear, the centre of gravity moved from search to monitoring — which is to say **this product's most valuable moment happens while the user is not looking at it.**

That judgement set everything that followed: alerts became a primary tab rather than a side feature; search stays free forever and only alerting is paid; and the quality of the push notifications became the single most important engineering problem in the project.

Prove it, then build it

Before the first line of interface code, I wrote three scripts.

FR / EN / 中文

The whole plan rested on one assumption: **the availability data on both platforms can be read without logging in**. If that were false, every design decision after it would be built on air.

So between 19:18 and 20:41 on 19 August I wrote three throwaway scripts that did exactly one thing — pull real availability down and see what shape it was in, how fast it moved, and whether the fields were enough.

- ① `v.py` — can the municipal availability pages be fetched at all?
- ② `v2.py` — can what comes back be parsed into structured slots?
- ③ `v3.py` — can both platforms be normalised into one model?

Why this belongs in a portfolio

Because of what the **order** says. Most people's instinct is to draw screens first — screens look good, they can be shown to people, they feel like progress. But if the data cannot be had, the prettiest screen in the world is worth nothing.

Ninety minutes spent trying to falsify the most fatal assumption, then six days building. That sequence is not an efficiency trick. It is a judgement call.

Only once the scripts ran did I learn the numbers: 43 municipal tennis centres, 202 courts — and **exactly 2 of them are clay**. That number turned directly into an interface decision — see section 09.

Design intent

Four pain points, four principles that had to hold.

FR / EN / 中文

Design does not start at "I would like to make a nice-looking screen". It starts at "where does the user get stuck". So I laid the observed pain points out one by one, and derived from each a principle the product was not allowed to break.

WHERE USERS STALL You only learn whether a slot exists by opening it — 301 page round trips just to see one week	SO THIS MUST HOLD Put availability in front of the choice, not behind it	WHAT IT LOOKS LIKE One list showing every matching slot at once; filtering narrows, it does not probe
WHERE USERS STALL Two platforms, switched back and forth, impossible to compare	SO THIS MUST HOLD Compare inside one view, not inside one head	WHAT IT LOOKS LIKE One data model — municipal and Anybuddy share a screen, a sort and a filter set
WHERE USERS STALL Tennis gets diluted inside a multi-sport app	SO THIS MUST HOLD Promote what tennis players actually care about to the first screen	WHAT IT LOOKS LIKE Surface, roof, distance and time as toggles; the scraper reads only the <code>/tennis</code> sub-page
WHERE USERS STALL The real opportunity is a random cancellation nobody can sit and watch for	SO THIS MUST HOLD The product has to work while the user is not looking	WHAT IT LOOKS LIKE Continuous background monitoring plus push; alerts as a primary tab

Behind every design principle there has to be a specific person at a specific moment.

Information architecture

Three tabs for three states: actively looking, passively waiting, managing myself.

FR / **EN** / 中文

Find

I want to play right now

Six filter dimensions + one cross-platform list + three sort orders

Alerts

I am waiting for someone to cancel

Watch conditions + "when I can play" + push

Settings

I want to change something

Home address, language, notifications, subscription

One real, unedited run



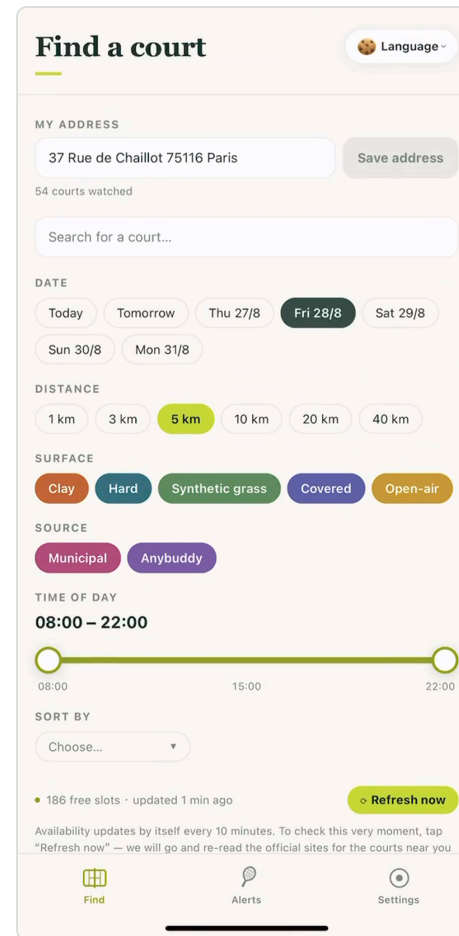
00:00

Cold start lands straight in Find



00:06

Filters too tight → empty state, with a way out



00:15

Switch to English: no reload, every filter kept



00:21

Switch to French, list unchanged

Twenty-seven seconds, no cuts. From cold start to results **not one step asks who you are**. Language switching is instant — no restart, no lost filters, no bounce back to the home screen — because language is only a copy layer and never touches state.

One deliberate duplication

Find and Alerts use the same filter vocabulary, but each keeps **its own independent state**.

The cost is that users configure things twice. I accept it, because "what I am looking for right now" and "what I want to be told about" are two different intents — one is momentary, the other standing. Sharing state would let a casual search silently poison a long-lived alert rule.

To compensate, the Alerts setup is deliberately lighter: its time window is bound to a standing "when I can play" schedule, set once per weekday and left alone.

08

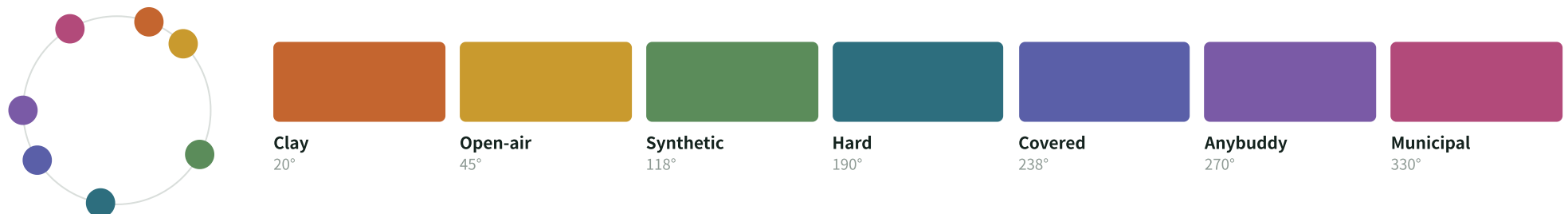
The colour system

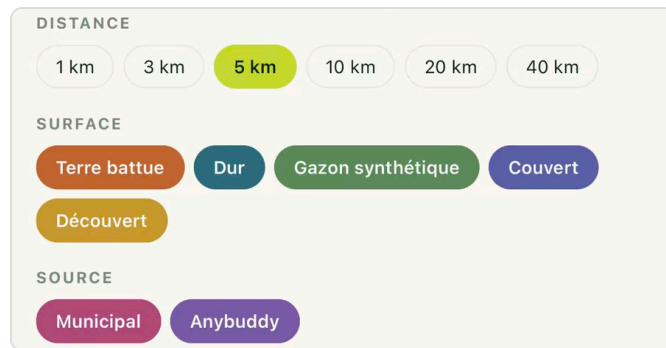
Seven filter dimensions share one screen. The colours were not picked. They were computed.

