Nazarbayev University

School of Engineering and Digital Sciences

PowerBook

A Web Platform for Habit-Building and Competitive Reading

CSCI-409 Senior Project II — Final Report

Spring 2026

https://power-book.app/

Group 44

Team Member	Role
Toktar Sagyngali	Database & Documentation
Nurislam Bakhtybayev	Backend Development
Bekzhan Aktoreev	Leaderboard & Performance
Bektas Keldibayev	Frontend Development & UI/UX
Nurdaulet Otegenov	Telegram Bot Integration

Project Advisor: Askar Boranbayev

Table of Contents

1. Executive Summary 3
2. Introduction 5
3. Identification of Constraints 6
4. Background and Related Work 7
5. Project Approach 9
6. Project Execution 21
7. Evaluation 25
8. Conclusion and Future Work 27
9. References 29

1. Executive Summary

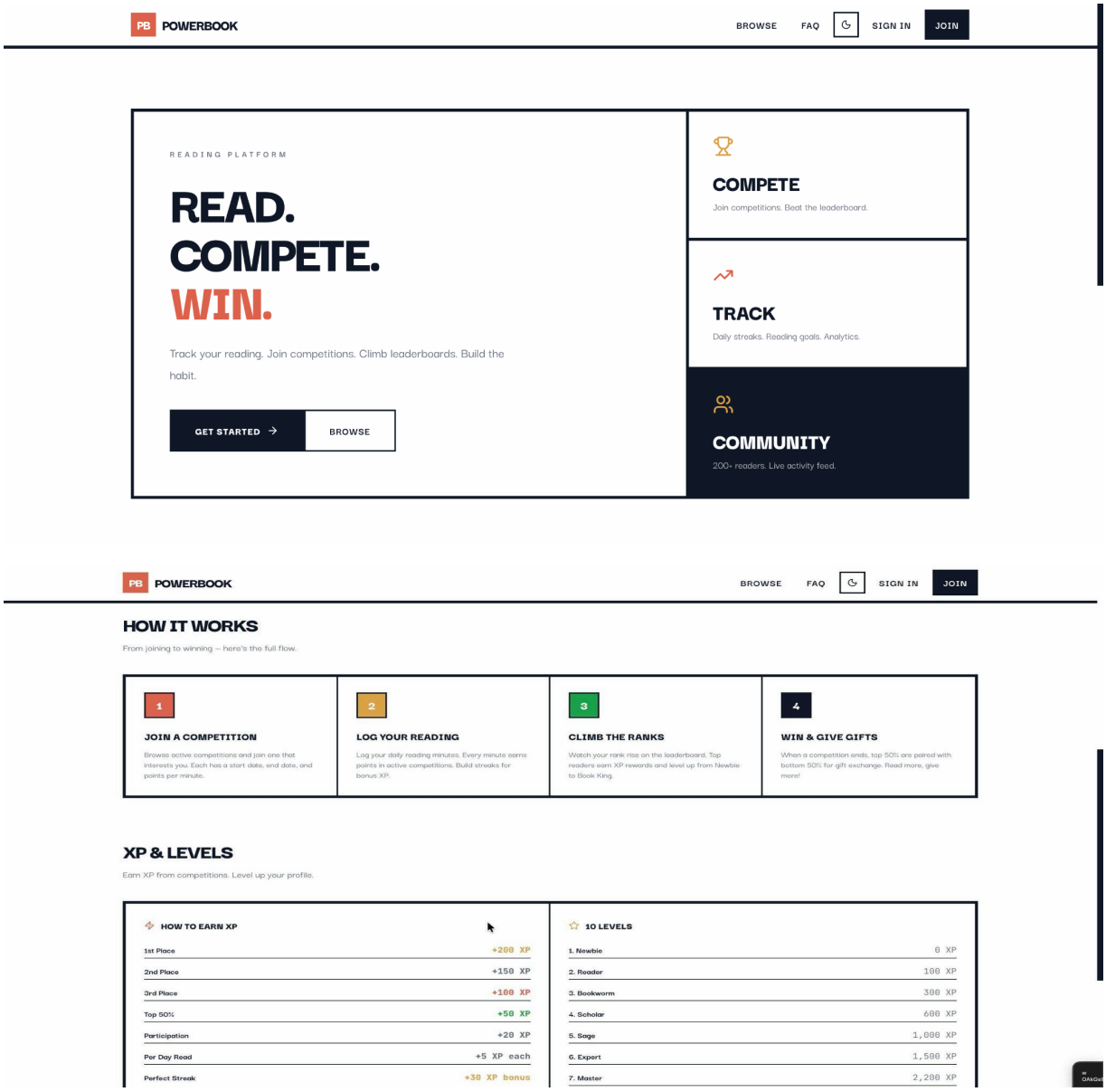
PowerBook is a web platform for building a reading habit through short, head-to-head competitions. Users log how long they read, join competitions that run for a fixed number of days, pick up XP and badges along the way, and - when the competition ends - take part in a gift exchange that pairs the heaviest readers with those who read less. Reading apps usually treat reading as a solo activity; our bet was that a bit of friendly pressure and a live leaderboard would do more for consistency than another streak counter.

The project was created over two semesters (Fall 2025 and Spring 2026) in the CSCI-409 Senior Project course at Nazarbayev University. The back server is implemented in Go using the Gin framework, using PostgreSQL as persistent data and Redis as live leaderboards. It is an app written in Next.js 16 / React 19 and styled with Tailwind CSS and deployed on Vercel. A friend bot, developed in Python with python-telegram-bot, sends daily notifications and allows users to record time reading via an inline-keyboard menu, meaning that they do not need to open the site to be entertained. The backend is hosted on a single DigitalOcean VPS as a Docker with Caddy serving as the frontend with TLS and automatic renewal of the Let's Encrypt.

The backend is based on Domain-Driven Design, with a hexagonal (ports-and-adapters) design, where the business logic is not dependent on the database, the cache, and the HTTP layer. Authentication is by JWT and bcrypt-hashed passwords. Logs of reading are logged on the server and limited to 1,440 minutes per day, largely to ensure that the numbers are honest.Competitions follow a create join active closed lifecycle and automatically close themselves down via a background scheduler when the end date is reached.sses. Redis sorted sets support the leaderboard, which can be paginated, has a Find Me jump-to button and a banner informing the user of whether they are in the top or the bottom half of the ranking. XP has 10 levels of users, starting with Newbie and ending with Book King, and 30 badges in the reading, social, and competition categories. The ranked halves are paired up in the gift exchange; with an odd number of participants, the median user will be labeled as neutral and indicated in grey rather than being compelled to form a pair. Above that is an admin panel, user profile pages, and a FAQ, dark and light themes, and the Telegram integration.

The site has been opened at the address of power-book.app. We inserted 50 demo users and three closed competitions (20, 30 and 40 participants) into the database to test it end-to-end and make sure that everything worked. Production VPS Leaderboard reads were less than 20 ms, standard API endpoints less than 50 ms, and the three backend containers consumed less than 90 MB of RAM. A seeded internal run cannot inform us of whether the competitive features in fact alter the actual reading behaviour - that would require a user study which we are yet to undertake, but the platform itself is

operational and the architecture is prepared to support the next things we'd wish to add: friend networks, team competitions, and eventually institutional rollouts.



Main page

2. Introduction

The knowledge gained through reading, the clarity of thought, and the usefulness of reading are lifelong habits, yet sustained reading has been declining in young adults since the smartphone has become the display of choice. Long-form writing is more readily accessible to students than ever before and students are reading less for pleasure than their parents. Group chats, short video, and streaming are all vying against the same half-hours spent in a book. One study on the habit formation suggests two approaches that can consistently be effective against that drift: social accountability and gamification. There are not many reading platforms that are a mixture of both.

The platform is created on the basis of competitions. The users are in a group and record their time reading within a predefined time and track a live leaderboard. As soon as one competition ends, a gift exchange is provided between the top-half of the readers and the bottom half - the most read provide a gift to the least read. That transforms the outcome of winner-take-all into something that the entire group will be glad to have, which is important in terms of retaining people interested in participating in the next one. Off to the side of that are the supporting items, which include the ten-level XP system, thirty badges, a daily streak counter, and a Telegram bot to remind a user at the end of the day and allow them to record a session without visiting the site. The availability of a web app and a Telegram bot allows users to access PowerBook via either platform that is already open.

There were four objectives of the project:

Create a web app that makes reading a game.

Create a contest and gift-giving system which will keep people interested once the initial rush is over.

Include a Telegram bot such that even when the site is shut, the habit loop continues to work.

Install the platform and run end-to-end with seeded test data.

The remainder of the report is in the sequence of the working done. Section 3 takes us through the limitations which influenced our design decisions. Section 4 conducts a survey of related research in reading apps, gamification, and research of habit-formation. Section 5 discusses the system architecture, technology stack, database schema, subsystems and frontend. Section 6 is a chronicle of two semesters of growth, what choices we made, what issues we ran into and how the work was distributed among five of us. The evaluation is given in section 7. Section 8 wraps up and explains what we would construct next.

