Gymnetics: Design, Implementation, and Evaluation of a Personalized Supplement and Fitness Application

Marlen Cuevas Duarte, Jia Lau, Adam Miftahelidrissi, Diya Patel, Samiha Shareef
Seidenberg School Of Computer Science and Information Systems
Pace University

Abstract

We conceptualized, developed, and deployed Gymnetics, a trainer-led web-based solution that connects clients with structured training programs and training products to supplement the training plans. The final product/MVP lends an easy onboarding and like to regular and new users and they get a consistent and measurable end to end experience where a user learns about and gets to understand which trainer approach is best suited to them and trains them on products and get recommendations on new products with clear explanations of the rationale behind the recommendations. The production of human-readable, rules-based linking of trainer focus tags to product metadata provides concise and understandable explanations as to why this item as the use of opaque scoring was avoided during the MVP stage[7]. The user experience delivered, as designed and screen-readable, supported and resistant, orderly hierarchy, readable high-contrast palettes, access using keyboard only, relatively constant layouts under load, clear empty states and error states, and actions set up to be guarded to prevent inconsistent transitions. We captured style and design in a team wiki and we planned best-in principle seams with adjacent components like containerization and payments, and we also continued with the narrative burndown with actual delivery. The outcome sees a lean MVP slice that we can demo without any last-minute fix-ups, returning value back to users and trainers and building a solid foundation on which we can later drive machine-learning and product enhancements.

Introduction

In today's technology driven and health focused society, people are constantly searching for smarter and more tailored ways to manage their fitness and nutrition. While countless fitness applications exist, many fail to offer meaningful customization. These platforms often provide generic workout plans, basic meal logs, or surface level guidance that does not adapt to individual goals or body types. As a result, users are left feeling lost, unsupported, and inconsistent in their progress.

The Gymnetics web application, developed by Team Byte Squad, addresses these challenges through a dynamic fitness platform powered by artificial intelligence. Gymnetics is designed to generate workout plans and supplement recommendations based on a user's unique information, including their height, weight, age, and fitness goals. Whether someone is new to the gym or preparing for a competition, the application adjusts its recommendations accordingly to help users reach their full potential. This paper explores the design, development, and broader impact of Gymnetics as a modern solution to a common problem in health and fitness technology.

Gymnetics is a comprehensive fitness and nutrition planning platform that uses artificial intelligence to generate workout splits, macro nutrition breakdowns, and supplement recommendations for each user. Upon registration, users input their physical details and training objectives. The system analyzes the input using DeepSeek AI and instantly provides a suggested routine, meal guidance, and supplements that support those goals. Users can also browse a supplement store, add products to a cart, and go through a mock checkout experience. The platform is built using React for the user interface, Spring Boot for backend operations, and integrates services such as Stripe and AWS S3. This project focuses on the development and usability of Gymnetics and evaluates its effectiveness in providing an intelligent and accessible fitness experience.

Many popular health and fitness applications offer limited personalization and require users to manually track workouts or nutrition. MyFitnessPal, for example, is commonly used for calorie tracking, but does not provide intelligent workout suggestions or supplement guidance. Another popular web application, Fitbod offers some level of adaptive training, but does not include support for nutrition or product recommendations. Additionally, another application called JEFIT offers structured routines, but lacks the ability to generate plans based on unique physiological input. Some academic research has explored the use of artificial intelligence in fitness and nutrition, but most implementations are theoretical and not yet applied in a public facing tool. Gymnetics builds on these ideas and presents a unified solution that blends data analysis, AI decision making, and user centered design into one platform.

Gymnetics advances the field of health technology by combining several major features into one functional product. It provides smart workout planning, nutrition support, and supplement suggestions all within a single platform. This reduces the need for users to switch between multiple apps or consult different sources. By using artificial intelligence to tailor its output, Gymnetics simplifies the decision making process and helps users stick to a consistent routine.

