Supplementary Material: Technical Implementation of the Digital Twin Construction System

This document provides the comprehensive technical specifications, software architecture, and implementation details of the Digital Twin Construction (DTC) system described in the main manuscript. While the primary paper focuses on the system's conceptual framework, experimental validation, and impact on production control, this supplementary material details the software engineering aspects required to reproduce the experimental setup. The following sections describe the client-side interface logic, the server-side database schemas and API endpoints, the data fusion algorithms used for monitoring, and the specific heuristic logic employed by the optimization engine.

1 Worker and Supplier Interfaces

The DTC system relies on a suite of client-side applications to capture data and display directives. At the execution level, two web-based interfaces support field operations: a worker interface for guiding construction activities and a delivery interface for managing shipment preparation. Both interfaces maintain real-time synchronization with the digital twin server, ensuring that all operational decisions reflect the current planning state.

The worker interface provides assembly instructions to the construction crew and enables the crew to report completion to the digital twin. For instruction delivery, the interface retrieves the current assembly sequence from the server and tracks which elements have been completed, thereby identifying the next piece requiring erection. The interface displays this information through a floor plan visualization that highlights the target element's location with the element ID and type ID that workers use to verify they have selected the correct piece type from site inventory. A skip function allows workers to temporarily bypass pieces that are missing from inventory, deferring their installation to the following day while proceeding with available elements further in the sequence.

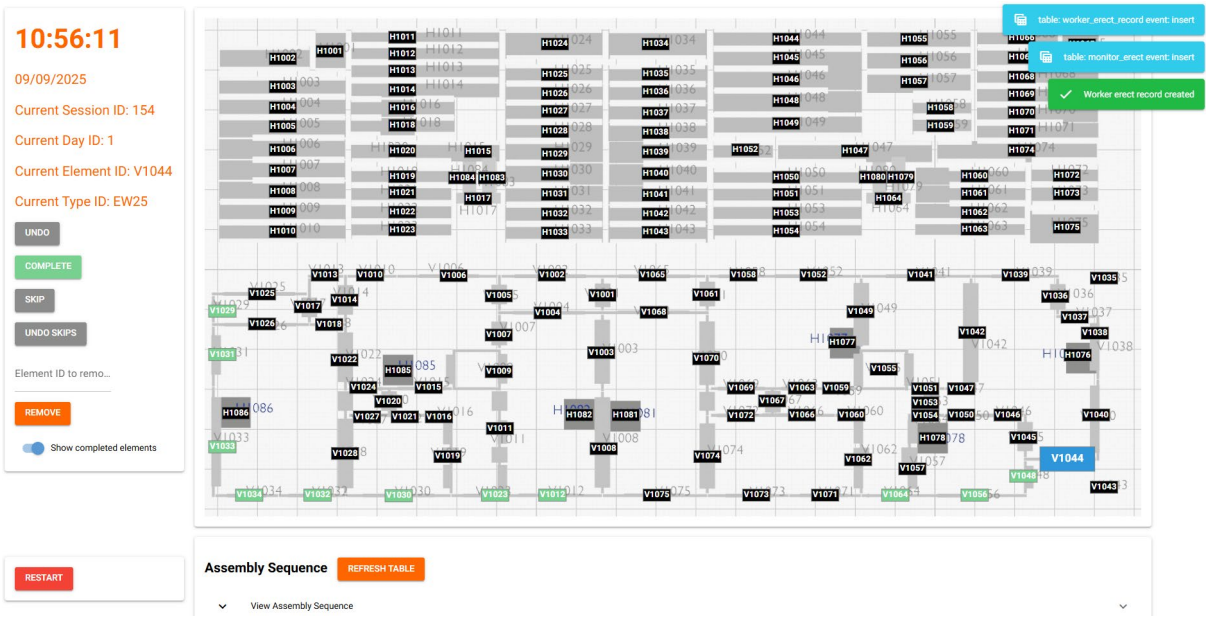

Figure 1 – Worker's assistance and reporting interface

Technical implementation of the worker interface used WebSocket connections for real-time notification of schedule updates. When the assembly sequence changes, the interface receives an event notification and sends an HTTP request to retrieve the updated schedule. Upon completing an element installation, workers press a complete button that triggers an HTTP POST request containing the element ID and timestamp, updating the element's status in the digital twin system.

