Assignee Part I

David Wu

Contents

1. Overview

1.1. Introduction

This report details Assignee, a full-stack web application developed for the ICT SBA task of implementing an assignment management system.

Items marked with † denote features unimplemented in initial phases, primarily due to lower priority. Accompanying dagger symbols in subsequent paragraphs provide relevant details where applicable.

Within this chapter, we:

- Outline user capabilities of Assignee,
- Present core design systems guiding later sections,
- Discuss auxiliary systems extending the core functionality,
- Reference materials and validation resources conclude the chapter.

1.2. Capability

This section describes user roles in Assignee and their core workflows. It provides a high-level overview, omitting secondary capabilities (e.g. accessibility features) which are covered in later chapters.

- Role inheritance: $\in$
- Role transitions: $\rightarrow$
- Multi transition: $\overset{*}{\rightarrow}$

Where:

- Role inheritance indicates all parent capabilities are available to child
- Role transitions occur when a user performs a certain trigger action

1.2.1. User System

Roles: (Visitor, User)

Visitor

- Sign in — Visitor $\rightarrow$ User
- Sign up — Visitor $\rightarrow$ User

User

- Logout — User $\rightarrow$ Visitor
- Revoke — User $\rightarrow$ Visitor
- Modify email
- Modify password
- Modify display name
- Modify other settings †

The user system forms Assignee's foundation for account management. Key design principles:

1. Email-as-identifier:

- Uses email addresses as primary identifiers
- Eliminates need for unique usernames during signup
- Supports institutional emails (school/corporate domains)

2. Display name flexibility:
   - Users may customize display names post-registration

† Additional settings are excluded from initial versions because Assignee employs opinionated defaults optimized for core functionality.

1.2.2. Team System

Roles: (User, Member, Owner)

User

- Create team — User $\rightarrow$ Member
- Accept invite — User $\rightarrow$ Member

Member $\in$ User

- Leave team — Member $\rightarrow$ User

Owner $\in$ Member

- Disband — Member $\overset{*}{\rightarrow}$ User
- Invite members — User $\overset{*}{\rightarrow}$ Member
- Appoint owners — Member $\overset{*}{\rightarrow}$ Owner
- Dismiss owners — Owner $\overset{*}{\rightarrow}$ Member
- Modify team title
- Modify team description

The team system backs Assignee's flexible group mechanism. Key design principles:

1. Global invitation:
   - Eliminates need for complex authorization
   - Security imposed by rotating unique codes
2. Multiple owners' schema:
   - Avoids appointment complexity
   - Enables hassle-free role management
3. Team usage flexibility:
   - Promotes creation of scoped small teams
   - Not strictly limited to school assignments

1.2.3. Task System

Roles: (Owner, (Member,) Assignee)

Owner

- Create tasks — Member $\overset{*}{\rightarrow}$ Assignee
- Revoke tasks — Assignee $\overset{*}{\rightarrow}$ Member
- Modify instructions

- Attach reference file
- Review submitted file
- Modify feedback comments

Assignee ∈ Member

- Attach work file
- Return submission
- Revoke submission

The task system backs Assignee's powerful assignment features. Key design principles:

1. Atomic task assigning:
   - Promotes small scoped tasks
   - Enables simpler submission review and tracking
2. Task goal flexibility:
   - Allows arbitrary attachment types
   - Suitable for reference and works

1.3. Auxiliary

Apart from the above-mentioned three core systems, the flexibility of design of Assignee allows extending to extra systems.

For instance:

- Notification channels with long-polling
- Instant messaging via WS or server-sent events (SSE)

† Side systems remain intentionally undeveloped in initial phases, conserving resources while maintaining straightforward implementation paths via existing core architecture.

1.4. Resources

The following resources are provided for SBA invigilators' reference and validation purposes.

1.4.1. Repository

The complete project is hosted in a repository, accessible at

Repository° (https://github.com/wavim/assignee)

for inspection.

A modular approach ensures clear separation of concerns:

- `report/` : Contains the Typst source for this report
- `site/` : Houses the web application, organized into:
  - `server/` : Application layer
  - `schema/` : Communication layer
  - `client/` : Presentation layer

1.4.2. Prebuilt Binaries

To accommodate environments without development dependencies, prebuilt archives for all mainstream operating systems (primarily for x64-86 architecture) are available in

Assignee Releases° (https://github.com/wavim/assignee/releases)

for invigilators.

To execute the application:

1. Extract the archive for your operating system (`bin-{os}-{arch}.7z`) to a preferred location
2. Run the prebuilt binary `app.{os-ext?}` and follow prompts

Database records persist in the `app.db` file.

2. Data Layer

Backing the application is a relational database storing user data. This chapter covers the database design rationale first, followed by implementation details.

Within this chapter, we:

- Detail the design of tables, fields, and data types,
- Rationalize relational mappings between tables,
- Explain the partial adoption of normal forms,
- Conclude with actual implementation details.

Unless specified otherwise, all table fields are `NOT NULL` to prevent inconsistency, reduce anomalies, and simplify backend logic.

Since SQLite is adopted for the actual implementation, which employs dynamic typing, field types are described using generics. Specific data constraints are implemented in the communication layer instead. Besides, note that `INTEGER PRIMARY KEY` in SQLite implies Auto-Increment.

2.1. User System

Tables:

- `User` user information
- `Pass` user password
- `Sess` user sessions
- `Code` user 2FA codes †
- `Pref` user preferences †

`User` (uid, mail, name, created, updated)

`Pass` (**uid**, hash, salt, created, updated)

`Sess` (sid, **uid**, hash, salt, created, updated)

`Code` † (**uid**, hash, salt, created, updated)

`Pref` † (uid, **uid**, hash, salt, created, updated)

Relations:

`User` = `Pass` `1-1`

- User must have one password
- Password belongs to one user

`User` = `Sess` `1-N`

- User may have many sessions
- Session belongs to one user

`User` = `Code` † `1-1`

- User may have one code
- Code belongs to one user

`User` = `Pref` † `1-1`

- User must have one preference's set
- Preference's set belongs to one user

2.1.1. User

`uid` `INTEGER PRIMARY KEY`

Primary key chosen over candidate key (mail) for:

- Indexing speed: Magnitudes faster
- Efficiency: Smaller than text references
- Consistency: Guaranteed uniform values
- Flexibility: Immune to authentication changes

Recurring primary key pattern, omitted elsewhere.

`mail` `TEXT UNIQUE`

Unique authentication identifier.

`name` `TEXT`

User-defined display name.

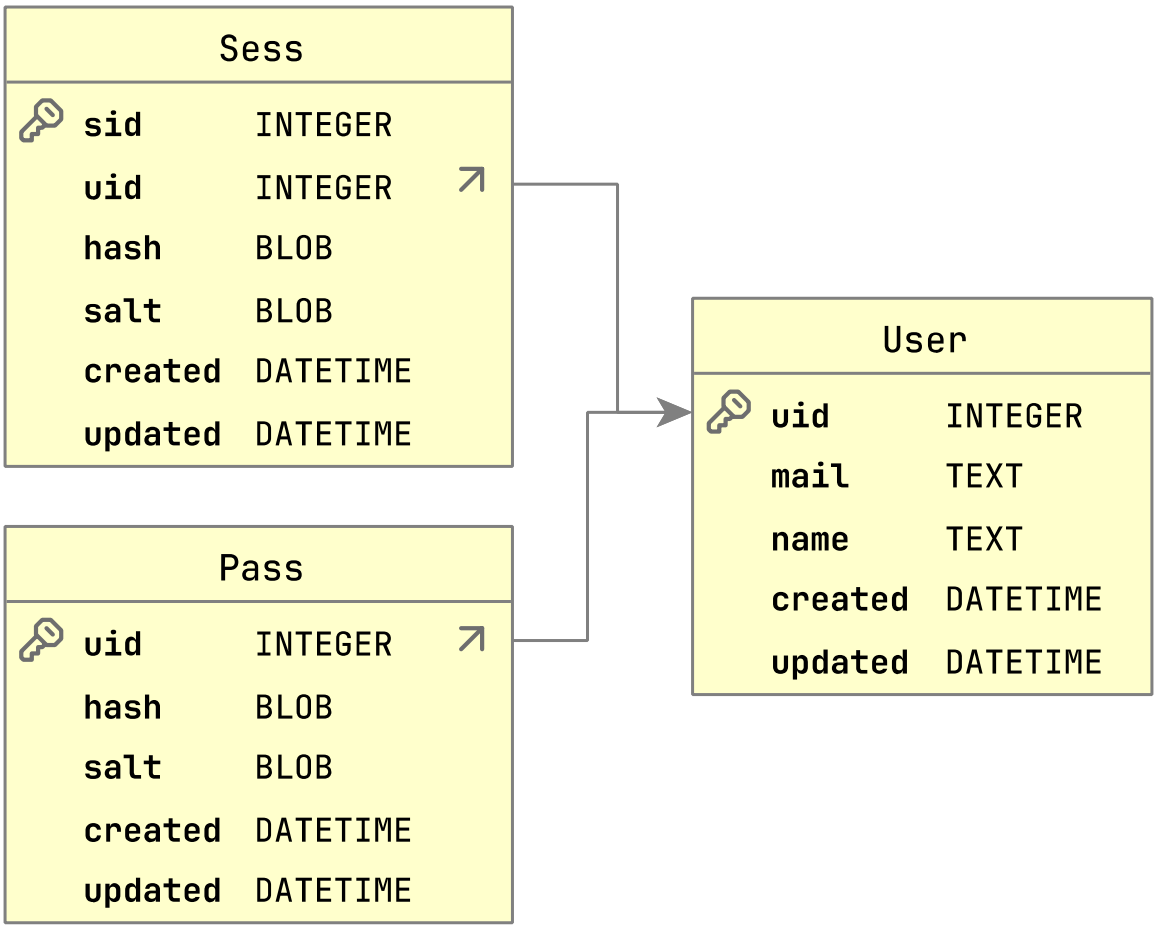

Figure 1: User System ERD

`created/updated` `DATETIME`

Entry creation/modification timestamps.

Recurring table metadata, omitted elsewhere.

2.1.2. Pass

`uid` (=User.uid) `INTEGER PRIMARY KEY`

Primary key and foreign key. (1:1 user mapping)

`hash/salt` `BLOB`

Secured credentials storage:

- Hashed via K12 (SHA3 Keccak-p variant, 256-bit digest, parallelism optimal)
- Salted (128-bit CSPRNG)

1. Append CSPRNG salt to key
2. Hash with K12, store digest + salt
3. Verification: Repeat with stored salt

Collision probability $\approx 1.5 \cdot 10^{-31}$ (negligible).

It would not be possible to reconstruct password from hash/salt, thus password reset endpoints must be present.

Recurring authentication pattern; omitted elsewhere.

2.1.3. Sess

`sid` `INTEGER PRIMARY KEY`

Primary key.

`uid` (=User.uid) `INTEGER`

Foreign key. (N:1 user mapping)

`hash/salt` `BLOB`

1. User authenticated through signin or signup
2. Token generated: sid reversible-hashed with pepper, CSPRNG 256-bit hex key
3. Bearer token sent to client as cookie with secure configurations

Authentication flow:

- Search cookie for session bearer token
- Compute sid from hashed ID, loop up session
- If session age passed expiration limit (i.e. 1 day):
  - Invalid session, respond with error
- Else:
  - If session age passed rotation limit (i.e. 1 hour):
    - Rotate token and return the new token
  - Else:

- Return original token and authenticate

Cron jobs are run on the server side to periodically remove expired tokens.

Session validity is checked on all API routes to protect Assignee from unauthenticated access.

2.1.4. Code

† Email 2FA omitted to prevent private API key leakage.

`uid` (=User.uid) `INTEGER PRIMARY KEY`

Primary key and foreign key. (1:1 user mapping)

`hash/salt` `BLOB`

Secured user 2FA code storage.

2.1.5. Pref

† Not implemented since Assignee is highly opinionated, adds implementation complexity.

`uid` (=User.uid) `INTEGER PRIMARY KEY`

Primary key and foreign key. (1:1 user mapping)

`data` `JSON`

Partial settings storage:

- Merged with global defaults
- Only stores user-modified values

Benefits:

1. Defaults flexibility
2. Space efficiency

JSON violates 1NF but enables nested organization (scholars have argued that this may not be a violation since 1NF allows any self-contained entity).

2.2. Team System

Tables:

- `User` user information
- `Team` team information
- `Invite` team invitation
- `Member` team membership

`Team` (tid, name, desc, created, updated)

`Invite` (**tid**, code, created, updated)

`Member` (**uid, tid**, auth, created, updated)

Relations:

`Team` = `Invite` `1-1`

- Team may have one invitation
- Invite code belongs to one team

`User` = `Member` `1-N`

- User may have many memberships
- Membership belongs to one user

`Team` = `Member` `1-N`

- Team may have many memberships
- Membership belongs to one team

2.2.1. Team

`tid` `INTEGER PRIMARY KEY`

Primary key.

`name` `TEXT`

Owner-defined display name.

`desc` `TEXT`

Owner-defined team description.