FR / EN / 中文

The Find screen has to show, all at once: three court surfaces, covered vs open-air, two data sources, and the date. Pick those colours by eye and they interfere on

screen, leaving the user to read every label word by word. So I spaced them **evenly around the hue wheel**:





Seven hues on one screen, as shipped. Five surface colours above, two source colours below. Any two neighbouring chips sit far enough apart in hue that you could shrink this image to a thumbnail and still tell them apart.

Note that the DISTANCE row above uses the tennis yellow instead — a different family entirely, because distance is a **single choice**, not a category.

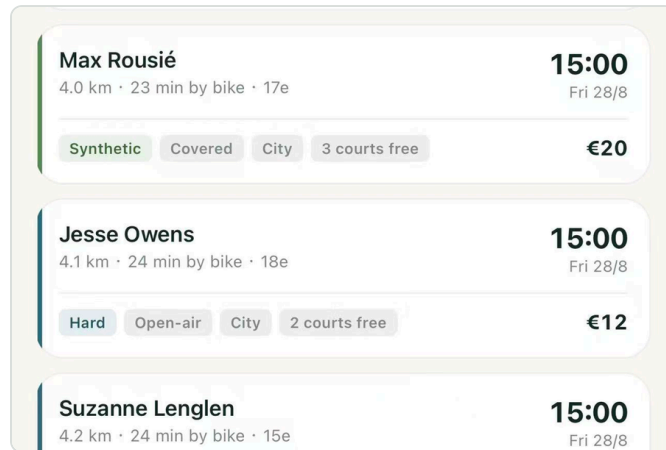
- The first five hues are roughly equidistant on the wheel, so **any two adjacent chips differ enough in hue** to be told apart without reading
- The two source colours were placed in the purple and magenta gaps that were left, so they never collide with a surface colour
- **Each hue carries its own memory hook:** terracotta for clay, grass green for synthetic turf, sunlit gold for open-air, shaded indigo for covered

The date row deliberately did not get an eighth hue

The date strip uses a deep pine green close to the body text, not another vivid hue — because **the date is an axis, not a category**. It is not the same kind of thing as the filter rows below it.

Give it a hue and users read it as a peer of surface and source. Using "a different kind of colour" to express "a different layer of structure" does more work than any extra divider line.

The payoff: every result card carries a vertical bar in its surface colour down the left edge. Scrolling fast, **peripheral vision alone rules out the surfaces you do not want**. The text labels are still there, but their job drops from identification to confirmation.



That bar is where the whole colour system gets paid off. The upper card is synthetic grass (green), the lower is hard court (blue) — the cards are otherwise identical, and the entire distinction is carried by 3 dp of colour

down the left edge.

Spend colour on being instant, not on being pretty.

Three details that improve nothing

The icon, the launch animation, the language button. None of them makes search faster - and together they decide whether the app reads as something a person made.

FR / EN / 中文

Utility products drift towards the same face: usable, correct, expressionless. I deliberately spent time in three places that move no efficiency metric at all.

I - The icon is not a tennis ball



A Paris sparrow in a tricolore headband, standing on a court, three balls beside it.

The default answer is to draw a tennis ball. It takes one second to think of and one second to forget. Search "tennis" on the App Store and you

get a screen of yellow-green circles.

The sparrow is the commonest bird on a Paris street; the tricolore band is the city. **An icon should not answer "which sport" - it should answer "which sport, in which city".**

II - The 0.86 seconds at launch

The native Expo splash is a **static image**, so its text cannot follow the language. I split the launch into two layers: the native one carries the icon and not a single

word; the instant it disappears, a code-rendered layer takes over, and only then can the tagline follow the system language into French, English or Chinese.



The ball falls 160pt from above the screen over 520 ms on `cubic-bezier(0.34, 1.56, 0.64, 1)`. **The second control point has $y = 1.56$, greater than 1 - which is why the ball overshoots its landing and springs back.** Once it settles, the text fades in over 340 ms. 0.86 seconds in total: not slow, but far more alive than a hard cut.

These frames are not screenshots. They were recomputed from the bezier parameters and the seam geometry in the source, so each one sits at the exact position of that millisecond.

The seams are calculated, not traced

Those two arcs are not drawn curves. They are **the edges of two much larger circles**. Make the circles small and the arcs cross near the centre of the ball, and the ball turns into an almond.

The values that work: seam radius $1.62r$, seam centre $1.96r$ from the ball centre - so each arc reaches at most $0.34r$ past the centre, which is what a tennis ball actually looks like. The whole ball is assembled from Views rather than an image, so it stays sharp at any screen density.

III - Languages get pastries, not flags

A flag is not a language. Which flag belongs to English - the United States, the United Kingdom, Australia? And which to Chinese? Worse, a row of national flags inside a French product reads more like politics than like language.



中文

Dumpling



English

Cookie



Français

Croissant



Language ▾

The three pastries stand for no country. They stand for **what tends to be on the table in the morning where that language is spoken**. No ambiguity, no position taken - and still recognisable shrunk to 15pt.

The pill sits in the same corner on all three tabs. Switching language is something people do at any moment; it does not belong three levels deep in Settings.

10

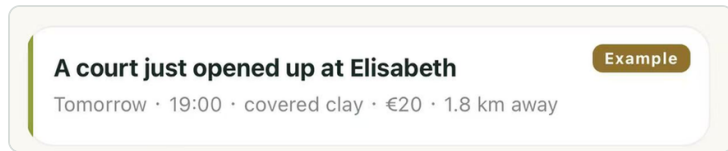
Three interface decisions

Each one can answer "why not the obvious alternative".

FR / EN / 中文

I The paywall shows the product itself, not adjectives about it

- WHAT The first thing on the subscription screen is not a feature list. It is a sample notification card that looks exactly like the real thing.
- WHY Before deciding whether to pay, a user has to see what they are actually buying. What three lines of adjectives cannot convey, one sample card conveys in two seconds.
- COST / EDGE Precisely because it looks real, the "Example" badge in its corner cannot be dropped — without it someone will believe a court has genuinely opened up and make a wasted trip. Trust earned by realism must not be cashed out through misdirection.

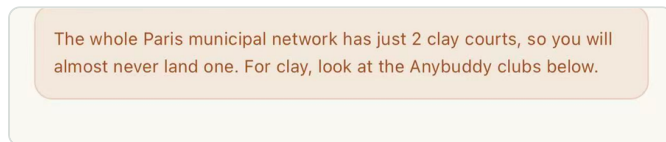


This is the sample card that opens the subscription screen. It uses the same component and the same typography as a real alert - venue name, tomorrow 19:00, covered clay, EUR 20, 1.8 km away.

The "Example" badge in the corner is the one stroke this decision cannot do without: the more real the card looks, the higher the cost of leaving it unmarked.

II The clay warning appears only when it is relevant

- WHAT Selecting "Clay" surfaces an explanation — but only when **Municipal and Clay are both selected**.
- WHY Of 202 municipal courts in Paris, exactly 2 are clay. Without the explanation, users conclude the app is broken.
- COST / EDGE But if the user is not looking at municipal courts at all, the same message is pure noise. The tighter the condition, the more the message is worth.

