Sistem de Licitatii Live

Platforma distribuita de licitatii in timp real

Sisteme Distribuite

Autor: Echipa proiectului
An universitar: 2025–2026

19 ianuarie 2026

Cuprins

1 Introducere **4**
1.1 Contextul proiectului . . . 4
1.2 Obiectivele proiectului . . . 4
1.3 Functionalitati implementate . . . 4

2 Arhitectura sistemului **5**
2.1 Prezentare generala . . . 5
2.2 Fluxul principal de date . . . 5
2.3 Diagrame suplimentare . . . 6

3 Componente si servicii **7**
3.1 Auction Service . . . 7
3.1.1 Endpoint-uri REST . . . 7
3.2 Worker-ul de expirare . . . 7
3.3 Notification Service . . . 8
3.4 Frontend React . . . 8
3.5 MongoDB si Redis . . . 9
3.6 Kubernetes si Ingress . . . 9

4 Tehnologii utilizate **10**
4.1 Backend si runtime . . . 10
4.2 Frontend . . . 10
4.3 Infrastructura si DevOps . . . 10

5 Fluxuri operationale **11**
5.1 Licitarea unui produs . . . 11
5.2 Expirarea licitatiilor . . . 11
5.3 Autentificare si autorizare . . . 11

6 Docker si containerizare **12**

7 Securitate **12**

8 Concept distributiv si rezilienta **12**
8.1 Separarea responsabilitatilor . . . 12
8.2 Comunicare asincrona si cuplaj redus . . . 13
8.3 Consistenta si coordonare . . . 13
8.4 Scalare orizontala . . . 13
8.5 Toleranta la defecte . . . 13

9 Ghid de instalare si rulare **14**
9.1 Cerintele mediului . . . 14
9.2 Pasii de deploy pe Minikube . . . 14
9.3 Actualizari dupa modificari . . . 14

10 Utilizare **15**
10.1 Flux administrator 15
10.2 Flux utilizator standard 15

11 Structura proiectului **15**

12 Concluzii **17**
12.1 Rezultate obtinute 17
12.2 Provocari intampinate 17
12.3 Directii viitoare 17

13 Bibliografie **18**

1 Introducere

1.1 Contextul proiectului

Platformele moderne de licitatii digitale ofera experiente interactive care combina procesarea de evenimente in timp real, persistenta datelor si notificari instant. Proiectul **Sistem de Licitatii Live** implementeaza aceste concepte folosind o arhitectura de microservicii rulata pe Kubernetes, cu accent pe coerenta datelor si scalare orizontala.

1.2 Obiectivele proiectului

1. **Arhitectura distribuita modulara** – separarea responsabilitatilor intre servicii independente: API, notificari, frontend si workeri de fundal.
2. **Procesare evenimente in timp real** – propagarea ofertelor si a schimbarilor de status prin Redis Pub/Sub si Socket.io.
3. **Management robust al licitatiilor** – validare optimista cu versiuni, scheduling al expirarii si evidenta istorica a ofertelor.
4. **Containerizare completa** – livrare prin imagini Docker si orchestrare cu manifesturi Kubernetes adaptate pentru Minikube.
5. **Experienta utilizator moderna** – interfata React cu Material UI, filtre dinamice si feedback instant pentru utilizatori si administratori.

1.3 Functionalitati implementate

Aplicatia ofera urmatoarele capabilitati principale:

- Autentificare si inregistrare cu token JWT si roluri `user` si `admin`.
- Administrare licitatii (creare, stergere, seed de date) disponibila doar administratorilor.
- Flux de licitare cu verificare increment minim, control concurential si istoric al ofertelor.
- Notificari live catre toti clientii conectati atunci cand apar oferte, se creeaza sau se finalizeaza licitatii.
- Clasificarea licitatiilor in taburi (active, licitatiile mele, castigate, finalizate) cu sincronizare a timpului de pe server.
- Expirarea automata a licitatiilor prin coada BullMQ si worker dedicat.
- Deploy reproductibil pe Kubernetes cu Ingress unic si sesiune sticky pentru Socket.io.

2 Arhitectura sistemului

2.1 Prezentare generala

Sistemul este compus din trei servicii Node.js containerizate, o aplicatie frontend React si resurse de infrastructura (MongoDB, Redis). Comunicarea intre componente se realizeaza prin HTTP, WebSocket si canale Redis.