The Gymnetics application is especially useful for individuals who struggle with routine, motivation, or lack of expert knowledge. For instance, Maria is a college student who finds it hard to stay consistent and doesn't know what supplements to take. James is an aspiring bodybuilder who works out frequently but has no structure in his nutrition. Gymnetics supports both users by providing structure, clarity, and curated product suggestions such as protein powders, creatine, and multivitamins. This helps users avoid guesswork and build habits that align with their goals.

This study demonstrates how artificial intelligence can be effectively applied to health and wellness in a way that is user friendly and scalable. It also shows how intelligent systems can simplify processes that are often confusing or time consuming. By building a tool that adapts to user data and offers clear guidance, Gymnetics serves as a model for future health applications. It proves that fitness planning can be both accessible and intelligent, and that the gap between knowing what to do and actually doing it can be closed through smart design.

III. System Design & Architecture

Gymnetics uses a three-tier design with a React and TypeScript frontend, a Spring Boot REST API backend, and a MySQL database. The frontend (React 19.1.0, TypeScript, Material-UI v7) uses a component-based structure with Context API for state. Authentication is handled with JWT. Other features include profile editing, settings (which the mvp includes password changes and account deletion), a supplement store with a cart for checkout, AI supplement recommender, and AI generated workout routines. This is all delivered in a responsive frontend that is mobile friendly. The backend (Spring Boot 3.2.0 using Java 20 with Spring Security) utilizes clean REST endpoints and handles business logic in the following services. 'AuthService' handles sign-in, sign-up ,and profile management. .SupplementService' is for product management. 'StripeService' handles the flow for payments. 'S3Service' for the storage of images. Lastly the 'CustomUserDetailsService' is for security integration. The persistence of data is handled by MySQL 8.0 using JPA/Hibernate with core entities for 'User' (auth/profile/goals), 'Supplement' (pricing/inventory), 'Purchase' (orders/payments), and `Product` (Stripe mapping). Additional integrations using API's include Stripe for secure payments, AWS S3 for image storage, and DeepSeek AI to generate personalized workout plans.

Decisions and rationale for the design and architecture fall to practicality, flow between tech, and the team member's skillset. React and TypeScript were chosen for the frontend because of reliability and ease of use in a rapidly growing tech tech product. Not too many team members have experience using these two technologies, but with the extensive documentation to aid the group, progress could be made even without prior experience. For the backend, Spring Boot was a clear choice due to our collective experience using java due to the requirement of proficiency in java that Pace University requires of us as students. Outside of this spring boot also allows for integrated security with JWT and supports our desired backend architecture. For the database we have chosen MySQL because of another collective experience with using MySQL as students at Pace University. It is also a reliable relational database that aligns well for Gymnetics transactions. Authentication uses stateless JWTs with Spring Security for authorization. Continuing to that point, passwords are hashed with BCrypt, and CORS is configured to enable safe frontend to backend communication. The frontend state relies on Context API and local storage for a lightweight persistence of sessions and carts. The API adheres to the structure set by the DTOs. Images are stored using S3 due to prior experience handling files using S3. Payments occur through Stripe with webhooks and are used as well due to ease of set up and prior experience using Stripe for payment. The benefit of this system architecture is the ability to build quickly by allowing stripe to handle payments and Spring Security to handle security. The frontend and backend allow for horizontal scaling.

The repository includes a class diagram of entities and relationships (`documents/class-diagram.png`), a high-level component view (`documents/architecture-diagram.png`), sequence diagrams for authentication and profile updates (`documents/sequence-diagram.png`), an ER diagram for the schema (`documents/er-diagram.png`), and a context diagram showing boundaries and integrations (`documents/context-diagram.png`).

IV. Implementation

In terms of implementation, when we started working on user login authentication our initial plan was to use Firebase. After some time, we decided to move away from Firebase and switch to JWT authentication since it was something our teammates were more familiar with. For JWT authentication we created two APIs, one for login and one for registration. Both APIs use an auth service where the JWT token is sent, which allows user creation and login validation.