The delivery interface guides the factory worker in preparing shipments and scanning the element QR codes before delivery. The interface displays type IDs and the required quantities for upcoming deliveries and displays the quantity of pieces scanned. A status symbol indicates completion when scanned quantities match requirements for each type. As a safety mechanism, deliveries can only be marked complete when all types and quantities align, preventing human error in reporting.

The experimental design incorporates a one-day lead time for delivery commitment, meaning the next day's delivery schedule is fixed while subsequent days remain modifiable. Consequently, factory personnel only see delivery schedules extending through the next day, preventing premature preparation of deliveries that may still change. The workday cannot commence until factory personnel confirm completion of the current day's delivery.

Both interfaces maintain synchronization through WebSocket connections that monitor schedule changes. When modifications occur, the delivery interface updates all schedules beyond the following day while preserving the deliveries committed for the next day. This ensures that both construction and delivery operations continuously align with the digital twin's current planning state, enabling the closed-loop control that characterizes the digital twin construction approach.

2 Server and Database

The monitoring systems (detailed in the main manuscript) generate construction status data that must be stored, processed, and distributed to support comparative analysis across different automation levels.. This section describes the database and server infrastructure.

The system architecture employs a containerized design using *Docker*, a platform that packages applications and their dependencies into portable containers for consistent deployment across different computing environments. A *FastAPI* application provides REST endpoints for data operations, deployed with *Uvicorn* workers. An *Nginx* reverse proxy server, which distributes incoming network requests across multiple application instances, routes traffic by separating read operations from write operations to prevent blocking. The infrastructure includes a WebSocket service for real-time bidirectional communication, exposed via *ngrok* for external access. The complete deployment uses *docker-compose* for orchestration.

The database uses *PostgreSQL*, an open-source relational database system, with *SQLAlchemy* ORM (Object-Relational Mapping), which translates between database tables and programming objects. The system uses an asynchronous engine built on *asyncpg* for non-blocking database operations. The schema organizes around two identifiers: *SessionId* for tracking experimental runs and *PlanId* for versioning construction schedules. Tables are grouped into functional categories. Experiment and runtime tables log session events and timestamps. Scan and Monitoring tables track Delivery Scans, Erect Scans, and the Monitor

Erect timeline. The Element Status table maintains production, delivery, and erection days for each component. Planning tables include *AssemblySequence*, *ProductionSchedule*, and *DeliverySchedule*. Additional tables store *PieceType* classifications, *GlobalParameters*, *SessionParameters*, and *OnsiteInventory*. Execution records capture *WorkerErectRecord* entries and Disruptions. The *OptimizationResults* table stores algorithm outputs as JSONB payloads, a PostgreSQL data type for storing structured JSON data, while *OptUserActions* tracks manual interventions. A *UsersStateTable* persists interface state between sessions. An entity relationship diagram of the data schema is available in the supplementary materials.

The REST API implements standardized CRUD (Create, Read, Update, Delete) endpoints for each database table. Session-scoped endpoints filter results by the current *SessionId*. The API supports filtering through query parameters using syntax such as field operator value patterns. Control endpoints enable session management, day progression, and plan versioning. Schedule orchestration functions handle baseline plan insertion from Excel files, production and delivery schedule generation, and rescheduling operations that increment the *PlanId*. The system provides batch update capabilities for element status modifications and functions to compute *ElementStatus* from schedules and scans. Transaction handling uses FastAPI dependencies to manage Async Session objects with commit and rollback support.

Real-time communication uses a separate WebSocket application that connects to PostgreSQL and listens on notification channels. When database operations trigger NOTIFY commands, the WebSocket service broadcasts these events to subscribed clients. This push-based approach allows digital twin services to receive status changes without polling. Clients receiving notifications can then make REST API calls to retrieve detailed data or initiate planning updates.

The data flow follows a pattern where construction events generate REST API calls that persist to *PostgreSQL*. Write operations may emit NOTIFY messages that the WebSocket service distributes. The server computes derived values such as status updates and inventory levels either on-demand or after writes. Session-scoped operations provide filtered views for each experimental scenario. The *Nginx* proxy's traffic separation ensures that read operations do not impact write performance.

This infrastructure supported all experimental scenarios, capturing performance metrics while maintaining the responsiveness required for digital twin operation. The standardized API design facilitated development of interfaces for various levels of automation of the DT control, while the real-time capabilities enabled the feedback loops necessary for testing closed-loop control.