3. Identification of Constraints

PowerBook was constructed under a combination of the technical, economic, organisational and ethical constraints. They both dictated certain design choices and precluded others. They are discussed below respectively and their trade-offs as well.

3.1 Economic Constraints

There was no budget in the project. Its backend is hosted on a single DigitalOcean VPS at an approximate cost of $6 a month, which is paid using student credits; the frontend is hosted on the free tier of Vercel, and Redis is hosted in a Docker container on the same VPS as opposed to a dedicated Redis service like Redis Cloud. It is that budget that has made the stack almost entirely composed of free and open-source software - Go, PostgreSQL, Redis, Next.js, Tailwind CSS, python-telegram-bot - and made the use of paid services such as Auth0, hosted databases, and push-notification services never even a consideration.

3.2 Time Constraints

The project was done in two semesters and each member of the team was balancing the project with other course work. That necessitated actual scope reductions. OAuth sign-in, a native mobile application, and a recommendation engine were all mentioned during the planning process and moved to the future. It was also much more constrained than we would have preferred user testing on real participants and that is the reason why the numbers in Section 7 are based on seeded demo data and not a group of actual users.

3.3 Social Constraints

Time of reading is self reported. The system cannot possible check that a user that logs 90 minutes actually read 90 minutes, and that is important in a competitive platform. The scoring system operates based on this, where consistency, the number of consecutive days of consecutive workouts or consistent logging is rewarded, not the individual marathon sessions, to which there is minimal incentive to inflate a single entry. The reason why the gift exchange works is different: by matching heavy and light readers rather than having a clear-cut winner, the pressure to fabricate numbers to win is diminished. The motivational banner (to inform the users about being in the first half or the last half of the leaderboard) is there to ensure that even users who are on the bottom end receive recognition instead of a silent, low ranking.

3.4 Technical Constraints

The system must be able to support simultaneous updates to the same leaderboard. The reason why redis sorted sets were selected over the others: the atomic ZINCRBY operation remains consistent when multiple writers are involved and has a runtime of O(log N). Go was selected as the backend though not all members of the team had released production Go prior. The learning price was tangible, yet Go

concurrency paradigm and one-binary implementation were justified by a server which needs to respond to the requests of the website and the Telegram bot. Next.js 16 was a fairly new version on the frontend when we were developing, so the official documentation and examples in the community were still behind and we were sometimes forced to read source to verify behaviour. Its current deployment is a single VPS and this limits horizontal scaling. The bottleneck is not the application layer itself - the leaderboard runs in Redis and the authentication is stateless JWT, so a hosting change (not rewrite) would be moving to multiple instances behind a load balancer.

4. Background and Related Work

4.1 Reading Habit Formation

Lally et al. (2010) discovered that habitation develops by repetition within a stable environment - the behaviour is repeated frequently, in similar enough conditions, that it ultimately becomes automatic and requires no conscious effort. In the case of reading, the practical implication of it is that frequency is more important than the length of the session: reading twenty minutes five days a week will develop a habit more efficiently than reading two hours once a week on Sundays. There are a few obvious ways in which a digital platform can help support that kind of frequency, and PowerBook relies on each of them. Streaks highlight missed days, as a missed evening will appear on the chain as a break instead of just fading away. The Telegram bot reminds users who choose to receive a reminder in the evening. And each logged session rewards a small dose of XP, which builds up towards the next level or badge - not so much as to induce a sense of manipulation, but just enough to provide some form of reward to show up.

4.2 Gamification in Education and Self-Improvement

Gamification refers to acquiring the game-like elements of points, progress bars, badges, leaderboards, and applying them to an underlying task that is not a game. The definition adopted by Deterding et al. (2011) remains in use in the literature today: game design components in a non-game setting, implemented to make it more engaging. The question of whether such mechanics actually change behaviour is less determined. The meta-analysis by Hamari et al. (2014) found that gamified systems tend to boost engagement, although with a significant caveat - this effect is highly dependent on the mechanics design and its alignment with the activity on which it is affixed. Badges which seem arbitrary, such as those which tend to leave the needle stationary, do not generally move the needle. The gamification toolkit that PowerBook utilizes is relatively standard (XP, levels, badges, a leaderboard), and one of the aspects that is unique to the project is a gift exchange that follows each competition, which is addressed in Section 5.6.

4.3 Self-Determination Theory and Motivation

Self-Determination Theory (SDT), which is normally attributed to Ryan and Deci (2000), posits that intrinsic motivation relies on the three psychological needs being satisfied, namely, autonomy, competence, and relatedness. All three have PowerBook weighted, even if not equally. Relatedness is the best fit - competition and the gift exchange are there primarily in order to achieve the effect that the reading is something that people do together but not alone. Competence manifests itself in the form of XP and badges, not because points themselves are motivational but because they render small steps to progress noticeable, which is what the theory points to in the first place. The most non-involved of the three is autonomy: users decide what, when, and how much to read, and the platform does not intentionally tell users what to read. The only aspect that introduces an extrinsic reward into the intrinsic ones is the gift exchange. In the article that still constitutes the primary reference point to this concern, Deci et al. (1999) established that in some cases, external rewards may diminish the internal drive they are supposed to support - but the impact of social and non-contingent rewards is even less significant than that of transactional rewards. The act of a gift between equals at the end of a contest is nearer to the social extremity than the transactional one, and that is why we tapped into this when we chose to retain the mechanic.

4.4 Existing Reading Platforms

A number of the current platforms provide an aspect of reading tracking and social reading, but none of them offers all the features that PowerBook does.

Goodreads. The biggest social reading site, which includes an option to catalog the book, leave a review, and a yearly reading challenge. Nevertheless, Goodreads emphasizes the number of books read, but not the amount of time spent reading, and its social aspect is geared towards discussion and not competition in real-time. It does not have a leaderboard or a streak system or a reward system except a fixed annual badge.

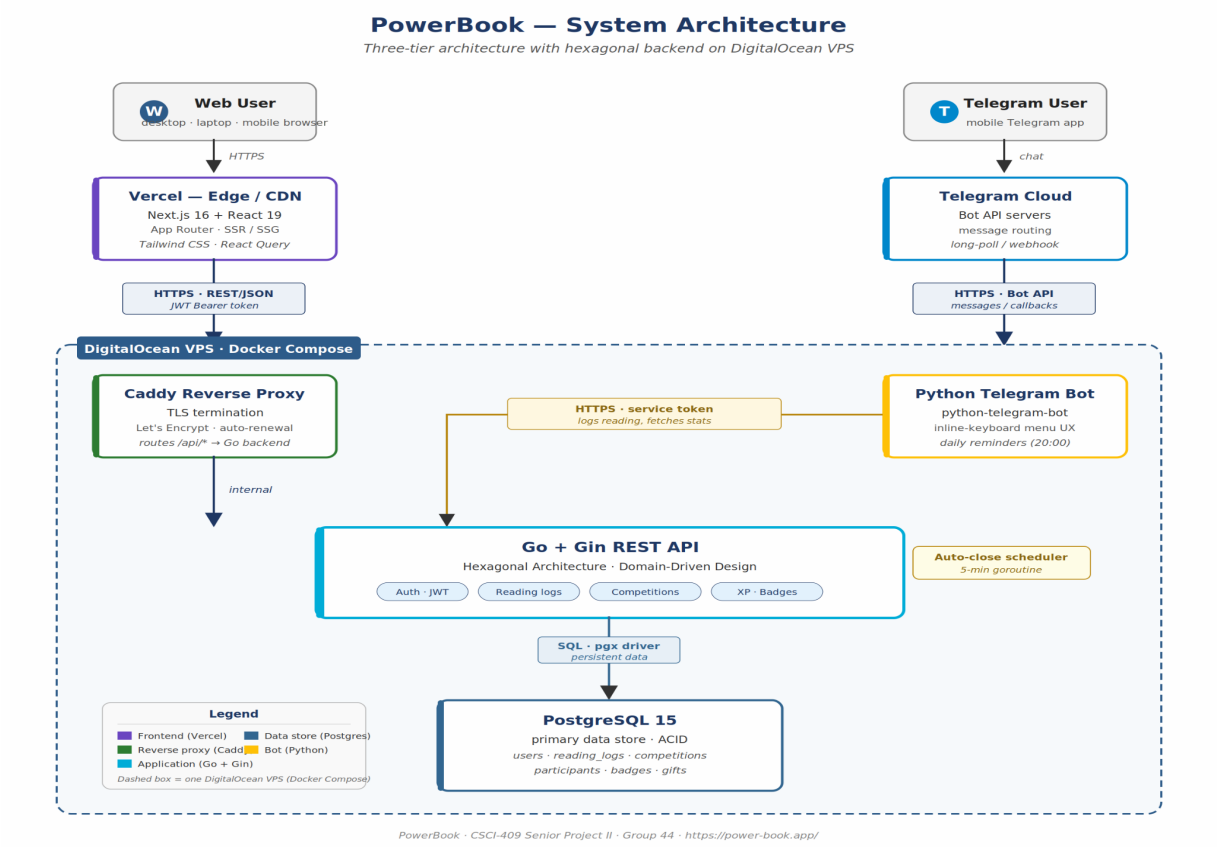
StoryGraph. An alternative to Goodreads that is more modern with improved analytics and recommendations based on mood. Although StoryGraph is used to track reading progress and pacing, it lacks head-to-head competitions, time-based tracking, and badges and streaks.

