



Caza

caza.sivarn.com

CS3216 Software Product Engineering for Digital Markets

Assignment 3 Milestones Report

Team Info:

Bharath Chandra Sudheer	A0218550J
Gan Hong Yao	A0217912H
Han Geng Ning	A0222055U
Ruppa Nagarajan Sivayoga Subramanian	A0217379U

Note to graders	4
Milestone 0	4
Milestone 1	4
Milestone 2	5
Bring in new users for the product	6
Gathering feedback to retain users	6
Milestone 3	7
Milestone 4	8
REST API	8
GraphQL	8
REST API vs GraphQL: Pros and Cons	8
Data underfetching and overfetching	8
Error handling	9
Caching	9
When to use which?	9
Our Choice	10
Milestone 5	10
Milestone 6	17
View Application List	17
Create Application	18
Company Autocomplete	20
Milestone 7	21
Milestone 8	23
Milestone 9	31
Strike a balance between key security and performance	31
Keep private keys well-protected	31
Renew certificates in a timely manner	32
Milestone 10	32

Milestone 11	33
Overview of Token-Based	33
Overview of Session-Based	34
Caza Implementation	34
Milestone 12	35
Frontend Framework	35
Backend Framework	35
UI Framework	36
5 Mobile site design principles	36
Making search visible	36
Designing Finger-Friendly Touch Targets	37
Keeping the user in a single browser window	37
No pinch to zoom	37
Optimize site for mobile	38
Milestone 13	38
Milestone 14	41
Milestone 15	41
Milestone 16	42

Note to graders

Some of our features use aggregated user data to display information about roles. Information like how many people are at which stages, and how long they've spent at each stage is collected and rendered for the user. We have seeded the production database with some fake data so that testers can see the feature as if this app already has the users it needs for collecting this data..

Milestone 0

Describe the problem that your application solves.

University students have a lot on their plate, and for those who aspire to break into software engineering, the internship season can be quite the headache. Caza reduces their burden by helping students track their internship applications.

Applying for internships sucks. Tracking these applications sucks more. Most people right now use Excel sheets, note-taking applications or other general purpose organizing solutions. With any of these options, they need to configure and force the application to fit their requirements. With Caza, they will get an app and a beautiful interface built specifically for their needs. It even comes with additional useful features like seeing user statistics on roles they've applied to.

Milestone 1

Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make the most sense to implement your application as a mobile cloud application?

Caza is an internship or job application tracker for computing students. It allows users to track the progress of the applications, note down tasks, and access global statistics of applications for various job roles.

Firstly, users could get updates about their application at any time, over a phone call, or while reading their emails on their desktop. Mobile cloud computing allows users to quickly update their application status through any device and at any location with internet connection before they forget.

Next, users benefit from seeing global statistics about the roles they have applied to. Mobile cloud computing is ideal to store every user's data in the cloud so that global statistics can be computed and delivered to any user easily.

Moreover, tracking applications could happen over years and users may change devices over time. Mobile cloud computing is platform independent and allows users to continue accessing the app regardless of changes in device. In the first place, Computer Science students are likely to use a range of different mobile and desktop devices so an application that is platform independent can cater to most of our target users.

Lastly, as users would track application-related tasks on the app, storing data in the cloud could easily allow extensions to set up notifications across different devices. This allows users to be reminded promptly at any point in time.

Milestone 2

Describe your target users. Explain how you plan to promote your application to attract your target users.

Our main target users are computing students in university. Computing students often look for internships every academic year. They have to apply for many roles and complete application tasks by their deadlines. The applications often involve many phases from online assessments, to different rounds of technical and interviews. They need an organized way to track their applications and ensure deadlines are met. They also want to have an idea of the application progress. Currently, it is frustrating to track progress of application statuses over time on standard note-taking apps as it does not display the trajectory of their application progress. It also cannot highlight pending tasks. Our users appreciate a convenient tracking solution that can be accessed from multiple devices. They want an organized interface while minimizing time spent on tracking applications.

We decided to break up the marketing campaign into two phases that are in tandem with the product launch. These are our main aims:

1. Bring in new users for the product.
2. Obtain feedback and track growth to improve user experience over time to better serve our users.

Bring in new users for the product

There have been many of such groups in the CS student community that have gained in popularity over the last couple of years, such as subreddits [r/csMajors](#) and [r/cscareerquestions](#). We have been browsing these subreddits for quite a while and we have seen many posts where people ask around about one another's current stages. We aim to broadcast this application to these social media channels to gain traction.

Closer to home, there is the Telegram group, [Project Intern](#), created by the NUS Hackers core team in NUS. It would help that we are NUS students ourselves, and we could get more people that we know personally to try our application so that we can gather feedback and some initial data for us to iterate on.

Gathering feedback to retain users

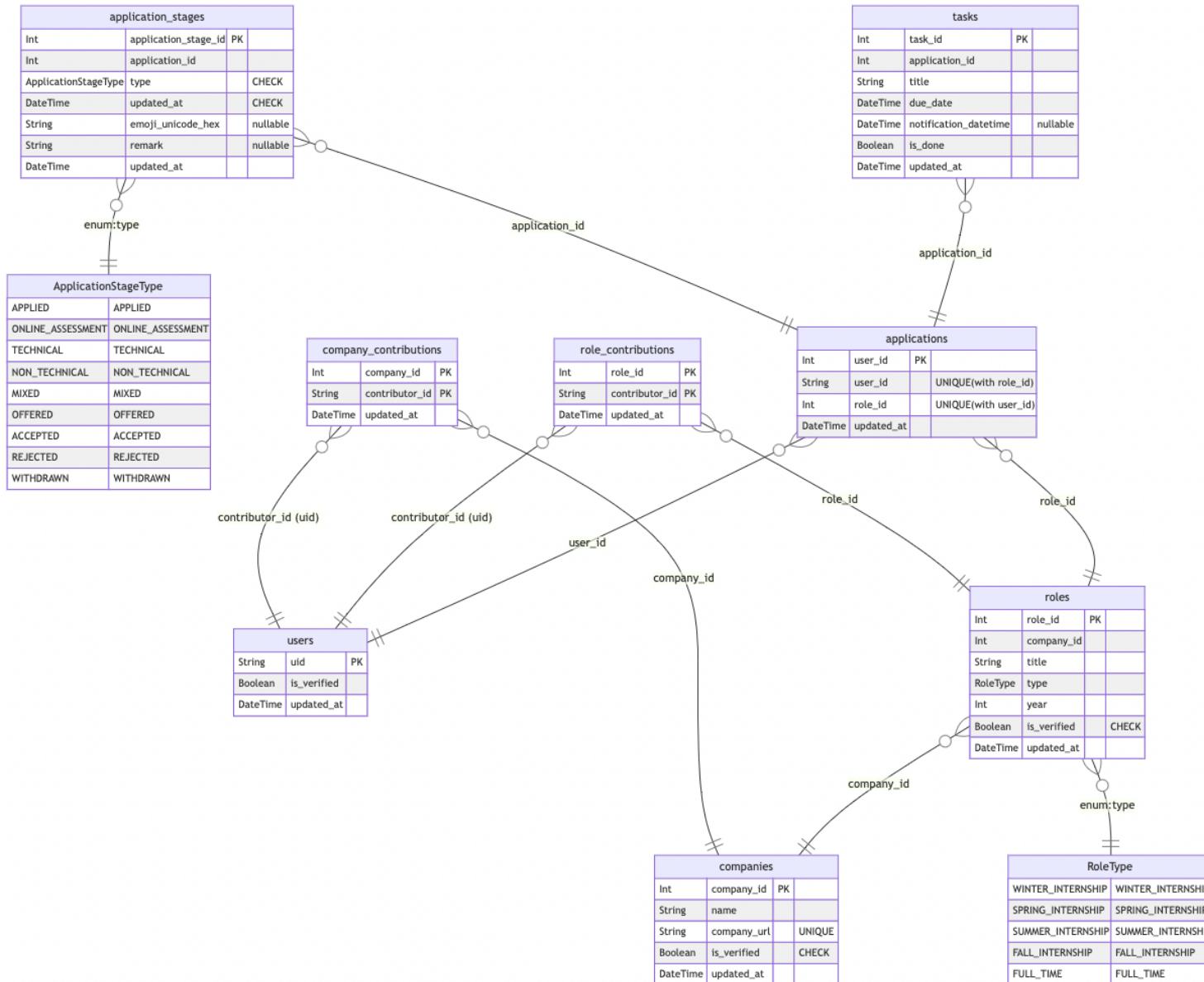
We have set up a Google Analytics dashboard for us to monitor page view statistics as well as track the events that we have associated with various user actions in the application. We will use these insights to analyze common user patterns and investigate if the user flows are what we expect to see.