3 Monitoring System

The digital twin's representation of construction progress relies on integration of multiple data streams that track each building element through its lifecycle. The system combines three primary sources of information: factory delivery scanning, on-site worker scanning, and worker completion reporting. These data streams, collected through the worker and factory interfaces, are processed and stored in the database. The challenge is to associate piece identifiers assigned during production with element identifiers that define specific positions in the building structure, a relationship is only fixed once an element is installed.

Tracking begins at the factory when pieces are prepared for delivery. When factory personnel scan a piece using the delivery interface, the system captures the piece ID along with its Type ID through the QR code. At this stage, the relationship between piece and final building position remains open, any piece of a given type may be installed in multiple element positions in the building that require that type. Each position for a piece in the design model is identified with a unique element ID and carries a type ID that determines which pieces can be installed. Upon arrival at the construction site, the system updates the piece status to "delivered," maintaining the temporal record of its progression through the supply chain.

Once pieces arrive on site, the worker interface coordinates the erection process while capturing tracking data. When a worker receives an instruction to install a piece in a specific position (identified by its element ID), they locate a piece with the matching type ID from the site inventory. The worker scans the QR code on the selected piece, transmitting both the piece ID and type ID back to the server. This scanning event triggers a status update in the *MonitorErect* table to "erection in progress," and records the erection start timestamp. The raw scanning data is preserved in the *ErectScans* table, maintaining an audit trail of all scanning events.

The completion of element installation marks the point where the system resolves the piece-to-element relationship. When the worker presses the completion button on their interface, the system captures the element ID, type ID, and completion timestamp, recording this information in the *WorkerErectRecord* table. This completion event provides the link that allows the server to associate the specific piece ID (from delivery and erection scans) with the element ID (from the completion record). This establishes which physical piece was installed in which building position.

The server employs data fusion logic to interpret these three data streams into a coherent status representation. The association algorithm examines the temporal relationships between delivery scans, erection scans, and completion records, using timestamps to establish the logical flow of each piece through the construction process. By matching type IDs across all three data sources and applying temporal constraints, the system determines the path of each element from factory to final position. The *MonitorErect* table contains this fused data, transforming raw scanning events into project status information that supports decision-making.

Recognizing the importance of status accuracy, the system incorporates fail-safe mechanisms to ensure data reliability in the presence of connection errors or scanning failures. The completion button on the worker interface includes a confirmation feedback mechanism, alerting workers if their completion message was not received and processed by the server. This confirmation loop ensures workers cannot proceed to the next element until the current completion is recorded. The system acknowledges that scanning events from delivery or erection may fail to reach the server due to network interruptions. To address these gaps, the fail-safe logic examines incoming completion records and, when unable to correlate them with corresponding piece IDs from earlier scans, creates partial records in the *MonitorErect* table with missing piece IDs. These incomplete records can be healed through subsequent data reconciliation while ensuring that the completion status is preserved.

4 Optimization Server and Algorithm

The optimisation server provides the core algorithmic support for the Plan and Check phases. It functions as a discrete microservice. It is implemented as a Python FastAPI application, packaged and deployed in a Docker container, and connected to the operational system through both RESTful and WebSocket endpoints.

Conceptually, the optimisation workflow combines two complementary components: a parallel assembly-sequence enumerator and a downstream scheduler that converts chosen sequences into production and delivery plans. The workflow starts from four inputs that define the current construction state and priorities: (i) a directed graph of buildings, floors, zones, and prefabricated elements, including spatial adjacencies and structural support relationships; (ii) the set of already-installed elements, representing partial progress; (iii) priority elements grouped by readiness level; and (iv) priority zones that encode project objectives such as earlier completion of certain apartments or building zones. These inputs are derived from the digital twin's project status and intent representations, initialising the dependency graph, updating zone and floor states, and seeding the search for feasible next steps.

The enumerator explores possible erection sequences in parallel using a manager–worker pattern. The manager identifies eligible starting elements, typically structurally foundational nodes satisfying support and stability requirements, and assigns them as tasks to worker processes. Each worker maintains their own local copy of the dependency graph to avoid synchronisation overhead and extends a partial sequence step by step. At each step, the worker selects an eligible element that respects structural dependencies (supports in place before supported elements), spatial continuity (avoiding scattered "islands" of progress), and type precedence (e.g., walls and columns before bathroom modules, bathroom modules before hollow-core slabs). Once these safety and structural rules are satisfied, priority elements and priority zones bias the choice among admissible options so that exploration remains aligned with project objectives rather than wholly random branching. When a worker encounters branching points with multiple admissible next elements, it continues along one branch while returning the others to the manager as new tasks, combining depth-first progress along a candidate path with breadth-wise coverage of alternatives. Randomised tie-breaking with fixed seeds ensures that repeated optimisation runs on the same input state remain reproducible, and indexing structures keep vertex lookups efficient. The enumerator stops when a preconfigured time threshold is reached or when the space of tasks is exhausted, returning a collection of complete, valid assembly sequences from the current construction state through to project completion.