Another feature we built was the DeepSeek API, which provides users with recommended workouts based on the information they enter about themselves. For example, a user can input basic details along with the areas they want to improve and the workout duration. DeepSeek then generates a custom workout session tailored to those goals. Along with personalized workouts, we also included a supplement recommendation that suggests products from our store. All DeepSeek API calls are made from the frontend through deepseekService.ts and openaiService.ts, and to run these features we needed to include DeepSeek secret keys.

We also implemented Stripe in the backend to handle payments. To run Stripe, the keys had to be added to the application settings in Spring Boot. Once configured, we could create items in Stripe with a price, image, one-time payment setup, and other metadata. Each item created is treated as an object with a PRICE_ID, which can be queried when making a purchase. Since our project uses a cart system rather than one-time purchases, we needed to work with a list of PRICE_IDs. For this we built a checkoutCart API in the StripeController. When the API is called with a list of PRICE_IDs, it redirects the user to a Stripe checkout link where they can complete the purchase. One of the biggest challenges was that, at first, we could only purchase a single item and not multiple ones from a cart. We overcame this by researching Stripe's documentation and using tutorials until we got the cart checkout working properly.

Lastly, we integrated AWS S3 to store images of the Stripe items. We chose this approach because storing images as blobs directly in a database is inefficient, especially in an enterprise setting. One challenge we faced was configuring the S3 bucket so that uploaded images could be made public. After adjusting the settings, we were able to upload product images, make them accessible through public links, and store those links in the database. This allowed us to display product images in the frontend while keeping storage efficient and scalable.

V. TESTING & EVALUATION

Our test work was focused on validating user stories against their acceptance criteria and testing the critical paths to deliver the onboarding, authentication, catalog, checkout, profile management, and visuals

A. Approach and tooling

Table 1.

Story (ID)	**Acceptance Criteria (summary)**	**Test Case IDs**	**Result**
US_01: Sign up	Create account; redirect to dashboard; profile plan saved	TC_03_02	Pass
US_04: Login	Valid credentials sign in; invalid shows error; session persists	TC_02_01/02	Pass
US_09: Checkout	Stripe success marks order paid; failure shows error; no double charge	TC_09_01/02	Pass
US_12: Edit profile	Update name/info; email change confirms; password change requires re-auth	TC_12_01/03	Pass
US_13: Delete Account	User logged out; data removed; resets blocked	TC_13_01/02	Pass

Figure 1

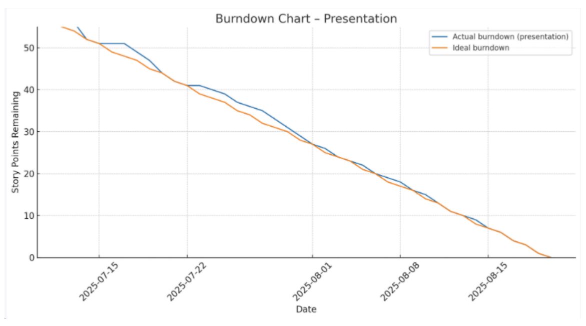

Figure 2

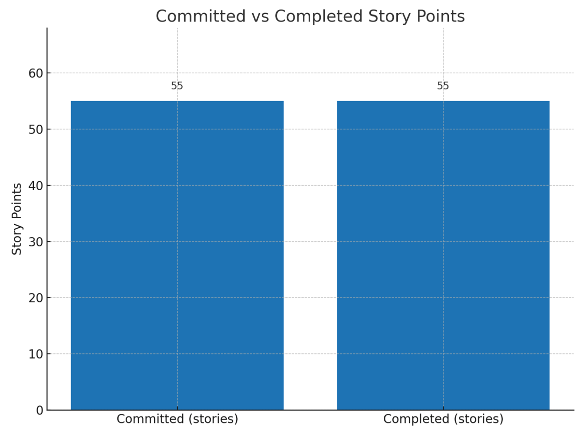

Table 1. Example story-to-test traceability and outcomes (expand if space allows).
Fig. 1. Sprint-2 Burndown Chart. Insert PNG from slide 27; two-column width; y-axis = 55; line touches ideal 5+ times (non-stair-step).
Fig. 2. Committed vs. Completed Story Points. Insert PNG from slide 28; two bars — committed = 55, completed = 55 — with legend.

