City of Toronto

311 Service Tracking

How a joined-up approach solved one large problem

Role	Senior UX/UI Designer, individual contributor
Team	30+ people, about ten roles. Service designers wrote the content. The UX/UI of the tracking page was mine.
Period	Six months
Context	Public sector. 311 Toronto — 9 divisions, 600+ problem codes

− 89%
Duplicate requests
SR data after launch · production

− 46%
Repeat contacts
Call centre logs · production

− 28%
Call centre load, against a −30% target
Beta measurement at launch

Overview

This is the story of how one person’s bad experience set off a chain of changes across ten city departments and improved the quality of public service delivery on Toronto’s main request portal, 311.

A city councillor once contacted 311 Toronto with an ordinary household request, as any resident would, and hit two problems that forced him back to the call centre several times: at the point of creating the request, and at the point of status updates. That led to an uncomfortable discovery.

— Residents cannot find the service they need on the portal. After an hour of searching they call the contact centre to place the request for them.

— Having placed a request, the resident receives no status updates. They do not know what happens next, and call again.

— Different residents routinely create identical requests without knowing about each other. Requests duplicate.

— There are more than 50,000 such calls a month.

All of it overloads the call centre and delays responses. The initial analysis produced the brief: simplify finding and submitting a service on the portal, and cut overall call centre load by 30%.

My role

I owned the UX/UI of the tracking pages, from research through handoff to development. I also owned the decomposition and presentation of the special instructions — sorting them by importance and by lifecycle, and setting the placement rules — and the mapping of street-test questions across the stages of a request’s life. Service designers wrote the content. Research was run jointly. The duplicate-removal mechanism was built by the team; I took part in the decisions around it. The work ran in several iterations: from an MVP concept to alpha and beta,

with functionality added and page complexity growing along the way.

The log study that set the direction

The target was a number. A number does not tell you what to build. The first step was two studies: an analysis of call centre logs and a survey of its staff. Design waited.

Problems appeared at different points along the service request journey — from finding the service to post-resolution. Each point generated its own calls, often several from the same people. Logs and interviews broke them down by cause.

— Service search is broken. Among 600 options the resident cannot find the right one, and calls to have the request created for them.

— Request creation is unclear. The resident does not understand how to submit, and calls for help.

— There is no tracking page. They submit and then see nothing, and call to ask what is happening with their request.

— Cancellation without explanation. A Cancelled status arrives, and they call to ask why.

— Duplicates. Several residents submit a request about the same thing, unaware of each other.