Each generated sequence is then evaluated by a scoring function that reflects construction logic, spatial feasibility, and alignment with project priorities. The score aggregates penalties and rewards for, for example, respecting structural precedence, maintaining compact spatial fronts, avoiding unnecessary zone switching, progressing priority zones earlier, and keeping the number of problematic or delayed elements low. The manager aggregates these results, ranks all evaluated sequences by their score, and selects a subset of top-ranked alternatives. These alternatives are stored in the database as optimisation results associated with the current session and plan identifiers and exposed to the recommendation dashboard through the central server. The recommendation dashboard presents these alternatives to the construction manager, who can inspect their trade-offs and, if they so desire, adopt one as the new baseline erection sequence. Once an alternative is selected, the corresponding assembly

plan is written back into the planning tables and becomes the basis for subsequent directive generation to workers and supplier interfaces, thereby closing the loop from algorithmic exploration to human decision-making.

The scheduler consumes a chosen assembly sequence and translates it into concrete production and delivery schedule that respects factory throughput and transport constraints. Each row in the assembly plan is converted into a “piece” object enriched with attributes such as step order, target day, factory type, production group, and component type. Pieces are partitioned by production line (slabs, walls, pods and stairs) and organised into production groups, which are then ordered so that earlier erection steps and earlier target dates are pushed to the front of the production backlog. Configurable production tables represent parallel production resources; each table draws pieces from its assigned group until its batch size or shift capacity is reached, applying cycle times per piece and setup times when switching between groups. When a table exhausts its group, it is reassigned to the largest remaining backlog to reduce idle time. The resulting per-table traces are aggregated into daily production quantities per factory and group, forming the production schedule.

For ongoing projects, the scheduler incorporates historical production and delivery records before computing new schedules. Pieces that have already been produced are removed from remaining demand, and production restarts after a configurable lead time, with historical rows preserved for analysis. Delivery planning then walks the assembly sequence in step order, releasing deliveries only when inventory contains the required piece types and the current day meets or exceeds the target date implied by the assembly plan. A daily delivery-capacity cap prevents unrealistic delivery bursts; when a required item cannot be shipped because the cap would be exceeded or inventory is insufficient, the attempt still consumes capacity to reflect handling and organisational overheads even when the shipment is incomplete. Inventory levels are initialised from cumulative production minus past deliveries to avoid double counting. The scheduler outputs include daily production plans per precast production table, daily delivery plans, any assembly rows that could not be scheduled under the given constraints, and table-level traces that can be used to diagnose capacity bottlenecks. When disturbance scenarios are modelled, a disruption log records delivery errors and their effects.

End to end, the optimisation server orchestrates this combined enumerator–scheduler workflow as a single optimisation cycle. When triggered via the recommendation dashboard, it pulls the current construction state and planning context from the digital twin, enumerates and scores feasible erection sequences, forwards ranked alternatives for human review, and, once an alternative is selected, generates compatible production and delivery plans that honour both structural logic and operational constraints. Because the server is deployed as a containerised FastAPI service with REST and WebSocket endpoints, it can be invoked repeatedly during an experiment to support replanning as the site state evolves, enabling systematic study of how different levels of algorithmic support and monitoring influence the performance of the closed-loop DTC control system.

5 Planning and Control Interface

The planning and control interface supports the construction manager. It presents the current delivery and assembly plan as an interactive, multi-day scheduling board (Figure 2). Each column corresponds to a production day in the simulated project, and each card within a column represents one building element. The card shows the element ID in bold, together with the type ID and production group, so that the manager can reason both about individual

elements and about groups of similar components. The position of a card under a given day encodes the requested delivery and erection day for that element: placing an element under “Day 6”, for example, specifies that this element is requested to be available on site and erected on Day 6. This plan is therefore a delivery and assembly request sent to the factory rather than a guaranteed commitment; the production system subsequently attempts to satisfy this request subject to its own capacity and lead-time constraints.

