

Чому план дня ніколи не збувається, і що з цим робити

Ми щоранку складаємо план, а ввечері дивуємося, куди пішов день. Справа не в лінощах і не в слабкій силі волі. Ось що я зрозумів, коли почав порівнювати свій план із тим, що відбувалося насправді.

У мене довго був один і той самий ритуал. Зранку — список справ на день, акуратний і реалістичний, як мені здавалося. Увечері — половина пунктів не закреслена, а відчуття таке, ніби працював без перерви.

Довго я списував це на те, що «день видався такий». Потім помітив, що такими видаються майже всі дні. І вирішив подивитися на це не як на проблему дисципліни, а як на задачу з даними. Колись я вже писав тут про [шлях створення власної системи обліку](#) — і ось до чого він зрештою привів.

Ми систематично недооцінюємо час

У психології для цього навіть є назва — **помилка планування** (planning fallacy). Її описали Деніел Канеман і Амос Тверські ще в 1979 році: люди стабільно недооцінюють, скільки часу займе задача, навіть коли знають, що минулого разу схожа задача тривала довше.

Найцікавіше тут — «навіть коли знають». Тобто досвід сам по собі не виправляє оцінки. Ми пам'ятаємо, що «колись затягнулося», але не пам'ятаємо, наскільки. А без цифри мозок щоразу підставляє оптимістичний сценарій.

Перемикання з'їдають більше, ніж здається

Друга причина — перемикання. Дослідниця Глорія Марк з Каліфорнійського університету роками вивчає, як люди працюють за комп'ютером. В її дослідженнях після того, як людину відволікли, вона в середньому поверталася до початкової задачі лише через двадцять із чимось хвилин.

У плані дня перемикань не існує. Там є «задача — 1 година». А в реальності є задача, повідомлення, «швидке питання» від колеги, знову задача, пошта. Кожне перемикання коштує не тільки хвилину на відповідь, а й час, щоб повернутися в контекст.

Що допомогло мені

Нічого революційного, але саме ця комбінація спрацювала.

1. Записувати факт, а не тільки план. Не з пам'яті ввечері, а в моменті: почав задачу — відмітив, закінчив — відмітив. Пам'ять дуже погано відновлює день: вона стискає перерви і розтягує «продуктивні» шматки.

2. Класти план і факт поруч. Це головне. Коли план в одному місці, а облік часу в іншому, їх ніхто не порівнює. Коли вони поруч на одній шкалі дня, розбіжність просто видно: ось тут я

планував одну годину, а вийшло дві з половиною.

3. Дивитися на точність, а не на кількість годин. Питання не «скільки я працював», а «наскільки мій план збігся з реальністю». Через кілька тижнів з'являються закономірності: певні типи задач я стабільно недооцінюю вдвічі, а наради завжди тривають довше, ніж у календарі. Після цього планувати стає набагато легше: я просто чесно закладаю реальний час.

4. Працювати фокус-сесіями з ціллю. Не «посиджу над задачею», а «50 хвилин на ось це, без месенджерів». Коли є ціль і видно, скільки лишилося, перемикатися хочеться значно менше.

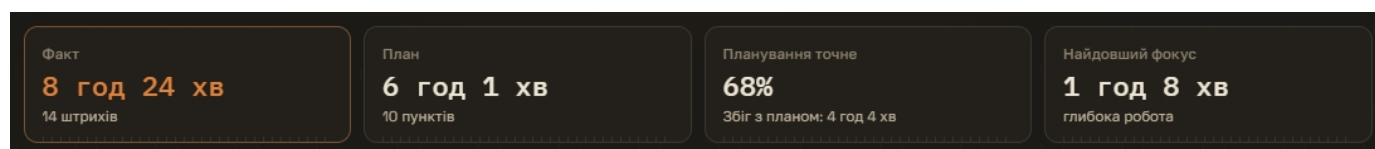
5. Не карати себе за «рваний» день. Дані потрібні, щоб планувати краще, а не щоб почуватися винним. Буває день, коли все розсипається на дрібні шматки, — це нормально. Важливо, щоб таких днів ставало менше і щоб ви розуміли чому.

Те саме, але в команді

У командах ця проблема множиться на кількість людей. Керівник питає «скільки часу пішло на клієнта?», і кожен згадує свою частину приблизно. Звіт збирається з пам'яті в п'ятницю ввечері, а помилки планування повторюються на кожному наступному проєкті.

При цьому рішення «давайте ставити програму, яка робить скріншоти екрана» зазвичай робить гірше: люди відчують нагляд, а не допомогу. Набагато краще працює, коли кожен сам веде облік, а команда бачить загальну картину: хто над чим працює, скільки пішло на проєкт, де план розійшовся з фактом.