We also plan to collect explicit user feedback by displaying prompts on our application so that the user will see the prompt when they browse our application. This helps to gather suggestions from real end-users.

Google Analytics is easy to set up and provides out-of-the-box dashboards that provide digestible insights. By collecting these data, we can identify demographics of users and the devices on which they use our application. This helps us understand our users better and gives us ideas on how we can optimize the user experience.

What Google Analytics does not provide is explicit user feedback, which we will aim to obtain using feedback from components on the application. With concrete recommendations, we can have a rough sense of what the users wish to see in future iterations of the app.

Milestone 3



Additional Notes:

Check for application_stages:

The set of application_stages for a given application is only valid if the chronology follows these rules:

- Only 1 'APPLIED' stage, and it must be the earliest.
- 1 or 0 of 'OFFERED',

'ACCEPTED', 'REJECTED' or 'WITHDRAWN' stages, and it must be the latest.

Verifying user-created roles and companies:

The tables company_contributions and role_contributions keep track of user-created unverified companies and roles.

Once at least 5 users contribute to the same role or company, the property `is_verified` for that entity in roles and / or companies is set to true.

Milestone 4

Explore one alternative to REST API (may or may not be from the list above). Give a comparison of the chosen alternative against REST (pros and cons, the context of use, etc.).

Between REST and your chosen alternative, identify which might be more appropriate for the application you are building for this project. Explain your choice.

REST API

In REST API, **resources** form the basis of the API design. A resource in REST API is just like a class in object-oriented programming. A typical REST-oriented API would expose a few standard methods (e.g. LIST, GET, CREATE, PUT and DELETE) which act upon the resources.

GraphQL

In GraphQL, the server writes a **schema** to describe the shapes of the available data. Read operations are done using **queries** and write operations are done using **mutations**.

REST API vs GraphQL: Pros and Cons

Data underfetching and overfetching

GraphQL has one major plus point over REST API - queries allow fetching and delivering exactly what the client requires which reduces data transfer overhead. This is in contrast to REST API where resources live on different routes. In the case that multiple entities are required, in REST, we will have to either make multiple requests to different endpoints or to write new endpoints to serve the request, which may violate the essence of REST API if this new endpoint does not encapsulate a resource.

Conversely, sometimes we just want a small subset of the entity. It is then expensive and large overhead is incurred to fetch the full entity in a REST design. Queries in GraphQL provide the flexibility to cater to any kind of data requests.

Error handling

In a REST API, errors are handled using HTTP response status codes and the codes are grouped in five classes. For example, status codes 5xx indicate a server error. These status codes make it clear to the client about the type of error that has occurred.

In a GraphQL API, status codes are not used by default. It is common practice to use the 200 OK status code for all requests. Non-200 response codes indicate issues with the HTTP transport layer instead of the GraphQL layer. When errors happen, the standard error handling mechanism is to place the errors in an `errors` array as part of the response body. It is difficult to understand where the error came from - some errors may be domain-specific with client-facing error messages (like 400 Bad request, while others could be more technical such as database errors. This makes it difficult for the client to display the appropriate messages to the user.

Caching

In GraphQL, queries are mostly done through POST requests. This breaks some caching mechanisms like intermediate proxy caching and service worker caching which only supports caching of GET requests. Although the GraphQL spec mentions that both GET and POST should be supported, when doing so, the query has to be sent as a URL query parameter since a GET request cannot have a request body. This becomes problematic in bigger queries and one may run into `URI Too Long` errors on certain servers. There have been workarounds such as using the query string as a cache key and having the service worker cache the response in IndexedDB, but one would have to implement such a solution themselves.

When to use which?

REST API is a rather straightforward approach if the web application is a CRUD-style application, which makes it easy to model the objects as resources and actions as HTTP methods. It is also more preferred over GraphQL if out-of-the-box caching is desired.

GraphQL would be the obvious choice if the API has to serve clients with data requirements that vary a lot by only slight differences. GraphQL allows the API to be highly flexible in terms of customizing requests.

Our Choice

We decided to use the REST API for our application. Our application is based on only a few entities which could be easily separated into distinct resources. We also perform CRUD operations on these entities and separating these operations into different HTTP methods on different endpoints makes it cleanly separated.

More importantly, since we were building a Progressive Web App, we needed our API to be highly cacheable by the service workers so that the application can still perform well on low-quality networks and offline.

Milestone 5

Design and document all your REST API. If you already use Apiary to collaborate within your team, you can simply submit an Apiary link. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with an explanation of the purpose of each request for reference. Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices (if any). You will be penalized if your design violates principles for no good reason.

Note

- Every request's authorization header must contain a Bearer Token and the token must be valid in order for api requests to be authorized.
 - Authorization: Bearer <TOKEN>
- Every request's response follows the following format
 -

```
{
  payload: PayloadType;
  messages?: { type: 'ERROR' | 'SUCCESS'; message: string }[]
}
```

- The table below specifies payload for each response
- The format of requests and response in the table below is stated as { fieldName : type }. It follows Typescript type notations. The actual api request and response will be as **JSON payloads**.

Request Method + Relative URL	Purpose	Request Parameters / Body	Response Values (Payload only)
GET /companies	To find companies when	{ companyNames:	{ id: number;

	creating applications.	string; }	name: string; companyUrl: string; }[]
POST /companies	To create companies that do not exist in our database yet before creating an application for it.	{ name: string; companyUrl: string; }	{ id: number; name: string; companyUrl: string; }
GET /roles	To find roles when creating applications.	{ companyId?: number; searchWords: string; }	{ id: number; title: string; type: RoleType; year: number; isVerified: boolean; company: CompanyData; }[] RoleType: { 'WINTER_INTERNSHIP', 'SPRING_INTERNSHIP', 'SUMMER_INTERNSHIP', 'FALL_INTERNSHIP', 'FULL_TIME', } CompanyData = { id: number; name: string; companyUrl: string; }
POST /roles	To create roles that do not exist in our database yet before creating an application for it.	{ companyId: number; title: string; type: string; year: number; }	{ id: number; title: string; type: RoleType; year: number; isVerified: boolean; } RoleType: { 'WINTER_INTERNSHIP', 'SPRING_INTERNSHIP', 'SUMMER_INTERNSHIP', 'FALL_INTERNSHIP', 'FULL_TIME', }
GET /roles/world	To get an overview of the number of people at each stage of each	{ searchWords: string[]; roleTypeWords: string[]; }	{ id: number; title: string; type: RoleType; year: number; }

	role.	<pre>shouldFilterForCurrentUserApplications : boolean; }</pre>	<pre>isVerified: boolean; company: CompanyData; applicationStages: { type: ApplicationStageType; count: number }[]; }[] RoleType: { 'WINTER_INTERNSHIP', 'SPRING_INTERNSHIP', 'SUMMER_INTERNSHIP', 'FALL_INTERNSHIP', 'FULL_TIME', } CompanyData = { id: number; name: string; companyUrl: string; } ApplicationStageType: { 'APPLIED', 'ONLINE_ASSESSMENT', 'TECHNICAL', 'NON_TECHNICAL', 'MIXED', 'OFFERED', 'ACCEPTED', 'REJECTED', 'WITHDRAWN' }</pre>
<pre>GET /roles/world/:roleId</pre>	To get a detailed breakdown of the number of people at each stage and the general progression for a role.		<pre>{ role: RoleApplicationListData; numberOfApplications: number; nodes: string[]; edges: RoleSankeyEdgeData[]; } RoleApplicationListData = { id: number; title: string; type: RoleType; year: number; isVerified: boolean; company: CompanyData; } RoleSankeyEdgeData = {</pre>