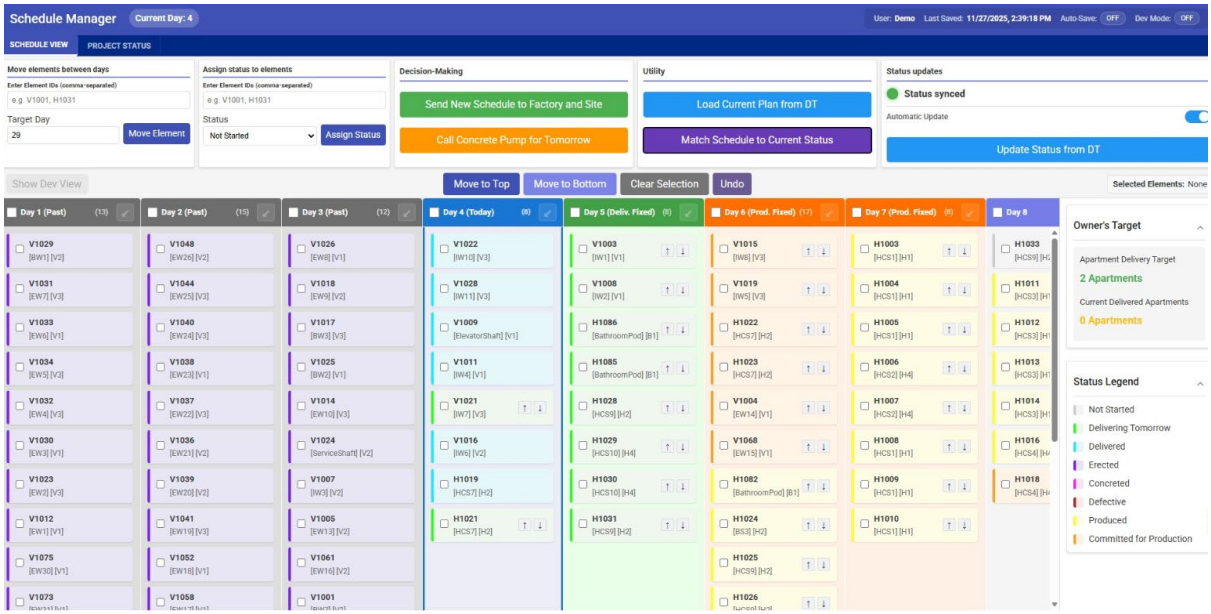

Figure 2 – Planning and Control Interface

Colour-coding on the cards communicates the status of each element (“not started”, “delivering tomorrow”, “delivered”, “erected”, “concreted”, “defective”, “produced”, and “committed for production”). In Figure 2, the current day is Day 4. all elements scheduled on Days 1–3 and successfully installed are marked as “erected”, elements that have been delivered but not yet installed appear as “delivered”, and elements scheduled for the next day are highlighted as “delivering tomorrow”.

Schedule changes are made directly on this board. The manager can move single elements or selected groups across days, reorder elements within a day, and clear or undo recent changes. When the manager confirms a revised plan, the updated assembly sequence and delivery requests are written to the digital twin via the REST API. This operation versions the plan in the database, updates the worker and supplier assistance interfaces so that they follow the new sequence, and triggers the production-side logic that attempts to realise the requested deliveries.

In addition to the schedule changes, the interface enables the manager to order a concrete-pump truck for slab concreting on the following day, based on the current state of slab erection and the manager’s preference. The consequences of this decision are reflected in subsequent days through the updated status information and schedule.

6 3D Visualization Interface

A visualization interface (Figure 3 and Figure 4) was implemented in the Unity engine to provide an interactive 2D/3D view of the building model and to spatially contextualize both (i) the planned delivery and erection sequence and (ii) the as-performed status streamed from site monitoring. The interface supports rapid progress inspection in a top-down 2D plan view and supports spatial reasoning in a navigable 3D view, enabling managers to assess installation dependencies and the consequences of disruptions.