B. Coverage by user story

Our priority was to prove that what we deliver, actually works on all ends. We tested and executed scenario based tests that were directly tied to the acceptance criteria. We kept it manageable such as manual runs against the latest build, Stripe test cards to check the payment flows, and realistic product data for catalog checks. Negative paths such as invalid credentials, declined payments, and edge conditions for account changes were also tested.

Media from S3: Images are able to load without layout shift..

Account deletion: Deleting an account results in logging the user out, it removes their profile data, and blocks subsequent password reset attempts for that email.

Profile management: Customers are able to edit their name/info. They're also able to change their email although it requires confirmation. Changing password also requires re-authentification. Both invalidate old sessions.

Checkout via Stripe: Using Stripe test cards, success marks order paid and shows confirmation. Failure shows an error and leaves the order unpaid. Refreshes do not double-charge.

Cart and quantity changes: Customers can add, remove, and edit quantity amount which updates re calculates subtotals and totals correctly. Removing the last item shows an empty cart.

Catalog and product details: Supplements list shows title, price, and image. The detailed view includes description and price. Scroll position is saved if the customer chooses to return. Infinite scroll works as expected.

Forgot password: Registered email is used for the reset. It requires re-authentification, and an unregistered email returns a response indicating that it is unregistered. Password policy feedback rejects weak passwords.

Authentication basics: New registration and returning user login succeed. Invalid credentials display a clear error. Users are able to refresh and reset on logout.

AI workout generator and personalized recommendations: When launching the web application, users can create an account with verified email sign-in and use the AI assistant which recommends workouts and supplements based on your goals and experience.

C. Metrics

We planned 55 story points and completed 55 (100%). The Sprint-2 burndown shows steady, non-stair-step progress and touches the ideal line multiple times. The committed-vs-completed bar chart reflects a 1.0 ratio for this iteration. Both visuals were displayed as figures above. with captions per IEEE formatting.

D. Traceability

Every high-priority story maps to at least one passing test. Table 1 lists examples showing how I tied acceptance criteria to concrete tests and outcomes.

XI. Deployment Manual (Completed Approach)

We started with making the runtimes of both tiers standard, letting each of our teammates replicate the environment with ease. The service layer was also pinned to a long-term-support stack to freeze security patches and library behavior and the web client a modern stable toolchain on which development machines and the build server were consistent. This congruency removed the problem of work on my machine and lowered new contributor setup time. We listed the specific technologies and version details, and system package dependencies, and environment requirements in the project wiki with a short smoke-test checklist. The runtime was treated like the rest of the source code-described, versioned and reviewed, so drift was limited and upgrades became intentional as opposed to haphazard.

Access control and data integrity were principles. We have built our own specialized schema of relational databases and created an application account limited to those permissions and limited to schema access only. As opposed to learning the model and then freezing it soon, we used a migration early strategy that enabled the schema to develop safely as the feature work progressed. This implied small, auditable transformations rather than big and dangerous reorganization. We tested every migration on a staging instance prior to promoting, and maintained a light record of the then state of the schema and its intended invariants in the wiki. This minimized costs as it eliminated surprises in the process and it guaranteed that local, staging, and production were structurally aligned.

The media use-case required an object-storage bucket to store profile and product images and a limited scope of permissions on programmatic credentials. Public access was also turned off by default; access policies restricted to only those read and write operations the application actually had to do. We recorded the expected file storage structure, file naming scheme and caching requirements in such a manner that although easy image rendering was performed, we did not risk stale images. This provided the predictability of asset delivery without the hidden hangups that you can get when using any storage in an ad-hoc manner like broken links based on refactoring or cache control difference across deployment.

The security issues were taken into consideration prior to any component live working. We established a secret approach by which the sensitive values did not go to the source control and were centralized in a secure configuration store. The wiki records the location of secrets, who has permission to rotate them and the method used to rotate the secrets without any outage. In the case of third party integrations, we got test-mode credentials of the payment provider and the API key of an AI provider so that all initial checks and balances would take place in a non-production-ready environment. We also demarcated ownership lines: the owner of each integration was the one who had to renew credentials and update documentation after a provider changed its policies or endpoints.