5. Project Approach

5.1 System Architecture

PowerBook is designed around a typical three layer division: a presentation layer at the client, business logic at the middle and data storage at the bottom. The backend is based on hexagonal architecture, or the ports-and-adapters form of Domain-Driven Design, which is a formal but in reality a business rules are written against interfaces (as opposed to writing against Postgres and Redis directly). The reward is testability: we can exercise domain logic without booting up a database, and when we replaced our initial in-memory leaderboard with Redis partway through the project, nothing in the domain layer needed to change.

The system runs with three processes: a Go REST API which opens up the platform, a Next.js web application which the browser interacts with, and a Python Telegram bot which calls the API using a service token with HTTPS. In front of the Go service is a Caddy reverse proxy that deals with TLS, which automatically renews the Let's Encrypt certificate.



5.2 Technology Stack

Layer	Technology	Purpose
Frontend	Next.js 16 + React 19	Web UI with SSR/SSG and App Router
Styling	Tailwind CSS v4	Utility-first styling, dark/light theme
Client state	TanStack React Query	Server-state caching and refetching
Backend API	Go + Gin framework	RESTful API with JWT authentication
Database	PostgreSQL 15	Users, competitions, reading logs, badges
Cache / Leaderboard	Redis 7	Real-time sorted-set leaderboards
Telegram Bot	Python + python-telegram-bot	Mobile logging and daily reminders
Frontend hosting	Vercel	Edge CDN, serverless deployment
Backend hosting	DigitalOcean VPS (Docker)	Container-based deployment
Reverse proxy	Caddy	Auto-TLS with Let's Encrypt
CI/CD	GitHub Actions + webhooks	Automated redeploy on push to master

5.3 Backend Architecture

The backend is structured around four layers of a hexagonal-architecture:

Domain layer. Basic Go classes of the core objects User, Competition, ReadingLog, Badge, Gift, Participant and the business rules governing them. No third-party library is imported in this layer.

Application layer. Use-case services which organize the domain entities: registration of a user, reading log, joining competition, competition closure, badge awarding.

Ports. Dependences on interfaces: UserRepository, CompetitionRepository, ReadingLogRepository, Leaderboardservice, Badge Evaluator.

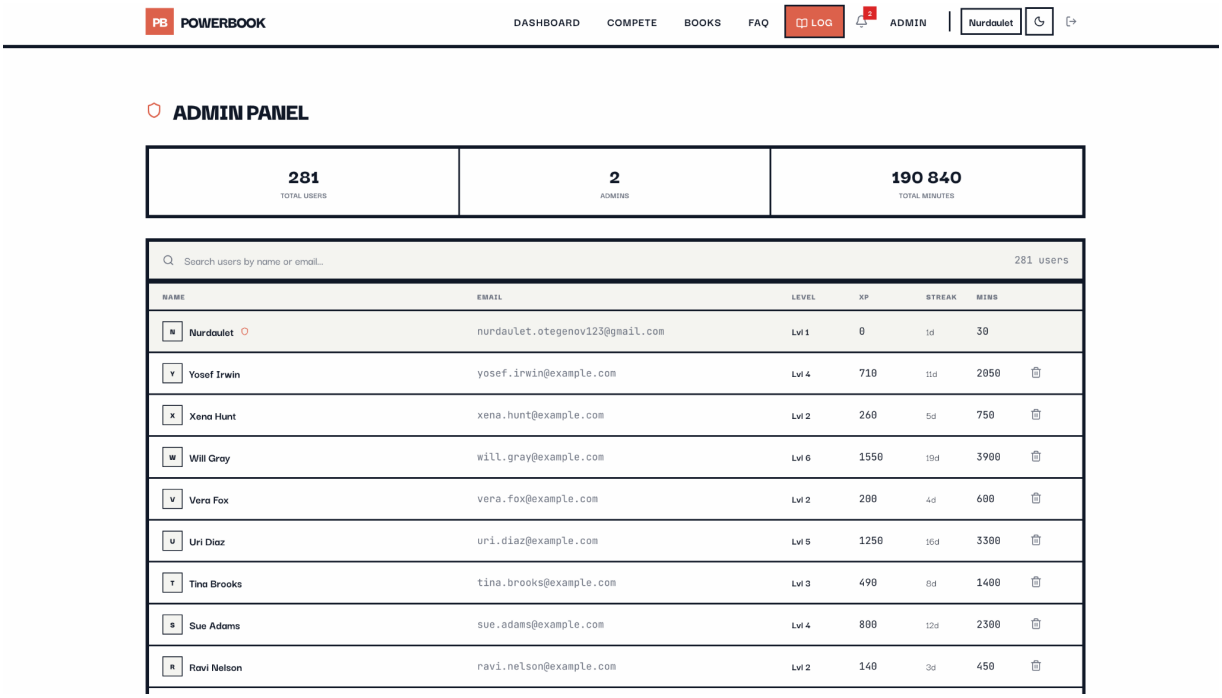
Adapters. Practical applications of such interfaces: PostgreSQL to the repositories, Redis to the leaderboard, Gin to the HTTP, and a JWT middleware adapter to authenticate.

There are thirteen REST API endpoints in four families. Authentication includes register, profile and login. Reading management deals with reading history and reading log. Competition management reveals create, join, close, list and get-by-id. The leaderboard queries reveal paginated rankings, a single-rank look up, and the find-me endpoint. All but registration and login must have a JWT in Authorization. All responses are the same envelope of JSON: a success flag, a possible field of error, and a data object.

The daily reading cap is one guard that should be called out since it imposes the 1,440-minute limit of Section 3.5. When a submission would take the user above the limit, the API does not decline the request immediately instead it sends out HTTP 200 with the number of minutes left in the payload and the frontend silently updates the entry. Hard rejection would be like being punitive; to back off the remaining budget would allow the user to correct his/her input without resistance.

Competitions close themselves. A background goroutine (the codebase refers to it as the auto-close scheduler) is activated every five minutes and it inspects competitions whose end date has elapsed and closes each competition, giving the final XP, re-calculating competition-specific badges, and running the gift-pairing algorithm. No one has to press a button at midnight and a creator who has forgotten that there is someone to compete with does not leave it suspended in the active state.

The admin panel is at /admin and is authorized by the is_admin flag in the users table. It presents a list of users that is searchable, a platform-wide statistics (total users, number of admins, total minutes read on the platform), and a delete-user option. The route is visible only to users whose flag is set; the presence of the Admin link in the navigation bar only shows up to them.



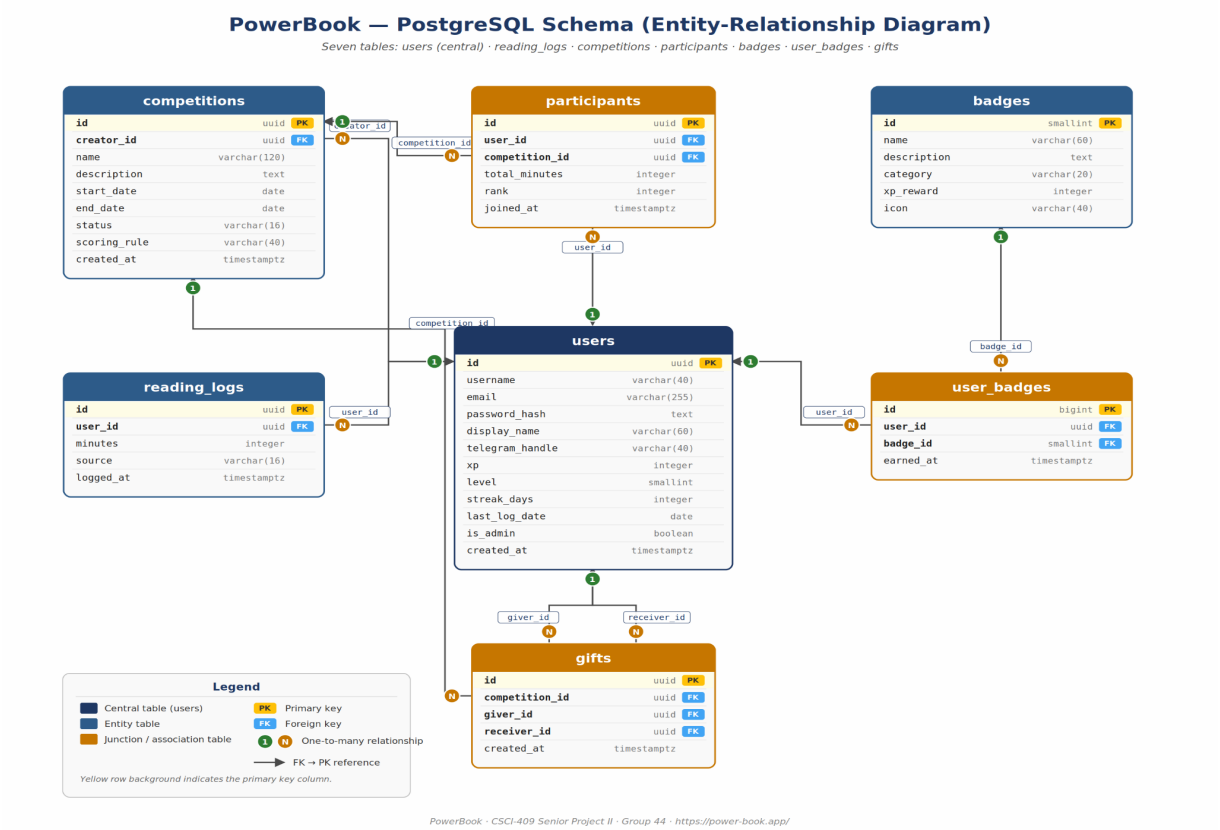
5.4 Database Design

PostgreSQL schema PostgreSQL schema consists of six tables: users store the basic account information: a bcrypt-hashed password,XP, level, streak counters, an is-admin flag and an optional Telegram handle of users who have connected their bot session. reading-logs store the reading session information: user-

id, minutes, a source column, which indicates whether the log was created in the web or in the Telegram bot and a server-