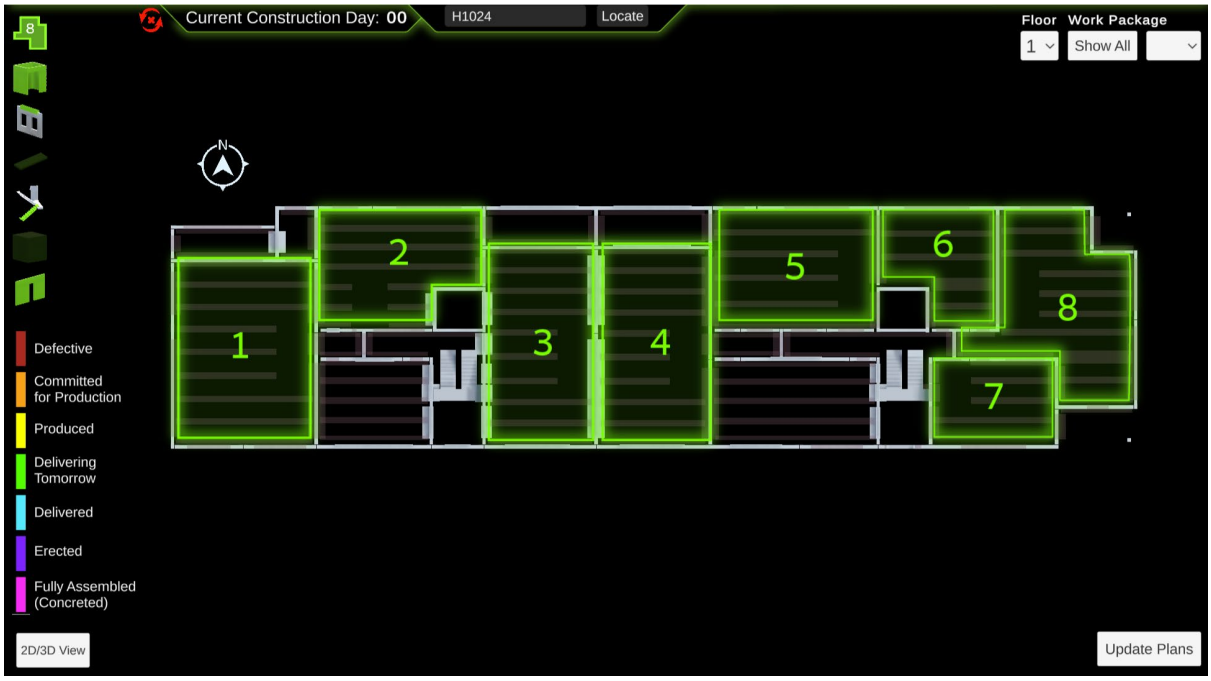

Figure 3. 2D plan view in the Visualization Interface at Day 0. The interface shows floor 1 layout with apartments (1–8) highlighted in green, orientation indicator, and the color-coded status legend on the left. At this stage, no construction progress has been re

The interface exposes element-level context on demand: when a user hovers over an element, the system retrieves and displays the element's planned erection step and work-package association together with upstream production attributes (e.g., factory-related identifiers and production grouping). Managers can reduce visual complexity by filtering the displayed model by element type, floor, apartment association, and production status, allowing focused diagnosis of delayed, missing, or defective elements and of the downstream elements that depend on them.

The Unity client synchronizes with the digital-twin services using a hybrid communication scheme. It issues REST requests (UnityWebRequest) for structured data retrieval (e.g., loading the current plan, querying element metadata) and maintains a persistent WebSocket listener for low-latency propagation of monitoring updates. Because site monitoring pushes status changes automatically, the visualization reflects updated element states in near real time without manual refresh.