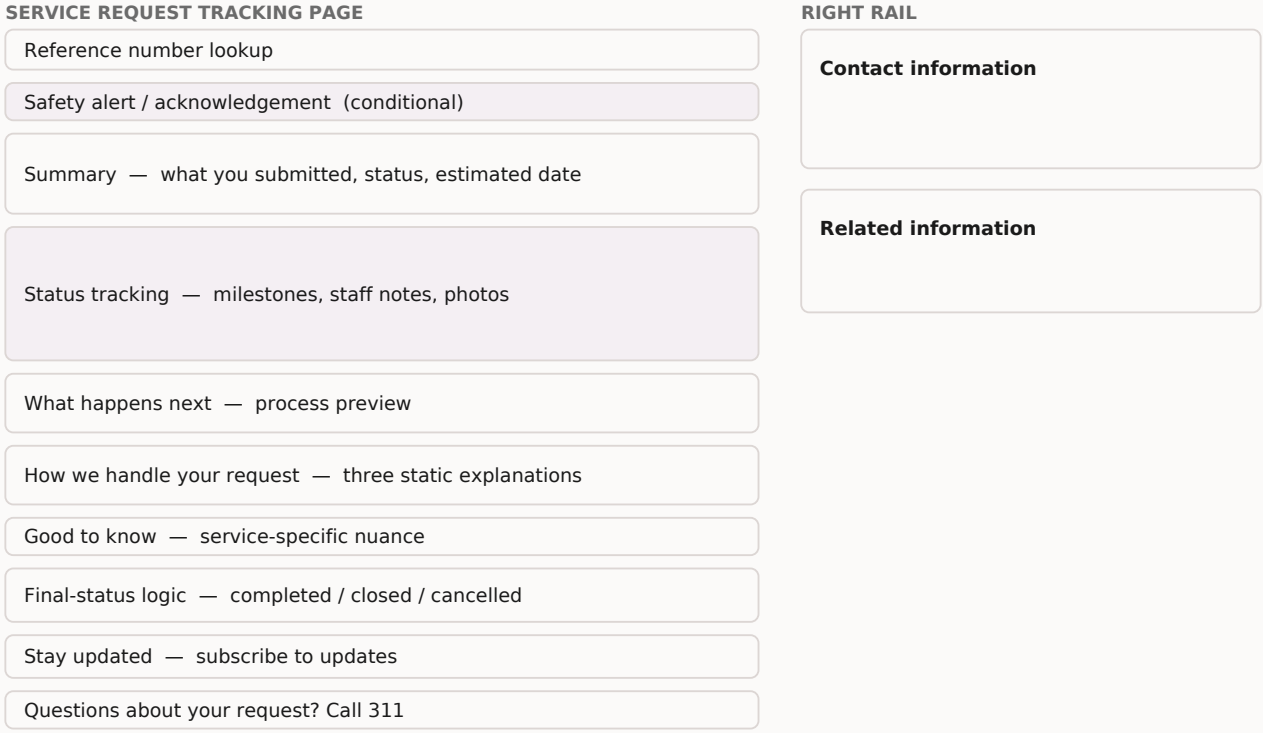
The first two points — search and creation — belonged to other teams. The rest stayed with ours: tracking, cancellation, duplicates. Each cause went on to get its own mechanism.

The tracking page: silence after submission

The largest driver of calls was that the resident saw nothing at all. They submitted a request and waited in a vacuum.

The solution came from the team, not from above. Working with the service designers we put together a tracking page concept, took it to leadership, and it went into production.

I designed the page to give residents what had been missing. Request statuses. Milestones showing where the request sits on its path. Notes from city staff. Photos from the location: what is happening there, how the problem is being resolved. A person opens the page and sees movement instead of silence.



Anatomy of the tracking page as designed. Redrawn from the working file; no production screenshots, no real problem codes.

The page closed the biggest driver of calls — the silence after submission. It is one component of the combined reduction in call centre load.

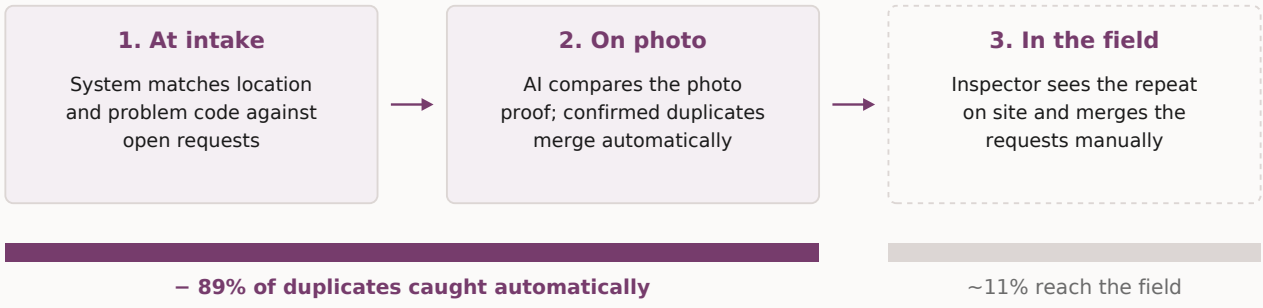
Duplicates: the city paying twice

A separate problem from the logs: several neighbours create a request about the same thing.

A pothole in the road. Everyone who walks past sees it and files their own service request. For the city each duplicate is a new request: an inspector drives out again to the same spot, crews arrive several times to fix one problem. This is not a spare row in a database. It is a double or triple spend of field resource.