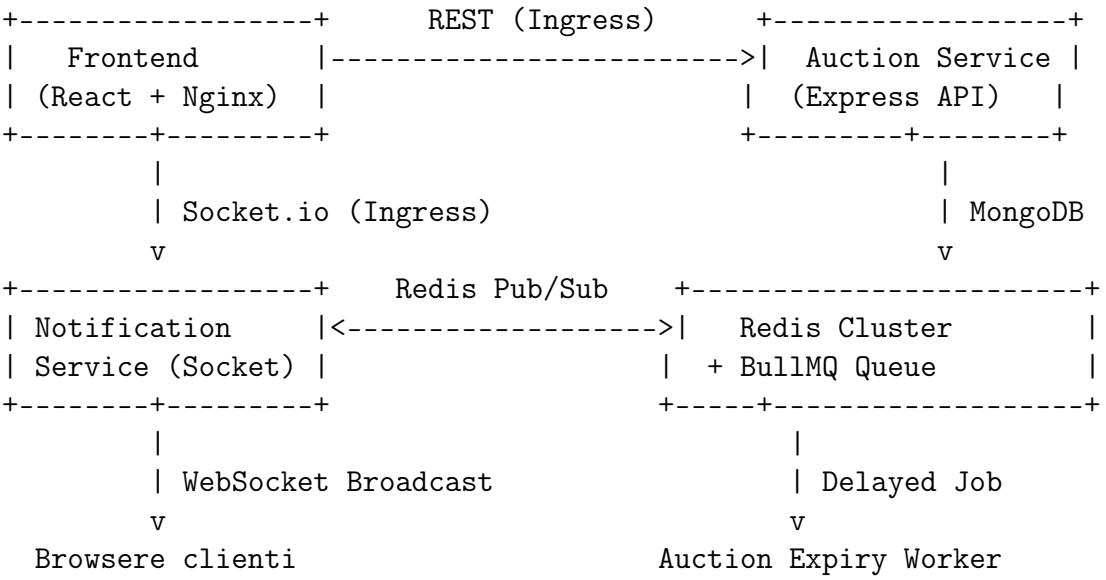

Figura 1: Arhitectura logica a sistemului

2.2 Fluxul principal de date

1. Utilizatorii acceseaza interfata web prin Ingress-ul Kubernetes care redirectioneaza traficul catre serviciul frontend.
2. Actiunile REST (autentificare, listare, creare si licitare) sunt trimise catre `/api/*` si ajung la Auction Service.
3. Auction Service valideaza licitatia, persista datele in MongoDB, programeaza expirarea in BullMQ si publica evenimente in canalul `auction.events.v1` din Redis.
4. Notification Service este abonat la canal si translateaza evenimentele in mesaje Socket.io (`price_update`, `auction_created`, `auction_ended`).
5. Frontend-ul primeste notificarile in timp real si actualizeaza starea locala fara a necesita refresh manual.

2.3 Diagrame suplimentare

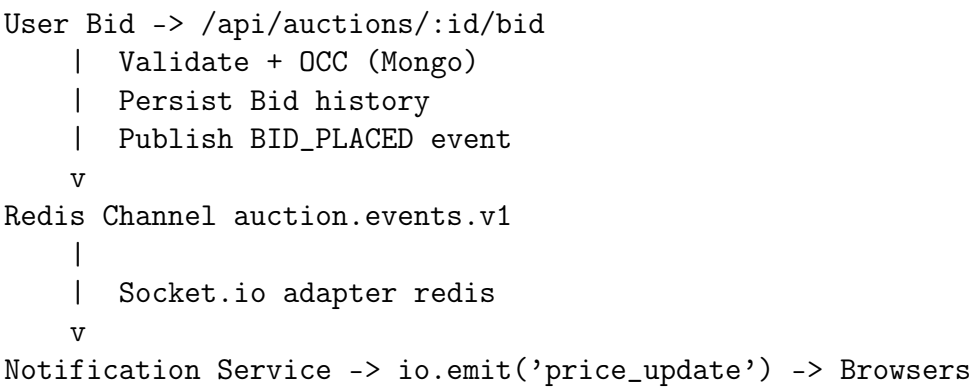

Figura 2: Fluxul unui eveniment de licitare

3 Componente si servicii

3.1 Auction Service

Responsabilitate: expune API-ul REST pentru autentificare, gestiunea licitatiilor si plasarea ofertelor. Ruleaza pe portul 3000.