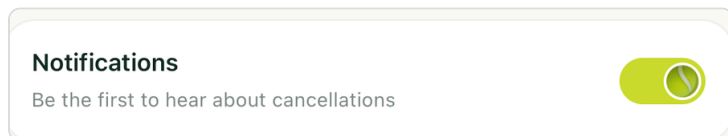


The message does not just deliver the bad news; it hands over **the next move**: go look at the Anybuddy

clubs. A message that only says "no" is a complaint. One that offers a route is design.

III The toggle knob is a tennis ball

- WHAT Everyone has seen the system default switch. I replaced the knob with a tennis ball.
- WHY It is the one purely decorative change in the entire app, and it improves no efficiency metric whatsoever.
- COST / EDGE But it makes the app read as **something a person cared about**, rather than something assembled from a component library. I keep it in the portfolio because I believe that judgement is also part of design.



The knob is a tennis ball - seams and all. Everyone has seen the default white dot; replaced by a ball, this switch belongs to this app and no other.

It improves no efficiency metric. But users touch it every time they turn alerts on or off, and for that moment the product has a face.

Three interaction decisions

Good interaction usually shows up as "the user never noticed something was handled".

FR / EN / 中文

IV Zero sign-up — the first screen is the product

WHAT	Opening the app drops you straight into search, usable immediately. No sign-up, no login, no onboarding carousel. Identity is a random string generated locally on first launch.
WHY	This product proves its value in thirty seconds — "oh, there really is a court at that time". Anything standing in front of those thirty seconds extracts a cost before the user has any reason to trust me.
COST / EDGE	The cost is that changing phones loses your settings. I accept it: the sign-up friction, password recovery and GDPR exposure that an account system brings all outweigh what it would solve.

V Data freshness is stated out loud

WHAT	Every result is stamped "updated X minutes ago", with a manual refresh, a cooldown on that refresh, and a note that the official pages are the final authority.
WHY	The biggest trust risk in any aggregator is "is what you are showing me stale?"
COST / EDGE	Writing "this may lag by a few minutes" into the interface beats pretending to be live. A user who hits one mismatch from a product that never warned them loses trust all at once; warned in advance, the same mismatch reads as normal.

VI Fail open — rather let one through than wrongly lock one out

WHAT	When subscription status cannot be read (network hiccup, system error), the product lets the user through instead of blocking them.
WHY	A network blip locking out someone who paid — and leaving them thinking their subscription is broken — is an unacceptable amount of damage.
COST / EDGE	The cost is that, very rarely, an unsubscribed user briefly sees paid content. Weighed against the alternative, this one is clearly the lighter harm.

From design to code

Design decisions do not stop at the mockup. Below, the idea is on the left and what it became is on the right.

FR / EN / 中文

There was nobody to hand this project off to — I did both the design and the implementation. That has one real benefit: **every design decision had to be buildable, or it died on the spot.**

Tennis only is not a slogan

THE INTENT

"Take the vertical layer out of a horizontal platform and rebuild it." Left in a document, that sentence is empty.

```
app/sources/anybuddy.py

CLUB_PAGE = ORIGIN + "/fr/club/{slug}/tennis"
```

The URL literally ends in `/tennis`. Padel, badminton and squash inventory never enters our data in the first place. Day three has a commit named, in so many words, "exclude padel courts".

Not one word of the price or trial may be hard-coded

THE INTENT

The price on the paywall has to match the actual charge — otherwise it is a documented rejection reason. Apple's pricing, exchange rates and tax handling differ by country.

```
src/iap.js

// Prices come from StoreKit at run time — never hard-coded
export async function loadPrices() {
  const list = await fetchProducts({ skus: SKUS, type: 'subs' })
  return Object.fromEntries(list.map(p => [p.id, {
    price: p.displayPrice,      // Apple's localised price
    currency: p.currency,
    trial: trialOf(p),         // the trial is read live too
  }]))
}
```

If no trial is configured in App Store Connect, the word "free" never appears anywhere in the UI — the button simply becomes "Subscribe" and the auto-renewal disclosure swaps to the version without a trial. The interface has no way to lie.

The entitlement gate is a physical fact, not a conditional

THE INTENT

Someone without a subscription should not receive push. The usual approach: the server validates the receipt, decides entitlement, decides whether to send.

```
entitlement is decided on-device

# Not entitled → pull the push token off the server.
# From then on the server physically cannot reach this device.
if not entitled:
    await registerPushToken(user_id, token=None)
```

The server never needs to know who paid. **No receipt validation, no third-party subscription service, no payment data stored anywhere.** Revoke the token and push stops — that is not an `if`, it is a physical fact.

One capitalisation bug made subscriptions immortal

THE INTENT

Testing the subscription flow before release, I found the one path I could never reproduce was expiry.

```
src/iap.js

// Apple's field is expirationDateIOS – all three letters capital.
// The expirationDateIos I had written never resolved.
const v = p.expirationDateIOS ?? p.expirationDate
```

JS property names are case-sensitive, so the value was always undefined — and the "no expiry field means still valid" fallback then held for every transaction. **Any account that ever bought once was unlocked forever.** This class of bug never crashes and never logs. It just quietly takes your revenue.

Why this chapter exists

The usual problem with a design portfolio is that beautiful screens do not show whether the thinking behind them was real, and show even less whether it could actually be built.

This chapter is here to demonstrate two things: **that my design decisions each have a concrete implementation path**, and **that I can walk that path myself**.

Shipping

The last day: taking in-app purchase from "looks like it works" to "actually works".

FR / EN / 中文

On 25 August I worked through five purchase paths in order. The first three passed quickly. The last two forced out four real bugs — every one of them affecting whether users can pay.

1 The payment sheet never appeared

CAUSE	expo-iap	5.x expects	request.apple.skus; the code passed the 3.x shape	request.ios.skus
IF	The user taps Buy, nothing happens, and the UI shows a vague			
UNFIXED	"purchase failed"			

2 Subscriptions never expired

CAUSE	expirationDateIos	was miscapitalised, so expiry was never read
IF	Any account that bought once stayed unlocked forever	
UNFIXED		

3 "Restore purchases" did nothing for the people who needed it

CAUSE	It only read the local cache and never synced with the App Store
IF	New phone or fresh install — exactly the people who need restore —
UNFIXED	could never restore

4 The trial length was hard-coded

CAUSE	"7 days free" was a literal in the code rather than a value read from StoreKit
IF	With no trial configured, that is false advertising and a documented
UNFIXED	rejection reason

The first three bugs never crash and never log an error. They only quietly cost revenue, or get you rejected at review.

With those fixed, all five paths passed: purchase, entitlement unlock, kill-and-relaunch, restore, cancel mid-term, and fall back on expiry. Submitted for App Store review at 17:12 the same day.

Seven days

The full run, reconstructed from file timestamps and git history.