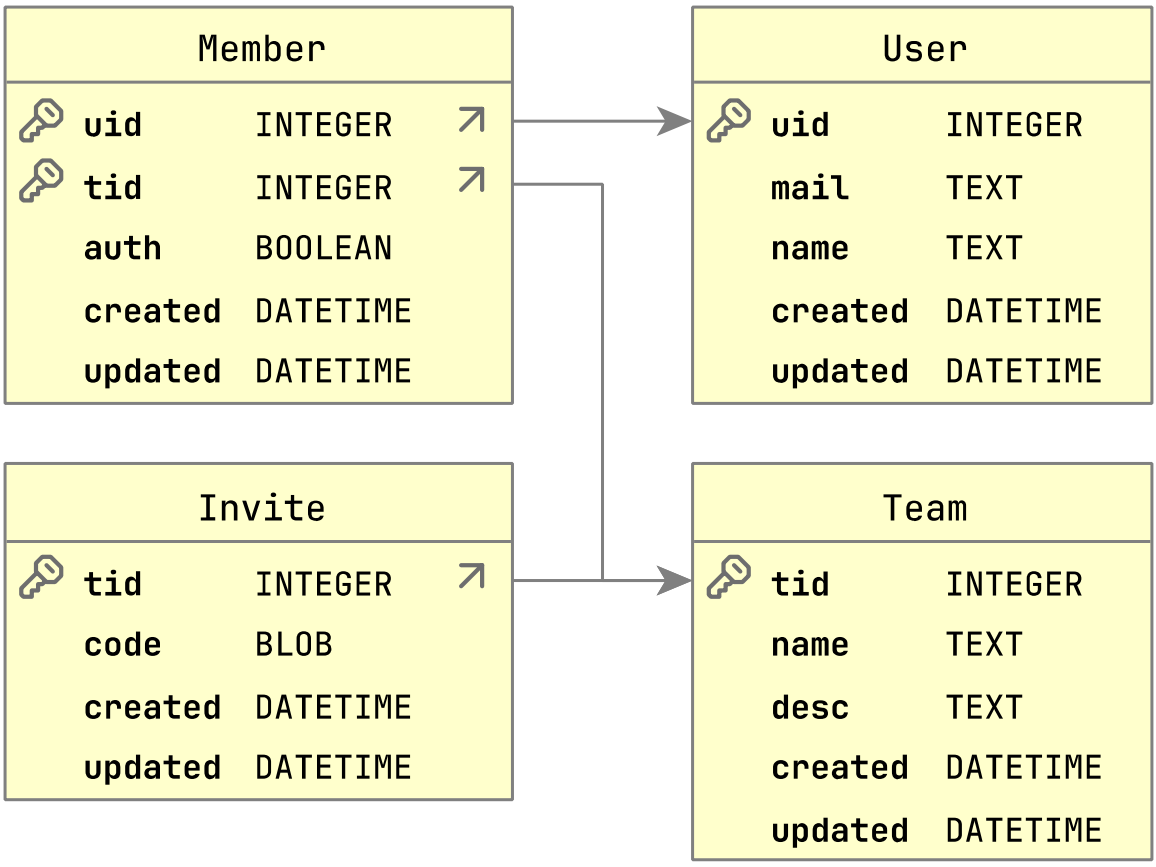

Figure 2: Team System ERD

2.2.2. Invite

`tid` (=Team.tid) `INTEGER PRIMARY KEY`

Primary key and foreign key. (1:1 team mapping)

`code` `BLOB`

Globally unique team invitation code stored as raw binary.

Benefits:

- More straightforward i.e. CSPRNG returns buffer
- Faster than TEXT for query and uniqueness checks

The bytes would be converted to hex as code, this is particularly helpful since it is case-insensitive and hard to confuse.

Cron jobs are run on the server side to periodically remove expired tokens.

Invitation codes are created lazily, while being validated and rotated with a similar mechanism to session tokens.

2.2.3. Member

`uid` (=User.uid) `INTEGER COMPOSITE KEY`

Composite key and foreign key. (N:1 user mapping)

`tid` (=Team.tid) `INTEGER COMPOSITE KEY`

Composite key and foreign key. (N:1 team mapping)

`auth` `BOOLEAN`

If checked, the member would be considered an owner of the team.

Rationale:

- Flexible role management framework
- Eliminates the need for embedding team owner

This flag, and the entry in general, is used in endpoints to ensure authorized access.

2.3. Task System

Tables:

- `User` user information
- `Team` team information
- `Task` task information
- `Work` work information
- `TaskFile` task attachment file
- `WorkFile` work attachment file

`Task` (aid, **tid**, name, desc, dead, created, updated)

`Work` (sid, **uid**, **aid**, done, comm, created, updated)

`TaskFile` (**aid**, name, mime, blob, created, updated)

`WorkFile` (**sid**, name, mime, blob, created, updated)

Relations:

`Team` = `Task` `1-N`

- Team may have many tasks
- Task belongs to one team

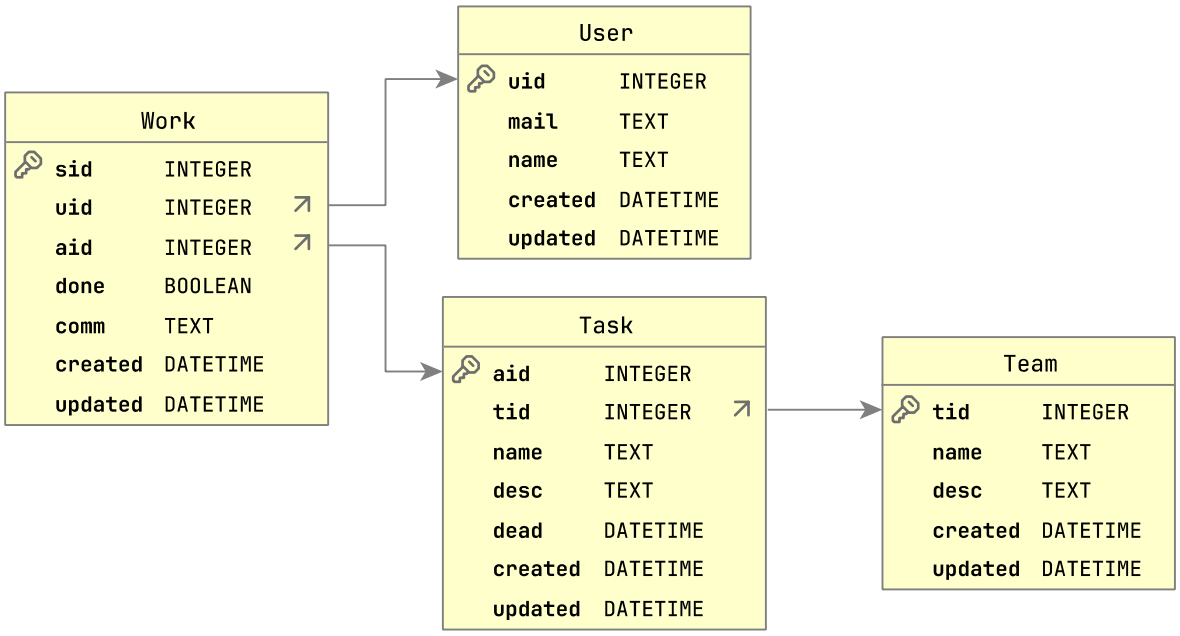

Figure 3: Task System ERD (1)

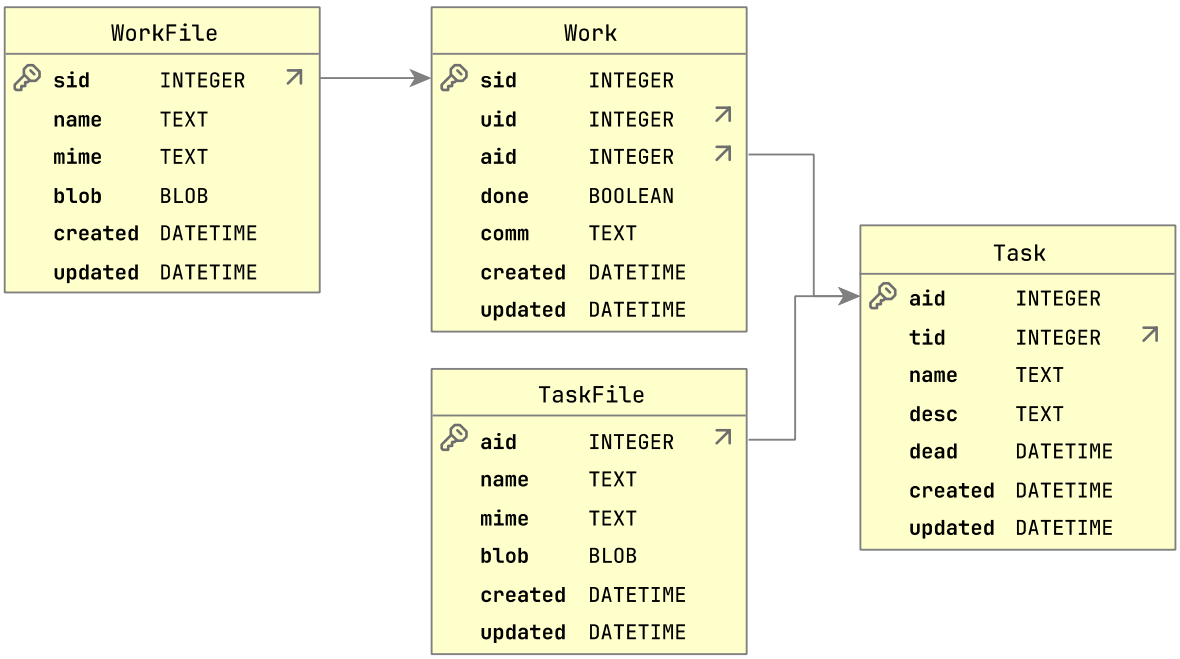

Figure 4: Task System ERD (2)

`Task` = `Work` `1-N`

- Task may have many works
- Work belongs to one task

`User` = `Work` `1-N`

- User may have many works
- Work belongs to one user

`Task` = `TaskFile` `1-1`

- Task may have one attachment
- Attached file belongs to one task

`Work` = `WorkFile` `1-1`

- Work may have one attachment
- Attached file belongs to one work

2.3.1. Task

`aid` `INTEGER PRIMARY KEY`

Primary key. (‘a’ for assignment)

`tid` (=Team.tid) `INTEGER`

Foreign key. (N:1 team mapping)

`name` `TEXT`

Assigner-defined task name.

`desc` `TEXT`

Assigner-defined task instructions.

`dead` `DATETIME`

Assigner-defined task deadline.

2.3.2. Work

`sid` `INTEGER PRIMARY KEY`

Primary key. (‘s’ for submission)

`uid` (=User.uid) `INTEGER`

Foreign key. (N:1 user mapping)

`aid` (=Task.aid) `INTEGER`

Foreign key. (N:1 task mapping)

`done` `BOOLEAN`

Flag indicating submission status. Work entries are lazy created upon certain user actions (e.g. viewing task), rather than being created all at once on task creation. This is more performant and flexible for larger teams.

`comm` `TEXT NULLABLE`

Optional feedback comment set by assigner after returning.

2.3.3. TaskFile

`aid` (=Task.aid) `INTEGER PRIMARY KEY`

Primary key and foreign key. (1:1 task mapping)

`name` `TEXT`

Attachment file name.

`mime` `TEXT`

Attachment file Multi-purpose Internet Mail Extensions type.

`blob` `BLOB`

Attachment file raw binary data.

2.3.4. WorkFile

`sid` (=Work.sid) `INTEGER PRIMARY KEY`

Primary key and foreign key. (1:1 work mapping)

`name` `TEXT`

Attachment file name.

`mime` `TEXT`

Attachment file Multi-purpose Internet Mail Extensions type.

`blob` `BLOB`

Attachment file raw binary data.

2.4. Normalization

Tables satisfy 3NF/4NF (extended normal forms) in general, with practical compromises:

- JSON values (e.g. Pref): Space efficiency > strict 1NF, debatable violation of scalar values
- Separate tables (e.g. Pass): Security metadata tracking and flexibility concerns

All exceptions could be justified by performance/maintainability.

Otherwise, all tables satisfy:

- Unique rows
- Scalar values
- No partial dependencies
- No transitive dependencies

2.5. Implementation

The database is implemented in SQLite for:

- Simplicity of setup allowing quick iterations
- Dynamic data typing system easing configuration
- Portability backs prebuilt binaries for reviewers' inspection

2.5.1. Table Index

SQLite automatically indexes primary keys and unique fields. The following is manually indexed:

- Foreign keys: For efficient 1-N relational queries
- Composite's subset fields: [B] for composite [A, B], A via prefix-index

Modern DBMS utilizes B-Trees, instead of relying on binary search. B-Trees are self-balancing and the time complexities for search, insert, and delete are all $O(\log(n))$. No re-indexing is required in most cases.

2.5.2. Data Integrity

All foreign keys in Assignee are set to `ONUPDATE: RESTRICT, ONDELETE: CASCADE` to maintain referential integrity:

- Updating referenced parent fields is prohibited
- Deleting a parent entry would remove all related child entries

2.5.3. Miscellaneous

Advanced features like views and triggers are deliberately not implemented. The data layer of Assignee is meant to be simple yet efficient, thus complex logic is transferred to the application layer instead.

3. Application Layer

Backing the application logic is an JavaScript Express.js server leveraging Prisma ORM for type-safe database interactions. This chapter outlines the architectural design principles first, followed by concrete implementation patterns.

Within this chapter, we:

- Detail Express.js services design, route organization, and middleware stacks,
- Rationalize Prisma's integration for seamless data access and type safety,
- Address application performance and security considerations,
- Conclude with implementation specifics for portability.

3.1. Server Framework

The application is implemented in Express.js (Node.js library) instead of PHP servers for:

- Control: Express middleware
- Async I/O: Non-blocking requests
- Full-stack: Shared TypeScript interface
- Ecosystem: Rich tooling (ESLint, Prisma)

Being a superset of JavaScript, TypeScript provides extraordinary static typing. In contrast, PHP is getting obsolete and has been lacking ecosystem support for several years. JavaScript backed by Google's V8 engine is simply a more performant and developer-friendly choice.

3.2. Resource Serving

Assignee uses a client side router, thus only minimal static asset is served. This includes:

- HTML skeleton
- Frontend JavaScript
- External CSS stylesheets
- Other assets e.g. fonts

Fonts used by the frontend are self-hosted to reduce reliance on Google Fonts for higher robustness, and potentially improving performance.

Appropriate HTTP Cache-Control headers are set to ensure proper static resource caching, e.g. flagging assets as immutable.

3.3. DB Integration

Prisma ORM (JavaScript object-relational mapping library) is used for interacting with the SQLite database. Prisma accepts a schema defined in a custom language and generates database CRUD interaction functions for client usage.