Users are connected to competitions through two junction tables, participants which contains the points awarded per competition, the rank of a particular user in that competition and the time they joined that competition, user_badges does the same, and the thirty badges themselves are stored in a fixed badges catalogue and are rarely updated. The gifts table is placed above all this and captures the pairings that each exchange of gifts generates with the giver and receivers being identified by giver and receiver ID and competition being identified by competition ID as columns.

Indexes include where the read pressure is: userID on the reading-logs, competitionID on the participants, and a composite (userID, timestamp) index on the reading-logs to the reading-history query that drives the dashboard. All foreign keys are defined as ON DELETE CASCADE, and this is what makes it a one-statement deletion of users instead of a five-step cleanup.



5.5 Real-Time Leaderboard

The only component of the system where PostgreSQL did not fit is leaderboards. When a user records reading time in an ongoing competition, the backend makes a single call to ZINCRBY towards the Redis

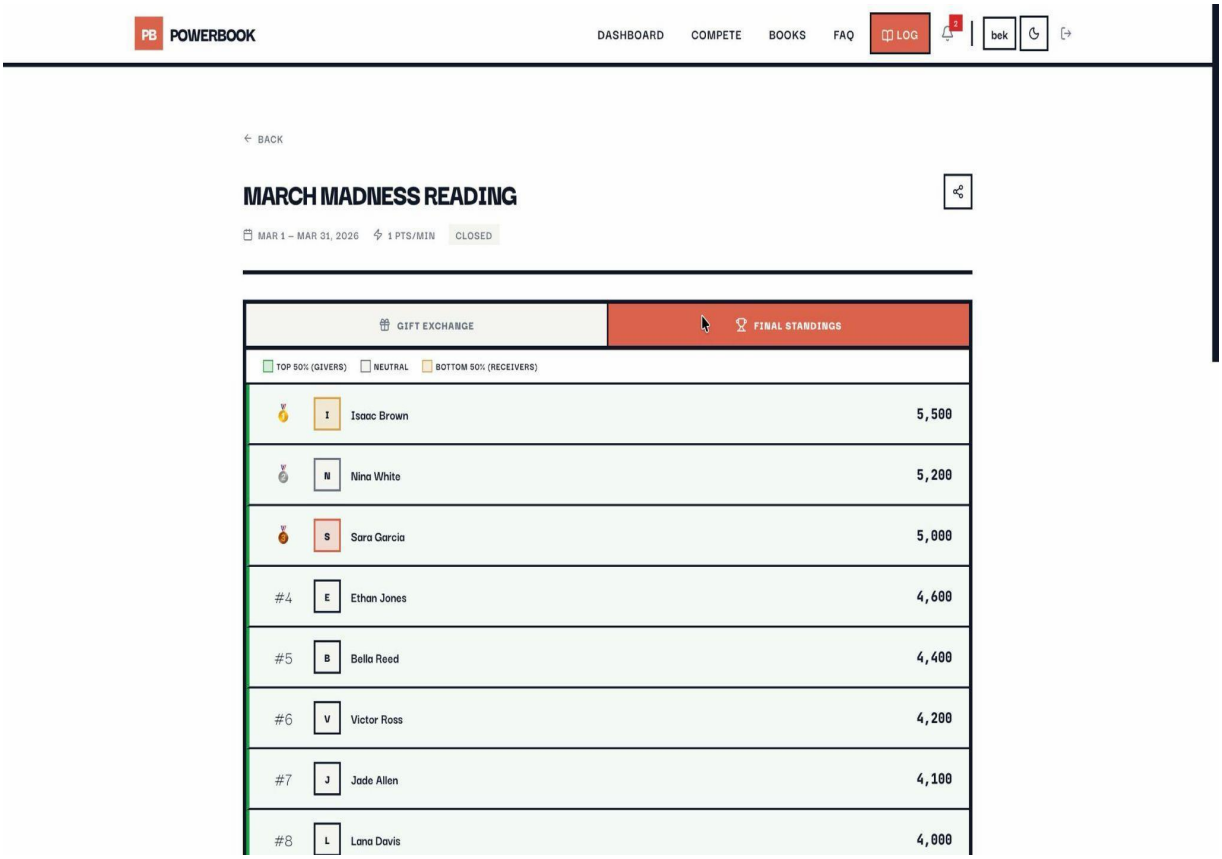
sorted set of that competition, which atomically increments their score. Reads are implemented by ZREVRANGE taking scores and run in O(log N + M) time, where N is the number of participants and M is the page size. This implies that query time remains constant even as the number of reading logs below it continue to increase. The previous implementation used the ranking with a PostgreSQL aggregate, which was fast enough when using only a few logs, but slowed visibly when we had seeded a few hundred; that is what drove us to Redis.

There are three leaderboard features that can be described individually.

Pagination. There are twenty-five participants per page. The frontend displays previous and next buttons and page number controls, allowing users to navigate large competition without the client downloading all the participants initially.

Find Me button. The authenticated user is taken to the page with their row and it is scrolled with a highlight into view using a single "Find Me" button. Compared to paging through looking to find yourself, as was the chief complaint during our internal testing, this is incredibly quicker in a competition with forty or more participants.

Motivational banner. During active competitions there is a contextual banner above the leaderboard. The users in the upper half of the list are presented with a green banner that indicates that you are in the top 50 percent; the users on the lower half of the list are presented with an amber banner that gives an encouragement. The idea is to provide one with an idea of his or her position without having to scroll through a list of forty names to figure out what the user is, and to make users with lower ranks feel recognized instead of being sidelined.



5.6 Competition and Gift Exchange System

A competition passes in 4 states:

Created. The user enters the name, description, start and end date, and chooses a scoring rule.

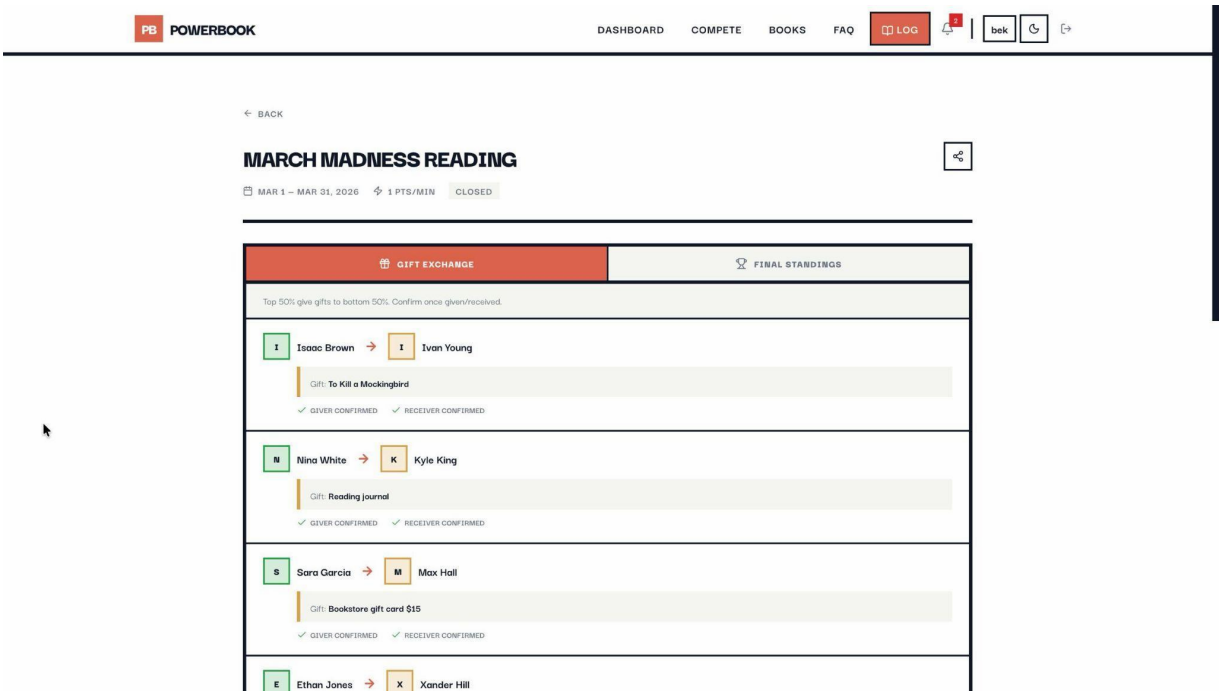
Joinable. Other users may join until the start date and in certain scoring rules during the active window.