Чому я зрештою зробив свій інструмент



Я довго робив усе це в таблицях і нотатках, і це було незручно саме в найважливішому місці — план і факт жили окремо. Тому зрештою зробив собі інструмент, де вони лежать на одній шкалі дня: план заштрихований, факт суцільний, а розбіжність видно з першого погляду.

Він називається **Штрих**, бо кожна задача лишає свій штрих у дні. Наприкінці день можна побачити як малюнок: коли фокус був довгим, штрихи лягають рівно, а день із постійними перемиканнями виходить рваним — і це видно без жодних цифр.

П'ЯТНИЦЯ

2026

2 ЖОВТНЯ

«рівний аркуш»



14

штрихів

1:08

найдовший фокус

68%

план точний

08:30–22:13

робочий день

 ШТРИХ

// 2 дні поспіль

shtrykh.net

Для однієї людини він безкоштовний, працює в браузері. Якщо захочете спробувати підхід «план поруч із фактом» — ось shtrykh.net. А якщо ні — спробуйте хоча б тиждень записувати факт поруч із планом будь-де, навіть на папері. Результат вас здивує.

А як у вас? Збігається план дня з тим, що відбувається насправді? Поділіться в коментарях — цікаво, які способи працюють в інших.

Крок 2. Створення системи обліку

*Спочатку ремарка. В мій кругозір потрапив один **product manager**, з яким ми мали цікаву бесіду відносно створення чогось з нуля, я отримав рекомендації щодо послідовності та деяких кроків, хочу залишити це у вигляді шпаргалки тут. Це все буде оформлено відносно до моїх потреб. Якщо хтось має якісь рекомендації - можете поділитись в коментарях.*



І так, даний проєкт має декілька основних блоків і вимагає розробки веб-додатка, який буде забезпечувати функціональність для кожного з них. Нижче наведено загальний план проєкту:

Етап 1: Аналіз вимог

1. Провести детальний аналіз вимог та створити докладний опис функціональності для кожного блоку проєкту.

Етап 2: Проектування

1. Розробити архітектуру додатка і бази даних на основі технологій Django, HTML, CSS і JavaScript.
2. Визначити структуру бази даних PostgreSQL та створити моделі Django для зберігання даних.
3. Створити макети інтерфейсу для фронтенду.

Етап 3: Розробка бекенду

1. Почати розробку бекенду за допомогою Django або Flask.
2. Реалізувати функціональність блоку клієнтів, включаючи зберігання інформації про клієнтів та цін за годину.
3. Реалізувати функціональність блоку проєктів (задач), включаючи можливість розподілення завдань на підзадачі та об'єднання їх в блоки задач.
4. Реалізувати функціональність фінансового блоку для створення звітності та рахунків для оплати, як загальних, так і з розбивкою на окремі проєкти та завдання.

Етап 4: Розробка фронтенду

1. Створити користувацький інтерфейс за допомогою HTML, CSS і JavaScript, який відображатиме дані та дозволить користувачам взаємодіяти з додатком.
2. Підключити фронтенд до бекенду за допомогою API (наприклад, RESTful API).

Етап 5: Тестування

1. Провести тестування функціональності кожного блоку проєкту та інтерфейсу.
2. Виправити будь-які помилки або недоліки, виявлені під час тестування.

Етап 6: Внесення змін та покращення

1. Завершити розробку будь-яких додаткових функцій або можливостей, які можуть виникнути під час тестування.
2. Оцінити можливості для оптимізації проєкту та покращення продуктивності.

Етап 7: Деплоймент та підтримка

1. Розгорнути додаток на сервері та забезпечити його належну роботу.
2. Провести навчання для користувачів, які будуть використовувати додаток.
3. Забезпечити підтримку та обслуговування додатка після впровадження.

Це загальний план, який може бути дороблений та адаптований відповідно до вашої конкретної ситуації та потреб. Ключовими технологіями для цього проєкту будуть Python (Django або Flask), HTML, CSS, JavaScript і база даних PostgreSQL.

Звісно, я сумніваюсь, що зможу дотримуватись саме такої послідовності, бо в мене немає команди між якою можна розділити хочаб фронт та бек, але вже є від чого відштовхуватись.