FR / **EN** / 中文

08-19	18:47	● Requirements v0.1 — the auto-booking plan
08-19	19:01	● Requirements v0.2 — overruled myself; aggregate + alert
08-19	19:18	● Validation scripts: confirm the data can be fetched
08-20	06:41	● UI prototype + three background-colour variants
08-20	11:18	● First back-end commit
08-20	14:09	● First front-end commit
08-21	11:53	● Anybuddy: tennis only, with real surface detection
08-22	01:04	● Time-of-day slider + new icons + full price display
08-22	08:24	● Exclude padel courts
08-23	19:55	● Anybuddy pinned down to individual courts
08-24	07:28	● Push notifications working end to end
08-25	08:00	● In-app purchase wired up
08-25	13:20	● Fixed the four real IAP bugs
08-25	17:12	● Submitted for App Store review

6d 22h

Total span

52

Commits

14 min

From v0.1 to v0.2

1

Person

Looking back

What I think I got right, and what I would do differently.

FR / EN / 中文

What I think I got right

- **Falsify the most fatal assumption before building.** Ninety minutes of validation scripts removed the risk of six wasted days.
- **Killing the sexier plan with nobody watching.** No supervision, no audience — and still the only correct call.
- **Treating colour as information structure rather than decoration.** The hue-wheel derivation is the part of this project I am happiest with.
- **Implementing every design decision myself.** It made "that cannot be built" part of the design process rather than a surprise discovered after handoff.

What I would do differently

- **Wire up in-app purchase far earlier.** I left it to the final day and four bugs surfaced at once. It is the part of the product least able to tolerate mistakes and the hardest to test — it belonged on day three.
- **Run a production build on a real device sooner.** A development build and a signed production build are not the same thing; some problems only appear under a real signature.

- **Seven days to a finished product costs you real users.** Every pain point here comes from me and from the players I know. The next step has to bring real user feedback in — what I most want to measure is the delivery rate of the alerts, and the conversion from a push to an actual booking.

What this project means to me

I am still a student and I want to be a product manager. I know that writing "product thinking" on a CV convinces nobody, so I chose instead to say it with something that was really built, really shipped, and will really be downloaded by strangers.

In a domain where the information was already public, reorganising it is by itself worth something real.

Users still pay on the official platforms. The courts still belong to the city. The only thing that changed is that people finally know when one is free.

Paris Tennis Dispo

UX Case Study • Jiayue Liu • August 2026



UX CASE STUDY

FR EN 中文

Paris Tennis Dispo

在巴黎,最先知道有人取消

6 天 22 小时 从第一份需求文档到提交 App Store 审核

独立完成 · 产品 / 设计 / 前后端

刘嘉悦 Jiayue Liu

2026 年 8 月

目录

FR / EN / 中文

01 项目速览

03 第 14 分钟

我写完第一版方案,十四分钟后推翻了它。

05 先证明,再动手

在写第一行界面代码之前,我先写了三个脚本。

07 信息架构

三个 Tab,对应三种状态:主动找、被动等、管自己。

09 三个不提升指标的细节

图标、开屏、语言按钮。没有一个能让搜索变快——但它们决定了这个 App 像不像一个人做的。

11 三个交互决策

好的交互常常表现为「用户没注意到有什么被处理掉了」。

13 上线

最后一天,把内购从「看起来能用」测到「真的能用」。

15 复盘

做完之后,我认为哪些判断是对的,哪些下次会不一样。

02 起点

这不是一个假想的用户故事。这是我自己的周三晚上。

04 两个战场

放弃抢票,反而让我看见了一个没人在打的战场。

06 设计目的

三个痛点,推导出四条设计原则。

08 色彩系统

七个筛选维度要同屏出现。颜色不是挑的,是算的。

10 三个界面决策

每一个都能回答「为什么不是别的做法」。

12 从设计到代码

设计决策不是停在图上的。下面每一条,左边是想法,右边是它在代码里的样子。

14 七天

文件时间戳和 git 提交记录还原的完整过程。

项目速览

七天,一个人,把「我自己订不到网球场」这件事,做成了一个提交上架的 iOS 产品。

我的角色	独立完成 —— 需求、设计、iOS 前端、Python 后端、上架
周期	2026 年 8 月 19 日 - 8 月 25 日,共 6 天 22 小时
成果	iOS 1.0.0 已提交 App Store 审核;三语界面;订阅制
技术栈	React Native (Expo) • StoreKit 2 • Python / FastAPI • PostgreSQL • Railway
规模	54 个场馆持续监控 • 202 片市政球场 • 每 10 分钟一轮(截至 2026 年 8 月)

6 天 22 小时

从想法到提交审核

52

次提交(前端 29 / 后端 23)