Benefits:

- Straightforward relational queries
- Sanitization to prevent SQL injection
- Strict typing interfaces for validation

An example schema definition for Prisma:

```
model Member {
  uid Int
  tid Int

  auth Boolean

  created DateTime @default(now())
  updated DateTime @updatedAt

  User User @relation(fields: [uid], references: [uid], onUpdate: Restrict, onDelete: Cascade)
  Team Team @relation(fields: [tid], references: [tid], onUpdate: Restrict, onDelete: Cascade)

  @@id([uid, tid], name: "pk")
  @@index([tid])
}
```

Notice that Prisma also helps with enforcing relational integrity via `@relation` definitions.

After defining the schema, relational queries could be performed in TypeScript with ease:

```
await prisma.task.findMany({
  select: {
    aid: true,
    name: true,
    dead: true,
    Team: { select: { name: true } },
    Work: { select: { done: true }, where: { uid } },
  },
  where: { Team: { Member: { some: { uid, auth: false } } } },
});
```

3.4. Middleware

Assignee uses a middleware stack to ensure proper authentication and authorization for endpoints, and enforces correct usage of response headers.

All responses of Express (no matter static or API) are compressed with Brotli for bandwidth and response speed.

3.4.1. Authentication

Validates against the cookie from request to see if the user bears a valid session cookie.

3.4.2. Membership

Validates against the team ID to see if the authenticated user bears a team membership.

3.4.3. Assignment

Validates against the assignment ID to see if the authenticated user is involved within.

3.4.4. Cache Control

Sets the HTTP Cache-Control header to no-cache for API endpoints, forcing validation.

3.5. Route Services

Services implement different functions that interact with the database, and are called by API endpoints. For example:

- Creating a user with the provided data
- Rotating a session and returning token
- Updating a work file with the new payload
- Logging out from a session

The actual authentication logic is also implemented here with cryptography utilities, including K12 hashing and CSPRNGs.

An example extract of authentication logic:

```
async function createSession(uid: number): Promise<SessionCookie> {
  const key = randk();
  const { sid } = await prisma.sess.create({ select: { sid: true }, data:
{ uid, ...chash(key) } });

  return { sid: configs.hashSID.encode(sid), key };
}

export async function signin(req: SigninRequest): Promise<SessionCookie> {
  const user = await prisma.user.findUnique({
    select: { uid: true, Pass: { select: { hash: true, salt: true } } },
    where: { mail: req.mail },
  });

  if (!user) {
    throw new HttpError("UNAUTHORIZED", "Invalid Email or Password");
  }

  if (!user.Pass) {
    throw new HttpError("INTERNAL_SERVER_ERROR", "No Password Data");
  }

  if (!match(req.pass, user.Pass.hash, user.Pass.salt)) {
    throw new HttpError("UNAUTHORIZED", "Invalid Email or Password");
  }

  return await createSession(user.uid);
}
```

3.6. Route Endpoints

Routers are the primary way we define API endpoints in Express.js for RPC/REST requests. After authentication, authorization, and validating the payload, corresponding services are called to perform the requested action.

Most endpoints are GET or POST requests, but some use PUT. Both GET and PUT are assumed to be idempotent (the same request yields the same results) and thus enables better caching. Actions that could not be safely cached usually goes with the POST method, e.g. authentication. Additionally, endpoints with payloads that could be too large for the URL query string must avoid using GET.

It is worth noting that although the CCache middleware is used to set Cache-Control to no-cache, it doesn't really mean to force no caching (which is the case for no-store). Instead, data validity must be checked before proceeding with the cached asset. This is typically not required for static assets, but inherently important for API endpoints.

Having a global configuration file, there are rate limiters on certain routes such as signin and signup to prevent abuse and enumeration attacks e.g. DoS. The rate limiters are set to use key generators suitable for the case, i.e. email address for signin/signup, and uses client IP otherwise.

3.7. Quality Assurance

Google Issues is used extensively to enforce HTTP response best practices, including response headers and caching directives. Repeated tests on different scenarios are done to ensure good baseline response time.

3.8. Server Deployment

For invigilators' reference, the Node.js application is bundled and compiled into prebuilt binaries for all mainstream operating systems, by packing in Node.js internals into a single executable.

The server would try to host on 0.0.0.0, which is the reserved wildcard address in IPv4. It would then resolve to the client's public broadcast IPv4 address (typically 192.168.x.x), and start Assignee on port 5450 (a number I love personally, avoids port conflicts). All computers in the same LAN would be able to access the web application.

By correctly configuring path resolution, static files are retrieved inside a virtual file system at runtime. The `app.db` file is resolved relatively, this allows database records to be preserved.

4. Communication Layer

This chapter is on the validation of requests and response between the server and clients. The layer is the primary way of validating complex data, instead of relying on rigid database constructs.

By using TypeScript end-to-end, a full-stack application allows global TS schema validators. The library is Zod, which adds support for sophisticated runtime data validation.

The rules are used to enforce business rules and maintain data consistency, blocking further actions if failing to parse payloads. They include:

- Checking for valid email address format
- Checking for password security strength
- Checking for dates earlier than expected

And more. These even include validating complex data types such as regular expression strings, arrays, objects, and discriminated object unions, which is impossible to validate solely with database constraints.

Validation schemas are structured and named using API endpoints for ease of management, e.g. `PostTaskRequest, PostTaskResults`, validating data both coming to and from the server, by sharing the Zod schema between the server and client. This prevents bypassing JavaScript while maintaining immediate response otherwise.

Zod schemas fill up the inadequacy of SQLite dynamic data types, and allows even more specific constraints. This ensures development goes smoothly, and different layers agree on the same interface, catching errors early in development.

4.1. Example Validators

Some example validators defined for authentication:

```
// POST /auth/verify
// POST /auth/logout

export const SessionCookie = z.object({
  sid: z.string().check(z.regex(/^[0-9a-zA-Z]{8,}$/)),
  key: z.string().check(z.regex(/^[0-9a-f]{64}$/)),
});
export type SessionCookie = z.infer<typeof SessionCookie>;

// POST /auth/signin

export const SigninRequest = z.object({
  mail: z.string().check(z.trim(), z.email(), z.toLowerCase()),
  pass: z.string().check(z.regex(/^[\x20-\x7E]{8,}$/)),
});
export type SigninRequest = z.infer<typeof SigninRequest>;
```

Notice the use of `z.trim()` and `z.toLowerCase()`, Zod allows schemas to preprocess and post-process payloads to meet specific requirements without throwing an error. In the case, email addresses that start or end with spaces could be refined as needed, but not throwing an error and make users frustrated.

TypeScript types are inferred from Zod schemas via `z.infer<typeof ...>`, and exported for use of both the backend and frontend.

Instances using discriminated unions (simplified for clarity):

```
// GET /teams/:tid/tasks

export const GetTeamTasksResults = z.discriminatedUnion("auth", [
  z.object({
    auth: z.literal(true),
    data: z.array(
      z.object({
        // ...
        done: z.number().check(z.nonnegative()),
      }),
    ),
  }),
  z.object({
    auth: z.literal(false),
    data: z.array(
      z.object({
        // ...
        done: z.boolean(),
      }),
    ),
  }),
]);
export type GetTeamTasksResults = z.infer<typeof GetTeamTasksResults>;
```

Greet the usage of request parameters i.e. `:tid`! By deciding the user's authenticated status in a group, Assignee would return different payloads. In this case, team owners would receive completion status as figures, while members would only have access to their own completion status as a boolean. This greatly increases the flexibility, security, and performance of the application, as only a selection of all data would be needed in different cases.

5. Presentation Layer

The essence of an application is the interface that presents data to users. This chapter covers the frontend website design first, followed by implementation details.

Within this chapter, we:

- Detail the philosophy of theme, palette, and font,
- Demonstrate the landing pages and the application,
- Detail the accessibility options of the application,
- Detail user experience and mobile-friendliness,
- Conclude with actual implementation details.

5.1. Design Language

Essence, Clarity, Calm

Assignee is a deliberate rebellion against digital noise. It champions radical simplicity, cognitive ease, and undistracted focus to transform task management into a serene, intentional ritual.

Inspired by the tactile honesty of paper and the precision of modernist typography, every element serves a purpose—nothing more, nothing less.

5.1.1. Palette

Assignee uses a neutral palette carefully crafted by expert designers. The theme colors bear OKLCH values with zero chroma and hue, only varying the lightness.

Figure 5: Neutral Palette

Employing a neutral palette within a minimalist website design is fundamentally driven by the core principle of intentional reduction. Colors like whites, grays, beiges, and blacks inherently possess low chromatic distraction, allowing the essential elements to command attention without visual competition.

This absence of strong color saturation minimizes cognitive load for the user, fostering a sense of calm, clarity, and sophistication. Furthermore, a neutral foundation enhances readability, promotes timelessness over fleeting trends, and creates a sense of spaciousness and order that aligns perfectly with minimalist goals of focusing purely on essential content and user experience.

5.1.2. Font Face

Assignee uses Plus Jakarta Sans as the sole typeface, directly reinforces the core values of elegance, timelessness, and superior legibility. Its clean, geometric structure embodies elegance through balanced letterforms with subtle, sophisticated details, avoiding sterility while maintaining refined simplicity.

Whereas disregard and contempt for human rights have resulted

Figure 6: Plus Jakarta Sans

This elegance complements a neutral palette, ensuring typography becomes a harmonious element of the aesthetic rather than a distraction. The font's timelessness stems from its blend of contemporary clarity and humanist proportions, ensuring relevance and sophistication for years, much like the neutral backdrop it sits upon.

Most critically, Plus Jakarta Sans excels in legibility: its generous x-height, clear letter differentiation, and well-considered spacing optimize readability across devices and sizes, reducing user strain and ensuring content remains effortlessly accessible.

5.1.3. Logo Design

Assignee uses a logo directly referencing its name.

Figure 7: Assignee Logo

The **ΔSSIGNEE** logo embodies minimalist principles through potent symbolism and intentional restraint. The delta (Δ) serves a dual function: its sharp, geometric form acts as a pen tip, instantly evoking the core action of assignment, and injects dynamism into the neutral "paper-like" backdrop, while also representing the letter "A", an elegant, efficient integration of brand identity.

It achieves essential harmony: the precision of the delta anchors the word mark, while the neutrality ensures both symbol and text remain crisp, uncluttered, and effortlessly comprehensible. Ultimately, the logo exemplifies minimalist power: using fundamental forms like geometric symbols and clear letterforms, to communicate core purpose with timeless elegance.

5.2. Demonstration

It should be noted that images might not be up-to-date, as the report is authored during development. However, it would not deviate significantly from the end result of Part I.

5.2.1. Hero Section

The landing page starts with the hero section. A compelling hero section serves as the immediate visual and functional gateway to the platform's purpose. This section bridges elegance and utility—leveraging minimalist principles to reduce cognitive load while using asymmetry to create movement, making the interface feel alive and intentional from the first glance.

The strategic use of Bauhaus L Fade tiles as section separator embodies a functional asymmetry that energizes the minimalist framework while honoring modernist principles of dynamic composition, creating deliberate visual tension against the neutral backdrop. Their asymmetry prevents rigid predictability without overwhelming users.

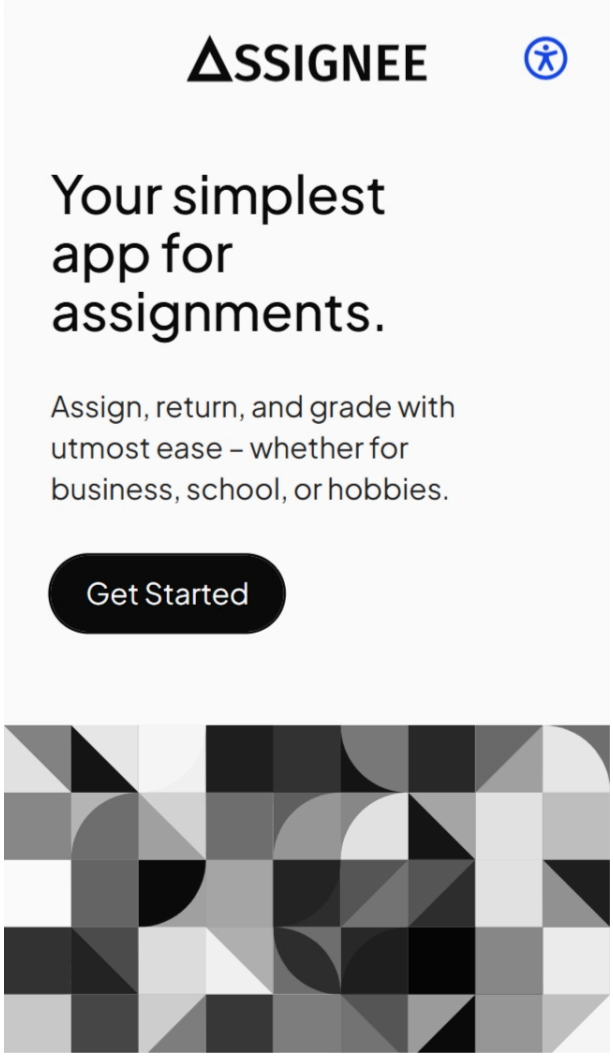

Figure 8: Hero Section

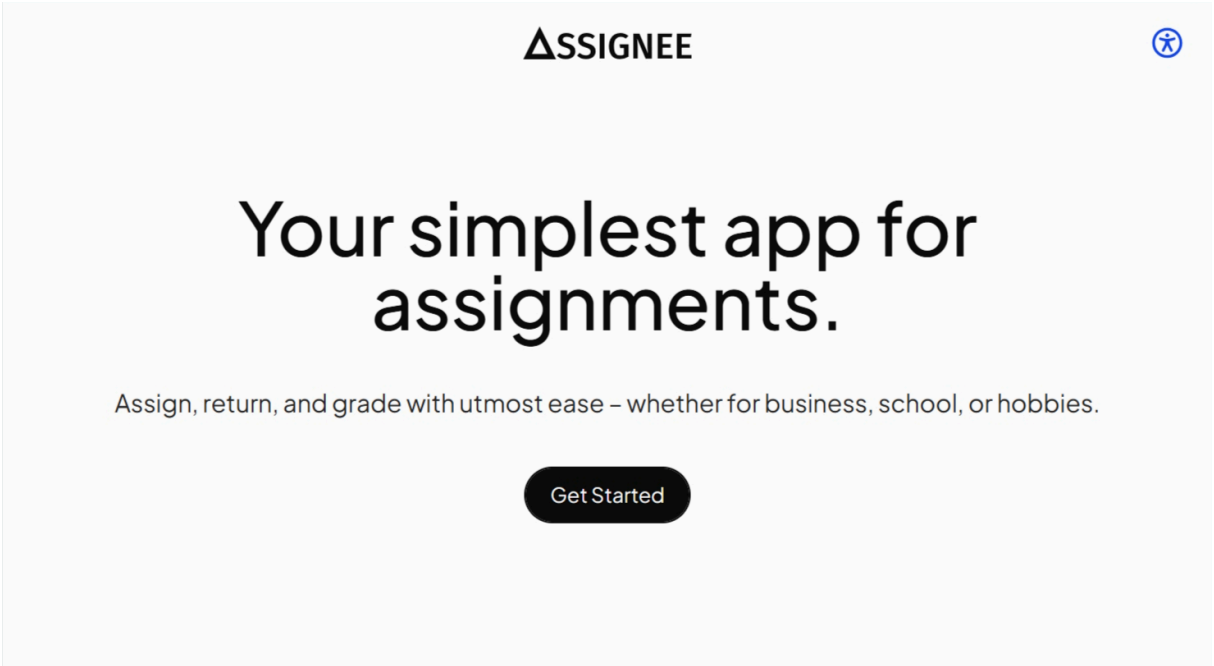

Figure 9: Hero Section
MD Variant

Uncompromising geometry of interlocking squares and triangles structure the interface: they anchor content zones, guide the eye toward the critical action, and partition whitespace with rhythmic tension. It also acts as a dynamic visual threshold, using sharp angles and staggered collisions to signal transition without disruption.