			<pre> source: RoleSankeyNodeId; dest: RoleSankeyNodeId; userCount: number; totalNumHours: number; } </pre>
GET /applications	To get a user's applications. Certain filters are allowed to facilitate search.	<pre> { searchWords: string; roleTypeWords: string; stageTypeWords: string; } </pre>	<pre> { id: number; role: RoleApplicationListData; latestStage?: ApplicationStageApplicatio nListData; taskNotificationCount: number; } RoleApplicationListData = { id: number; title: string; type: RoleType; year: number; isVerified: boolean; company: CompanyData; } RoleType: { 'WINTER_INTERNSHIP', 'SPRING_INTERNSHIP', 'SUMMER_INTERNSHIP', 'FALL_INTERNSHIP', 'FULL_TIME', } CompanyData = { id: number; name: string; companyUrl: string; } ApplicationStageApplicatio nListData = { id: number; type: ApplicationStageType; date: string; emojiUnicodeHex: string null; } ApplicationStageType: { 'APPLIED', 'ONLINE_ASSESSMENT', 'TECHNICAL', </pre>

			<pre>'NON_TECHNICAL', 'MIXED', 'OFFERED', 'ACCEPTED', 'REJECTED', 'WITHDRAWN' }</pre>
<p>POST /applications</p>	<p>To create a new application for a user.</p>	<pre>{ roleId: number; applicationDate: string; }</pre>	<pre>{ id: number; role: RoleData & { company: CompanyData }; } RoleData = { id: number; title: string; type: RoleType; year: number; isVerified: boolean; } CompanyData = { id: number; name: string; companyUrl: string; }</pre>
<p>GET /applications/: applicationId</p>	<p>To get details of an application, including its list of stages and tasks.</p>		<pre>{ id: number; role: RoleData & { company: CompanyData }; applicationStages: ApplicationStageApplicatio nData[]; tasks: TaskData[]; } RoleData = { id: number; title: string; type: RoleType; year: number; isVerified: boolean; } CompanyData = { id: number; name: string; companyUrl: string; } ApplicationStageApplicatio nData = { id: number;</pre>

			<pre> type: ApplicationStageType; date: string; emojiUnicodeHex: string null; remark: string null; } ApplicationStageType: { 'APPLIED', 'ONLINE_ASSESSMENT', 'TECHNICAL', 'NON_TECHNICAL', 'MIXED', 'OFFERED', 'ACCEPTED', 'REJECTED', 'WITHDRAWN' } TaskData = { id: number; title: string; dueDate: string; notificationDateTime: string null; isDone: boolean; } </pre>
DELETE /applications/: applicationId	To delete an application.		{}
POST /applications/: applicationId/s tages	To create a stage in an application.	<pre> { type: string; date: string; emojiUnicodeHex: string null; remark: string null; } </pre>	<pre> { id: number; applicationid: number; type: ApplicationStageType; date: string; emojiUnicodeHex: string null; remark: string null; } </pre>
PATCH /applications/: applicationId/s tages/:stageId	To update a stage in an application.	<pre> { type?: string; date?: string; emojiUnicodeHex?: string null; remark?: string null; } </pre>	<pre> { id: number; applicationid: number; type: ApplicationStageType; date: string; emojiUnicodeHex: string null; remark: string null; } </pre>
DELETE /applications/:	To delete a stage in an		{}

<code>applicationId/stages/:stageId</code>	application.		
POST <code>/applications/:applicationId/tasks</code>	To create a task in an application	<pre>{ title: string; dueDate: string; notificationDateTime: string null; }</pre>	<pre>{ id: number; title: string; dueDate: string; notificationDateTime: string null; }</pre>
PATCH <code>/applications/:applicationId/tasks/:taskId</code>	To update a task in an application.	<pre>{ title?: string; dueDate?: string; notificationDateTime?: Nullable<string>; isDone?: boolean; }</pre>	<pre>{ id: number; title: string; dueDate: string; notificationDateTime: string null; }</pre>
DELETE <code>/applications/:applicationId/tasks/:taskId</code>	To delete a task in an application.		<pre>{}</pre>
POST <code>/user</code>	To create a user to maintain the user's data.	<pre>{ oldToken?: string; }</pre>	<pre>{ uid: string; }</pre>

How does our API conform to REST?

- Our request and response payloads maintain a uniform interface.
 - The requests follow HTTP request method, param and body formats. The responses have a standard format as mentioned in [Notes](#). This allows clients to handle all responses similarly, in a systematic way.
 - The API is structured in the format of handlers, routing requests from their headers, then their path, then methods, then validation and conversion to the database's data types, then performing database operations, and creating the response.
- Our API maintains the client-server design pattern. The request and response types are defined explicitly, so it can be separate from our database's data types. This allows scalability to support more clients without affecting server operations. Likewise, it allows maintenance and change in the server or database without affecting clients.

3. Our API is stateless. Each request contains all information necessary for completion and does not depend on the state of previous requests.

Milestone 6

Share with us some queries (at least 3) in your application that require database access. Provide the actual SQL queries you use (if you are using an ORM, find out the underlying query and provide both the ORM query and the underlying SQL query). Explain what the query is supposed to be doing.

View Application List

When a user first enters the application, they will see a list of applications that they have applied to. A HTTP GET request to the `/api/applications` endpoint to get the user's application list.

To find the current user's application list, the following Prisma code is used. The underlying SQL query generated is also included below.

```
const queriedApplications = await prisma.application.findMany({
  where: {
    userId: userId,
    role: {
      AND: [{ OR: roleTypeFilters }, { OR: roleTitleOrCompanyFilters }, { OR:
roleYearFilters }],
    },
  },
  select: {
    id: true,
    role: { include: { company: true } },
    applicationStages: { orderBy: { date: 'desc' }, take: 1 },
    _count: {
      select: {
        tasks: {
          where: { notificationDateTime: { lte: new Date() }, isDone: false },
        },
      },
    },
  },
});
```

```

prisma:query SELECT "public"."applications"."id",
"public"."applications"."role_id", "aggr_selection_0_Task"."_aggr_count_tasks"
FROM "public"."applications" LEFT JOIN (SELECT
"public"."tasks"."application_id", COUNT(*) AS "_aggr_count_tasks" FROM
"public"."tasks" WHERE ("public"."tasks"."notification_datetime" <= $1 AND
"public"."tasks"."is_done" = $2) GROUP BY "public"."tasks"."application_id") AS
"aggr_selection_0_Task" ON ("public"."applications"."id" =
"aggr_selection_0_Task"."application_id") WHERE
("public"."applications"."user_id" = $3 AND ("public"."applications"."id") IN
(SELECT "t0"."id" FROM "public"."applications" AS "t0" INNER JOIN
"public"."roles" AS "j0" ON ("j0"."id") = ("t0"."role_id") WHERE (1=1 AND
"t0"."id" IS NOT NULL))) OFFSET $4 /* traceparent=00-00-00-00 */
prisma:query SELECT "public"."roles"."id", "public"."roles"."company_id",
"public"."roles"."title", "public"."roles"."type", "public"."roles"."year",
"public"."roles"."is_verified", "public"."roles"."updated_at" FROM
"public"."roles" WHERE "public"."roles"."id" IN ($1,$2) OFFSET $3
prisma:query SELECT "public"."companies"."id", "public"."companies"."name",
"public"."companies"."company_url", "public"."companies"."is_verified",
"public"."companies"."updated_at" FROM "public"."companies" WHERE
"public"."companies"."id" IN ($1,$2) OFFSET $3
prisma:query SELECT "public"."application_stages"."id",
"public"."application_stages"."application_id",
"public"."application_stages"."type", "public"."application_stages"."date",
"public"."application_stages"."emoji_unicode_hex",
"public"."application_stages"."remark",
"public"."application_stages"."updated_at" FROM "public"."application_stages"
WHERE "public"."application_stages"."application_id" IN ($1,$2) ORDER BY
"public"."application_stages"."date" DESC OFFSET $3

```

The above query gets all applications where `user_id` equals to the current user's id. We also have a `where` clause to filter the applications based on search filters. Specifically, we allow filtering by role type, role title or role company or role year. Since our application list needs information from other tables like the application's role's company, the latest application's stage and also the count of tasks that are due and not done, the Prisma ORM performs more select queries to get the relevant information based on our schema relationships.