Active. The reading logs posted in this window are fed into the leaderboard through ZINCRBY.

Closed. The creator may close the competition manually or it will be closed automatically by the auto-close scheduler after the end date. Concludes awards XP and executes gift exchange.

The gift exchange works in the following way. The participants will be ranked according to their reading time during the competition period. The list is sorted into two parts; the first half will be givers the second half receivers. These are matched up in parallel; top-1 bottom-1, top-2 bottom-2, etc. down the list. In cases where the number of participants is odd, the participants cannot be paired off evenly hence the median participant will be noted as neutral and will not be part of the exchange in respect to that competition and it will be indicated in grey in the UI. The gifts table supports all the pairings and appears on the profiles of both the participants under a confirmation flow.

The logic behind the exchange being this mechanism is that a single-winner competition is likely to create a consistent issue: the individual at the bottom of the leaderboard silently ceases to open up the app. The combination of the best readers and the worst readers takes the place of that dynamic with a more like-minded approach of the heavy readers receiving satisfaction of giving and a competition-badge and the light readers receiving recognition and a reason to participate in the next competition instead of thinking they have lost already. Neither is it an ideal incentive scheme, but it is quantifiably less aggressive than winner-takes-all.



5.7 XP and Leveling System

Users earn experience points (XP) for reading activity, competition participation, and badge acquisition. The levelling system comprises ten tiers, each with increasing XP thresholds and a thematic title:

Level	Title	XP Required
1	Newbie	0
2	Page Turner	100
3	Bookworm	300
4	Chapter Chaser	700
5	Novel Navigator	1,400
6	Literary Explorer	2,500
7	Story Sage	4,200
8	Reading Maestro	6,500

9	Grand Scholar	9,500
10	Book King	14,000

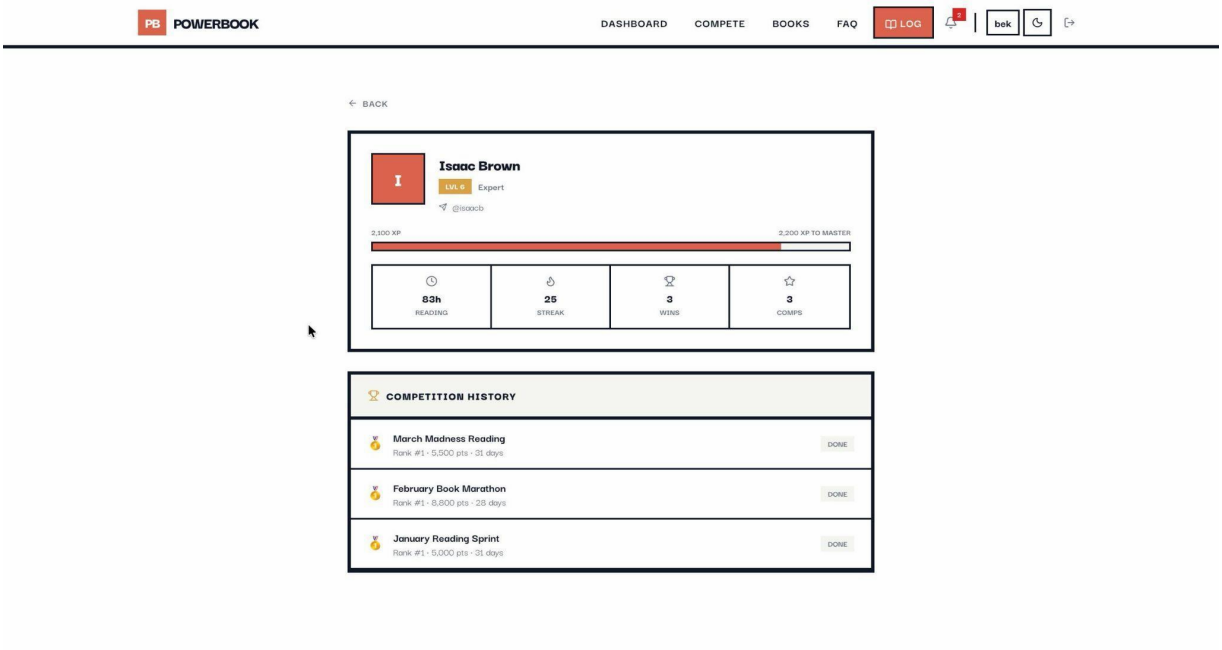
5.8 Badge System

The total number of badges is thirty and they are divided into three categories based on the triggering factors.

Badges Reading badges are cumulative: cumulative minutes read, number of separate reading sessions, number of days reading, and streaks (three-day streak, seven-day streak, thirty-day streak, etc.). These are the ones that a user will most likely get without necessarily trying to get them - they come by default when using them in normal use.

Interaction with other users, such as joining your first competition, entering a number of competitions, sending or receiving a gift, inviting a friend via a shared competition link are social badges. This category is aimed at rewarding a user something basic when they do anything sociable on the platform the first time and not volume.

Outcome based badges are competition badges: winning a competition, placing in the top three, completing a month long competition or having a streak throughout a full competition window. Competition badges have a higher XP value compared to the other two categories, which are on the basis that they are more challenging to obtain.



5.9 Telegram Bot

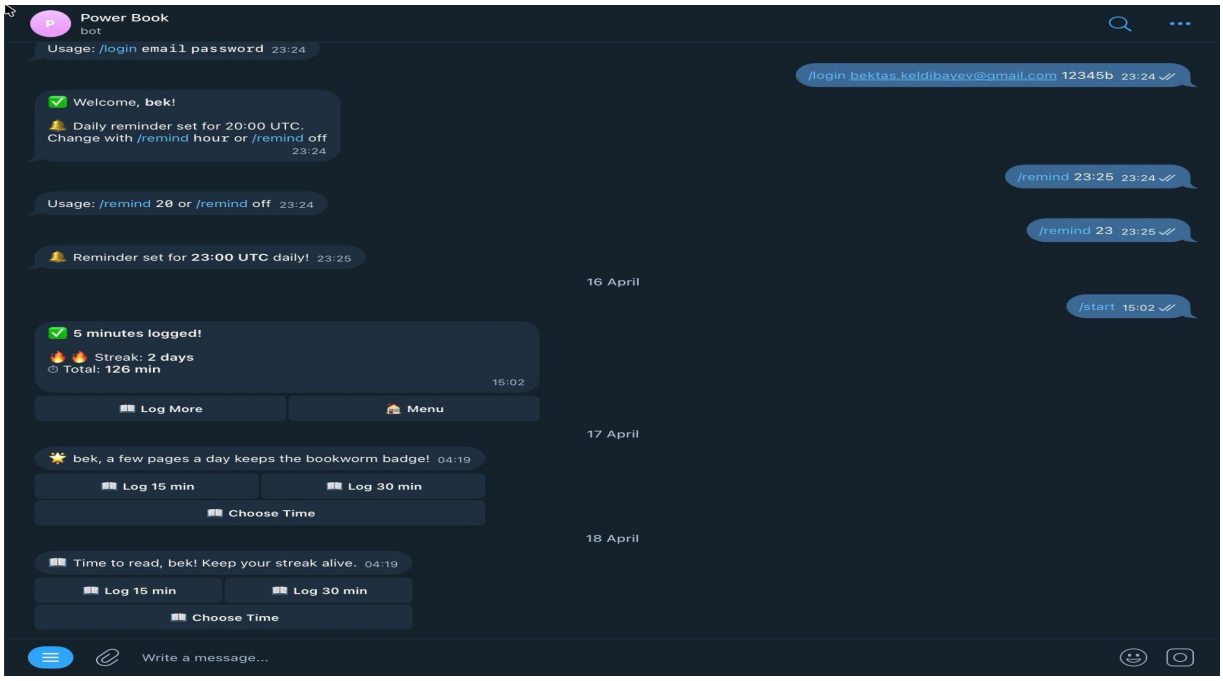
The Telegram bot is an independent Python service, developed using the python-telegram-bot library. The initial one was a slash-command interface. There was no way to learn what commands were available, and even experienced users continued to mistype arguments frequently enough to be frustrated. The entire bot was rewritten with an inline-keyboard menu, and this is the version that is described here.

Menu-driven interface. Clicking /start now results in a main menu containing six inline buttons, which are: Log Reading, My Stats, Streak, Competitions, Reminders and Settings. There is a sub-screen with each button and a Back to Menu button is always at the same position, hence navigation can always be reversed. The BotCommand API also registers the same commands, allowing anyone who prefers typing to still use the built-in command picker on Telegram to view what can be used.