In larger screens, these tiles recede entirely, allowing the hero to expand edge-to-edge, signifying the presence of space and forces focus.

5.2.2. Home Header

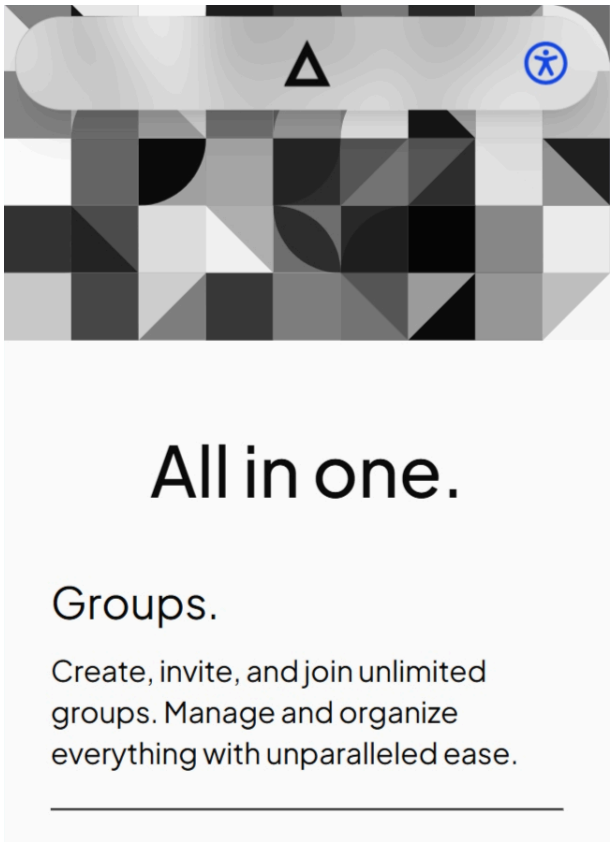

Figure 10: Header Scrolled

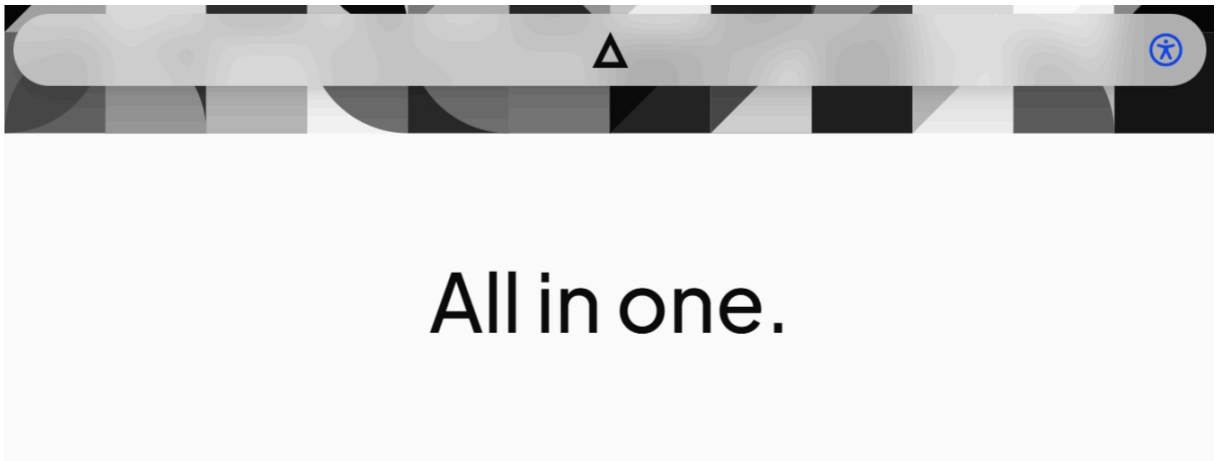

Figure 11: Header Scrolled
MD Variant

The header's transformative scroll behavior embodies minimalist utility through adaptive elegance. Initially transparent and embedded within the hero's natural flow, it avoids visual imposition. As scrolling commences, it transitions into a subtle overlay with soft elevation shadow, signaling a shift from introduction to action.

Although the logo navigates to the homepage, a "navigation bar" is not present per se. This is due to Assignee's minimal amount of landing pages.

5.2.3. Features Section

Following the hero, the features section leverages medium-screen asymmetry through a purposeful split: a persistent "Features" anchor sticks rigidly to the left viewport edge, while feature details scroll independently on the right.

This creates a content-rich rhythm: the anchored text acts as both a minimalist compass orienting users within the page, and a structural counterweight to the dynamic content flow. Meanwhile, the right side's scrollable column allows features to unfold with breathing room, ensuring no pixel is wasted.

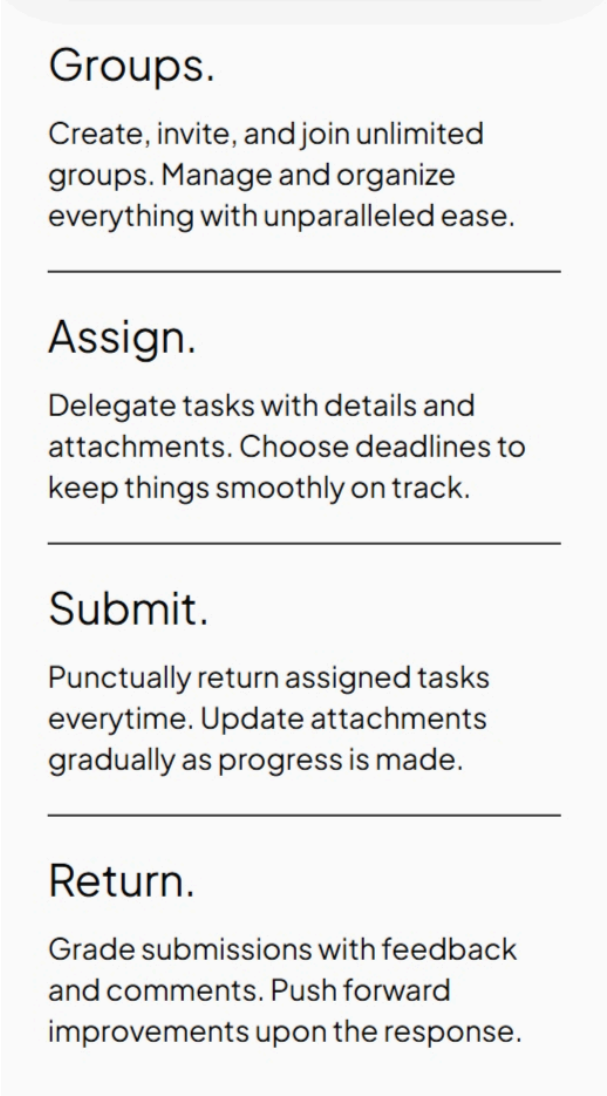

Figure 12: Features Section

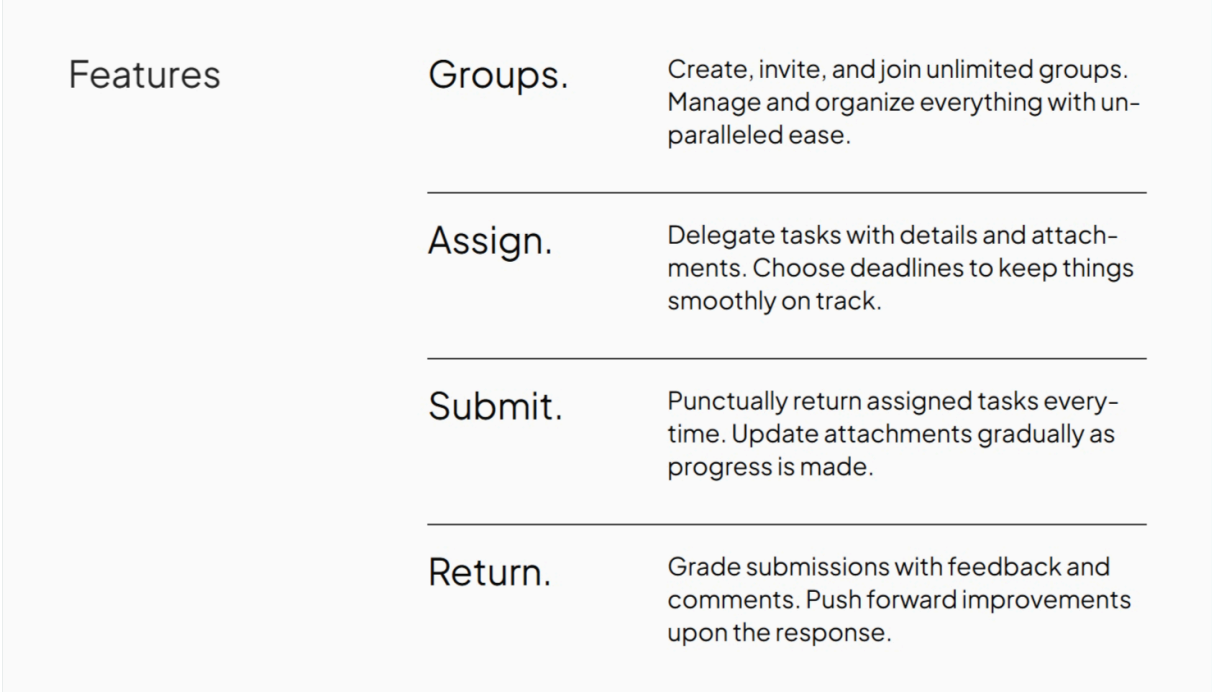

Figure 13: Features Section
MD Variant

5.2.4. Call To Action

The landing page ends with a Call-to-Action section, deliberately mirrors the hero's text positioning and spatial logic, creating rhythmic closure that bookends the user journey with purposeful familiarity.

Its mirrored structure triggering subconscious recognition while amplifying urgency. By echoing the hero's architecture, the CTA transforms functional symmetry into psychological momentum: a final, frictionless pivot from exploration to commitment.

Symmetry, Asymmetry. Rigid structural symmetry beneath the surface enables bold asymmetric expression in Assignee's visual language. Where symmetry whispers order, asymmetry shouts intention, together creating an experience that feels simultaneously anchored and alive. Every imbalance is calculated, every fracture purposeful, and every pixel a testament to minimalist discipline wielded with avant-garde audacity.

Did this beautiful philosophy not reign the grounds of Assignee?