Create Application

Users will want to create a new application when they have applied to a new role at a company. They can fill up the application details in a form in the frontend and submit it. Upon submission, A `HTTP POST`

request is made to `/api/applications` endpoint with the request body containing all the relevant data.

To create a new application, the following Prisma code is used. The underlying SQL query generated is also included below.

```
const newApplication = await prisma.application.create({
  data: {
    roleId: applicationPostData.roleId,
    userId: userId,
    applicationStages: {
      create: {
        type: ApplicationStageType.APPLIED,
        date: applicationPostData.applicationDate,
      },
    },
  },
  select: {
    id: true,
    role: {
      select: {
        id: true,
        title: true,
        type: true,
        year: true,
        isVerified: true,
        company: { select: { id: true, name: true, companyUrl: true } },
      },
    },
  },
});
```

```
prisma:query BEGIN
prisma:query INSERT INTO "public"."applications"
("user_id","role_id","updated_at") VALUES ($1,$2,$3) RETURNING
"public"."applications"."id" /* traceparent=00-00-00-00 */
prisma:query INSERT INTO "public"."application_stages"
("application_id","type","date","updated_at") VALUES ($1,$2,$3,$4) RETURNING
"public"."application_stages"."id" /* traceparent=00-00-00-00 */
prisma:query SELECT "public"."applications"."id",
"public"."applications"."role_id" FROM "public"."applications" WHERE
```

```

"public"."applications"."id" = $1 LIMIT $2 OFFSET $3 /* traceparent=00-00-00-00
*/
prisma:query SELECT "public"."roles"."id", "public"."roles"."title",
"public"."roles"."type", "public"."roles"."year",
"public"."roles"."is_verified", "public"."roles"."company_id" FROM
"public"."roles" WHERE "public"."roles"."id" IN ($1) OFFSET $2
prisma:query SELECT "public"."companies"."id", "public"."companies"."name",
"public"."companies"."company_url" FROM "public"."companies" WHERE
"public"."companies"."id" IN ($1) OFFSET $2
prisma:query COMMIT

```

After the data in the request body have passed data validations, the data is passed into a Prisma `create` method. The request body contains the `roleId` and `applicationDate`. Using these, a new application is created. Note that a application stage is also created at the same time as they application as they have applied to this role at the specified `applicationDate`. After a successful `INSERT` operation, the relevant fields required for the response body are selected and returned using `SELECT` operations. The whole SQL query is performed in a database transaction where if any of the `INSERT` or `SELECT` queries fail, none of the operations will be committed making the entire create application flow atomic.

Company Autocomplete

When the user creates an application, they would be required to fill in a form. One of the form fields includes the company name. The company name supports autocomplete as the user types in the text field. Multiple HTTP `GET` requests are made to the `/api/companies` endpoint. The search filters (the words that the user has already typed) are sent as query params in the `GET` request.

To search for the matching companies, the following Prisma code is used. The underlying SQL query generated is also included below.

```

const companies: CompanyListData[] = await prisma.company.findMany({
  where: {
    AND: [
      {
        OR: [
          { isVerified: true },
          {
            contributions: {
              some: { contributorId: userId },
            }
          }
        ]
      }
    ]
  }
})

```

```

    },
    },
  ],
},
{ OR: companyNamesFilters },
],
},
take: 5,
orderBy: {
  name: 'asc',
},
select: { id: true, name: true, companyUrl: true },
});

```

```

prisma:query SELECT "public"."companies"."id", "public"."companies"."name",
"public"."companies"."company_url" FROM "public"."companies" WHERE
(("public"."companies"."is_verified" = $1 OR ("public"."companies"."id") IN
(SELECT "t0"."id" FROM "public"."companies" AS "t0" INNER JOIN
"public"."company_contributions" AS "j0" ON ("j0"."company_id") = ("t0"."id")
WHERE ("j0"."contributor_id" = $2 AND "t0"."id" IS NOT NULL))) AND
"public"."companies"."name" ILIKE $3) ORDER BY "public"."companies"."name" ASC
LIMIT $4 OFFSET $5 /* traceparent=00-00-00-00 */

```

After the data in the query params have passed data validations, it is passed into the Prisma `findMany` query as a filter. Prisma performs `SELECT` queries with `ILIKE` clause to find all the companies that matches the string that the user had typed. Since there can be a lot of companies that can be a potential match, we only return 5 of them in ascending order. We also have other filters applied to this query as we want only verified companies or non-verified companies created by the current user in the result. All autocomplete `GET` endpoints in Caza are debounced in the frontend to minimise the number of `GET` request made to the server.

Milestone 7

Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly. Include an image of the icon and a screenshot of the splash screen in your write-up. If you did not implement a splash screen, justify your decision with a short paragraph. Add your application to the home screen to make sure that they are working properly. Make sure at least Safari on iOS and Chrome on Android are supported.



Figure: Application icon

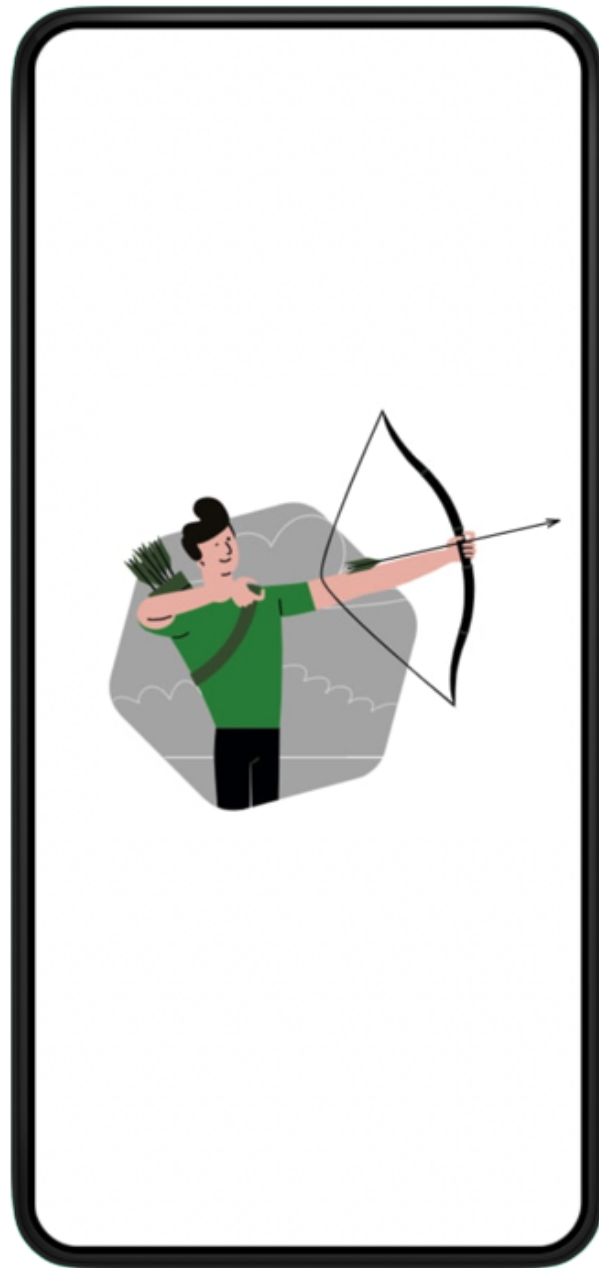
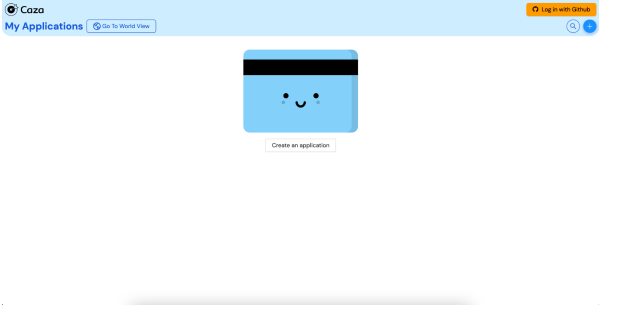
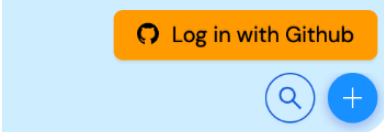
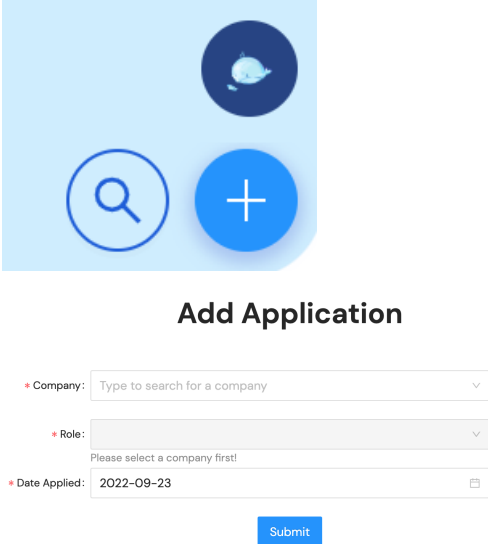



