Of course. Here is a long, detailed transcript of a developer team meeting.

**Meeting Transcript: Project Sentinel - Weekly Technical Sync**

* **Date:** July 30, 2025
* **Time:** 10:00 AM
* **Location:** Tel Aviv Office (Meeting Room "Kinneret") & Zoom
* **Project:** Sentinel (Customer Data Platform)
* **Attendees:**
  + Maya Cohen (Engineering Manager)
  + David Levy (Product Manager)
  + Liam Barak (Senior Backend Engineer)
  + Chloe Shapiro (Senior Frontend Engineer)
  + Ben Amir (DevOps Engineer)
  + Noa Mizrahi (Junior Developer)

**[10:01 AM]**

**Maya Cohen:** Okay everyone, good morning. Looks like everyone's here, on Zoom and in the room. Let's kick off. Yalla. First on the agenda is our quick weekly round-robin. What you worked on, what you're doing today, and any blockers. David, you can skip the formal update since you'll be leading the feature discussion. Chloe, let's start with you.

**Chloe Shapiro:** Sure. Good morning. So yesterday, I finished up the refactor of the main dashboard's state management. Moved everything from context-based state to a centralized Redux Toolkit store. The performance improvement is noticeable, especially on the larger data tables. Today, I'm starting to spec out the UI for the new "User Audit Log" feature. My main blocker is that I need the final data schema for the log entries from the backend. Liam, I think that's on your plate. I need to know what fields will be available—like actor, event\_type, timestamp, target\_entity, etc.—so I can build the table and filters correctly.

**Liam Barak:** Yep, morning. I've got that. It's in progress. The initial schema is drafted in Confluence, I'll send you the link. I was finalizing the choices for the event\_type enum yesterday. I should have the initial GET endpoint for you on the staging environment by end of day tomorrow. Yesterday, I also deployed a hotfix for the CSV export timeout issue, JIRA-721. It seems stable so far. Today, my main focus is on the architecture for the new file upload feature David is about to discuss.

**Maya Cohen:** Great, thanks Liam. Ben, how are things in DevOps-land?

**Ben Amir:** Hey. Yesterday was mostly spent wrestling with Terraform for the new analytics cluster. The provider had some breaking changes in the latest version. It’s sorted now. I also tweaked our CI/CD pipeline to run the end-to-end tests in parallel, which cut down the main branch deployment time by about six minutes. Today, I'm monitoring the production environment after Liam's hotfix and preparing for our planned database migration from PostgreSQL 14 to 15, which is scheduled for next weekend. It’s a big one, so I'm running multiple dry-runs on a cloned staging environment. No blockers, but I need everyone to be available for a final check next Friday.

**Maya Cohen:** Noted. We’ll schedule that. Noa, how about you?

**Noa Mizrahi:** Good morning. Yesterday, I finished my first "good first issue" ticket, JIRA-715, which was fixing the CSS on the login button for Safari. It's been merged after Chloe’s review, thank you for that. Today, I'm picking up a new ticket to add validation to the "Create New Segment" form. I was just reading the requirements on Jira. I might have some questions later about the best way to handle the asynchronous validation against the API.

**Maya Cohen:** Perfect, that's a great next step. Feel free to pair with Chloe or Liam on that. Okay, thanks everyone. That brings us to our main topic. David, the floor is yours to introduce the "Custom Report Upload" feature.

**[10:08 AM]**

**David Levy:** Thanks, Maya. So, team, this is a big one that our enterprise customers have been asking for. The user story is this: "As a marketing manager, I want to upload my own custom CSV or XLSX reports into the Sentinel platform, so I can merge our internal sales data with the customer behavior data that Sentinel tracks."

**David Levy:** Right now, all data gets in via our integrations. This feature gives users direct control. The MVP (Minimum Viable Product) scope is as follows:

1. A new page in the "Data Sources" section with an upload interface.
2. The user can drag-and-drop or select a CSV or Excel file.
3. The system will parse the file, and the user will have to map the columns from their file to the fields in our system (e.g., their "Customer ID" column maps to our user\_id, their "Sale Amount" maps to our transaction\_value).
4. Once mapped, we ingest the data and it becomes available in the segmentation and analytics tools.