Tehnologii: Node.js 18, Express 5, Mongoose 9, JWT, BullMQ, Redis client oficial.

Caracteristici cheie:

- Validare increment minim prin constanta `MIN_BID_INCREMENT` si control optimist folosind campul de versiune `__v`.
- Semantica de evenimente cu envelope semnate cu `eventId`, `traceId` si `occurredAt`.
- Endpoint-uri de healthz si readyz integrate cu probe Kubernetes.
- Creare automata a unui utilizator admin la prima pornire a serviciului.

```
const publishEvent = async (type, payload, traceId) => {
    const envelope = {
        eventId: randomUUID(),
        type,
        occurredAt: new Date().toISOString(),
        traceId,
        payload,
    };
    await redisPublisher.publish(EVENT_CHANNEL, JSON.stringify(envelope)
   );
};
```

Listing 1: Publicarea unui eveniment de licitare

3.1.1 Endpoint-uri REST

Metoda	Ruta	Descriere
POST	/api/auth/register	Creeaza cont nou si emite token JWT
POST	/api/auth/login	Autentifica utilizator si returneaza token
GET	/api/auth/me	Datele utilizatorului curent (token necesar)
GET	/api/auctions	Lista licitatiilor active
POST	/api/auctions	Creaza licitatie *(admin)*
POST	/api/auctions/:id/bid	Plaseaza o oferta *(autentificat)*
DELETE	/api/auctions/:id	Sterge licitatie *(admin)*
GET	/api/time	Timpul serverului pentru sincronizare client
POST	/api/seed	Populeaza baza cu licitatii demo (unica rulare)

Tabela 1: Endpoint-uri expuse de Auction Service

3.2 Worker-ul de expirare

Componenta `worker.js` ruleaza separat si porneste un **BullMQ Worker** conectat la aceeasi baza Redis. Acesta preia joburi din coada `auction-expiry`, marcheaza licitatiile

ca inactive si publica evenimentul `AUCTION_ENDED`. La startup ruleaza reconciliere pentru a inchide licitatiile deja expirate dar ramase active in baza.

```
const auctionExpiryQueue = new Queue(QUEUE_NAME, { connection });
const createAuctionExpiryWorker = () => new Worker(
  QUEUE_NAME,
  async (job) => {
    const { auctionId } = job.data;
    const auction = await Auction.findOneAndUpdate(
      { _id: auctionId, isActive: true },
      { isActive: false },
      { new: true }
    );
    if (!auction) return;
    await redisPublisher.publish(EVENT_CHANNEL, JSON.stringify({
      eventId: randomUUID(),
      type: 'AUCTION_ENDED',
      occurredAt: new Date().toISOString(),
      payload: {
        auctionId: auction._id,
        winner: auction.highestBidder,
        finalPrice: auction.currentPrice
      }
    }));
  },
  { connection }
);
```

Listing 2: Configurarea cozii de expirare

3.3 Notification Service

Responsabilitate: creaza conexiunea WebSocket (Socket.io) catre clientii web si traduce evenimentele Redis in notificari in timp real. Ruleaza pe portul 4000.

Tehnologii: Socket.io 4, adapter Redis, Express 5.

Functionalitati:

- Adaptor Redis pentru scalare orizontala (replici multiple sincronizate prin Pub/-Sub).
- Lista de evenimente permise cu validare de schema si dead-letter list in `auction.events.deadlett` pentru mesaje invalide.
- Broadcast periodic al timpului serverului catre toate clientele pentru sincronizarea contoarelor de expirare.

3.4 Frontend React

Aplicatia React (build Vite) ruleaza in Nginx si expune UI-ul pentru autentificare, filtrare si licitare.

- Context de autentificare cu persistenta token-ului in `localStorage` si atasarea header-ului `Authorization`.