Authentication flow. The user logs in or logs out by sending username and password as a typical chat message; the bot guides the user through the process in a conversational manner, and does not require them to type in command arguments. Upon authentication, the bot maps the Telegram chat ID to the backend user ID thus allowing the session to continue across messages without the user having to log in again after each message.

Daily reminders. A task is scheduled to execute every day at a specifiable time - 20:00 server time default and a reminder is sent to each user who has subscribed. Users who have already read that day are not shown, since notifications about something that the user has already done are nothing but

noise. The reminder is also provided with built-in quick-log buttons (15, 30, 45, 60 minutes) in such a way that a user can use only one tap to create a session without the necessity to open the app.



5.10 Frontend Design

The visual direction of the frontend is brutalist: big typography, high contrast, minimal ornamentation, and a clear visual hierarchy per page. It was an intended call. A reading platform, which hides the real data under gradients, soft shadows, and illustrative drawings, struggles with itself. Colour is generally kept to the minimum and only items that really have meaning are coloured, such as green to signify positive state, amber to warn of possible dangers, and red to signify destructive behaviour. The rule is the only thing that is stretched to the limit here, namely the twelve-colour avatar palette that is described below.

There are ten pages in the frontend.

Landing page. An introductory page having a hero section, a four-step guide on how it works, and an overview of the XP and levels system. It is primarily there to provide a first-time visitor with enough background to make a decision of whether to sign up.

Login and Register. Minimal forms with inline validation. A click on either of them substitutes the button with a skeleton loader to enable the user to immediately receive feedback that they have clicked.

Dashboard. Authenticated home page. Displays the streak, the number of minutes logged today, the XP bar, and competitions that the user is in or about to join.

Competitions list. Status Tabs sort by status All, Active, Upcoming, Completed. Each tab has a count badge indicating to users that they can see at a glance whether there is anything new to join.

Competition detail. The competition metadata, the leaderboard (paginated), a join button (when joinable), and (after the competition is over) the gift-exchange results.

Create competition. Name, description, start and end date and scoring rule form.

Public profile. Accessible without the need to log in. Displays the XP bar, level name, connected Telegram handle (where the user has an account), badges, and a list of competitions with final position.

FAQ. Accordion-based questions about registration, competitions and gifts, and the Telegram bot.

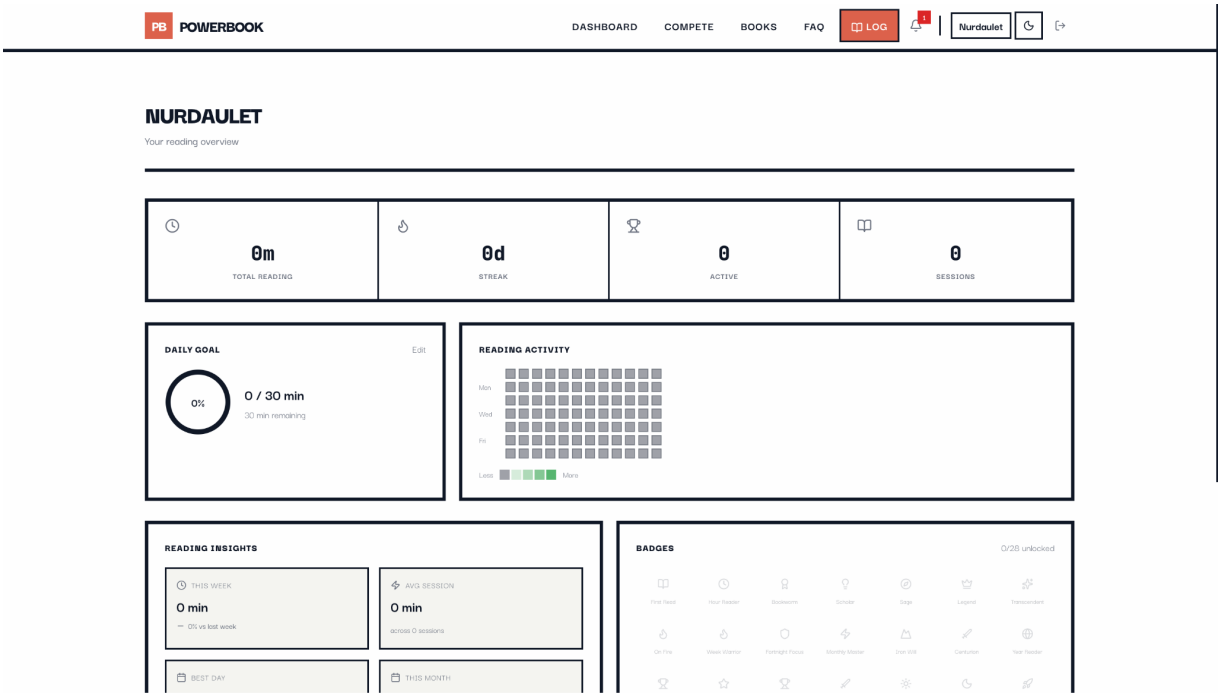
Admin panel. Covered in Section 5.3.

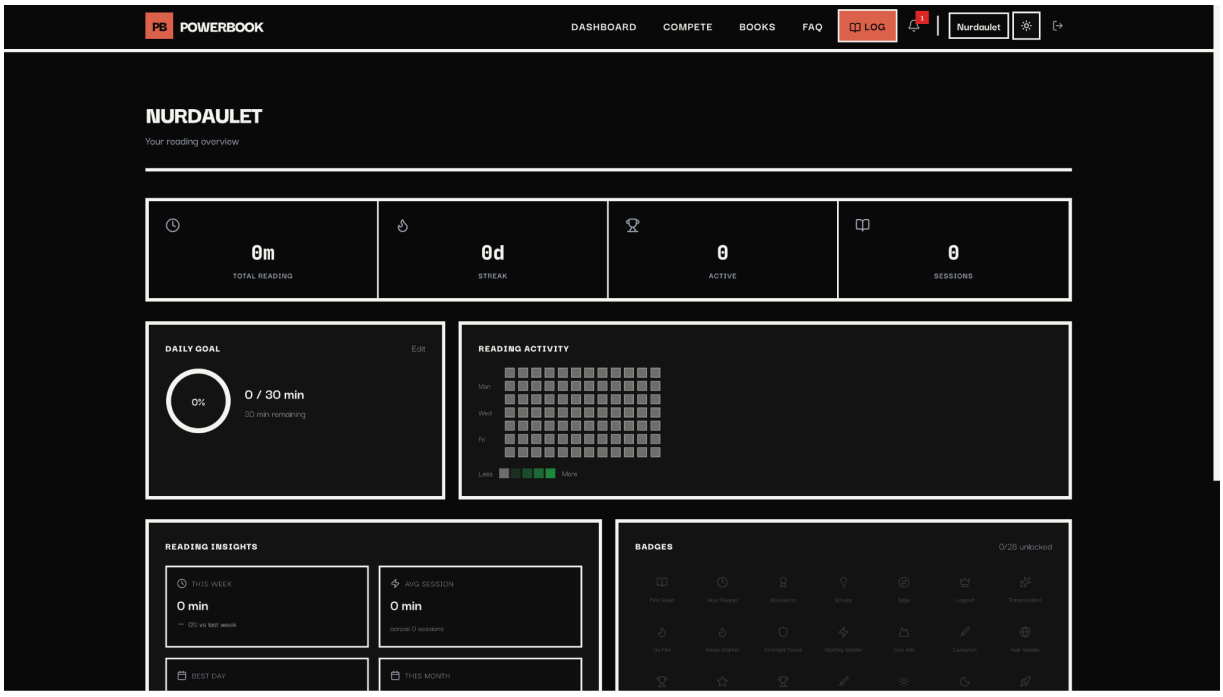
There are a few cross-cutting UI details that can be mentioned.

Skeleton loading states. All data-dependent views use skeleton placeholders - card skeletons in the dashboard, row skeletons in the leaderboard - as the actual content is being loaded, instead of the default text of Loading. The idea is to convey the form of what is in the pipeline such that the layout does not physically leap as the data is received.

Avatar colours. All users have a uniform avatar colour, based on a hash of the display name, with twelve different colours in the palette. It is the most inexpensive method of visual scannability of leaderboards and participant lists, not requiring users to upload a picture that they do not wish to upload.

Light and dark theme. Themes can be switched on a toggle in the header, and the option is saved in localStorage to be remembered next time the page is visited.

- 



Light and dark theme of “Your reading overview”

6. Project Execution

The development was over two semesters. Fall 2025 was primarily research and design and getting the skeleton of the system standing up. It became a product in the spring of 2026: Leaderboards, the Telegram bot, gamification, deployment, and the real work of getting the thing ready to a demo.