Figure: Splash screen

Milestone 8

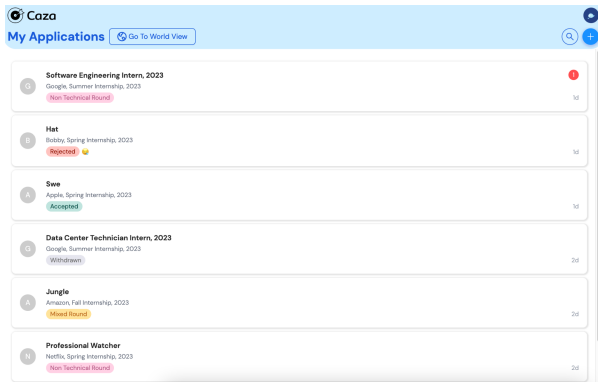
Style different UI components within your application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application. Choose one of the CSS methodologies (or others if you know of them) and implement it in your application. Justify your choice of methodology.

UI Component	Why it is the best
<p>Prompt for first time users to create application</p> 	<p>When users enter Caza for the first time, they are prompted to create an application. This lets them know the main action and start using Caza for tracking applications immediately.</p>
<p>Prominent login button</p> 	<p>When users are not logged in, the log in button is prominently displayed to encourage users to log in.</p>
<p>Primary action buttons</p> 	<p>The primary action button is highlighted to let users know the main action of each page/component. Examples of primary action buttons include</p> <ul style="list-style-type: none"> Buttons to add new application/ stages/ tasks Form submit/save buttons
<p>Secondary action buttons</p> 	<p>The secondary action buttons are outlined to show they are still actionable, but are not the main focus.</p>

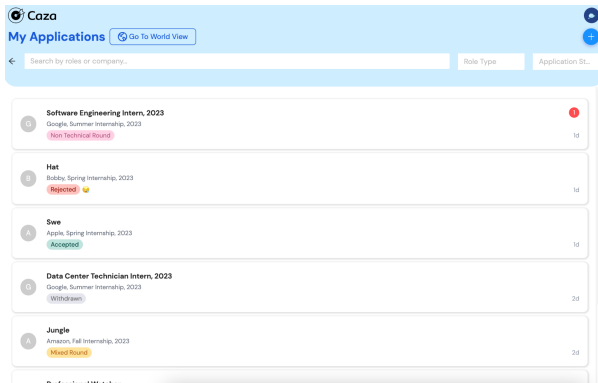


Search bar

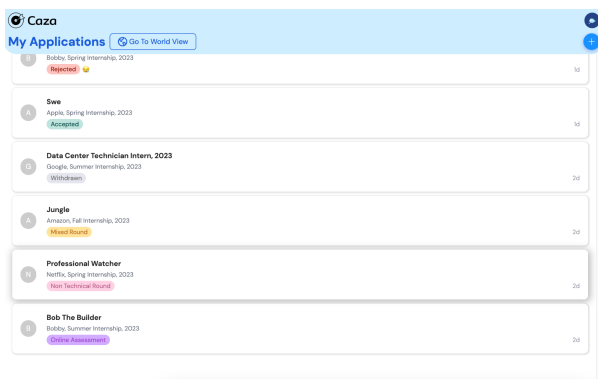
Not in use



In use



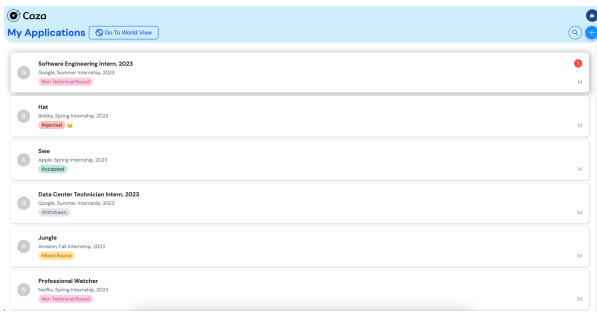
Minimized when scrolling down



Hover effects

The search bar is closed by default to increase screen real estate. Users can click the search icon to see the search filters. When they scroll down, it is automatically minimized to increase screen space.

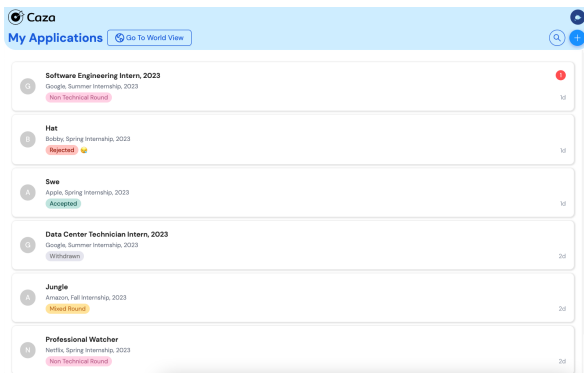
When on desktop, hovering over actionable



components will show effects to let users know that the component is interactive. Examples include

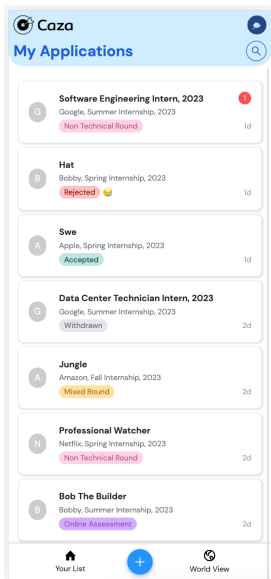
- Hovering over cards display a darker background shadow to show they are clickable
- Hovering over clickable elements changes the cursor to a pointer to show they are clickable

Responsive layout for different screen sizes



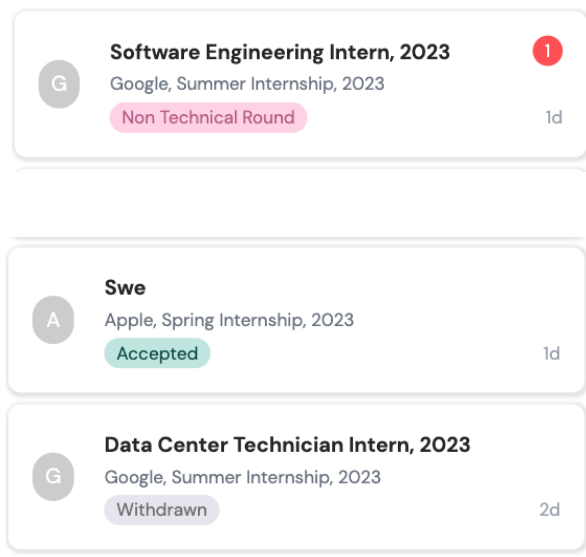
When screen size changes, elements are resized to show content without any cut-offs. This is useful as users can use the app on both desktop and mobile.

The navigation buttons are moved to the bottom for mobile to make it easier to reach.