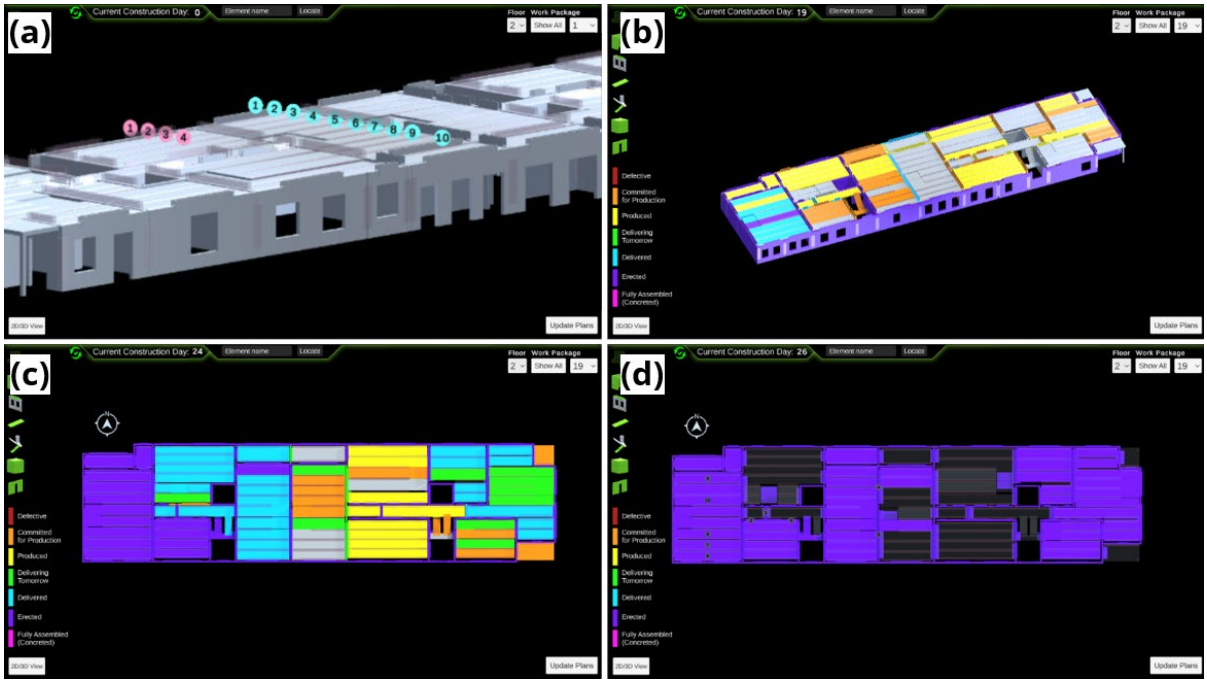

Figure 4 – (a) : Zoomed-in 3D view of the erection step sequence, highlighting the planned order of installation; (b): 3D full status view of all elements on a selected floor, providing a spatial overview of production, delivery, and erection; (c): 2D status view; (d): 2D view of currently erected elements.

By combining planned data, live updates, and interactive filtering, the visualization interface supports the manager's ability to (1) maintain situational awareness across work packages, (2) detect emerging delays, errors, or defective elements and identify impacted dependencies, and (3) reassess sequencing decisions considering the current construction state.

7 Location-based Visualization

The location-based visualization provides a dashboard-style view that summarizes real-time progress at the work-zone level using status information maintained in the digital twin (Figure 5). Implemented as a JavaScript web application and deployed in Docker, the interface combines a time-series view of apartment completion progress and a tabular, zone-organized status view. The progress chart tracks, by project day, apartments in progress (orange), erected (blue), and concreted/closed-off (green), and juxtaposes these observed curves against stakeholder targets (dark purple delivery-rate line) and the planned completion trajectory extracted from the current schedule (pink dashed line). The companion table reports per-apartment progress percentage and status together with planned end dates and any recorded actual erection and concreting dates, while also exposing element-level states (e.g., in factory, in site inventory, erected) aggregated by work zone. This representation enables managers to monitor unit completion, detect divergence from plan and targets, and localize blocking constraints without requiring element-by-element inspection.