6.1 Semester 1 — Fall 2025

Weeks 1-4: Planning and research. The initial month was devoted to reading and not writing code. Gamification, habit formation and current reading app literature review; a competitor pass through Goodreads, StoryGraph, Bookly and Habitica to understand what portions of the design space had been occupied already. These requirements were the result of a MoSCoW exercise - must-have, should-have, could-have, won't-have - where the first decision to postpone a native mobile app and OAuth sign-in was taken. The final technology stack to become locked down was the last. Both Node.js and Go would have been good choices as the backend; we chose Go primarily due to its concurrency and single-binary executability, and we were willing to bear the learning cost associated with it. The Vue vs. React debate was less protracted and React prevailed due to more members of the team having used it.

Weeks 5-8 Architecture and database design. The hexagonal construction was completed during this stage, as well as the initial variant of PostgreSQL schema. An OpenAPI document was written describing API endpoints prior to any code being written that defined the envelope of the response and how errors would be returned, which proved a valuable discipline as it caused us to make important decisions before either half of the team was committed to an implementation. The development environment around Docker Compose was established, and PostgreSQL and Redis were both deployed in containers alongside the Go service.

Weeks 9–12: Core backend. The first endpoint was user authentication, which included registration, log in, issuing JWTs, and the reading-log CRUD endpoint, and the competition lifecycle (create, join, close). The domain layer was also written with unit tests; the hexagonal architecture was justified in this case, since the tests were run with bare structs without accessing the database.

Weeks 13-16: First integration and frontend. The app router was used to scaffold the Next.js project. We developed the authentication flow, reading-log page and competition-list page and then linked them to the backend. End-to-end integration found a few minor discrepancies between the OpenAPI spec and the implementations - nothing critical, but enough to confirm that the document was worth the effort of writing.

End of Semester 1 deliverables: a functional authentication system, reading-log system, a simple competition system, three main frontend pages and complete project documentation prior to mid-point review.

6.2 Semester 2 — Spring 2026

Weeks 1-4: Leaderboard and gift exchange. The PostgreSQL-aggregate backed leaderboard was substituted with the Redis-backed version and connected with the competition system. Pagination, the Find-Me button and top-50% / bottom-50% banner all appeared here. The gift-exchange algorithm was implemented in parallel, with the neutral-median treatment of an odd number of participants. This was the initial phase in the project where the competitive features began to be more of a product and not a set of endpoints.

Weeks 5-8: Gamification and Telegram bot. Over these four weeks, the XP system, the ten levels, and the thirty badges were in. The other significant work was the Telegram bot. It was initially designed to be a slash-command interface, and we re-implemented it around the six-button inline menu (Log Reading, My Stats, Streak, Competitions, Reminders, Settings) when it became apparent that the command form could not be found. That rewrite resulted in the authentication by plain text messages, the Back to Menu button, which remains constant, and BotCommand registration. Bot-to-API authentication was set up using service tokens in the Authorization header.

Weeks 9-12: Polish and deployment. This was the most prolonged sprint and the one that contributed most to the platform being complete. This window shipped the admin panel, public profile pages, the FAQ, the landing page with its How It Works section and XP overview and the dark/light theme. All "Loading." placeholders in the codebase were substituted by skeleton loading states. The competitions list received badge counts on the tabs. Display-name hashes were used as avatar colours. In the backend, the 1,440-minute daily limit and the auto-close scheduler were introduced, React Query caching was modified, and a separate GET /competitions/:id endpoint was sliced off, so that the detail page did not need to query and filter the entire list. It was all brought together at the end: the backend was containerised and deployed on DigitalOcean behind Caddy, the frontend was deployed to Vercel, and a CI/CD pipeline was connected via GitHub webhook to ensure that each push to the master led to an automatic redeploy. The cycle of release had changed to run these steps on the server to merging the PR in a single afternoon.

Week 13-16: Testing and evaluation. Seed data (50 demo users and 3 completed competitions (20, 30 and 40 participants) that completed a complete gift exchange) was tested internally. The performance indicators were recorded and the final report and presentation were compiled during the last two weeks.

6.3 Challenges and Solutions

Auth-token persistence. The interceptor on the frontend clearing the stored JWT on all HTTP 401 responses makes sense, until it dawns on you that a 401 can also be returned by routes that have nothing to do with authentication - a permissions check on the admin panel, a stale token just in time before it was renewed. Users were being logged out in the middle of a session due to reasons that were not related to their credentials being invalid. The bug was to make the interceptor narrow: just clear the token on 401 responses that were sent by the auth endpoints themselves, and leave the rest of the codebase to treat other 401s as normal errors.

Leaderboard data-format mismatches. In the instances where the frontend had been coded expecting a populated list to always be returned, the API returned empty arrays and {"found: false" envelopes. The outcome was that a few select pages went white-screen crash, such as empty competitions, brand-new users, competitions that had no active participants yet. We put explicit TypeScript type guards and null checks at the deserialisation boundary and made the empty and not found cases first-class states of the UI instead of exceptional cases. The implication behind this was that the vacant case was worthy of design and not a guard.

Excessive prefetching. The default behaviour of TanStack React Query is to pre fetch data when the user hovers over or otherwise expects a link. This, together with an extensive list of competitions, had the effect of scrolling through the page silently making ninety-plus API calls in the background, many of which were never utilized. We disabled prefetch on all of the types of links and increased the default stale time to two minutes. The solution is three lines of code; it took an afternoon of poking at the Network tab to find out where the requests were originating.

CI/CD pipeline. The initial deployments were done manually, SSH in, pull, rebuild the container, restart. This served a five-person team, until it didn't: somebody would forget to migrate, or push to a non-master branch, or forget to update an env file and take the API offline fifteen minutes. We configured a GitHub webhook that would call a redeployment script on the server, and after that redeployed the backend automatically with each master push. Releases ceased to be a source of errors.

6.4 Team Collaboration

The team used two GitHub repositories - the powerbook and senior-project code repositories - where code was reviewed on all non-trivial pull requests. The day-to-day organization occurred in a Telegram group. Components were divided by responsibility, and wherever a change affected more than just one component, there was overlap:

Toktar Sagyngali - PostgreSQL schema, migrations and the documentation that would later become the majority of this report.

Nurislam Bakhtybayev - the Go backend: authentication, competitions, reading logs, the admin endpoints.

Bekzhan Aktoreev - the Redis leaderboard service, the auto-close scheduler, and the performance benchmarking.

Bektas Keldibayev - the Next.js frontend, Tailwind design system, dark/light theming, and React Query configuration.

Nurdaulet Otegenov - the Python Telegram bot, menu-based UX, daily reminders, and deploying the bot.

The integration points - API contracts, deployment configuration and anything that traversed a repository boundary were addressed collectively typically via a short call and then a shared document which both sides could edit prior to the change landing.

6.5 What Could Have Been Improved

In hindsight, there are three things that we would have done differently.

The competition and management flow of gifts exchange became even more complex than we had planned. Both of these features are simple-looking on the surface, but they share numerous small states competition not yet started, competition live, competition closed but gifts not yet confirmed, odd number of participants with a neutral median, and each state had to be treated to UI differently. When we had completed that work, the admin panel and the profile pages were on a compressed schedule, and they both launched with less polish than we would have preferred.

The actual user testing did not occur as soon as it should have. An informal test was scheduled at the end of Semester 1 under the original plan, but the scope pressure caused it to be slipped and when we did actually run the actual sessions we were well into Semester 2. Some few onboarding problems, primarily relating to explaining the competition-join flow to a first-time user, would have been identified much earlier and fixed at a low cost should we have done the testing we intended to do.

Go was a calculated decision, and we are proud of it, but a few of our team members entered into production Go development without prior experience, and the learning curve consumed the first semester more than it should have. A more intensive Go deep-dive, that is, a week of shared reading and shared pair-programming, before a single line of production handler code was written, would have flattened that curve and would likely have saved us ten working days in the first semester alone.

7. Evaluation

7.1 Evaluation Methodology

It is cumbersome to assess an older project. Academic schedule does not provide the type of live-user study that would realistically determine whether the platform alters reading behaviour, and we were not able to afford a real cohort. We could have done what the other three axes needed to do, which is to test each feature the platform claims to work with, and also to test the system to what load it can bear in normal operation, and to test the interface with the devices that people would plausibly open it on. The testing below is a combination of team-based testing, synthetic testing using the deployed backing, and a demo test with a seeded database that would appear to be a real platform in action.

7.2 Performance Evaluation

The system deployed was tested in simulated load conditions. All endpoints were tested over 1,000 consecutive requests of a warm cache; p95 response times and per-process memory footprints were captured.

Metric	Result
Backend API response time (health, auth, competition list)	< 50 ms
Leaderboard query for 50 participants (Redis ZREVRANGE)	< 20 ms
Competition-close endpoint (including gift-pair computation)	~150 ms
Frontend initial page load (Vercel CDN, 4G network)	< 1.5 s
API Docker process memory footprint	37 MB
PostgreSQL container memory footprint	41 MB
Redis container memory footprint	4 MB
Total backend memory footprint	82 MB
Concurrent users tested	50, across 3 competitions

The leaderboard endpoint is the one we watched most closely, since it is the piece that drove us off PostgreSQL aggregates in the first place; at under 20 ms for a fifty participant query it is now comfortably faster than the rest of the stack. Competition-close is the slowest endpoint at roughly 150 ms, but this is acceptable because the work it does - gift pairing, XP awards, and competition-badge evaluation for every participant happens inside a single transaction and runs exactly once per competition lifecycle. The total backend footprint of 82 MB matters mainly because it fits inside a free-tier VPS with room to spare, which is a hard requirement rather than a nice-to-have for this project.