3

种语言,全部由我撰写

14 分钟

推翻第一版方案所用的时间



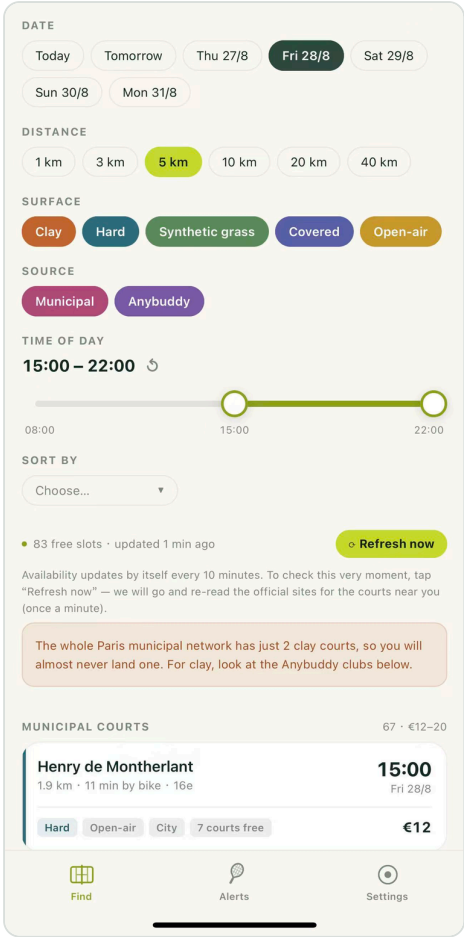
找场 · 中文

六个筛选维度全部摊在第一屏,不藏进抽屉



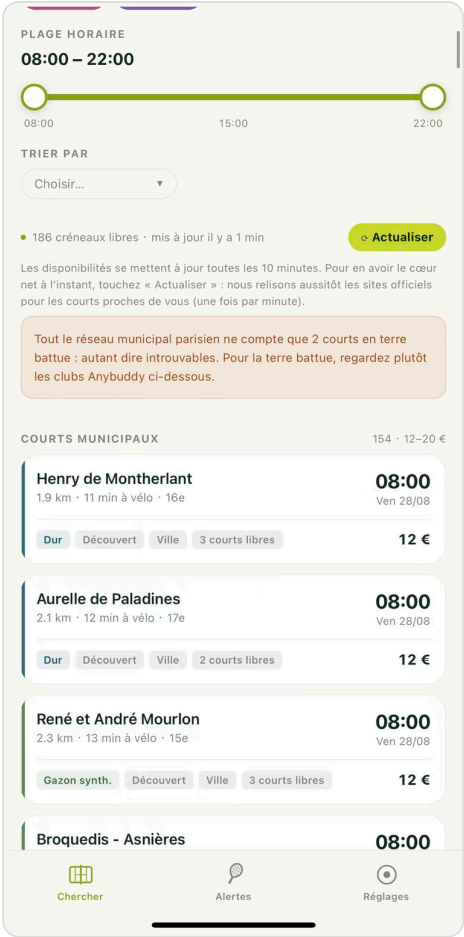
结果 · 中文

市政与 Anybuddy 同屏、同排序、同筛选



Find · English

同一套结构,三种语言;界面自己会重排



Chercher · Français

法语是本地用户的默认语言,不是「翻译版」

起点

这不是一个假想的用户故事。这是我自己的周三晚上。

FR / EN / 中文

我在巴黎打网球。想打球的时候,得去市政的预订网站上一格一格点:选球场、选日期、选时段、提交、等页面加载、看到「无可用地」,退回去、换一个时段、再来一遍。

点了二十几次之后,我通常会关掉网页,心里想的是那句所有产品都最怕的话——

「算了,今天不打了。」

这个念头出现的那一刻,就是产品要解决的问题。不是某一次点击太慢,而是**整件事的注意力成本高到让人放弃**。

我做这个产品,最初的动机非常自私:我想让自己能打上球。后来才发现,这个问题在巴黎的网球圈里是普遍的。

第 14 分钟

我写完第一版方案,十四分钟后推翻了它。

FR / EN / 中文

第一版方案叫「代订工具」:存下用户的账号密码,在放号的瞬间自动帮他抢场。**它能解决问题,但它不该被做出来。**

v0.1 · 自动代订

- 必须存储用户在第三方平台的密码
- 明确违反对方的反自动化条款
- App Store 大概率拒审
- 挤占其他人的公共资源
- 每个用户都要单独跑一份抓取

v0.2 · 聚合 + 提醒

- 完全不需要密码,不需要账号体系
- 只使用公开可见的数据
- 成熟品类,可以上架
- 不抢任何人的场,只是让信息更快到达
- 中心化抓一次,所有人共享

两栏并排一摆,就没什么可犹豫的了。文件时间戳记得很清楚:v0.1 写于 **18:47**,v0.2 写于 **19:01**。

这一页是我最想被看见的一页

不是因为我做对了,而是因为**我在只有自己一个人的时候,主动否掉了一个更「性感」的方案**。自动抢场听起来厉害得多,做出来也更容易拿去炫耀。但它把用户的密码、平台的规则和公共资源的公平性一起赌上了。

产品经理的价值不在于想出功能,而在于知道哪些功能不该存在。

这次转向的代价是实打实的:我放弃了在放号那一瞬抢热门场的能力。但恰恰是在掂量这个代价的时候,我看见了产品真正的机会 —— 见下一页。

04

两个战场

放弃抢票,反而让我看见了一个没人在打的战场。

FR / EN / 中文

「订到热门场」看起来是一件事。拆开之后,是两件完全不同的事。

战场 A

放号瞬间

- 每天固定时刻
- 几百人同时刷新
- 几秒内抢光
- 决胜靠手速和延迟

打不赢。主动放弃。

战场 B

别人取消腾出的场

- 全天随机发生
- 几乎无人盯着
- 谁先看到谁订到
- 决胜靠「谁先知道」

这才是机会。

人不可能一天刷两百次网站。服务器可以每十分钟扫一遍全城。

想清楚这一点之后,产品的重心从「搜索」挪到了「监控」——也就是说,这个产品最有价值的时刻,发生在用户没有打开它的时候。

这个判断定下了后面所有的设计:我把捡漏提醒做成了主 Tab,而不是一个附属功能;搜索永久免费,只有提醒收费;推送的质量控制,成了整个项目最难的工程问题。

05

先证明,再动手

在写第一行界面代码之前,我先写了三个脚本。

FR / EN / 中文

整个方案建立在一个假设上:两个平台的空场数据,不登录就能拿到。如果这个假设不成立,后面所有设计都是空中楼阁。

所以在 08-19 晚上 19:18 到 20:41 之间,我写了三版验证脚本,只做一件事——把真实的空场数据抓下来,看看结构是什么样、更新有多快、字段够不够用。

- ① `v.py` —— 能不能抓到市政的可用性页面
- ② `v2.py` —— 抓到的数据能不能解析成结构化的场次
- ③ `v3.py` —— 两个平台的数据能不能归一化成同一个模型

为什么这一步值得写进作品集

因为它体现的是**顺序**。很多人的第一反应是先画界面——界面好看、能给人看、有成就感。但如果数据拿不到,再好的界面也是废的。

先花一个半小时证伪最致命的假设,再花六天做产品。这个顺序不是效率问题,是判断力问题。

脚本跑通之后我才拿到这几个数:市政有 43 个网球中心、202 片球场,其中**只有 2 片红土**。这个数字后来直接长成了一条界面设计——见第 10 章。

06

设计目的

三个痛点,推导出四条设计原则。

FR / EN / 中文

设计不是从「我想做个好看的界面」开始的,是从「用户卡在哪里」开始的。我把观察到的痛点一条条摆出来,每一条推出一个必须成立的设计原则。

用户卡在哪 只有点开才知道有没有 —— 一周的市政市场要 301 次页面往返才看得完	于是必须成立 把「可用性」摆到选择的前面	产品里长什么样 一个列表,一次呈现所有符合条件的场次;筛选是收窄,不是试探
用户卡在哪 两个平台来回切,无法横向比较	于是必须成立 同一个视图里比较,而不是同一个脑子里比较	产品里长什么样 统一数据模型,市政与 Anybuddy 同屏、同排序、同筛选
用户卡在哪 网球在综合运动 App 里被稀释	于是必须成立 把网球用户真正在乎的维度提到第一屏	产品里长什么样 材质、有顶、距离、时段做成开关;抓取只读 /tennis 子页面
用户卡在哪 真正的机会是随机发生的取消,人盯不住	于是必须成立 产品必须在用户不看它的时候工作	产品里长什么样 后台持续监控 + 推送;捡漏做成主 Tab