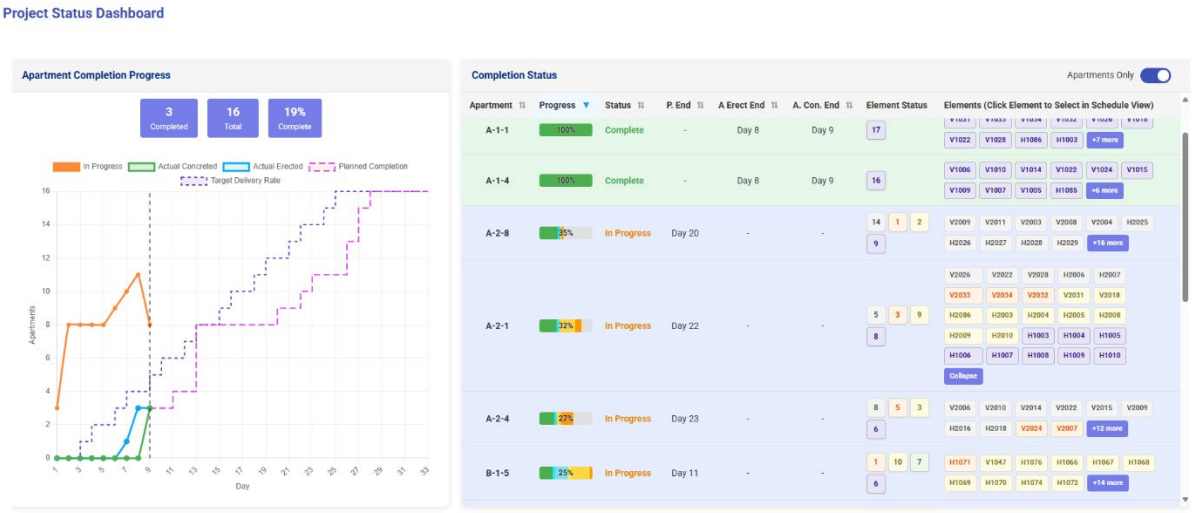

Figure 5 – Location-based visualization for manager decision support

8 Recommendation Interface

The recommendation interface presents alternative assembly schedules generated by the optimization service and enables the construction manager to accept (or reject) a proposed replanning action (Figure 6). The system operates on a daily cycle: at the start of each workday, the optimization service retrieves the current project state from the digital twin, generates a small set of feasible alternatives (configured to five in the experimental setup, utilizing the heuristic enumeration logic described in Section 1.1.3 above), and stores these alternatives on the server for review.

The interface summarizes each alternative using key performance indicators (projected total cost and a risk score) and provides complementary views that explain plan implications. A 3D sequence view visualizes the current state and the proposed remaining sequence, with element-level information available on hover and with filtering by floor and element categories to support targeted inspection. In parallel, the interface provides apartment-level plan explanations (e.g., predicted completion ordering) and time-based projections (e.g., completion progress and throughput) so that managers can compare alternatives against external priorities such as phased handover commitments.

When the manager selects an alternative, the interface issues an update request to the digital twin server, which versions the plan and propagates the revised schedule to the worker and delivery interfaces. This closes the decision-support loop by ensuring that an accepted recommendation immediately becomes the operational plan for subsequent production, delivery, and erection activities.

The interface was implemented as a JavaScript web application and deployed in Docker.

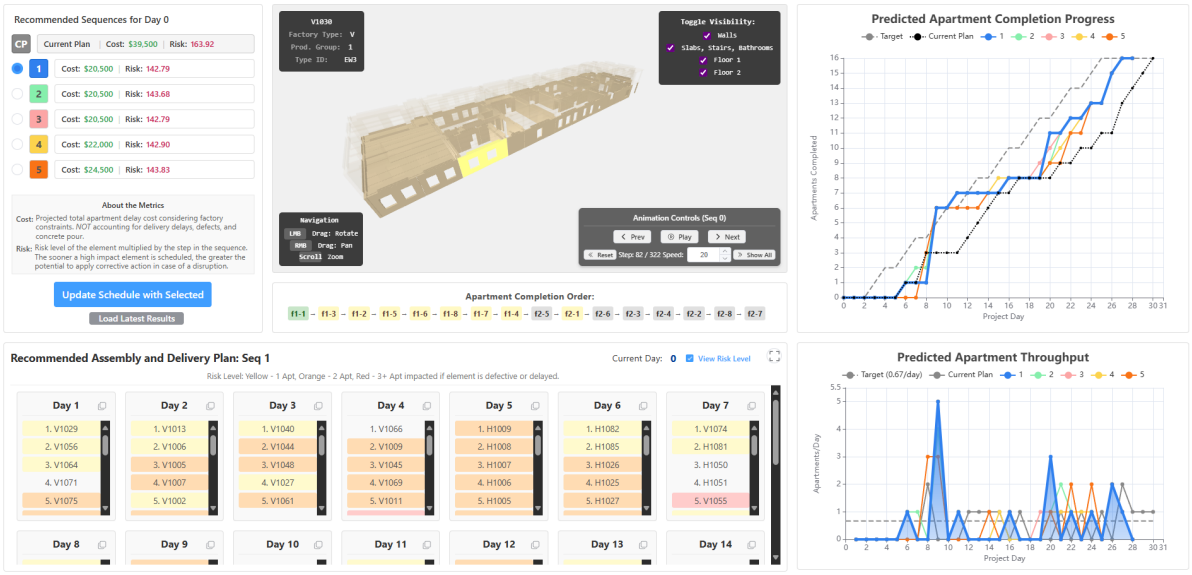

Figure 6 – Optimization and decision support interface