Шлях створення власної системи обліку

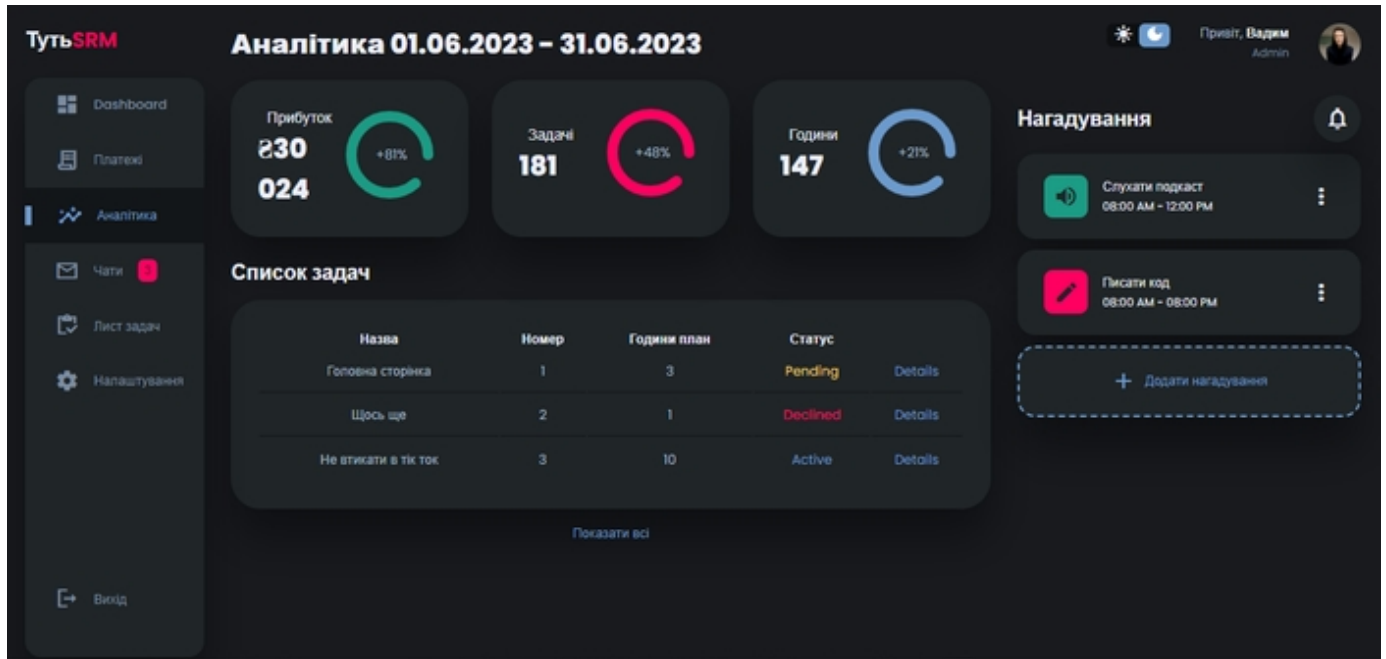
Трішки історії та планів.

Моя робоча діяльність, яка включає фріланс-проекти та роботу в компанії, де годинна оплата є загальною практикою, призводить до необхідності відстеження витрат часу та фіксації завдань, які будуть оплачуватися. Поза тим, не завадить аналіз фінансових виплат. Це включає в себе перевірку оплат, виявлення затримок та прийняття рішень про подальшу співпрацю з клієнтами.



На своєму шляху я випробував купу різних ресурсів, такі як [Asana](#), [ClickUp](#), [Neemb](#), та низку інших. Всі вони мали свої переваги, але жоден не забезпечував повного вирішення проблеми.

Таким чином, я прийшов до рішення спробувати створити власну систему управління проектами та фінансами на Web платформі. Це рішення дозволить покласти контроль над процесами управління власними проектами в мої руки. Додатково це поєднується з величезним бажанням відійти від розробки BAS систем, та постійного відчуття синдрому самозванця (бо як би всі причетні не кричали, що це до русні не має ніякого значення, але всі розуміють правду. Навіть, якщо гроші і не йдуть напряду в руснястан, все одно ну його нафіг) та перейти на іншу сторону розробки, а саме Web Dev.



Поки якась цілісна схема не сформувалась, по переду дуже багато роботи, але ось основні блоки, що є на поточний момент в голові:

- Блок проектів (задач): Цей блок для організації проектів та має включати в себе можливість розподілення завдань на підзадачі та об'єднання в блоки задач для полегшення формування звітів на оплату.
- Блок клієнтів: У цьому блоку зберігаються дані про клієнтів, включаючи назву компанії, ціну за годину, закріплену команду та ціну за годину для команди.
- Фінансовий блок: створення звітності та рахунків для оплати, як загальних, так і з розбивкою на окремі проекти та завдання.
- Віджети: надають інформацію про загальну кількість завдань та загальний дохід за місяць.

Планую використовувати такі технології для розробки:

- Мова програмування: Python (Django, Flask) для бекенду, HTML, CSS, JavaScript для фронтенду.
- База даних: PostgreSQL використовується для зберігання даних.