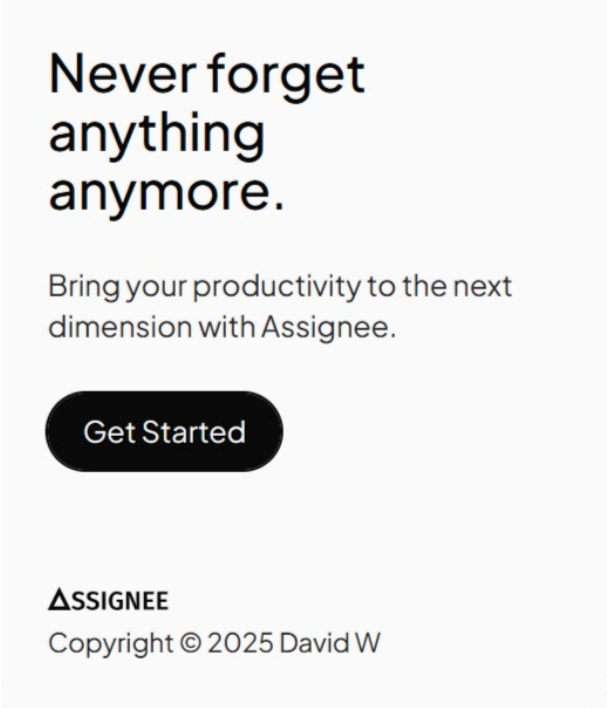

Figure 14: Features Section

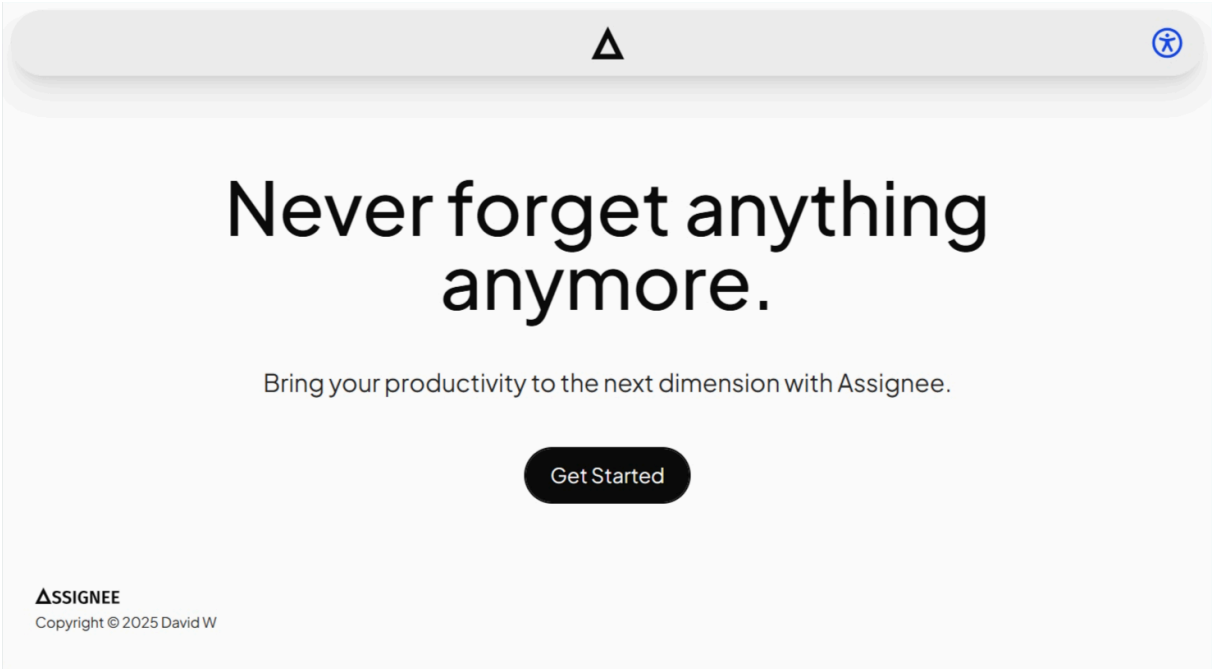

Figure 15: Features Section
MD Variant

5.2.5. Home Footer

The footer embodies a subtle logo paired with a clean copyright linking to my GitHub. No dividers, no social icons, no excess: just geometric purity and creator credit on a neutral canvas. What a final whisper of a system where every element serves purpose?

5.2.6. Authenticator

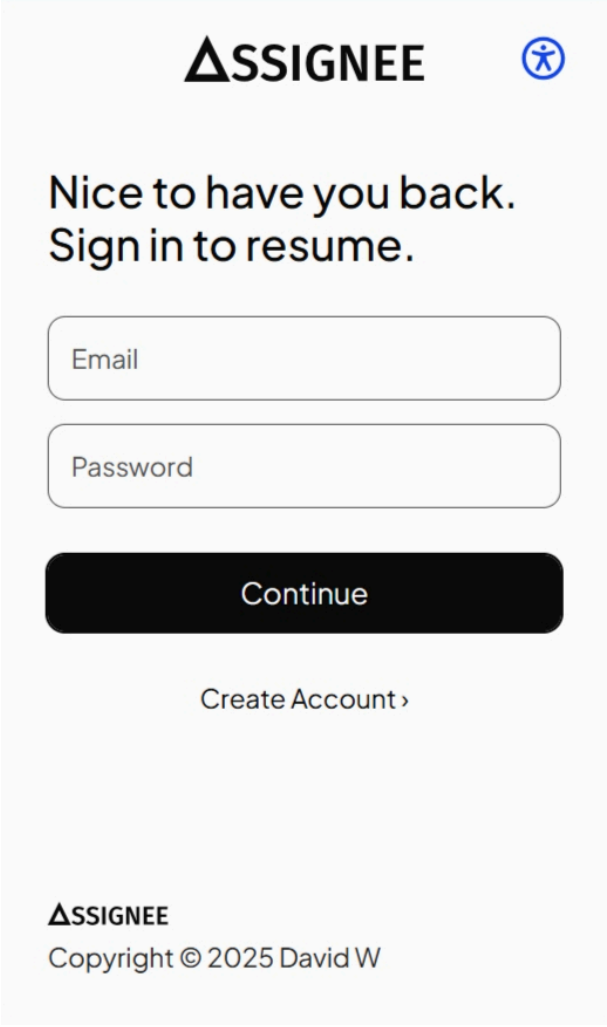

Figure 16: Dynamic Signer

Two fields, one button. No extras. Assignee strips authentication to its essence. Task clarity through disciplined design. The asymmetry is quiet, the palette calm, the interaction profound in its simplicity: a silent handshake between human and machine.

Placeholders shift to titles as one inputs, preserving context without cluttering the input zone. The asymmetry of moving labels against static fields creates kinetic hierarchy, providing the user generous interaction feedback.

The sole button transcends static labeling through adaptive text. Morphing dynamically to error messages, Assignee turns a button into a self-contained dialogue, eliminating pop-ups that disrupt flow: errors resolve within the element.

Only signin could be reached via the landing page, with signup accessible through it. Burying signup within the signin flow isn't a hidden feature, it's a strategic reduction of decision fatigue. One problem, one solution.

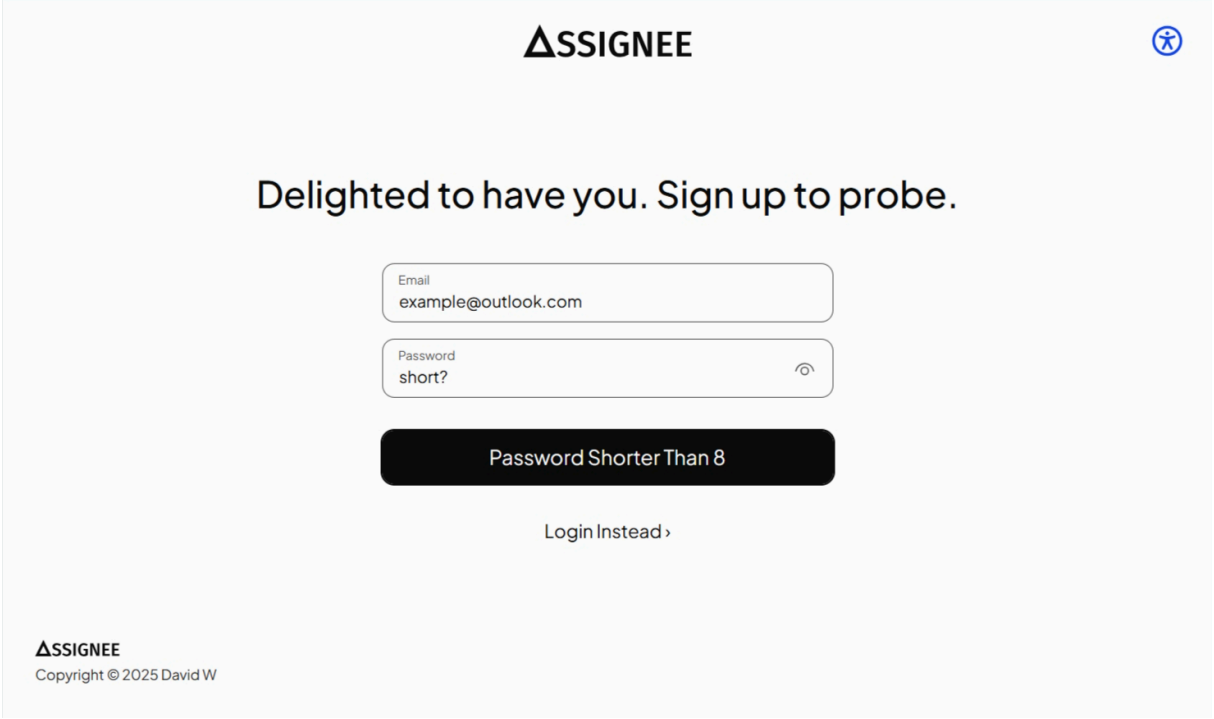

Figure 17: Dynamic Signer
MD Variant

In fact, all visible ASCII characters (and space `\x20`) are allowed characters in the password. This increases entropy and enhances security. This also aligns well with modern password managers.

5.2.7. Page Not Found

A generous 404 page is also implemented just in case users get lost (unlikely!)

Maybe, umm, try to navigate back to the landing page by clicking on that button...?

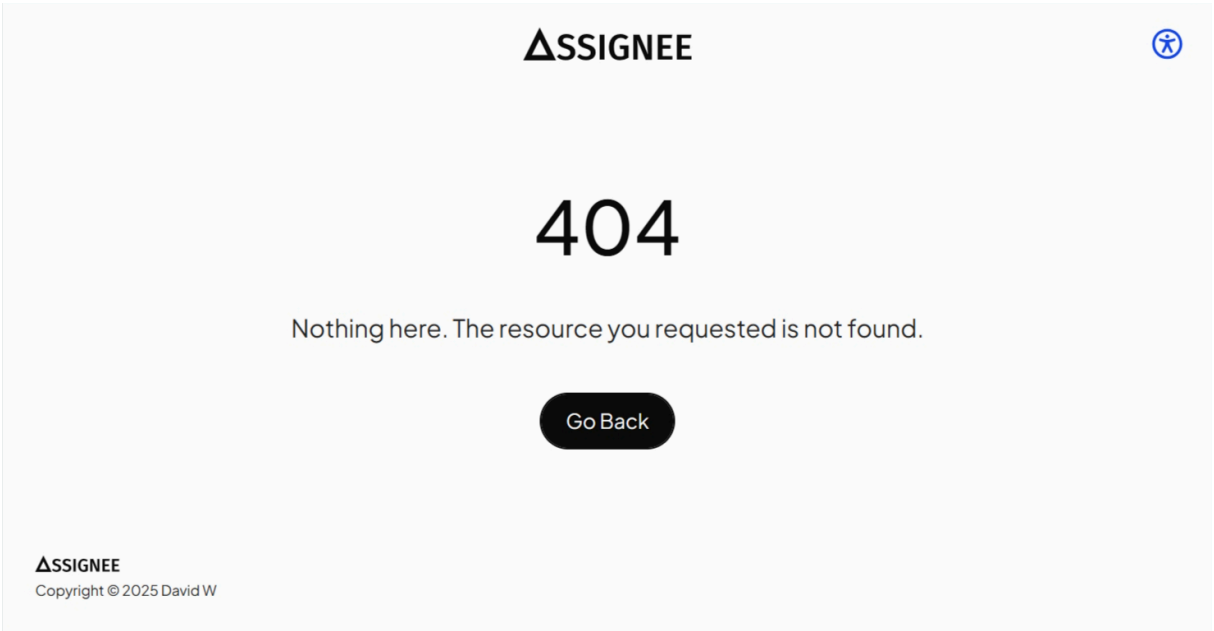

Figure 18: Not Found
MD Variant

5.2.8. The Application

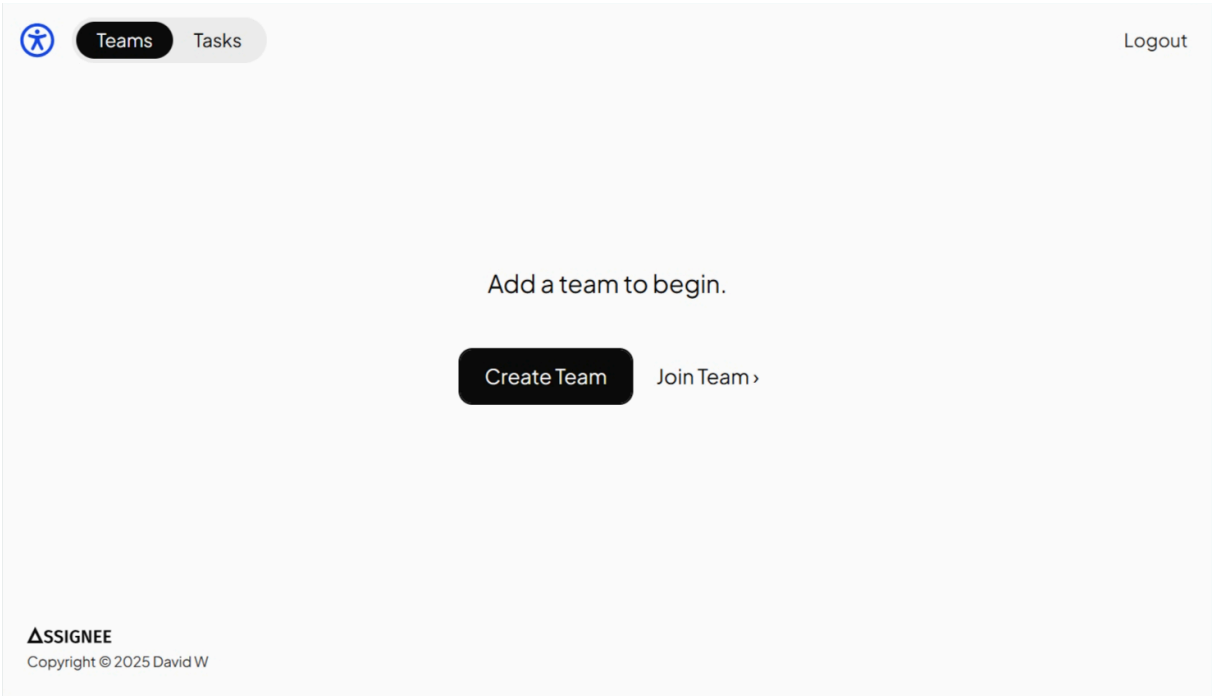

Figure 19: Dashboard, Teams
MD Variant

The dashboard elegantly weaponizes negative space, transforming emptiness into focused potential. Below the welcome prompt, the two action button set up a dynamic duo, pairing primary action and secondary action with asymmetry.

The emptiness is the metaphor: a pristine workspace awaiting action.