The implementation of the system online had followed a sequential formula The service was set up to connect to the database, storage, sign and validate tokens, and communicate with the external providers[8]. We manually tested until the service would start cleanly, have a stable connection to each one of its dependencies, and would reconnect when transient failures occurred sensibly and without necessarily duplicating attempts or failing to release user-state. When it was found that those baselines could be repeated, we proceeded to the web client. We directed the client to the base address of the service and saw that the product data and images are displayed in a consistent manner as well as noticed that product recommendations were delivered with brief descriptions when the data came through slowly. Such an approach helped to maintain troubleshooting by making sure that the front-end work was not a cover-up of back-end problems.

With the main user-facing path working reliably gate to gate, we made a production deployment and parked both tiers behind an easy reverse proxy on a single domain. Terminating the transport-level security at the proxy and internally forwarding application traffic minimized cross-origin complexity and simplified authentication, cookie scope and caching policies. We confirmed that static contents were delivered with reasonable cache control and that error pages were not allowing operational details. We achieved safety at release-time by capturing exact client commits and service artifacts deployed and we limited the number of configuration toggles to keep this manageable and easy to track.

A holistic exercise to the integration of the commerce was done in test mode prior to any involvement with real users. We had a webhook endpoint that was registered with a payment provider and pointed to a sensitive path on the service and stored the signing secret amongst other operational configurations. End-to-end testing established visual user confirmations, proper state transitions of orders, and inventory signals. All of the public traffic was redirected on to the proxy; however, the service itself never faced the internet directly. We had switched to this regime of small frequent releases over the larger infrequent ones, so that we were able to correctly carry out rollbacks or a roll-forward in the event a regression did squeak through. Every release incorporated a short change note and a reference to the checklist in which the validation was performed.

The level of operative procedures placed emphasis on visibility and not over collection. We stored short logs of request traces, coarse-grained recommendation decisions, and pointers to slow external dependencies but not personally identifiable attributes or sensitive attributes. After every deployment, we did a simple availability test that simulated the way the real users used it: the client loads, media loads, sign-up and sign-in work, recommendations are provided, and transaction works, e.g., confirmation screen appears. In the cases where latency was a concern, we also indicated the slowest response times of the endpoints and charted it against the informal service goals to guide tuning.

When personal errors were committed, we preferred minimal effective repair and we checked against that same checklist of user paths. Whether configuration or code, we just did a fix, updating the secure store and documenting the rationale; whether a thin hotfix with the reasons being clear from a cleanly scoped diff. Each resolution was followed by an update to the wiki entry on that particular subsystem so that the knowledge gained was not stored in memory but group knowledge. With time, such an approach lowered the mean time to restore and minimize repetitive errors by creating an easily accessible playbook of common failure modes.

Lastly we tested basic resiliency out of the happy path. We made sure that transient upstream faults would cause polite retries with limited backoff and idempotence was maintained with sensitive operations to avoid a second instance of the same operation being run. We had periodic dependency exercises- artificial storage slowness or temporary loss of a database- to see how the application responded to pressure. Such exercises confirmed that our protective patterns were not mere theory and gave confidence to the team to know that the system would not disintegrate under less than ideal conditions. These release and operational decisions made releases predictable, ensured the platform could be trusted in real world use, and left it primed to receive the addition of features described in the future-work roadmap.

VII. Results & Discussions

In terms of final features delivered, I would say we completed our task in the sense that we have a fully functional full-stack application. All MVPs were met, though sometimes with different technologies than originally planned. For example, we intended to use Docker at the start of the sprint but decided to pivot to a simpler setup so everyone could run the program more easily. Another pivot was moving away from Firebase. While Firebase seemed better for scalability, the time crunch made us stick with JWT, which still provided the same functionality for logging in and creating new accounts. For the checkout page, everything worked as intended with the Stripe implementation. When users add items to the cart, it redirects them to the checkout page where they can purchase supplements. The AI recommender that analyzes a user’s data also worked as expected, and we kept it through to the end without changes. Lastly, we used AWS S3 to store Stripe images. This worked smoothly as well, since we were able to save the image links in the database instead of storing them as blobs.