We worked out where duplicates come from and placed three barriers at different points in the life of a request.

A DUPLICATE REQUEST TRAVELS LEFT TO RIGHT UNTIL A GATE CATCHES IT



At intake the system matches location and problem code, and offers an anonymized update on a request that already exists. On photo, AI compares the proof and merges confirmed duplicates automatically. In the field, the inspector merges what got through. Neither resident learns who the other is.

The first two barriers produced a 89% drop in duplicates. The remainder — roughly 11% by the research data — reached the inspector and was merged in the field. Method: analysis of service

request data after launch. Production.

The street test: the next layer of questions

The page shipped and closed the main problem. But we were two points short of the target, and went to check what was left.

The method was direct. The service designers and I went out on the street with a laptop, stopped passers-by, showed them clickable prototypes and interviewed them. The first round produced 55 questions from real people. I mapped them onto the stages of a request's life.

Lifecycle stage	Questions
Submission, what next	13
Assessment, routing, priority	8
Work in progress	10
Completion	9
Cancellation and closure	15
Total	**55**

The largest pile gathered at the end. Fifteen questions about cancellation and nine about completion — over a quarter of everything was about how a request ends, not how it progresses. Was it fixed or just closed. Why close it if the problem is still there. What do I do if I disagree.

The page had already solved the main issue. The test showed where to dig for the rest: into explanations of the process. People still did not understand how the city handles their request at each stage.

Decomposing the special instructions

Those explanations were meant to come from the special instructions — texts attached to each request depending on division and problem code. They were broken.

Six hundred instructions across nine divisions. Each division had its own set, with its own unstructured description, piled together, often outdated and written in technical language. The whole array was dumped onto the page as-is.

Here is what the pothole instruction looked like — everything piled into one block at the bottom of the page: what happens next; how long to wait by road type; what is worth knowing; why the request may be closed. And a final line:

> "If you have questions or would like more information, please call 311 and we'll be happy to help."

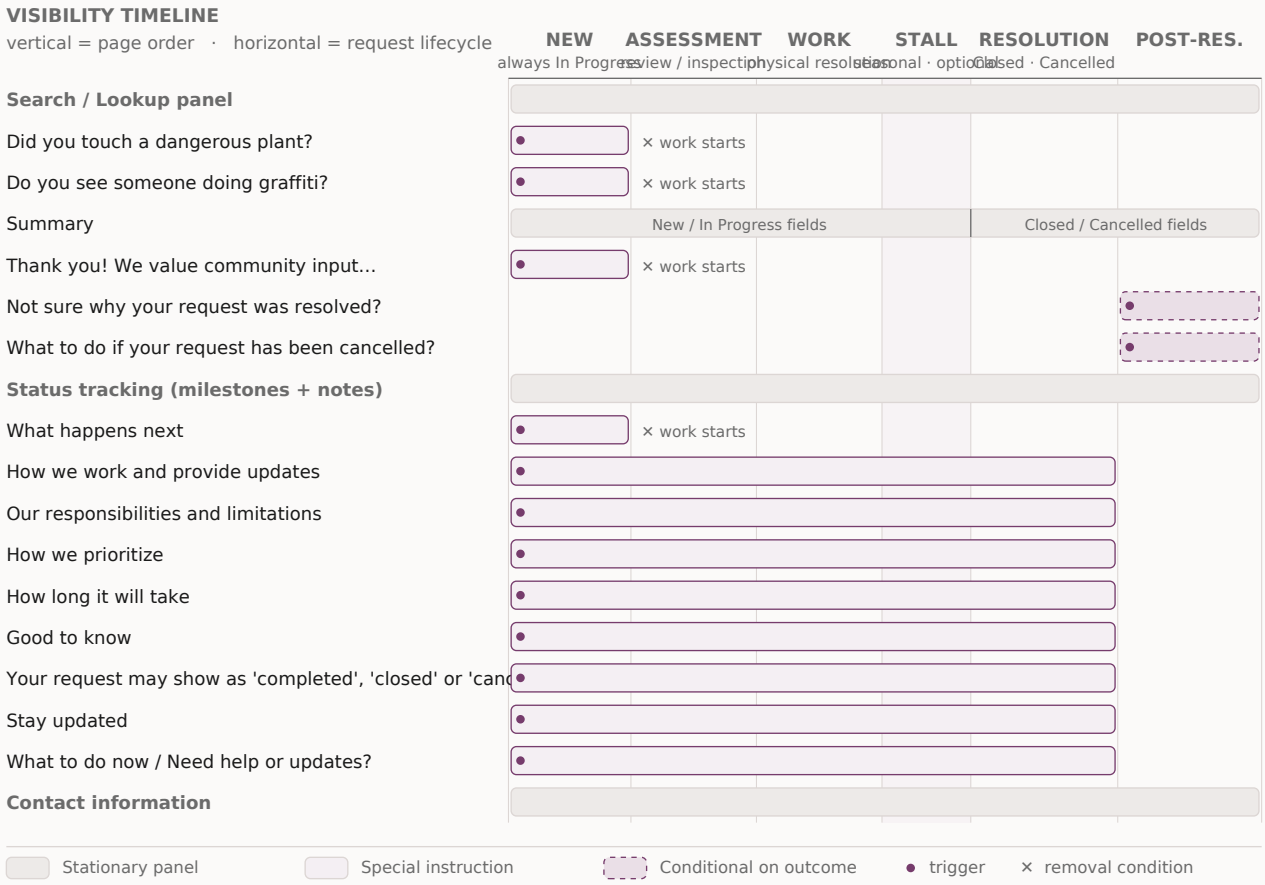
Text meant to take load off the call centre ended by offering to call the call centre.