每一条设计原则背后,都必须站着一个具体的人和一个具体的时刻。

07

信息架构

三个 Tab,对应三种状态:主动找、被动等、管自己。

FR / EN / 中文

找场

我现在就想打球

六个维度的筛选开关 + 跨平台统一列表 + 三种排序

捡漏

我等着有人取消

监控条件 + 「我能打球的时间」+ 推送

设置

我要改点东西

出发地、语言、通知、订阅

一次真实的连续操作



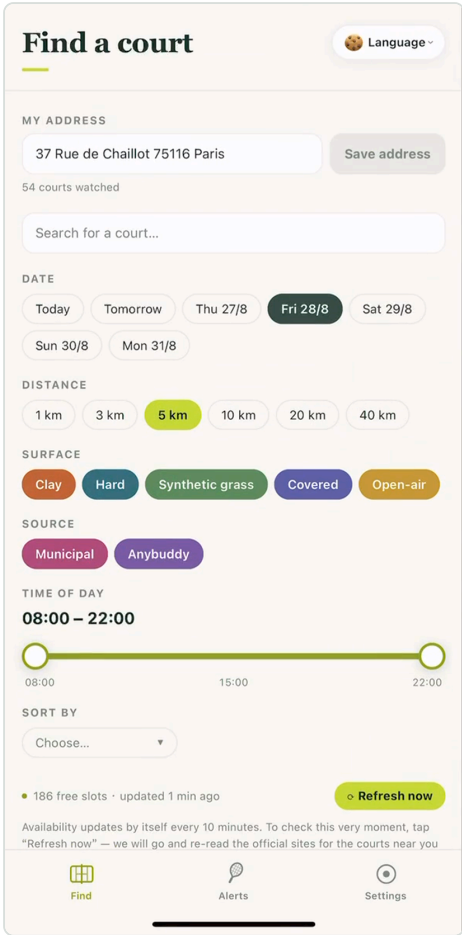
00:00

冷启动直接进「找场」



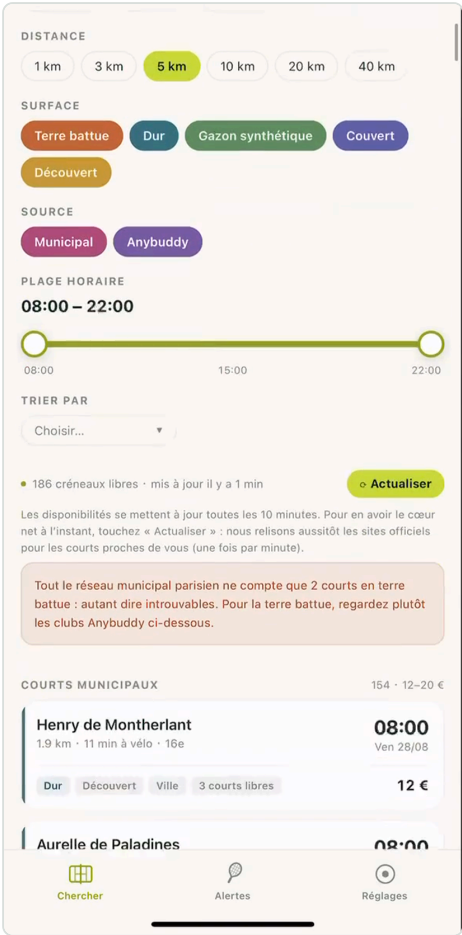
00:06

筛太窄 → 空态,并给出放宽建议



00:15

切英文,页面不重载、筛选条件原样保留



00:21

再切法文,列表照旧

整段 27 秒,没有剪辑。从冷启动到看见结果没有任何一步需要输入身份;语言切换是即时的,不重启、不丢筛选条件、不回到首页 —— 因为语言只是一层文案映射,不参与状态。

一个刻意的重复

「找场」和「捡漏」用的是同一套筛选维度,但它们各自维护独立的状态。

代价是用户要配置两次。我接受这个代价,因为「我现在想找什么」和「我希望被提醒什么」是两种不同的意图 —— 前者是临时的,后者是长期的。共用一套状态,会让一次随手搜索污染掉长期的提醒规则。

作为补偿,「捡漏」的配置被刻意做得更轻:时段绑定到一个长期的「我能打球的时间」,按星期几设一次,长期有效。

08

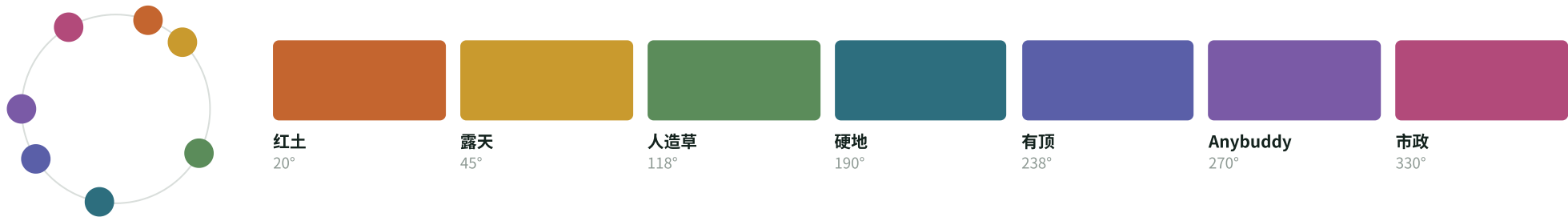
色彩系统

七个筛选维度要同屏出现。颜色不是挑的,是算的。

FR / EN / 中文

「找场」页要同时摆出三种场地材质、有顶 / 露天、两个数据来源,再加一条日期。颜色要是随手挑,它们挤在一屏里就会互相干扰,用户只能一条条去读文字。

所以我把它们按色相环等距排开:



七个色相同屏的真实样子。材质五色在上,来源两色在下。任意两个相邻标签的色相差都足够大 —— 把这张图缩到指甲盖大小,你依然分得清哪块是哪块。

注意最上面那排「距离」用的是网球黄,和下面所有分类色都不同族 —— 因为它是**单选**,不是分类。

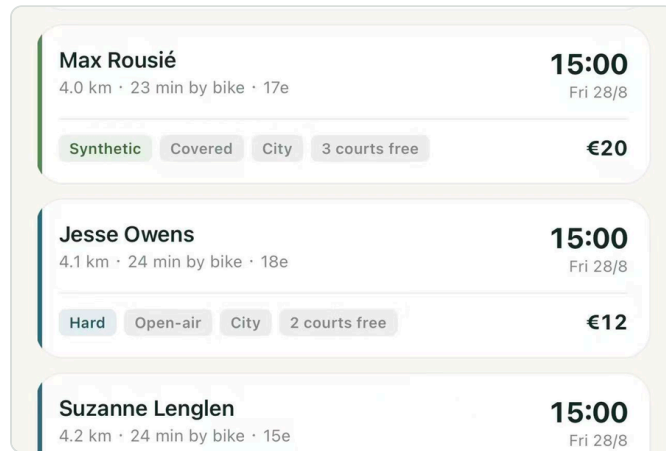
- 前五个色相在环上大致等距,任意两个相邻标签的色相差都足够大,不看文字也能区分
- 来源的两个颜色刻意落在紫和玫红这两个剩下的空档里,避免和材质色撞车
- 语义自带记忆点:红土用陶土色、人造草用草绿、露天用日晒的金、有顶用背阴的靛蓝

日期刻意没有拿到第八个色相

日期条用的是接近正文的深松绿,不是一个新的鲜艳色相。因为**日期是「轴」,不是「类别」**——它和下面那几排筛选不是同一种东西。

给它一个色相,用户会误以为它和材质、来源是并列的维度。用颜色的「不同类」来表达信息结构的「不同层」,比多加一条分隔线有效得多。

实际效果:结果列表里每张卡左边有一条材质色竖条。快速滚动时**靠余光就能筛掉不想要的材质**,不用逐条读标签。文字标签还在,但它的角色从「识别」降级成了「确认」。



