

HSS

A hotel support system for one 36-room property with a restaurant.

Stack	PHP on Laravel 12, MariaDB, Vite for the front end. Runs on shared hosting and, as five Docker containers, on a PC inside the hotel
Status	Complete. Second version; built, tested and in the hands of the hotel
Size	About 22,000 lines of PHP and 730 tests, plus seven typeset manuals for the people who use it

The problem

A hotel with thirty-six rooms and a restaurant is too small for the property-management systems built for chains and too busy to run on paper. It needs its own website to sell rooms; a front desk that knows which of those rooms is clean, occupied or out of service; a screen in the kitchen; a way for waiters to take orders from a phone; a housekeeping list; and one person with an admin password who can change a rate. It also needs all of this to keep working when the hotel's internet connection drops, which it does.

HSS is those things as one application. This is its second version. The first was built, used, and found wrong in ways that could not be patched; the second is the one described here.

The modules

Module	Who uses it, and how they get in
Website and booking	the public; no sign-in
Front desk	an enrolled desk machine plus a PIN
Kitchen display	an enrolled screen plus a PIN
Waiter orders	an enrolled handset plus a PIN
Housekeeping	an enrolled handset plus a PIN
Contacts	an enrolled handset plus a PIN
Admin	email and password

Staff do not have email-and-password accounts. A device is enrolled once by an administrator and thereafter a PIN is enough, because nobody at a front desk should be typing a password twice a shift, and because a lost handset can be un-enrolled without anybody having to change a password they share.

Two servers, and neither is in charge

The system runs in two places at once. A shared-hosting server is where the public website and its bookings land. A PC inside the hotel runs the same stack in containers and is also the front desk's workstation. They are peers: a sync spine reconciles them in both directions, and when the hotel's connection drops, the desk, the kitchen and the waiters carry on against the local machine and catch up when it returns.

Peers for data, but not for everything. Only the on-site machine can talk to hardware – a receipt printer, a key-card encoder – so the design gives it a role the cloud half cannot take over, rather than pretending the two halves are interchangeable. And where reconciliation cannot be settled by a rule (both sides changed the same booking while apart) it goes to a conflict inbox for a person to decide, with both versions shown. The system's job is to make sure nothing is silently lost; the judgement is a human's.

The on-site machine is set up by four idempotent scripts that turn a bare Ubuntu installation into a staffed workstation: the containers, the desk session, the printers, the enrolment. Idempotent matters. It means the cold spare in the cupboard is a desk, not a rebuild.

What the first version got wrong

Version one assumed staff would use the on-site server whenever they were on the hotel's network. That is false for anyone on a personal phone, which is everyone, so the sync was designed around a case that never

happened and the real case – a waiter on mobile data ten feet from the local server – was the exception path. It was rebuilt rather than patched.

Two structural fixes carry the second version.

Selling a room type and assigning a physical room are separate acts. A guest books “a deluxe double for three nights”; which of the deluxe doubles they get is decided at check-in, by the desk, with the housekeeping board in front of them. The first version assigned a room at booking time. Within weeks the thirty-six rooms were fragmented into a pattern where the hotel was showing as full for a stay that four different rooms could each have taken half of. The second version sells from a pool and assigns late, and a defragmenter proposes moves that would open up longer stays.

Exactly one piece of code counts rooms. Three places counted them in the first version, and they disagreed, which produced oversells and unsellable rooms at the same time. There is now one capacity service, and every screen that says “available” asks it. It is the least glamorous class in the codebase and the one I would rewrite last.

The website sells the desk’s own rooms

The public site is not a booking form bolted onto a subpath. It is the hotel’s website, and it sells rooms from the same table and the same rate calendar the front desk uses. It cannot advertise a room the hotel no longer has, or a price the desk would not honour, because there is no second copy of either for it to be reading from. A rate changed in admin is the rate on the website in the same minute. Payment is through a hosted gateway; the site never sees a card.

What I would tell the next person

Most of the difficulty in this project was not in any module. It was in the seams between them: what “available” means when two machines have each sold the same room while apart, what happens to an order when the kitchen screen loses its connection halfway through, how a handset knows it is still enrolled. The modules were each a few days; the seams were the months. That is the usual shape of a system that has to keep working when part of it is missing, and it is why the second version put the sync and the capacity count at the centre rather than at the edge.