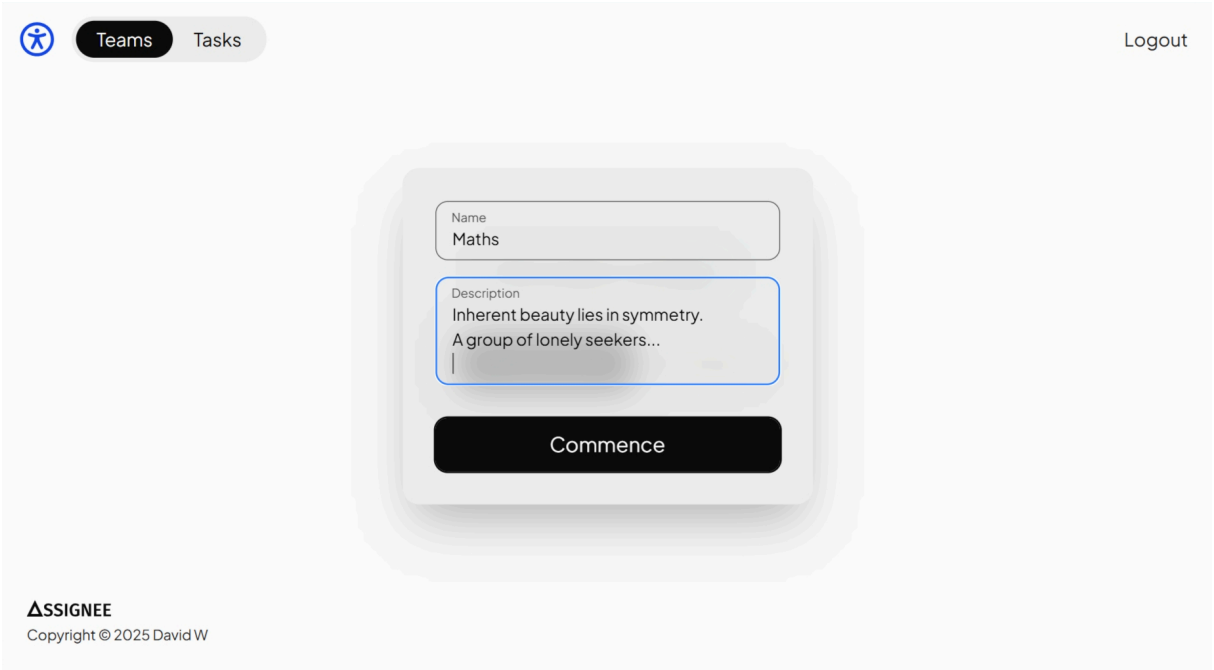

Figure 20: Dashboard, Create Team
MD Variant

Subtle shadows below the overlay and the backdrop blur gives a sense of elevation, directing users to focus on the immediate action.

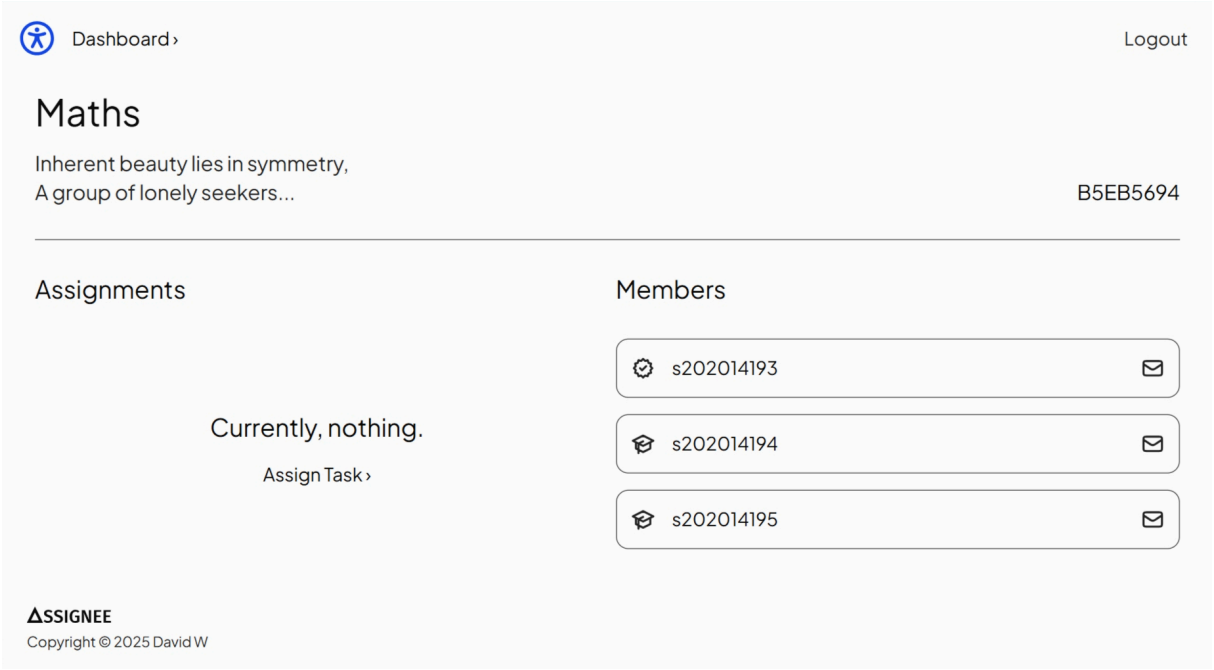

Figure 21: Team Hub
MD Variant

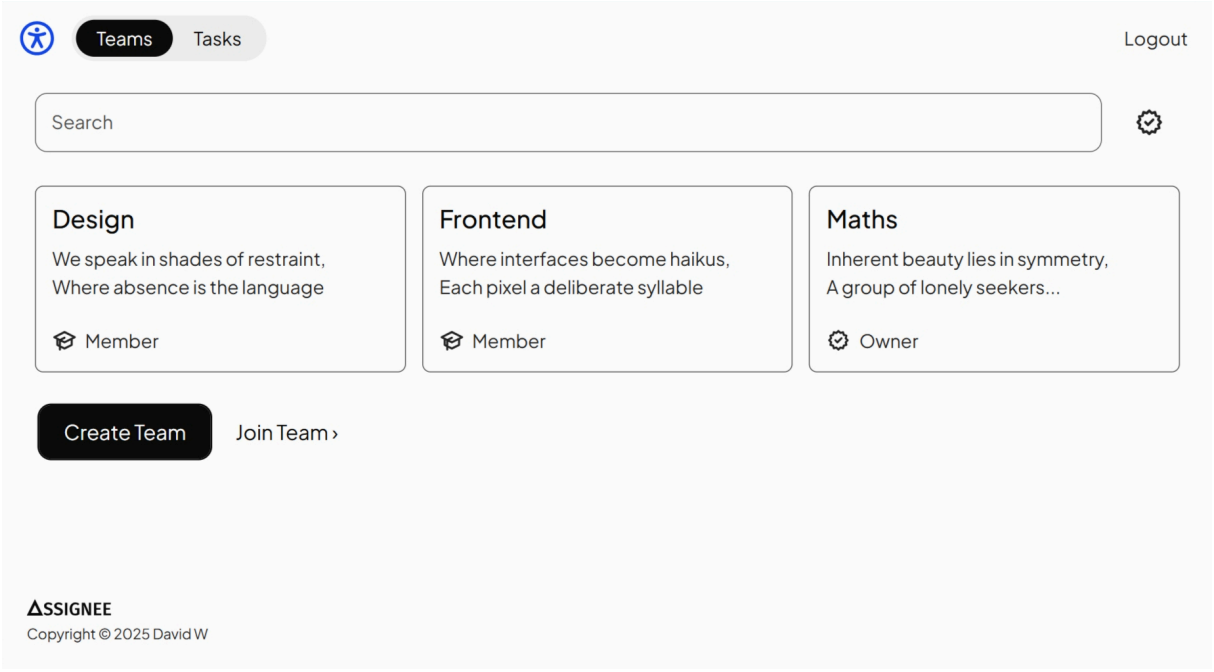

Figure 22: Dashboard, Teams
MD Variant

Cards flow in staggered columns, offset by deliberate white-space intervals. Hairline strokes with subtle radii soften combine to form gentle card edges.

By treating every stroke, curve, and alignment as deliberate concessions to human perception, the dashboard transcends utility to sanctuary.

Figure 23: Dashboard, Accept Invite

Embrace invitations via generous modal prompts. Enter code, done. Team invitation at its simplest, takes you to the team hub without demanding manual actions.

Team owner? Assign tasks through the simplest modal interface. Modals aren't interruptions, they are graceful extensions of the workflow, designed with monastic restraint and geometric intentionality. When invoking task assignment, the modal emerges not as a layer, but as a focused thought.

Click, click, done. Sensible defaults are already set for comment use cases. Autocompletion had never been so convenient. That being said, Assignee forces the presence of task instructions (also team description), to ease the experience of not only one, but everyone.

The team invitation code would be immediately visible to team owners on the right end of the team banner.

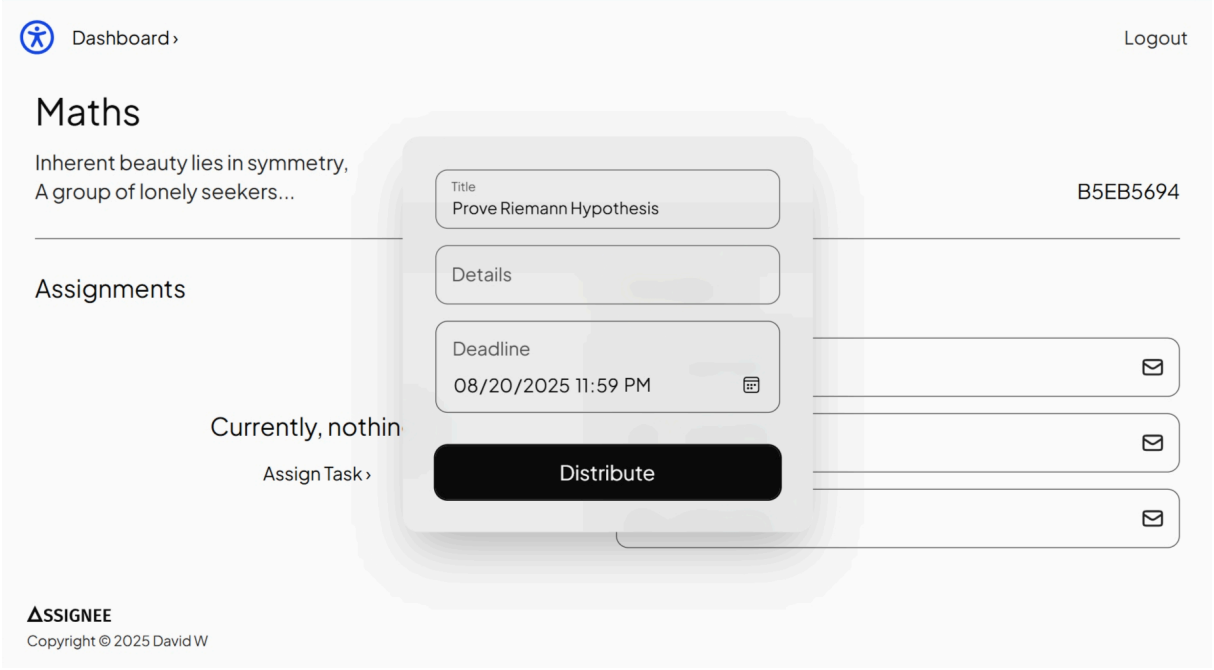

Figure 24: Team, Assign
MD Variant

Both team IDs and task IDs are reversible-hashed with pepper to prevent database data leakage, similar to hashed session IDs in bearer tokens.

- Easily update task reference file through the task overview
- Easily track member submission status
- Easily contact members who haven't finished yet via email (I'm sorry)