**David Levy:** The business impact is huge. It moves us from a passive data aggregator to an active workspace for our customers. It's a key differentiator against our main competitor. Any initial product questions before we dive into the technical implementation?

**Chloe Shapiro:** From a UX perspective, the column mapping is the most complex part. We'll need a very intuitive interface for that. David, can you share the wireframes from Figma? I'll need to see how we handle errors, like if they upload a file with a missing required column or mismatched data types.

**David Levy:** Absolutely. I'll post the link in the Slack channel. It has a full user flow, including error states.

**[10:15 AM]**

**Maya Cohen:** Okay, let's get into the technicals. Liam, you mentioned you started thinking about the architecture. What are your initial thoughts on the backend?

**Liam Barak:** Right. This is a multi-step process. First, the file upload itself. We absolutely cannot have users uploading potentially massive files directly to our API server. A 100-megabyte Excel file would block a Node.js worker thread and kill performance.

**Ben Amir:** I agree 100%. Don't send it to the API.

**Liam Barak:** Exactly. So, my proposal is a flow using pre-signed URLs for AWS S3. It works like this:

1. Chloe's frontend code makes a request to a new backend endpoint, let’s call it /api/v1/uploads/presign. It sends the filename and file type.
2. The backend generates a temporary, secure URL (a pre-signed URL) that grants write-only permission to a specific key in a dedicated S3 "ingestion" bucket. The URL is only valid for, say, five minutes.
3. The backend returns this URL to the frontend.
4. Chloe's code then uses that URL to upload the file directly from the user's browser to S3. This way, the heavy lifting is offloaded to S3 and our servers are untouched.

**Ben Amir:** Tachles, that's the standard, correct way to do it. From a DevOps perspective, I'll need to create the S3 bucket, configure its lifecycle policies to automatically delete old, unprocessed files, and set up the IAM role and policies that allow our backend service to generate these pre-signed URLs. I also need to configure CORS on the bucket so browsers don't block the upload request.

**Chloe Shapiro:** That works for me. A pre-signed URL flow is something I've implemented before. It's clean. So, my request to the backend will be a POST to /uploads/presign, and the response will be a JSON object containing the uploadUrl and any required headers?

**Liam Barak:** Correct. After the upload is complete, we need to process the file. This is the second major part. We can't parse a huge file in a synchronous web request.

**Ben Amir:** Lambda function.

**Liam Barak:** Exactly. We'll configure an S3 event notification. When a new object is created in the ingestion/ path of the bucket, it automatically triggers a Lambda function. This function's job is to:

1. Read the uploaded file from S3.
2. Parse the CSV or Excel data. This is the tricky part. We'll need a robust library that can handle different encodings and delimiters.
3. Perform initial validation. Does it have a header row? Does it have at least two columns?
4. Once it's parsed the headers, it needs to store this metadata somewhere so Chloe's mapping UI can fetch it. I propose a new uploads table in our PostgreSQL database. The Lambda would create a new record with a status of pending\_mapping, and store the extracted column headers as a JSONB field.

**Noa Mizrahi:** Sorry to interrupt, can I ask a quick question? What's JSONB?

**Liam Barak:** Good question, Noa. It's a data type in PostgreSQL for storing JSON data. The 'B' stands for binary. It's indexed, which means we can query the contents of the JSON structure very efficiently, which will be useful later.

**Chloe Shapiro:** Okay, so after my frontend confirms the S3 upload is successful, I'll then need to poll another endpoint, maybe /api/v1/uploads/{upload\_id}/status, until the status changes from processing to pending\_mapping. Once it is, I can fetch /api/v1/uploads/{upload\_id}/columns to get the headers and render the mapping interface for the user.

**Liam Barak:** Precisely. The user then submits the mapping. That's a POST to something like /api/v1/uploads/{upload\_id}/map. The request body will contain the mapping object, like {"user\_email\_column": "Customer Email", "transaction\_id\_column": "Sale ID"}.