Information is displayed in the most intuitive manner to users

Instead of displaying the latest stage date, we display the number of days since the latest stage as users would be more interested to track how long it has been since they last received an update

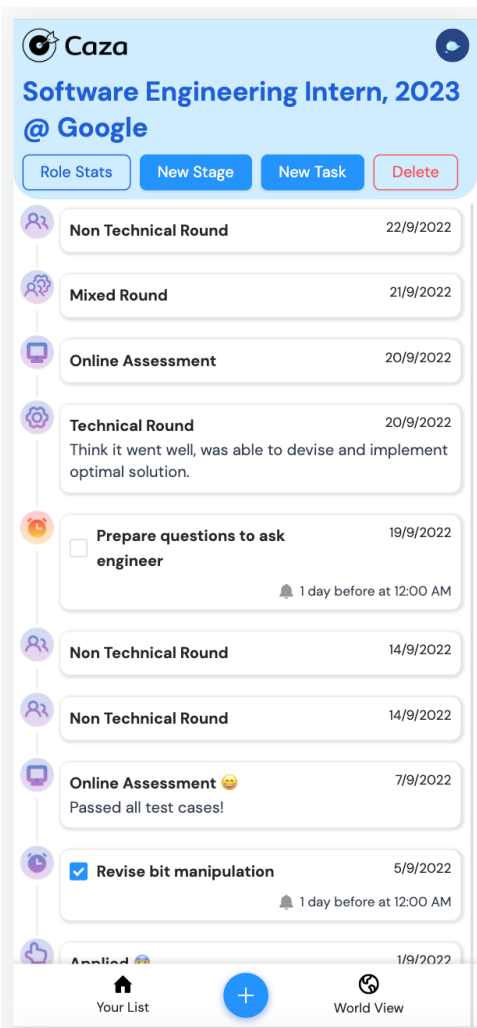


on their application status.

The red notification badge shows the number of notifications on alert for that application.

Applications with alerts are displayed higher in the list in descending order of alerts to bring them to the user's immediate attention.

Applications with a more recent latest stages are also displayed higher in the list as they are likely to be more active and more relevant to the user. This reduces the need for scrolling.



There is a shortcut button to access global role stats for that specific role so users do not need to find it from the world list.

Application stages and tasks are displayed in a timeline

The later stages and tasks are displayed on top as they are more relevant to the users. This reduces scrolling.

Tasks that have not been marked as done are prominently displayed with an orange icon to immediately draw the user's attention.

Tasks that are currently being displayed as notification alerts are highlighted in red to show them to the users

Users can mark tasks as done quickly from the application page to reduce hassle of opening and saving an edit form.



☐ Do online assessment

24/9/2022

1 day before at 9:00 AM



Caza

World View



F

Software Engineer Intern

Facebook, Summer Internship, 2023

1

Applied

Total applications: 1

F

Enterprise Engineer Intern

Facebook, Summer Internship, 2023

1

Applied

Total applications: 1

G

Software Engineering Intern, 2023

Google, Summer Internship, 2023

1

1

1

1

4

Applied Online Assessment

Non Technical Round Accepted Rejected

Total applications: 8



Your List



World View

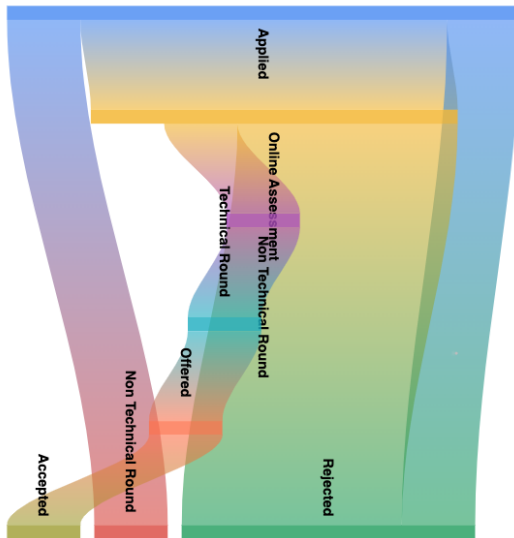
The distribution of people at each stage for each role is shown in a bar chart. This makes it easy for users to visualize the status of the role.



Caza



Software Engineering Intern, 2023 @ Google



Online Assessment to Rejected:

Based on 3 experiences, progression between these stages took an average of 3.7 days



Your List



World View

Upon clicking on a card in the world view list, users can see a detailed sankey. This shows the progression of people across different stages, so useBlock Element Modified rs can better gauge their performance.



World View

Log in with Github



Restricted Content

Log in with Github to view our community-sourced status aggregates to figure out where you stand among other applicants!



Your List



World View

Non verified users will see blurred out cards when they navigate to the world view. This encourages them to log in to find out more about what the world view offers.

CSS Methodology

The CSS Methodology used is based on Block Element Modifier (BEM). We used a UI library, Ant Design, to quickly get consistent styling across components. The style is arranged in blocks for things like lists and buttons. Each block also has elements like list containers. There are modifiers like `disabled` and `danger` to quickly change the style for certain themes. BEM is suitable as it is modular so styles applied to one component would not affect other components. This makes it easy to add components without breaking others. Many common components like form inputs are reused throughout the app so using BEM allows consistency.

Milestone 9

Set up HTTPS for your application, and also redirect users to the https:// version if the user tries to access your site via http://. HTTPS doesn't automatically make your end-to-end communication secure. List 3 best practices for adopting HTTPS for your application.

Strike a balance between key security and performance

A balance between key security and performance should be found. HTTPS uses Transport Layer Security (TLS) to encrypt HTTP content. As part of the TLS handshake, asymmetric encryption is used. Since encryption strength is directly related to the key size, one should make sure that the key size is large enough to be secure. However, it must be taken into account having larger keys like a 4096-bit key will require more compute power during the TLS handshake and performance will be affected. When in doubt, a 2048-bit RSA key is typically ideal.

Keep private keys well-protected

The private keys of the server should be well-protected. If the private keys are compromised, an example of an attack would be that an attacker could forge a site and force a way to redirect the user to the forged site (e.g. DNS spoofing) and thereafter, using the stolen private keys to pass the unilateral authentication during the TLS handshake, successfully tricking the client that the forged site is authentic. One should generate their own private keys on a secure environment, preferably on the server of deployment. Once a private key has been compromised or suspected to be

compromised, all certificates corresponding to the compromised key should be revoked by submitting a revocation request to the Certificate Authority (CA).

Renew certificates in a timely manner

SSL certificates come with an expiration date after which they will be invalid and can no longer be used to encrypt end-to-end communication. If the client proceeds with HTTP communication, the application layer data will be unencrypted and becomes vulnerable to man-in-the-middle attacks. This is a rather common issue that [even large corporations can overlook sometimes](#). As good practice, it is recommended to renew a certificate at least 15 days prior to the expiration date to leave ample time for testing and rolling back in case of any issues with the new certificate. Most CAs support auto-renewals of certificates close to the expiration date, which helps to make sure that a new certificate is installed before the previous expires.

Milestone 10

Implement and briefly describe the offline functionality of your application. Explain why the offline functionality of your application fits users' expectations. Implement and explain how you will keep your client synchronised with the server if your application is being used offline. Elaborate on the cases you have taken into consideration and how they will be handled.

On Caza, users are able to view their applications even when they are offline. This allows users to keep track of their application statuses, act upon any upcoming deadlines that companies have given them as well as complete tasks that they have created for themselves. This is done by precaching data for all of the user's applications whenever the user logs in or makes a change to applications.

Our client will always be synchronized with the server because the offline use cases in Caza do not mutate any state. We chose to go ahead with this approach as for application stages, there are constraints to be enforced with regards to the ordering of stages which are checked in the server-side code. This can quickly become complex and we do not wish to compute this client-side.

In addition, as the world view contains data that is aggregated from all Caza users, it would be more logical for users to only be able to visit the world view page so that they can view the latest statistics.

Milestone 11

Compare the advantages and disadvantages of token-based authentication against session-based authentication. Justify why your choice of authentication scheme is the best for your application.

Our authentication solution uses elements of both token-based and session-based strategies. We use Firebase's Authentication-as-a-Service' (AaaS) to create sessions for our users on Firebase databases and validate login credentials. At the same time, our backend uses token-based authentication using Firebase provided JSON web tokens (JWT¹) for each user.

Overview of Token-Based

The authentication server responds to a login request by validating the user credentials and generating a signed², short-lived authorization file (a token) for the client to store locally. If the token is about to expire, the client must request for a new token using the still valid token. If the client has no valid, non-expired tokens left, it will have to re-authenticate with a fresh log-in request.