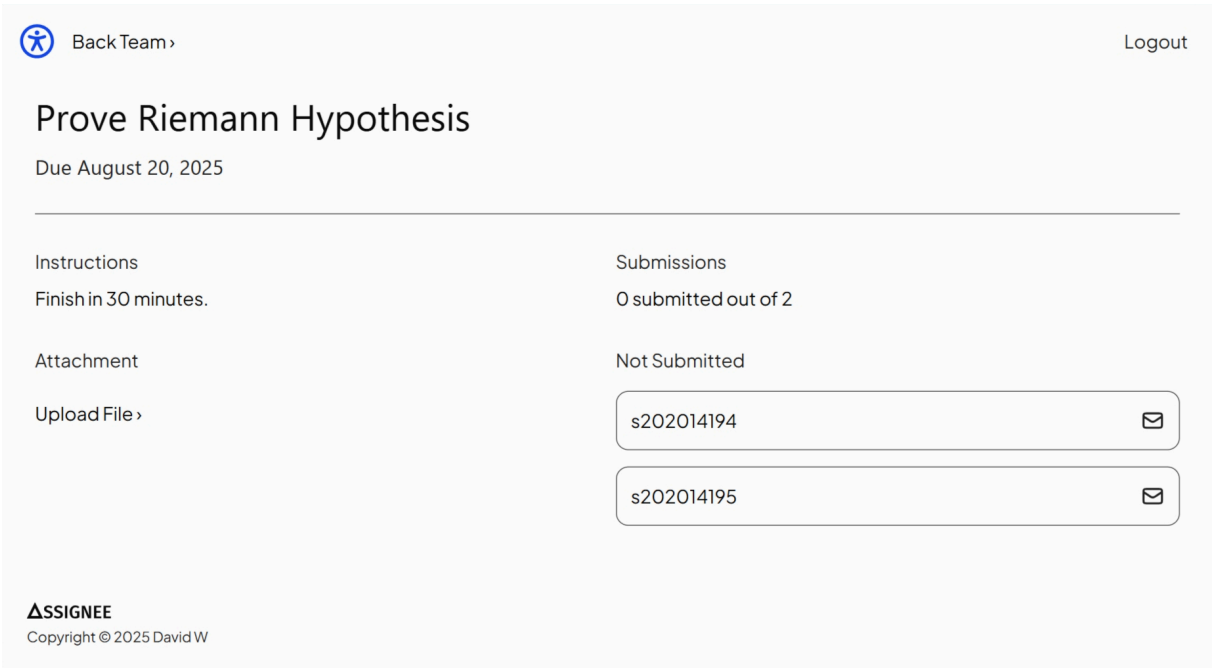

Figure 25: Task Overview
MD Variant

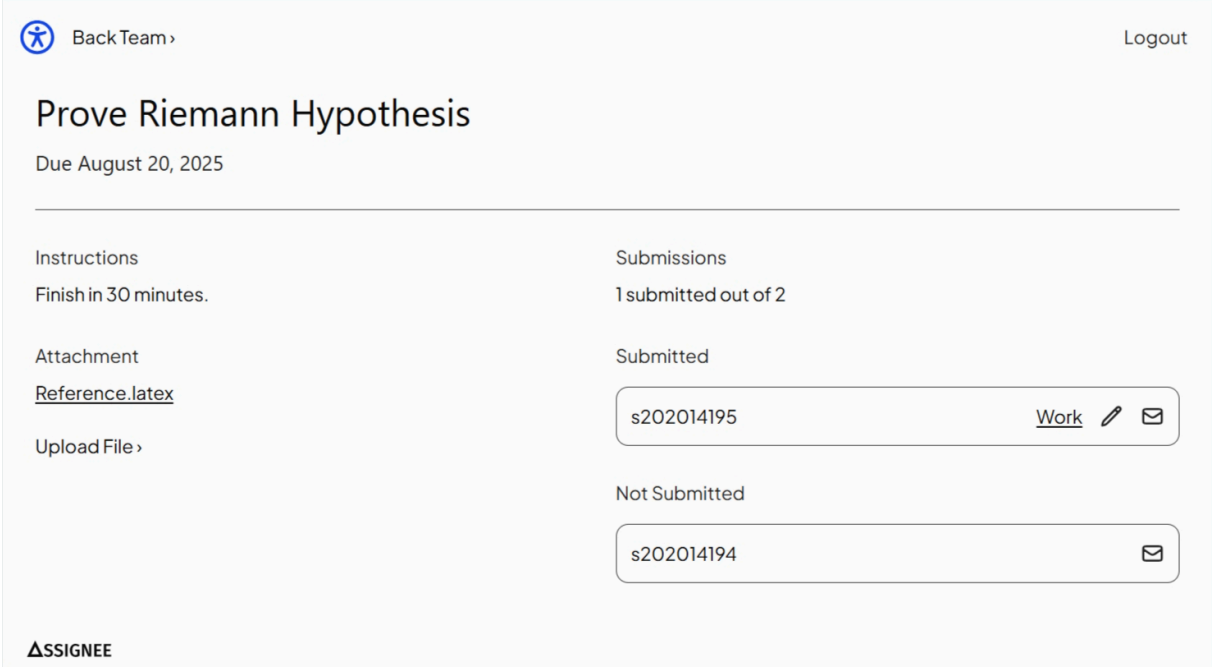

Figure 26: Submissions
MD Variant

Review members' work with unparalleled ease. Download work attachment for review (if any), and optionally provide encouraging feedback via comments.

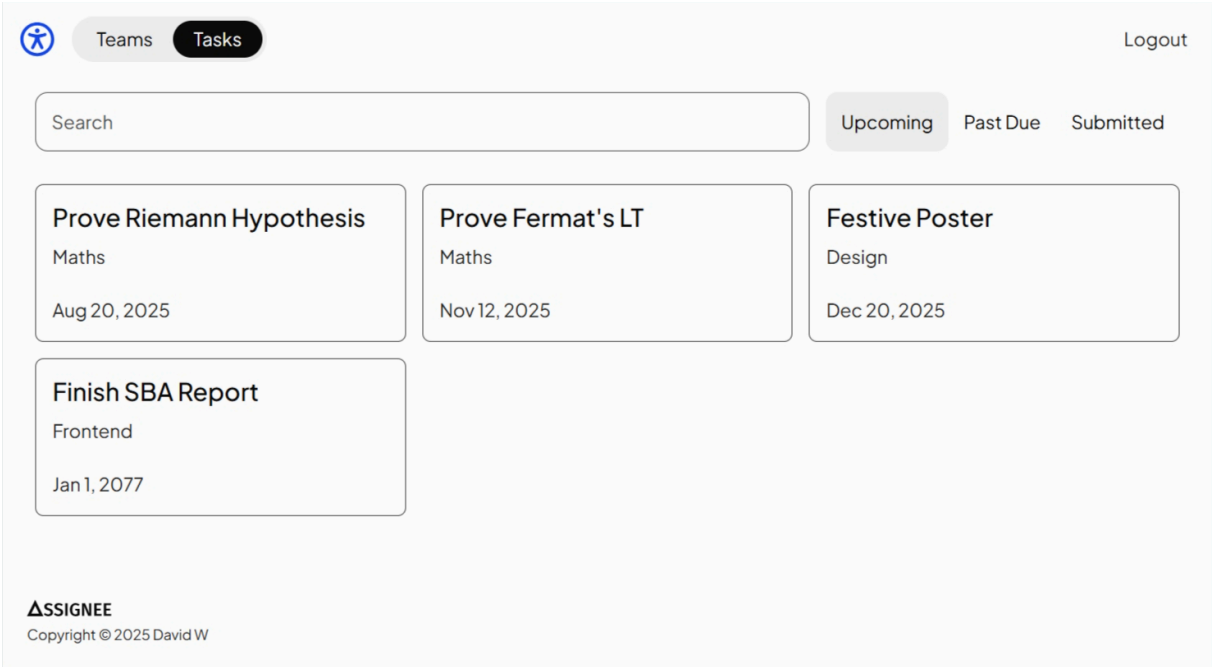

Figure 27: Dashboard, Tasks
MD Variant

Track assignments simply in the integrated dashboard. Filter by completion status, and search with name, team, or description (typos could also be caught and handled).

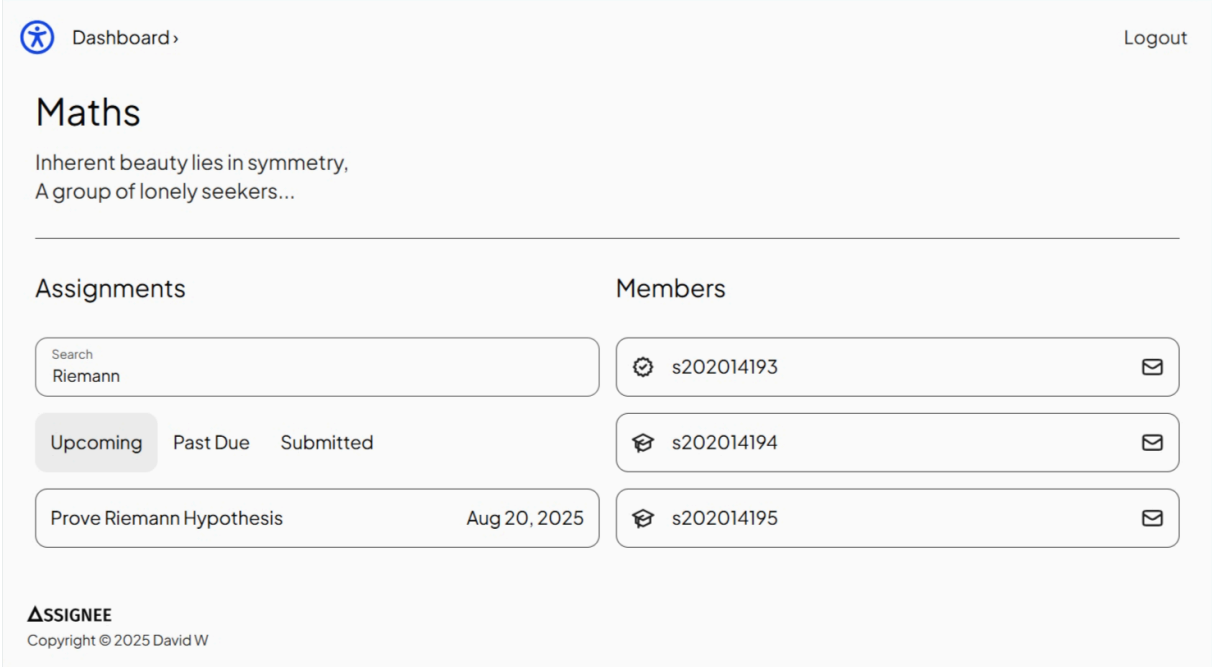

Figure 28: Team Hub, Tasks
MD Variant

Equivalently in the team hub. Team owners will not see the filter. Instead, they would be able to peek over tasks' current submission count.

5.3. Verification

Data validation is done via the communication layer, while data verification is implemented on the website in subtle ways.

Aligning with the core design language of the application, data verification constructs are not extracted to their own components. Instead, they exist within the input, similar to the dynamic form submission buttons. For instance, the "Show Password" toggle on password input fields allow trivial data verification, without relying on an extra "Repeat Password" field.

Subtleness, Correctness.

5.4. Navigation

Distinct from the landing pages (where navigation is too simple), the application views demand dedicated navigation. By blending simple navigation switches naturally in the header (notice that this header is different from the landing page header), Assignee avoids cluttering the view with excessive directives, reducing users' decision fatigue.

5.5. Responsive

Assignee is fully responsive to different screen sizes. This is achieved through the TailwindCSS `md:` media width breakpoint (roughly equivalent to tablet size).

Notice that demonstration images provided in the last section may come with an "MD Variant" postfix. This is because Assignee's styling is mobile-first, since the majority of Internet users nowadays are mobile users.

Refer to the previous section for examples (compare normal and MD variants). Apart from apparent UI changes, Assignee is also mobile user interface friendly, optimizing UX for hassle-free mobile usage.

A deliberate decision has been made to forgo the implementation of dedicated print variants. The rationale is grounded in several key considerations. The primary use case of a dynamic task assignment site resides inherently within its digital, interactive environment. User workflows center on real-time viewing, updating, and managing tasks directly within the application interface. The fundamental nature of the content possesses low inherent value in a static, printed format, which cannot reflect real-time updates or enable interaction.

5.6. Accessibility

Assignee provides numerous accessibility options to align with web standards.

(Default = +)

Font size
- Small
- Medium +
- Large

Language (Options shown in corresponding locales)
- System +
- English
- Chinese

Color theme
- System +
- Light
- Dark

Motion effects
- System +
- On
- Off

By default, Assignee detects system and browser configurations automatically to apply different settings. For instance, color theme, language, and motion could be inferred from the browser to provide optimal UX on first encounter.

The options are configurable through the iconic accessibility button (blue man), which is present on all pages. The configuration would be stored in local storage, allowing it to be persisted over sessions.

Figure 29: Accessibility
MD Variant

Although the options are fully interoperable with each other and plays well with responsive design (128 combinations), only a selected few could be demonstrated here to compact the report (mobile screenshots' aspect ratio would take up too much space).

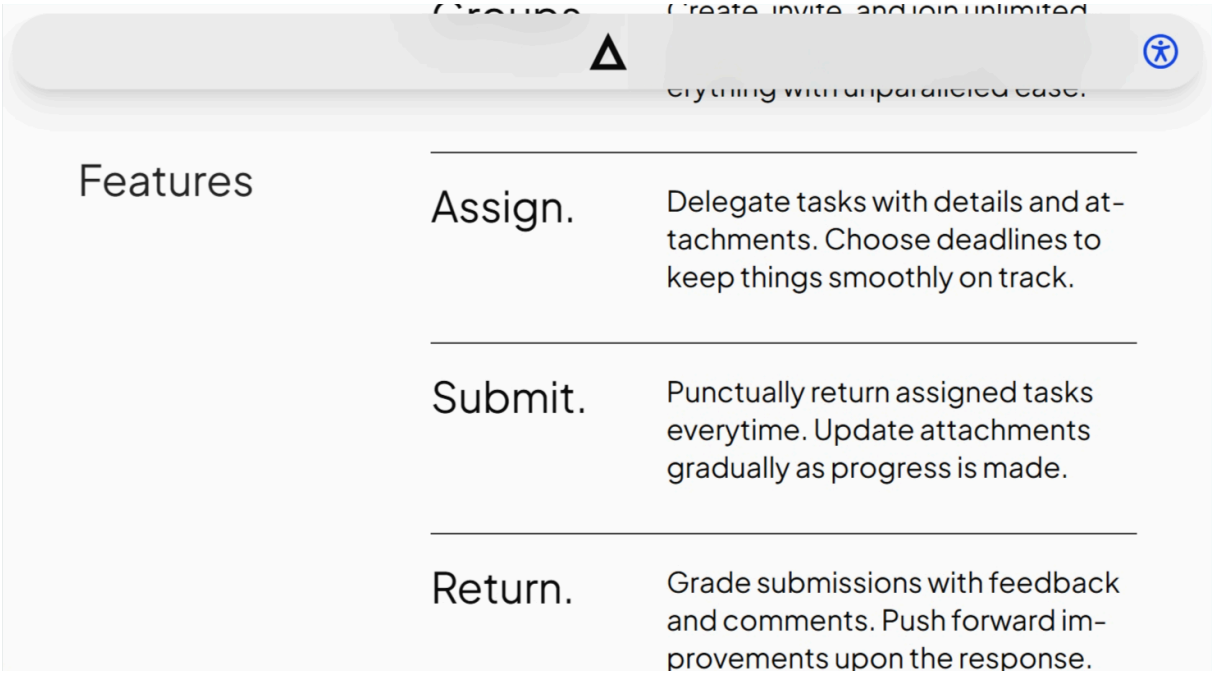

Figure 30: Accessibility, Font Large
MD Variant

Figure 31: Accessibility, Lang Chinese
MD Variant

In fact, even date, time, and hidden accessibility ARIA labels, would be translated to the specified locale. Translation is implemented everywhere across the application, not just specific parts. Doesn't rely on machine translate, localization is done by me manually to ensure conciseness.