这条竖线是整套色彩系统的兑现处。上面那张卡是人造草(绿),下面是硬地(蓝)——卡片本身长得一模一样,区分完全交给了左边 3 dp 宽的一条色。

把颜色花在「一眼」上,而不是花在「好看」上。

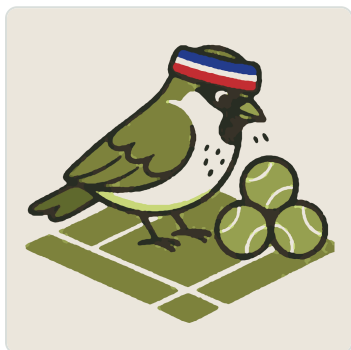
三个不提升指标的细节

图标、开屏、语言按钮。没有一个能让搜索变快 —— 但它们决定了这个 App 像不像一个人做的。

FR / EN / 中文

效率类产品最容易长成同一个样子:能用、正确、没有表情。我在三个地方刻意花了时间,它们不提升任何一条效率指标。

一·图标不是一颗网球



一只戴着三色头带的巴黎麻雀,站在球场上,旁边三颗球。

默认答案是画一颗网球 —— 一秒钟就能想到,也一秒钟就被忘掉。在 App Store 搜 tennis,一屏全是黄绿色的圆。

麻雀是巴黎街头最常见的鸟,三色头带是这座城市。图标要回答的不是「这是什么运动」,而是「这是哪座城市的什么运动」。

跟着系统语言切中 / 英 / 法。

二·开屏那 0.86 秒

Expo 的原生启动图是一张静态图片,没办法按语言换文字。所以我把开屏拆成两层:原生那层只放图标、一个字都没有;它一消失,由代码渲染的这一层立刻接上 —— 标语这才能



球从屏幕上方 160pt 落下,520 毫秒;缓动曲线是 `cubic-bezier(0.34, 1.56, 0.64, 1)`。第二个控制点的 y 是 1.56,大于 1 —— 所以球会冲过终点再弹回来。落定之后文字淡入 340 毫秒。全程 0.86 秒 —— 不拖沓,又比硬切多了一口气。

球的接缝是算出来的,不是描出来的

那两条弧不是画上去的曲线,是**两个大得多的圆的边缘**。圆小了,两条弧会在球心附近交叉,球就变成一颗杏仁。

最后取的是:接缝圆半径 $1.62r$ 、圆心离球心 $1.96r$ —— 这样每条弧最多只探到球心两侧 $0.34r$,才是网球该有的样子。整颗球用 View 拼,不依赖图片,任何屏幕密度下都不会糊。

上面这几帧不是截图,是按代码里的贝塞尔参数和接缝几何重新算出来的 —— 所以每一帧的位置都精确对应真实的那一毫秒。

三·语言不用国旗,用点心

国旗和语言不是一回事。英语该配哪面旗 —— 美国、英国、还是澳大利亚?中文呢?更何况在一个法国产品里摆一排国旗,读起来更像政治,不像语言。



中文

饺子



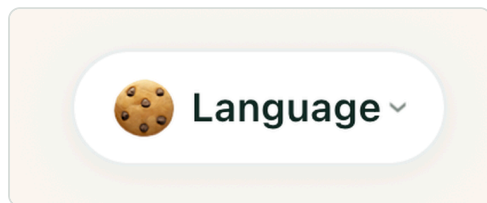
English

曲奇



Français

可颂



三样点心不代表国家,代表**说这门语言的人早上桌上会有什么**。没有歧义、没有立场,而且缩到 15pt 还是一眼能认出来。

右上角这个胶囊按钮在三个 Tab 里位置固定 —— 换语言是随时可做
的事,不该藏进设置的第三层。

10

三个界面决策

每一个都能回答「为什么不是别的做法」。

FR / EN / 中文

一 付费墙上摆的是产品本身,不是形容词

做法

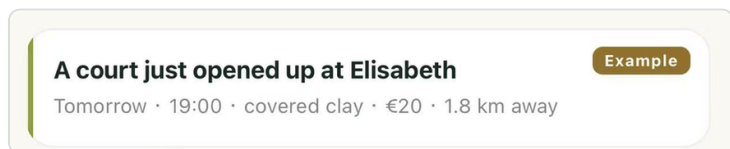
订阅页第一屏不是功能罗列,而是一张长得和真推送一模一样的通知样卡。

为什么

用户在决定付不付费之前,得先看见自己买的到底是什么。三行形容词说不清的事,一张样卡两秒钟就说清了。

代价 / 边界

正因为它长得像真的,右上角那个「示例」角标就不能省 —— 不标出来会有人以为真有场空出来,然后白跑一趟。真实感换来的信任,不能用误导来兑现。



这就是订阅页第一屏的样卡。 它和真推送用的是同一个组件、同一套排版 —— 场馆名、明天 19:00、有顶红土、€20、1.8 公里。

右上角那枚「Example」角标是这条决策里最不能省的一笔:卡片越像真的,不标出来的代价越大。

二 红土提示只在相关时出现

做法	勾选「红土」时会弹一条说明。但它只在 同时选中了市政来源 + 红土 的时候才出现。
为什么	全巴黎 202 片市政球场里只有 2 片红土。不解释,用户会以为 App 坏了。
代价 / 边界	但如果用户压根没在看市政场,这条提示就只是噪音。条件写得越细,提示越有价值。

The whole Paris municipal network has just 2 clay courts, so you will almost never land one. For clay, look at the Anybuddy clubs below.

提示不只说「几乎没有」,还给出**下一步**:去看 Anybuddy 的俱乐部。一条只说坏消息的提示是抱怨;带出路的才是设计。

三 开关滑块是一颗网球

做法	系统默认的开关谁都见过。我把滑块换成了一颗网球。
为什么	这是全 App 唯一一处纯装饰性的改动,不提升任何效率指标。
代价 / 边界	但它让这个 App 看上去 像是有人用心做的 ,而不是从组件库里拼出来的。我在作品集里留着这一条,是因为我认为「什么时候值得多花这一下」本身就是设计能力。

Notifications

Be the first to hear about cancellations



滑块是一颗网球 —— 连接缝都在。系统默认的黑圆点谁都见过,换成球之后,这个开关只属于这个 App。

它不提升任何效率指标。但用户每次开关通知都会碰到它 —— 而那一下,产品是有表情的。

三个交互决策

四 零注册 —— 第一屏就是产品

做法	打开 App 直接进搜索页,可以立刻用。没有注册、没有登录、没有引导页。用户标识是首次启动本地生成的随机串。
为什么	这个产品值不值,三十秒就能验完 —— 「哦,这个点真的有场」。任何挡在这三十秒前面的东西,都是在用户还没理由信我的时候先收一笔。
代价 / 边界	代价是换手机会丢配置。这个我认 —— 账号体系带来的注册摩擦、密码找回和 GDPR 责任,比它能解决的问题重得多。

从设计到代码

这个项目没有交接对象 —— 设计和实现都是我一个人。这反而带来一个好处:每一个设计决策都必须当场就能落地,落不了地就当场毙掉。

五 把数据的时效摊开来讲

做法	每一批结果都标着「X 分钟前更新」,旁边给一个「立即刷新」,刷新带冷却,并注明一切以官方页面为准。
为什么	聚合类产品最大的信任风险只有一句话:你给我看的,是不是已经过期了。
代价 / 边界	把「可能有几分钟延迟」明写在界面上,好过假装实时。同样一次对不上:产品从没提过,信任会一次性塌掉;提前说过,用户只会当它正常。