7.3 Usability Evaluation

The frontend was assessed on five pragmatic levels as opposed to a rubric.

Mobile responsiveness. Each page was pretested at various viewport widths starting with 320 pixels - an iPhone SE at its narrowest - and at a desktop monitor with the widest viewport of 1920 pixels. Most of the work was done in tailwind breakpoint utilities, and the leaderboard in particular would come to a crawl in a two-column display where the six-column wide one would be illegible.

Accessibility. Semantic HTML is utilized in navigation, forms and headings as opposed to styled divs. Colour contrast is covered on WCAG AA both dark and light and this is not a goal but an absolute requirement. All the interactive aspects are accessible through keyboard, so those who cannot use a mouse (or simply do not want to) can navigate through the whole application.

Loading feedback. The old text Loading. appears as skeleton placeholders on all data-dependent views. The operational difference is that the layout is no longer changed when data is received - the skeleton is the same shape as the final content, and the eye of the user does not need to refocus. Efficient actions display a toast notification to indicate to the user that the click has been registered.

Error handling. The occurrence of API errors is manifested in human-friendly messages as opposed to raw HTTP codes. The clearest of them is the daily-cap guard: rather than being rejected on an overflowing submission with a 400, it returns the number of minutes remaining to the day, and allows the user to edit the entry accordingly.

Navigation consistency. Each web page has a Home in the header and each Telegram-bot sub-screen a Back to Menu in the same location. Whether they are in the web app or in the bot, a user must always be able to revisit an already familiar place with a single tap.

7.4 Evaluation Summary

The assessment validates what we set out to demonstrate within the scope we were able to assess. All the fundamental requirements are deployed and operational on the deployed platform. Performance is comfortably within the free-tier budget with significant headroom - the leaderboard, the most of which we were concerned about is the fastest endpoint. The usability has been tested on sizes of the devices, themes, and channels of interaction. The seeded demo of power-book.app was given a load of fifty users and three full competitions with gift exchanges, as close to a real load as we could build without an actual cohort. What the evaluation is not able to determine - whether the competitive mechanics in fact enhance sustained reading in real users - is the subject of future research that would be required by the work.

8. Conclusion and Future Work

8.1 Conclusion

This project was meant to create a functioning web platform which transforms reading into an activity that people do together, and to do it with the normal limitations of a student group: no budget, two semesters, five individuals balancing the work with other coursework. That part we did. The system is functional at power-book.app, it has been subject to two complete competition-and-gift-exchange cycles with seeded data, and the benchmarks perform.

Some of the technical choices we had to make early on proved to be the correct ones due to reasons that we only realized much later. Go supported concurrent reading-log submissions without our necessarily having to think hard about it; the single-binary deploy model made the backend operationally boring. Redis sorted sets provided us with leaderboard reads that were within 20 ms and the original PostgreSQL-aggregate version had already started to become visibly slow. Our hexagonal structure meant that we could replace our in-memory leaderboard with Redis halfway through the project without domain code rewrites, a day of work rather than a week of work. And the GitHub-webhook deploy pipeline eliminated a category of errors on the release day, which had cost us real time in the first half of semester one of Spring.

What the project fails to show is that competitive features really cause more people to read. That is a statement that we would need to have supported through a live user study, and the test we could perform in a college schedule is not a test. The honest truth that we can tell is that the platform itself is functional, that it is scaling to the concurrent usage that it was meant to be used, and that the architecture is at a level where it can be expanded. And now to what we would construct the next.

8.2 Future Work

We also maintained a list of ideas of things we desired to create that we just could not fit into the portfolio of senior project. The more practical of them, roughly in the order in which we should go about them:

Book integration. The most apparent missing feature of the existing product is that PowerBook measures reading time, but not reading content. Integrating with the Open Library or Google Books API would enable the user to tie sessions to individual books, and opens an entire bandwidth of features that cannot be developed upon top of raw minutes alone: per-book progress, genre statistics, and badge of having finished the book, etc. This is what we would first do.

OAuth sign-in. Registration by means of passwords is an area of tension that fails to justify itself. The inclusion of Google and GitHub OAuth would reduce the registration process of four fields to a single

click and would eliminate the most frequent support query that we could possibly imagine had to be supported by us before we could even think of it (I forgot my password).

Friend system. This platform now only reveals other users within competitions. The service should add follow/unfollow relationships, friend-only contests, and a feed of what friends are reading to provide the platform with a second engagement loop without necessarily having to be in a competition when it happens.

Team competitions. One of these forms is the individual competition; another is the group-versus-group competition; and in our experience that of running early tests is a more lasting one--a team member who is not at his best this week is held up by the others instead of going off on his own. The existing schema would accommodate the introduction of aggregate reading time, team-level gift exchange, and team badge quite with slight adjustment.

Native mobile application. The Telegram bot takes much of the mobile aspect, yet it is not an alternative to a decent application. A native client based on the current REST API would include native push notifications (which are similar to Telegram reminders but not identical), home-screen notifications, and offline reading-log buffering in sessions that were logged without connection.

Machine-learning analytics. When we have book integration, the reading-cadence and competition-performance data that we have been gathering can then be used to provide personalised recommendations: what to read next, what time of day the user reads best, what types of books are likely to engage their interest. It is a feature that has the greatest potential payoff when the data to back it is available, which is not urgent but it is the one with the greatest potential payoff.

Full admin-panel tooling. The existing administration panel includes user management and statistics. Moderation of competition, badge issuance, and bulk user processes still need direct access to a database, and that is okay in a 5-person team but would fail in a system that runs on people other than its authors.

Multi-region deployment. Our existing single-VPS configuration has been good enough to handle all that we have been putting into it to date, but the implementation is already architecturally ready to scale horizontally stateless JWT authentication, Redis-stored leaderboards, a replicable database. What we would construct in the event the platform were adopted institutionally would be a multi-region deployment behind a load balancer, with a Redis replica set.

9. References

Caddy. (n.d.). *Caddy web server*. Retrieved April 19, 2026, from https://caddyserver.com/

Deci, E. L., Koestner, R., & Ryan, R. M. (1999). A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. *Psychological Bulletin, 125*(6), 627–668.

Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference* (pp. 9–15). ACM.

DigitalOcean. (n.d.). *DigitalOcean — The cloud for builders*. Retrieved April 19, 2026, from https://www.digitalocean.com/

Docker. (n.d.). *Docker: Accelerated container application development*. Retrieved April 19, 2026, from https://www.docker.com/

Evans, E. (2003). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley.

Fogg, B. J. (2019). *Tiny habits: The small changes that change everything*. Houghton Mifflin Harcourt.

Gin. (n.d.). *Gin web framework*. Retrieved April 19, 2026, from https://gin-gonic.com/

The Go Programming Language. (n.d.). *Go*. Retrieved April 19, 2026, from https://go.dev/

Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does gamification work? — A literature review of empirical studies on gamification. In *Proceedings of the 47th Hawaii International Conference on System Sciences* (pp. 3025–3034). IEEE.

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON web token (JWT)* (RFC 7519). Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc7519

Lally, P., van Jaarsveld, C. H. M., Potts, H. W. W., & Wardle, J. (2010). How are habits formed: Modelling habit formation in the real world. *European Journal of Social Psychology, 40*(6), 998–1009.

McGonigal, J. (2011). *Reality is broken: Why games make us better*. Penguin Press.

Meta Platforms. (n.d.). *React — A JavaScript library for building user interfaces*. Retrieved April 19, 2026, from https://react.dev/

The PostgreSQL Global Development Group. (n.d.). *PostgreSQL: The world's most advanced open source relational database*. Retrieved April 19, 2026, from https://www.postgresql.org/

PowerBook. (2026). *PowerBook — Competitive reading platform*. Retrieved April 19, 2026, from https://power-book.app/

python-telegram-bot Contributors. (n.d.). *python-telegram-bot*. Retrieved April 19, 2026, from https://python-telegram-bot.org/

Redis Ltd. (n.d.). *Redis*. Retrieved April 19, 2026, from https://redis.io/

Ryan, R. M., & Deci, E. L. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist, 55*(1), 68–78.

Tailwind Labs. (n.d.). *Tailwind CSS — Rapidly build modern websites without ever leaving your HTML*. Retrieved April 19, 2026, from https://tailwindcss.com/

TanStack. (n.d.). *TanStack Query — Powerful asynchronous state management*. Retrieved April 19, 2026, from https://tanstack.com/query

Vercel. (n.d.-a). *Next.js by Vercel — The React framework*. Retrieved April 19, 2026, from https://nextjs.org/

Vercel. (n.d.-b). *Vercel: Build and deploy the best web experiences*. Retrieved April 19, 2026, from https://vercel.com/