- Tabs dinamice pentru segmentele Active, Licitatiile mele, Castigate si Finalizate, populate din starea locala.
- Formular de creare licitatie disponibil doar administratorilor.
- Notificari UI (Material UI Snackbar) pentru feedback imediat.

3.5 MongoDB si Redis

MongoDB 6: stocheaza utilizatorii, licitatiile si istoricul ofertelor. Modelele Mongoose folosesc validari si hooks de hash pentru parole (bcryptjs).

Redis 7: indeplineste doua roluri esentiale: canalul de evenimente pentru Socket.io si backend-ul BullMQ pentru joburile intarziate. Configuratia implicita din Kubernetes expune serviciul `redis-service` pe portul 6379.

3.6 Kubernetes si Ingress

Manifestele din directorul `k8s/` definesc deploy-urile applicatiei, precum si servicii pentru MongoDB si Redis. Fisierul `ingress.yaml` concentreaza traficul HTTP astfel:

- `/api/*` catre `auction-service:3000`
- `/socket.io*` catre `notification-service:4000`
- Restul rutelor catre `frontend-service:80`

Ingress-ul activeaza `cookie affinity` pentru a mentine conexiunile WebSocket pe aceeasi replica.

4 Tehnologii utilizate

4.1 Backend si runtime

Tehnologie	Versiune	Rol
Node.js	18-alpine	Runtime pentru toate serviciile backend
Express.js	5.1.0	Framework REST pentru Auction si Notification Service
Mongoose	9.0.0	ODM pentru MongoDB
BullMQ	5.65.0	Coada pentru programarea expirarii licitatiilor
Redis client	5.10.0	Pub/Sub si conexiune BullMQ
jsonwebtoken	9.0.2	Generare si verificare token JWT
bcryptjs	3.0.3	Hashing parole utilizatori
dotenv	17.2.3	Incarcare configuratii din fisiere .env

Tabela 2: Stiva backend

4.2 Frontend

Tehnologie	Versiune	Rol
React	19.2.0	Biblioteca UI component-based
Vite	7.2.4	Bundler si dev server
Material UI	7.3.5	Componente si tema vizuala
Socket.io client	4.8.1	Conexiuni WebSocket cu serverul de notificari
Axios	1.13.2	Client HTTP pentru REST API

Tabela 3: Stiva frontend

4.3 Infrastructura si DevOps

Tehnologie	Versiune	Utilizare
Docker	Latest	Containerizare servicii si frontend
Kubernetes	1.32+ (Minikube)	Orchestrare si networking
MongoDB	6.0	Baza de date document
Redis	7.0	Cache, Pub/Sub si motor BullMQ
Nginx	Alpine	Servire aplicatie React compilata

Tabela 4: Componente infrastructura

5 Fluxuri operationale

5.1 Licitarea unui produs

1. Utilizator autenticat introduce suma si trimite formularul din UI.
2. Frontend-ul realizeaza un POST catre `/api/auctions/:id/bid` cu suma si versiunea curenta.
3. Auction Service verifica starea licitatiei, primeste suma minima acceptata, executa update atomic `findOneAndUpdate` si incrementeaza versiunea.
4. Se persista un document `Bid` cu istoricul ofertei, iar evenimentul `BID_PLACED` este publicat in Redis.
5. Notification Service emite `price_update`, iar clientii isi actualizeaza imediat interfetele.

5.2 Expirarea licitatiilor

1. La creare, licitatia este programata in BullMQ cu delay egal cu timpul ramas pana la `endTime`.
2. Worker-ul de expirare marcheaza licitatia ca inactiva atunci cand jobul este executat.
3. Evenimentul `AUCTION_ENDED` este publicat pentru a notifica toti clientii si pentru a afisa castigatorul.
4. Functia de reconciliere ruleaza la pornirea worker-ului pentru a evita licitatii blocate in stari invalide.

5.3 Autentificare si autorizare

- **Inregistrare**: parola este hashuita prin bcryptjs in hook-ul `pre('save')`.
- **Login**: validare credentiale si generare token cu payload `id`, `username`, `role` si expirare 24h.
- **Protectie rute**: middleware-ul `authenticateToken` verifica token-ul din header `Authorization`, iar `requireAdmin` limiteaza actiunile privilegiate.