As it only requires the public key and the cryptographic method to validate signed payloads, this operation can be completed without any lookups, minimizing storage requirements and operation time for each request. As a result, this is a scalable solution often used by mobile applications.

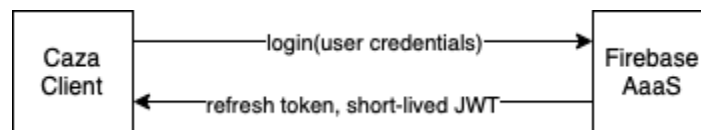
¹ JWT, or json web token is a well known open standard used to carry signed or encrypted payloads in JSON format.

² Signing puts a tamper-proof 'seal' on the data. This means if anyone attempts to modify a signed payload, the sign will no longer be valid. Often, public-private key pairs are used, which means a private key signs the data, while a public key is distributed so that anyone can verify the signer's authority on the data.

Overview of Session-Based

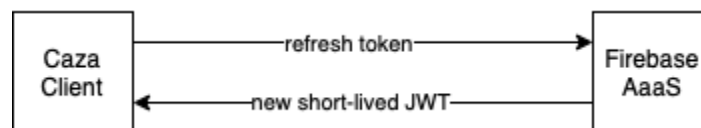
With session-based authentication, the authentication server authorizes the login attempt and creates a session to store in its database. The client is returned a cookie containing the session ID. On subsequent requests, the client will provide the cookie and the backend service must look up the session ID to verify that the session is still valid. Compared to token-based methods, this can also allow administrators to log a user out by ending a session.

Caza Implementation

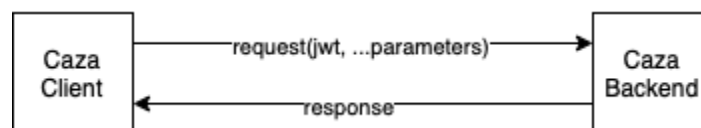


Whenever a user opens a fresh instance of our app (on a new browser, device or with the cache clearer), the browser will be automatically fingerprinted and a unique, stable UID will be generated by the Firebase service. They will then be logged in.

If the user is logging in with Github, they will use the OAuth standard to log in. This means a Github URL is provided to Caza to allow users to sign in using Github credentials, and their user data will be provided back to Caza. After this step, they will be automatically redirected back to Caza. This step is also handled by Github's OAuth implementation and the Firebase AaaS.



The refresh token will be used to get new short-lived JWT as and when needed by the client. These tokens are used to authenticate clients when making requests to the Caza backend.



The Caza backend servers are aware of the firebase public key and our firebase project ID, and these details are used to validate the JWT provided in each request. As these do not require any

database lookup, it is a very fast and lightweight authentication mechanism that plays to the strengths of our serverless backend functions.

Milestone 12

Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. Lastly, list down some (at least 5) of the mobile site design principles and which pages/screens demonstrate them.

Frontend Framework

React, Angular and Vue were our considerations for the frontend framework. We put heavy thought into the bundle sizes of the frontend frameworks because we wanted the application to have a good user experience even with poor internet connections. Having smaller bundle sizes for the framework means faster loading times for the user. Angular has the largest bundle size among all the 3 frameworks and therefore, we were hesitant to use it. React and Vue have almost similar bundle sizes with React having a slightly larger size. We decided to use React in the end to leverage on the React ecosystem for some of the functionalities that our application needed that would otherwise be hard to implement in Vue. All these frameworks allow us to create Single Page Applications where the JavaScript sent to the client makes the website feel like a native application due to the interactivity elements. Of course, we also considered how familiar all of us were with the frameworks, and we had a lot more experience with React altogether compared to Angular and Vue.

Backend Framework

We considered traditional backend frameworks like Django and Rails. Django and Rails are both very popular and fully-featured frameworks that have a thriving community. However, our main criteria in choosing the backend framework was the ability to have server-side rendering. The reason we believed that server-side rendering was important was due to that fact that mobile phones might not be as powerful as personal computers and having the JavaScript and CSS sent over to the client and having the HTML generated by the frontend framework would be a expensive operation that might be a problem on low-end devices. We researched on how we might go about implementing server-side rendering on the above frameworks and found out that it was rather complicated and

hard to understand. We then came across Next.js which is a full-stack framework that supports server-side rendering out of the box and uses an improved version of React as the frontend framework. We decided to use Next.js and take advantage of the server side rendering capabilities to improve our load times in mobile phones. However, we did not manage to implement any server-side rendered pages in our current iteration but our choice of framework will allow us to migrate to server-side rendered pages easily later on.

UI Framework

After looking through various UI libraries like Material UI, Bulma CSS, Ant Design, etc. We decided that Ant design was the best choice because it was most platform agnostic. For example, Material UI follows the Material design principle by Google which looks good on Android phones but looks out of place on iOS devices. Therefore, our goal was to choose a UI framework that looks great on both platforms and something that also provides the mobile look and feel. Therefore, for most of our components, we created custom CSS using TailwindCSS and used Ant Design when the components looked like they would go well with Android and iOS.

5 Mobile site design principles

Making search visible

Users will start on the application page and when they are looking for a specific application to view or update they might turn to search. Having the search feature visible in one of our mobile design principles to improve the user experience. In our application, the search button and the relevant filters are always located at the top of the device which means users will be instantly familiar as most native applications like Instagram, TikTok, etc all have their search buttons at the top.

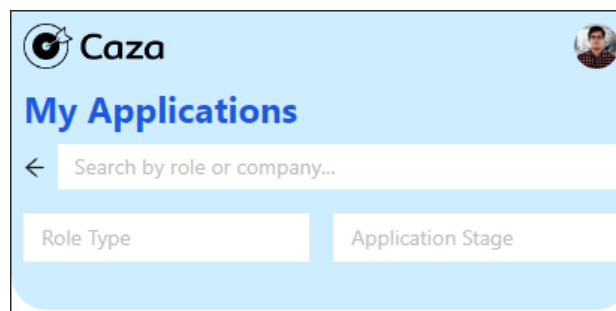


Figure: Expanded search Filters for the My Applications page

The search toggle for our application is always at the top of the screen where the user can click to expand it. It is always at the same location at every page and we always use the same icon so that the user is familiar with our search feature and it always visible.

Designing Finger-Friendly Touch Targets

Mobile users usually use fingers to navigate around their webpages (the rest of the time using stylus pens). Most people cannot achieve the same accuracy when touching a screen using their fingers as compared to a mouse or a stylus like the Apple Pencil. Having interactivity elements that have appropriate sizes are key to having a non-frustrating user experience. Therefore, we made sure that all UI elements like buttons, text fields and cards are appropriately sized for the end user.

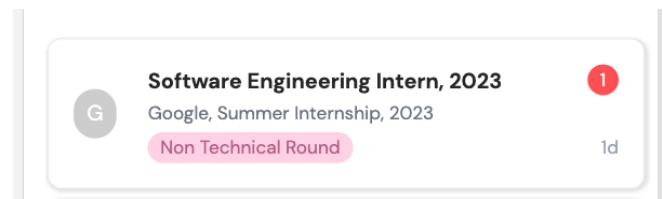


Figure: Full width cards that can be tapped

Keeping the user in a single browser window

Mobile users prefer to stay in a single tab because it is rather difficult to switch tabs when using mobile versions of popular browsers such as Chrome and Safari. It requires several steps just to switch tabs as compared to desktop versions. Therefore, not opening new tabs for features was key to making the user experience as seamless as possible.

No pinch to zoom

Pinch to zoom is one key feature in mobile phones. However, requiring users to pinch to zoom to see certain UI elements is not a good design principle as it is extremely frustrating for the user to have to zoom in and zoom out just to view the application. Ideally, all UI elements should be displayed with good font sizes so that the user can see everything at a single glance.

Optimize site for mobile

Mobile users do not have much real estate when it comes to screen size. Most devices have screen sizes smaller than 7 inches which means that the right content has to be displayed at the right time. Furthermore, most mobile users use their devices with one hand. This means that naturally, the things at the bottom half of the screen are easier to reach than the top elements. Therefore, placing frequently used items like the navigation bar at the bottom and less frequently used items like the logout and the search functionalities at the top will improve user experience as less handwork is needed by the user.