Look at one instruction and it is just a long piece of text. Look at a hundred and the system becomes visible. Each instruction was first put through a single question: what should it do after a request is submitted, and what in it is generic, from the wrong stage, or redundant. The array was then laid out along two axes.

— By hierarchy of importance. The page is built around the resident's questions: this is my request; what is the status; what has happened already; why is it taking so long; what does the outcome mean; do I need to do anything. The questions set the order, so safety, confirmation, progress, explanation, expectation and support stop competing for attention.

— By request lifecycle. Each instruction is mapped across the cycle: when it appears, how long it lives, on what condition it disappears. That makes visible where blocks overlap, which stages are overloaded, what already exists and what has to be created.

Every instruction lands at the intersection of the two: its place by importance, its moment in time.



Eighteen rows: four stationary panels and fourteen instructions, against six lifecycle stages. Each instruction carries its trigger and its removal condition. Redrawn from the working file.

The work was then split. Service designers brought the existing instructions up to a reasonable standard: readability, structure, consistency between divisions. I proposed additional instructions to close four gaps in user questions that the old array did not answer. Together with the service designers we refined them, and some went into production.

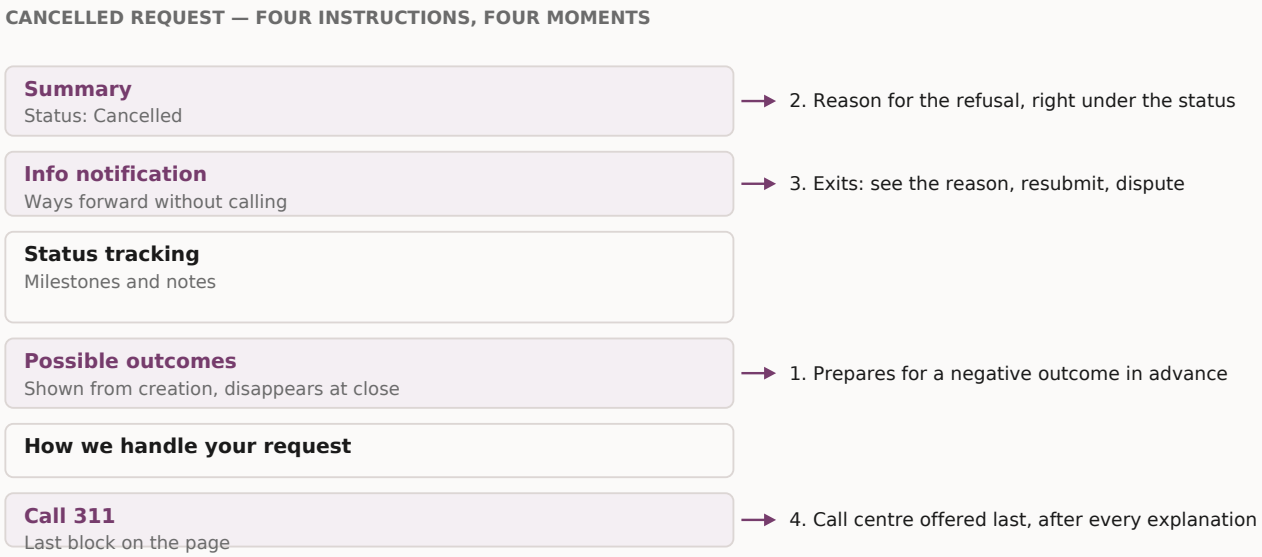
A scattered array became a structured system of UX content. The decomposition was taken through to concept and user testing; part of the decisions carried an "Open question for team review" flag. The case attributes no separate metric to it.

Cancelled: refusal without explanation

A negative ending produced the largest share of questions — and became the clearest case of the decomposition in action: four instructions from the array, positioned by importance and by lifecycle.

Cancelled arrived like a wall. The resident received a “request cancelled” email, did not understand why, and called. What is wrong. What do I do now. The standard “call 311” on the page answered nothing. It generated precisely the call that had to be removed.

Complaints about cancellation came in two kinds: the person disagrees with the decision, or the problem is physically still there.