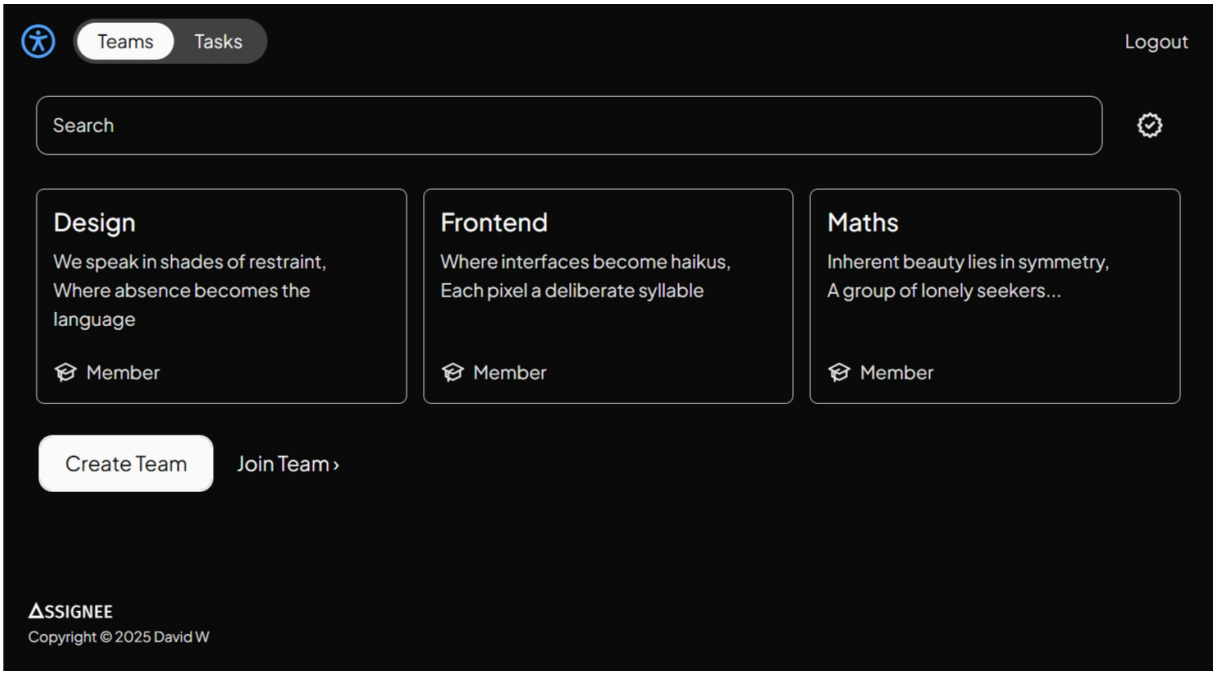

Figure 32: Accessibility, Dark Mode
MD Variant

Dark mode is achieved through global theme variable declaration in TailwindCSS, all colors in both themes are carefully curated by me manually to ensure contrast.

Reduce motion cannot be demonstrated directly, in words: it disables scroll smoothing, scroll-triggered animations, and CSS interaction-triggered transitions.

Apart from the aforementioned options, other practices are also implemented to ensure maximum accessibility.

This includes:

- Including ARIA labels for non-text components
- Separate signin/up pages to avoid confusing password managers
- Keyboard navigation support e.g. Esc to close modal, Tab to inputs
- Ensuring theme colors bear enough contrast (WCAG AAA compliance)
- Following form accessibility guidelines e.g. autocomplete, spellcheck, labels
- Using appropriate semantic elements to define content type e.g. `<header>`, `<footer>`, `<search>`

Together, Assignee helps to build a web accessible to everyone.

5.7. Implementation

The frontend is implemented in TypeScript with the SolidJS framework, with the following benefits:

- Simple JSX component reuse
- Reactive component updates
- Performant element renderer
- Modern client-side page router
- Efficient JSX caching mechanism

Only a skeleton HTML entry is needed when using dynamic JSX libraries:

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <title>Assignee</title>

  <link rel="preload" as="font" crossorigin="anonymous"
    type="font/woff2"
    href="assets/fonts/plus-jakarta-sans-v11-latin-regular.woff2"
  />
  <link rel="preload" as="font" crossorigin="anonymous"
    type="font/woff2"
    href="assets/fonts/plus-jakarta-sans-v11-latin-500.woff2"
  />
  <link rel="icon"
    type="image/png"
    href="assets/favicon.png"
  />

</head>
```

```
<body class="bg-main">
  <div id="root" class="flex min-h-dvh flex-col items-center"></div>

  <noscript>JavaScript Must Be Enabled to Run This App</noscript>
  <script src="src/index.tsx" type="module"></script>
</body>

</html>
```

Notice the inclusion of default page language, appropriate metadata tags, preload links, and message on disablement of JavaScript.

An example of reusable GUI components is a button variant:

```
export default (props: Props<"button"> & { pill?: boolean; full?: boolean }) => (
  <button
    {...props}
    type="button"
    class={clsx(
      props.class,
      "bg-button font-jakarta text-text-alter outline-button cursor-pointer px-6 py-3 text-xl
outline-2 transition-[outline-offset] duration-100 ease-out hover:outline-offset-2 focus:outline-
offset-2",
      props.full ? "w-full" : "w-max",
      props.pill ? "rounded-full" : "rounded-xl",
    )}
  ></button>
);
```

Look at how many styles and animations could be applied to a simple yet elegant button! TailwindCSS greatly boosts productivity and allows faster iterations and testing. JSX combined with TailwindCSS delivers maximum flexibility in component reuse, all you need to include the button is a single line of TSX (TypeScript React for HTML embedding): `<Button pill>Hello World!</Button>`.

5.7.1. Translation

I18n is implemented via a complex system of event listeners and dynamically retrieved dictionaries. Internals listen for language changes, and signals SolidJS contexts to replace their content at runtime by retrieving dictionaries we define. For instance:

```
export default defineI18n({
  en: {
    header: "Nice to have you back. Sign in to resume.",
    errors: {
      generic: "Invalid Email or Password",
      ratelim: "Too Many Requests",
    },
  },
  zh: {
    header: "...",
    errors: { /* ... */ },
  },
});
```

Additional languages could be defined in no time with this flexible framework.

5.7.2. Page Styling

Page styling is done primarily via TailwindCSS, which compiles inline classes into corresponding CSS statements (e.g. `h-4` ⟶ `{ height: 1rem }`). This reduces the total output external stylesheet size by algorithmically optimizing class usage. It also simplifies style reuse and promotes style consistency by facilitating global theme variables:

```
:root {
  --color-a11y: var(--color-blue-700);
  --color-border: var(--color-neutral-700);
  --color-button: var(--color-neutral-950);
  --color-holder: var(--color-neutral-600);
  --color-main: var(--color-neutral-50);
  --color-outline: var(--color-blue-500);
  --color-overlay: var(--color-neutral-200);
  --color-shadow: var(--color-neutral-950);
  --color-text-alter: var(--color-neutral-50);
  --color-text-major: var(--color-neutral-950);
  --color-text-minor: var(--color-neutral-800);
}

:root.darkmode {
  /* theme variable overrides ... */
}
```

Relative EM units i.e. `em` and dynamic viewport units e.g. `vh` are used instead of absolute units e.g. `px` (with some exceptions) in CSS to ensure style consistency. It also backs the font size accessibility option, allowing containers to grow together with text.

FOUC (flash of unrendered content) is prevented by resolving DNS early with font preloading.

5.7.3. Animations

Some subtle animations (e.g. modal popup) are created with GSAP, a sophisticated animation platform, which also features a scroll trigger (e.g. header scroll).

Smooth scrolling of the webpage is enabled by Lenis, a lightweight scroll-smoother library.

5.7.4. Media Usage

The deliberate use of minimal black and white SVGs within Assignee is a design rooted in essentialism and sophisticated clarity. The scalability and crisp precision of SVGs ensure icons and graphic elements remain sharp and adaptable at any size, perfectly complementing the clean lines and uncluttered spaces inherent in minimalism. It embodies functional elegance, enhancing readability, ensuring timelessness, and improves load times, resulting in an experience that feels both refined and effortlessly intuitive.

The design intentionally forgoes audio and video to uphold a minimalist, calming philosophy that prioritizes user focus and reduces cognitive load. The exclusion of these high-stimulus elements prevents disruptive context-switching, ensuring the interface remains a silent, efficient tool for organization. This enhances site performance and reliability across all devices and networks, broadens accessibility for users with sensory impairments, and guarantees professional universality for use in quiet environments like offices or libraries. Ultimately, the choice creates a more respectful, dependable, and focused user experience.

In place of multimedia, subtle animations are used to provide essential user feedback. These purposeful motions offer clear, immediate confirmation of actions. This maintains the calm aesthetic while ensuring the interface feels responsive and alive.

5.7.5. Backend Requests

Exchanging data with the backend is done via Axios, enabling async I/O operations with automatic request header configuration.

5.7.6. Performance

SolidJS is one of the most performant frontend libraries currently available. But to deliver maximum application performance, the application is bundled, tree-shaked, and minified. Certain assets are externalized to enable better static resource caching.

Simulated throttling are used while inspecting to ensure Assignee is performant even for devices with poor computational power or connection.

5.7.7. Quality Assurance

Google Issues and Lighthouse are used extensively to enforce web best practices. This includes loading speed, contentful paint (FCP/LCP), and accessibility.

By reducing loading time and network dependency chains, initial rendering is sped up. This allows Assignee to deliver the best level of user satisfaction and engagement.

Assignee is tested on both laptop, tablet, and phone on real devices. It is also tested comprehensively for other dimensions with responsive design tools.

6. Credits

The success of Assignee relied heavily on the extensive use of ecosystem tools. While only a fraction are listed here, it's important to acknowledge that even a quarter of them couldn't be fully covered.

Items marked with + are written by myself during development.

6.1. Common

- Git° (https://git-scm.com)
- GitHub° (https://github.com)
- VSCode° (https://code.visualstudio.com)
- Conventional Commits° (https://github.com/vivaxy/vscode-conventional-commits)
- PNPM° (https://pnpm.io)
- TypeScript° (https://www.typescriptlang.org)
- ESLint° (https://eslint.org)
- Prettier° (https://prettier.io)
- Husky° (https://github.com/typicode/husky)
- Lint Staged° (https://github.com/okonet/lint-staged)
- Catppuccin° (https://github.com/catppuccin/vscode)
- Alt Delete° (https://github.com/wavim/vscode-alt-delete) +
- IstrainLess° (https://github.com/wavim/vscode-istrainless) +
- NeoGit° (https://github.com/wavim/neogit) +
- Git Branch° (https://github.com/wavim/vscode-git-branch) +
- Better Memo° (https://github.com/wavim/vscode-better-memo) +

6.2. Database

- SQLite° (https://www.sqlite.org)
- SQLite Studio° (https://sqlitestudio.pl)
- DB Visualizer° (https://www.dbvis.com)

6.3. Backend

- Node.js° (https://nodejs.org)
- Prisma° (https://www.prisma.io)
- Express.js° (https://expressjs.com)
- Compression° (https://github.com/expressjs/compression)
- Express Rate Limit° (https://github.com/express-rate-limit/express-rate-limit)
- Multer° (https://github.com/expressjs/multer)
- HashIDs° (https://hashids.org)
- Noble Hashes° (https://github.com/paulmillr/noble-hashes)
- HTTP Error° (https://github.com/wavim/http-error) +
- TSX° (https://github.com/privatenumber/tsx)
- Rollup° (https://github.com/rollup/rollup)
- CPR° (https://github.com/davglass/cpr)
- PKG° (https://github.com/yao-pkg/pkg)

- Google Issues° (https://developer.chrome.com/docs/devtools/issues)

6.4. Schema

- Zod° (https://github.com/colinhacks/zod)
- Rollup° (https://github.com/rollup/rollup)

6.5. Design

- SVG Repo° (https://www.svgrepo.com)
- Site Inspire° (https://www.siteinspire.com)
- Pattern Pad° (https://patternpad.com)
- Google Fonts° (https://fonts.google.com)
- Plus Jakarta Sans° (https://github.com/tokotype/PlusJakartaSans)
- WebAIM Contrast Checker° (https://webaim.org/resources/contrastchecker)

6.6. Frontend

- Vite° (https://vitejs.dev)
- SolidJS° (https://solidjs.com)
- Solid Router° (https://github.com/solidjs/solid-router)
- Solid I18n° (https://github.com/solidjs-community/solid-primitives/tree/main/packages/i18n)
- Solid Filter° (https://github.com/wavim/solid-filter) +
- TailwindCSS° (https://tailwindcss.com)
- CLSX° (https://github.com/lukeed/clsx)
- Lenis° (https://github.com/darkroomengineering/lenis)
- GSAP° (https://greensock.com/gsap)
- String Similarity° (https://github.com/aceakash/string-similarity)
- Wrap JSX° (https://github.com/wavim/vscode-wrap-jsx) +
- GFont Loader° (https://github.com/wavim/gfont-loader) +
- Omnires° (https://github.com/wavim/omnires) +
- Natural Log° (https://github.com/wavim/natural-log) +
- Lighthouse° (https://developer.chrome.com/docs/lighthouse)
- Google Issues° (https://developer.chrome.com/docs/devtools/issues)

6.7. Report

- LaTeX° (https://www.latex-project.org)
- Typst° (https://typst.app)
- ILM° (https://typst.app/universe/package/ilm)
- Tiny Mist° (https://github.com/Myriad-Dreamin/tinymist)
- PDF Viewer° (https://github.com/tomoki1207/vscode-pdfviewer)
- DeepSeek° (https://chat.deepseek.com)
- LTeX+° (https://github.com/ltex-plus/vscode-ltex-plus)

While only a selection is mentioned here, I am deeply grateful for the entire ecosystem that made Assignee possible.

7. Final Remarks

This report reflects original research, analysis, and insights. While generative AI tools were selectively used to enhance the clarity of this report, all core ideas, findings, and conclusions remain the product of human effort.

In the AI-augmented era, I have taken care to uphold academic and professional integrity throughout this work, by acknowledging how to use AI tools responsibly.

As I wrap up this report, I extend my thanks to the many tools, contributors, and collaborators who made Assignee a reality. I want to affirm my commitment to respecting intellectual property and licensing rights. In an era of boundless digital collaboration, I believe progress thrives when credit is given fairly, and innovation is built responsibly.