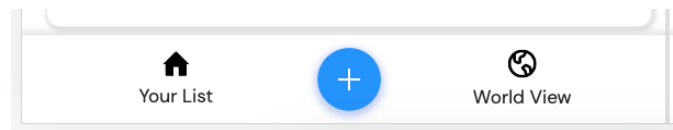


Figure: Large navigation buttons that are at the bottom so it is easier for the fingers to reach

Milestone 13

Describe 3 common workflows within your application. Explain why those workflows were chosen over alternatives with regards to improving the user's overall experience with your application.

First Workflow: First timer, non-verified user tracking their own applications

1. A user first enters Caza. They are on the home page which is also the applications list page. They are not logged in.
2. The user sees a blank space and a prompt to create a new application.
3. The user clicks the "+" or "Create an application" button to create a new application to track.
4. The user searches for the company of the role they want to track.
 - a. If it shows up in the dropdown, they select it.
 - b. If not found, they create a new company.
5. The user searches for the role they want to track.
 - a. If it shows up in the dropdown, they select it.
 - b. If not found, they create a new role.
6. The user selects their application date as they are likely to start tracking applications only after they apply for it.
 - a. The user sees that the application date is set to today's date. That is correct as they just applied right before using Caza. So the user does not need to pick a date.

- b. OR the user picks their application date.
7. The user saves the application. They are directed to the newly created application.
8. The user creates new tasks to remind themselves to prepare for future application stages.

Reason for first workflow

- When a user first enters Caza, they can immediately track applications without logging in. This reduces inertia of signing in and they can try the app and feel its benefits immediately.
- When users first use the app, the main desire is to track applications. They are immediately prompted to create an application to know how to start the journey.
- When users create an application, they can search for existing companies and roles without having to create companies and roles for every single application. As our target users are Computing students, most of them would be applying to similar companies and roles so providing that through autocomplete makes things easy.
- Default values in forms are set to make the tracking process as easy and fast as possible.

Second workflow: Non-verified user transitioning to a verified user

1. The user has tracked a number of applications on his phone.
 - a. Scenario A: The user navigates to the world page several times and sees it is blurred out for users who are not logged in. The user got curious.
 - b. Scenario B: The user started tracking applications on their phone. However, their phone ran out of battery when they received some updates to their application. They think it would be convenient if they could update their tracker through either a phone or desktop.
2. The user logs in to Github from the device they have been using. The data is automatically transferred to their account.
3. The user continues tracking their applications.
4. The user browses through the world view page.

Reason for second workflow

1. The user can transition from a non-verified user to a verified user (one who is logged in). This allows users to start trying the app quickly, but also have the flexibility of tracking their applications across different devices once they feel the need to.
2. When users log in for the first time from the device they have been using Caza on, the data is automatically associated with their account. This is convenient as users do not need to recreate all their applications again.
3. Users can log in using their Github accounts. This is convenient as they do not need to remember another password just to use Caza, or go through a cumbersome sign up process. Github is chosen as Computer Science students are the target users and it is likely that all of them have Github accounts.

Third workflow: Returning user updating their applications

1. The user receives updates on their applications. They open the app to update the respective applications.
2. The user notices some red notification badges on applications at the top of their list.
3. The user is reminded of tasks and clicks on the application to check what has to be done.
4. One of the tasks is complete an online assessment stage. The user completes it.
5. After that, the user opens the app and searches for the application. When scrolling down, the search bar is minimized.
6. The user finds the application, opens it up and marks the task as done.
7. The user updates the online assessment stage as well. The user adds an emoji to represent how they think they performed and adds remarks on the technical questions they faced and solutions they used so they can revise in future, or as a note for them to reflect on when reviewing applications.

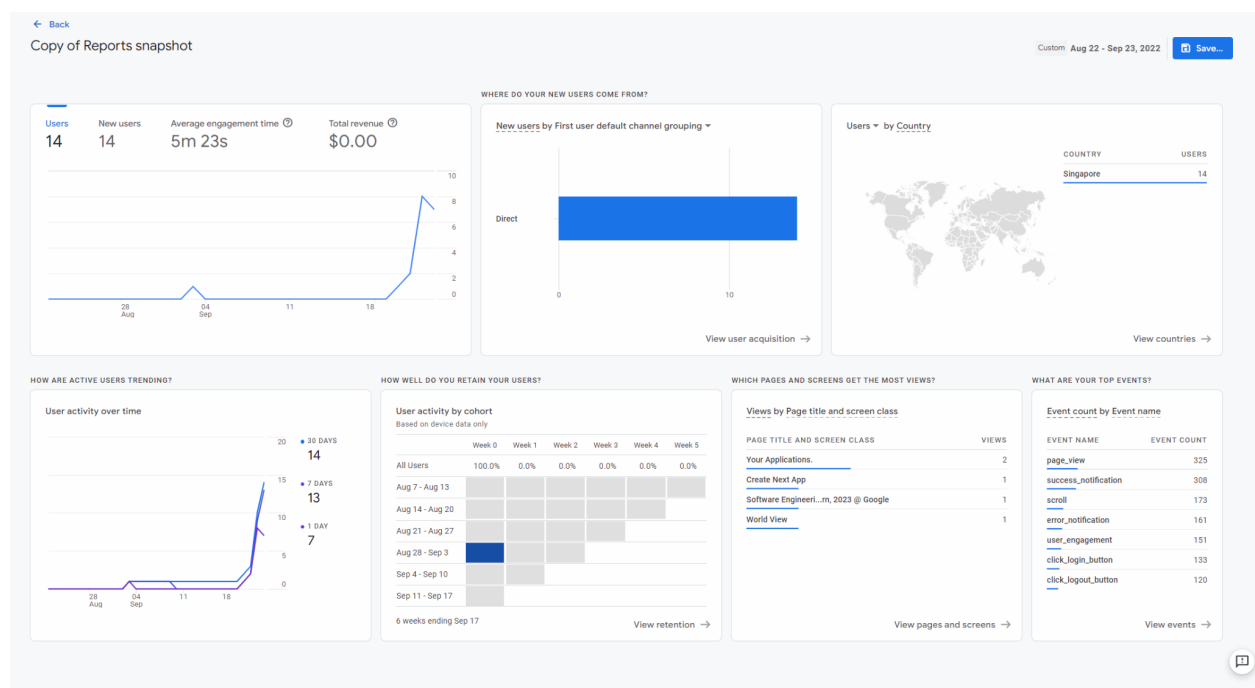
Reason for third workflow

1. Applications with task notifications that are not marked as done are placed at the top of the list and displayed prominently with a badge count. This immediately reminds users of urgent tasks so they can settle them.
2. When scrolling down, the open search bar is minimized. It reappears when users scroll up. This increases screen space for the users.

3. The user can mark tasks as done from the application, without having to click into the task to edit. The user is more likely to regularly and quickly mark tasks as done without editing the other fields. Allowing the user mark from the application page removes the hassle of opening a modal and clicking save.
4. The user can add emojis and remarks to stages to record their thoughts on their application progress. This is useful for revising questions to be used in other technical assessments and interviews.

Milestone 14

Embed Google Analytics or equivalent alternatives in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before the submission deadline as updates for Google Analytics are reported once per day.



Milestone 15



Achieve a score of at least 8/9 for the Progressive Web App category on mobile (automated checks only) and include the Lighthouse HTML report in your repository.



PWA

These checks validate the aspects of a Progressive Web App. [Learn more.](#)

INSTALLABLE

-  Web app manifest and service worker meet the installability requirements 

PWA OPTIMIZED

-  Registers a service worker that controls page and `start_url` 
-  Configured for a custom splash screen 
-  Sets a theme color for the address bar. 
-  Content is sized correctly for the viewport 
-  Has a `<meta name="viewport">` tag with `width` or `initial-scale` 
-  Provides a valid `apple-touch-icon` 
-  Manifest has a maskable icon 

Milestone 16

Identify and integrate with social network(s) containing users in your target audience. State the social plugins you have used. Explain your choice of social network(s) and plugins. (Optional)

We used Github as our social network and OAuth solution. As our target users are computing students, they are likely to be Github users as well, and this integration greatly improves their experience. Signing in with Github reduces the frictions involved in signing up, remembering passwords and simplifies regular use.