VIII. Future Work

Future aspirations of Gymnetics is to deepen the core experience while setting up for further scaling. Gymentics would move in a direction to be the all inclusive health and fitness hub for people to do all interactions with. For better user experience, social features such as further profile customization, friends, shared workouts, and some gamification. This would include badges, leaderboards, and challenges. Keeping a log on workouts and calories would be the next step as well. Gymentics would move in a direction to be the all inclusive health and fitness hub for people to do all interactions with. AI will further personalize supplement and workout recommendations from observed behavior and introduce an AI personal trainer. The supplement store will continue to grow with more partners for stock and low-stock alerts for the user. Additionally subscriptions for recurring purchases, and reviews for more user interactions.

IX. CONCLUSION

This work has a stable MVP that people can actually use. Overall, customers are able to sign up, log in, utilize the supplement and workout generator, browse products with images, manage a cart, and check out through Stripe. On the account side, users can also edit their profile, reset a forgotten password, and permanently delete their data.The features we committed to at the start of the sprint were the same ones we delivered at the end.

This effort was challenging in good ways. We had a few hiccups along the way related to choosing the stack, struggling with setup, and testing thoughts/ideas that looked promising but didn’t make the MVP such, parts of the trainer dashboard and nutrition features. Cutting those isn’t considered a failure but it was scoping. We got better at prioritizing our duties so that we could ship the core experience and test it end to end.

Along the way, our communication and organization improved immensely. We committed to our Jira board. Whenever we disagreed or had any sort of confusion, we referred back to the basics; our acceptance criteria. We had a smooth rhythm. We collaborated through Zoom, wrote down our tasks on Jira, executed them, and then met up on Zoom again to speak about what we did. We also learned to document decisions. There were times where we would say we were going to do something but not write it down in our group chat or on Jira. The struggles were obviously part of the process and they will aid us in our expectations for future work. We learned how to scope realistically, validate quickly, and reserve some time for testing and polishing. Communicating clearly and delegating are habits that will translate directly to any internships or full-time roles that we will have later on.

References

[1] Amazon Web Services, "Amazon Simple Storage Service (S3) Developer Guide", https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html, accessed August 2025.
[2] DeepSeek, "DeepSeek AI model overview and usage", https://www.deepseek.com/, accessed August 2025.
[3] Fitbod, "Fitbod – smart strength training plans", https://www.fitbod.me/, accessed August 2025.
[4] JEFIT, "JEFIT: Workout tracker and planner", https://www.jefit.com/, accessed August 2025.
[5] MyFitnessPal, "Calorie counter and diet tracker", https://www.myfitnesspal.com/, accessed August 2025.
[6] Stripe, "Stripe Payments – Testing cards", https://stripe.com/docs/testing, accessed August 2025.
[7]Johnen, G., Kley-Holsteg, J., Niemann, A., & Ziel, F. (2024). Optimising Water Supply–Application of Probabilistic Deep Neural Networks to Forecast Water Demand in the Short Term. *AI in Business and Economics*, 13. https://books.google.com/books?hl=en&lr=&id=CQ0ZEQAAQBAJ&oi=fnd&pg=PA1975&dq=rules-based+linking+of+trainer+focus+tags+to+product+metadata+provides+concise+and+understandable+explanations+as+to+why+this+item+as+the+use+of+opaque+scoring+was+avoided+during+the+MVP+stage.&ots=f8L3Mr0MJl&sig=t-TCWhd3zxpqVF_fu_g9OQ2J-pM
[8] Bansal, D., Sinha, D., Sinha, D., & Khandelwal, S. K. (2025). Design and Implementation of a Multiuser Enterprise Resource Planning Solution for Higher Education Institutions: Enhancing Accreditation and Administration. https://www.authorea.com/doi/full/10.22541/au.175443125.52110063