**Maya Cohen:** And the final step, the actual data ingestion?

**Liam Barak:** That will be another asynchronous process. Once the mapping is submitted, we'll push a job to our BullMQ queue. A dedicated background worker will pick up this job. The job will read the file from S3 again, apply the user's mapping, validate each row, and then insert the data into the appropriate tables. This needs to be a robust, resumable process. If it fails halfway through a million-row file, we need to be able to resume it without creating duplicates. This is the most complex part of the backend work.

**David Levy:** This sounds solid. It covers the user flow and seems robust. How long are we estimating for this? This feels like a full-sprint effort for at least two people.

**Maya Cohen:** I agree. It's a mini-epic. Liam, you'll lead the backend work. Chloe, you'll handle the frontend. Noa, I want you to pair-program with Chloe on the upload component; it's a great opportunity to learn about these flows. Ben, you'll support with all the needed infrastructure. We'll break this down into smaller user stories in the sprint planning meeting tomorrow.

**[10:35 AM]**

**Maya Cohen:** Okay, sababa. Let's move on. Next on the agenda is a quick review of a recurring bug: JIRA-650, the intermittent "Invalid Token" error on API calls. Ben, I know you and Liam were looking at the logs. Any updates?

**Ben Amir:** Yeah, we spent some time on this. We're seeing spikes of 401 Unauthorized errors in Grafana, but they're not tied to any specific deployment or user action. They seem random. My initial theory was a clock skew issue between our Kubernetes nodes, causing JWT validation to fail, but I've verified all nodes are in sync with the NTP server. So that's not it.

**Liam Barak:** My current hypothesis is that it's a race condition related to our token refresh mechanism. The access token lives for 15 minutes. The frontend has an interceptor that's supposed to use the refresh token to get a new access token just before it expires. However, if multiple API requests are fired in parallel right at the 14:59 mark, the frontend might fire multiple requests to the /auth/refresh endpoint simultaneously. The first one succeeds, gets a new token, and invalidates the old refresh token. The other parallel refresh attempts then fail because they're using an already-used refresh token.

**Chloe Shapiro:** That actually makes a lot of sense. I've seen that pattern before. In the network tab, I sometimes see a flurry of requests, and one or two of them will fail with a 401. So, on the frontend, I need to implement a mechanism to ensure only one refresh request is active at any given time. While a refresh is in progress, any other outgoing API calls should be queued and only sent after the new token has been received.

**Liam Barak:** And on the backend, I can add a small grace period, maybe 5-10 seconds, where the old refresh token is still considered valid to handle this exact race condition. It's a defense-in-depth approach.

**Maya Cohen:** Okay, that sounds like a concrete plan. Chloe, can you create a ticket for the frontend part? Liam, can you handle the backend part? Let's try to get this fixed in this sprint. It's been a persistent annoyance.

**[10:48 AM]**

**Maya Cohen:** Alright, we're almost at time. Any other business? Anything anyone wants to bring up?

**Noa Mizrahi:** I have one small thing. I'm trying to run the whole stack locally with Docker Compose, but the backend service sometimes fails to start because the database isn't ready yet. Is there a better way to handle this?

**Ben Amir:** Ah, a classic. You can use a tool called docker-compose-wait or write a small wait-for-it script. Basically, you need your backend service's entrypoint script to loop and try to connect to the database before it starts the main application. I can send you a link to a script we use in our CI environment. It will solve that for you.

**Noa Mizrahi:** Oh, thank you! That would be great.

**Maya Cohen:** Excellent. Okay, let's do a quick recap of the action items.

* **Liam:** You're on point for the backend architecture doc for the file upload feature. You'll also implement the backend fix for the token refresh race condition.
* **Chloe:** You're creating the frontend ticket for the token refresh and will lead the UI/UX for the file upload, with Noa's help.
* **Ben:** You'll set up the S3/Lambda/IAM infrastructure for the upload feature. You also need to schedule the final check for the production database migration and send the wait-for-it script to Noa.
* **David:** You'll share the Figma wireframes and help us break down the epic into smaller stories for sprint planning.
* **Noa:** You'll pair with Chloe and work on the form validation ticket.

**Maya Cohen:** Did I miss anything?

**[Silence]**

**Maya Cohen:** Great. Solid meeting, team. We have a clear path forward on our next big feature and a plan for that annoying bug. Thanks everyone. Let's get to work.

**[10:55 AM - Meeting Ends]**