六 宁可漏放,不可错锁

做法	订阅状态查不到时(网络抖动、系统异常),产品选择放行而不是拦截。
为什么	一次网络抖动就把付过钱的人挡在门外,他还会以为是自己的订阅坏了。这种伤害不可接受。
代价 / 边界	代价是极少数情况下会有未订阅用户短暂看到付费内容。两害相权,这个明显更轻。

只做网球,不是一句主张

设计意图

「从横向平台里把纵向那一层抽出来重做」——这句话如果只停在文档里,是空的。

```
app/sources/anybuddy.py

CLUB_PAGE = ORIGIN + "/fr/club/{slug}/tennis"
```

URL 末尾就写着 `/tennis`。匹克球、羽毛球、壁球的库存从抓取的第一步就不进我们的数据。第三天还专门有一次提交叫「排除 padel 场地」。

价格和试用期一个字都不能写死

设计意图

付费墙上显示的价格必须和实际扣款一致——否则是明确的拒审理由。苹果在各国的定价、汇率、含税与否都不同。

```
src/iap.js

// 价格必须从 StoreKit 现取,不能写死在代码里
export async function loadPrices() {
  const list = await fetchProducts({ skus: SKUS, type: 'subs' })
  return Object.fromEntries(list.map(p => [p.id, {
    price: p.displayPrice,      // 苹果给的本地化价格
    currency: p.currency,
    trial: trialOf(p),         // 试用期也现取
  }]))
}
```

后台没配试用时,界面上一个「免费」的字都不会出现——按钮直接变成「订阅」,自动续订说明也换成不含试用的版本。界面没法说谎。

权限闸门是一个物理事实,不是一段判断

设计意图

没订阅就不该收到推送。通常的做法是服务器验证收据、判断权限、决定发不发。

```
权限判定在设备本地完成

# 没订阅 → 把推送令牌从服务器撤掉
# 服务器从此发不出任何东西给这台设备
if not entitled:
    await registerPushToken(user_id, token=None)
```

服务器完全不需要知道谁付了钱。**不需要验证收据、不需要第三方订阅服务、不存储任何支付数据。**令牌一撤,推送自然停——这不是一个 if,是一个物理事实。

一个大小写引起的 bug,让订阅永远不过期

设计意图

上线前我逐条测试订阅流程,发现「到期」这条永远测不出来。

```
src/iap.js

// 苹果给的字段是 expirationDateIOS — IOS 三个字母全大写
// 原来写的 expirationDateIos 永远取不到值
const v = p.expirationDateIOS ?? p.expirationDate
```

JS 属性名区分大小写,取不到值之后,「没有到期字段就当作有效」那条兜底对每笔交易都成立 —— 买过一次的账号从此永久解锁。这类 bug 不会崩溃、不会报错,只会安静地让你少收钱。

为什么放这一章

设计作品集常见的问题是:漂亮的界面图,看不出背后有没有真的想清楚,更看不出能不能落地。

这一章想证明的是两件事:我的设计决策都有具体的实现路径,以及我能自己走完那条路径。

13

上线

最后一天,把内购从「看起来能用」测到「真的能用」。

FR / EN / 中文

8 月 25 日,我按顺序测了五条内购路径。前三条很快过了,后两条把四个真 bug 逼了出来 —— 每一个都会直接影响用户掏钱。

1 付款弹窗根本不出现

原因	expo-iap	5.x 认的键是	request.apple.sk	,代码里写的是 3.x 的	request.ios.sk
如果没修	用户点了购买,什么都不会发生,界面只显示一句笼统的「购买失败」				

2 订阅永远不过期

原因	expirationDateIos	大小写错,取不到到期时间
如果没修	买过一次的账号永久解锁	

3 「恢复购买」对目标用户无效

原因	只读本地缓存,没和 App Store 同步
如果没修	换手机、重装的人 —— 也就是真正需要恢复的人 —— 永远恢复不了

4 试用天数写死

原因 「7 天免费」是硬编码,不是从 StoreKit 取的
如果没修 后台没配试用就是虚假宣传,明确的拒审理由

前三个 bug 都不会崩溃,不会报错。它们只会安静地让产品少收钱,或者在审核时被拒。

修完之后,五条路径全部通过:购买、权限解锁、杀进程重开、恢复购买、中途取消、到期回落。当天 17:12 提交 App Store 审核。

14

七天

文件时间戳和 git 提交记录还原的完整过程。

FR / EN / 中文

08-19	18:47	● 需求文档 v0.1 —— 代订方案
08-19	19:01	● 需求文档 v0.2 —— 推翻自己,转向聚合 + 提醒
08-19	19:18	● 写验证脚本,确认数据抓得到
08-20	06:41	● UI 原型 + 底色三版对比
08-20	11:18	● 后端首次提交
08-20	14:09	● 前端首次提交
08-21	11:53	● Anybuddy 只取网球,识别真实场地材质
08-22	01:04	● 时段条 + 新图标 + 全价显示
08-22	08:24	● 排除 padel 场地
08-23	19:55	● Anybuddy 锁定具体场地
08-24	07:28	● 推送通知打通

08-25 08:00 ● 内购接入

08-25 13:20 ● 修好内购的四个真 bug

08-25 17:12 ● 提交 App Store 审核

6 天 22 小时

总跨度

52

次提交

14 分钟

从 v0.1 到 v0.2

1

个人

15

复盘

做完之后,我认为哪些判断是对的,哪些下次会不一样。

FR / EN / 中文

我认为做对了的

- **先证伪最致命的假设,再动手。** 一个半小时的验证脚本,挡掉了白做六天的风险。
- **在只有自己的时候否掉更性感的方案。** 没人监督,也没人会知道 —— 但那是唯一正确的选择。
- **把颜色当成信息结构来设计,而不是装饰。** 色相环那套推导是这个项目里我最满意的部分。
- **每个设计决策都亲手实现一遍。** 这让「做不到」变成了设计过程的一部分,而不是交接之后才发现的意外。

下次会不一样的

- **内购应该更早接。** 我把它放到最后一天,结果四个 bug 集中爆发。它是整个产品里最不能出错、也最难测的一块,应该在第三天就开始。

- **应该更早在真机上跑正式包。** 开发版和正式包不是一回事,很多问题只有正式签名下才暴露。
- **七天做完一个产品,代价是没有真实用户参与。** 所有的用户痛点都来自我自己和身边的球友。下一步必须把真实用户的反馈补进来 —— 我最想验证的是:捡漏推送的到达率,和「推送到实际预订」的转化率。

这个项目对我意味着什么

我还是学生,想做产品经理。履历上写「有产品思维」说服不了任何人,所以我选择用一个真做出来、真提交上架、真会被陌生人下载的东西来证明。

在一个信息本来就是公开的领域里,把信息重新组织一遍,本身就能创造真实的价值。

用户仍然在官方平台付款,球场仍然属于这座城市。唯一变了的是 —— 想打球的人,终于知道什么时候有空场。



Paris Tennis Dispo

UX Case Study · 刘嘉悦 Jiayue Liu · 2026 年 8 月