Each instruction appears at its own time and in its own place on the page. The call centre becomes the last option, after every explanation and every exit.

In most cases this was enough. The resident read the explanation, understood the refusal and accepted it — no repeat request was created. A 46% drop in repeat contacts. Method: analysis of call centre logs. Production.

Results

State as of my work on the project. The metrics differ in nature, so they are stated separately.

— Duplicate requests — down 89%. Produced by the first two automatic barriers: location and code matching at intake, and AI photo matching. The remainder, roughly 11% by the research data, reached the inspector and was merged in the field. Method: analysis of service request data after launch. Production.

— Repeat contacts — down 46%. Explaining the exits from a Cancelled state: the resident understands the refusal and does not resubmit. Method: analysis of call centre logs. Production.

— Call centre load — down 28% against a 30% target. The combined effect of several mechanisms. The tracking page closed the biggest driver, the silence after submission. The special instructions pointed residents from a cancellation to reference material and further exits. Duplicate removal cut the overall volume of requests. Service search and submission belonged to other teams and sit outside this case. Method: beta testing at launch. Beta.

Six months of work in total. All three directions share one source: the analysis of call centre logs at the start.

Reflection

Two lessons.

The first is about the number in the brief. A target of “minus 30%” contains no solution. It contains a question — why are there 50,000 calls in the first place. The logs showed that calls were born at different points along the request journey. Service search and request creation belonged to other teams. The rest was ours: silence after submission, unexplained cancellation, duplicates — and each cause got its own mechanism. The causes also fed each other: without merging, the inspector cancelled duplicates, and every cancelled duplicate became a fresh “why was this cancelled” question.

The second is about the instructions. Six hundred texts looked like a content problem. They were information architecture. When a person understands the process and its possible outcomes in advance, the need to call disappears. What decides it is where the text sits — by importance, and by moment in the life of the request.

All schematics in this document were redrawn from scratch. No production screenshots, no real problem codes, no verbatim internal copy.