6 Docker si containerizare

Fiecare serviciu dispune de Dockerfile dedicat. Auction Service si Notification Service folosesc imagini `node:18-alpine`. Frontend-ul utilizeaza un build multi-stage pentru a reduce dimensiunea imaginii finale cu Nginx.

```
# Build Stage
FROM node:18-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Production Stage
FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Listing 3: Dockerfile pentru frontend

Manifestele Kubernetes configureaza `imagePullPolicy: Never` pentru a utiliza imaginile construite local in Minikube. Notification Service ruleaza cu doua replici pentru a demonstra scalarea.

7 Securitate

- Hashing parole cu `bcryptjs` si interzicerea stocarii parolelor in clar.
- Token-uri JWT semnate cu `JWT_SECRET` injectat din Secret Kubernetes.
- Endpoint-uri `/healthz` si `/readyz` non-autentificate pentru verificari automate.
- Dead-letter list pentru evenimente Redis invalide, prevenind propagarea datelor corupte catre clienti.

8 Concept distributiv si rezilienta

Arhitectura proiectului urmareste principiile unui sistem distribuit: fiecare functie majora este izolata in servicii independente care comunica prin protocoale bine definite, iar infrastructura este pregatita pentru a raspunde la variatii de trafic si la defecte partiale.

8.1 Separarea responsabilitatilor

- **Microservicii independente**: Auction Service, Notification Service si worker-ul BullMQ ruleaza ca procese distincte, permitand cicluri de release separate si scalare specifica nevoilor fiecarui serviciu.
- **Servicii stateless vs stateful**: API-ul si serviciul de notificari sunt stateless, in timp ce starea persistenta este gestionata de MongoDB si Redis; aceasta separare faciliteaza inlocuirea rapida a podurilor fara pierdere de date.

8.2 Comunicare asincrona si cuplaj redus

- **Evenimente pe canalul** `auction.events.v1`: Auction Service publica mesaje in Redis fara a cunoaste consumatorii, iar Notification Service traduce aceleasi mesaje in WebSocket, demonstrand modelul *publish/subscribe*.
- **Coada BullMQ intarziata**: Programarea expirarii licitatiilor elimina dependenta de cronuri centralizate si permite redimensionarea worker-ilor in functie de volum.

8.3 Consistenta si coordonare

- **Control optimist al concurentei**: actualizarea licitatiei foloseste versiunea documentului Mongoose pentru a preveni suprascrierea ofertelor concurente si pentru a garanta monotonia pretului curent.
- **Reconcilieri la pornire**: worker-ul verifica licitatiile expirate la restart pentru a evita stari divergente intre baza de date si coada de joburi.

8.4 Scalare orizontala

- **Socket.io cu adaptor Redis**: replicile multiple ale Notification Service partajeaza aceeasi magazie de sesiuni, astfel incat orice replica poate trimite notificari catre orice client conectat.
- **Worker-i paraleli**: BullMQ permite rularea mai multor workeri pentru a procesa simultan expirari in scenarii cu volum ridicat.

8.5 Toleranta la defecte

- **Persistenta joburilor si dead-letter**: evenimentele si joburile ramase neprocesate sunt pastrate in Redis, iar mesajele invalide sunt mutate intr-o lista dedicata pentru inspectie.
- **Probe Kubernetes**: endpoint-urile `/healthz` si `/readyz` permit orchestratorului sa elimine si sa recreeze podurile defecte fara a afecta restul clusterului.

9 Ghid de instalare si rulare

9.1 Cerintele mediului

- Docker Desktop functional.
- Minikube cu driver Docker si `kubectl` configurat.
- Node.js 18+ (pentru rulare locala a scripturilor daca este necesar).

9.2 Pasii de deploy pe Minikube

1. Pornire cluster

```
minikube start --driver=docker
minikube addons enable ingress
# Ruleaza intr-un terminal separat
minikube tunnel
```

2. Configureaza Docker-ul shell-ului curent

```
& minikube -p minikube docker-env --shell powershell | Invoke-Expression
```

3. Construieste imaginile Docker

```
docker build -t auction-service:v1 ./auction-service
docker build -t frontend:v1 ./frontend
docker build -t notification-service:v1 ./notification-service
```

4. Aplica manifestele Kubernetes

```
kubectl apply -f k8s/mongo.yaml
kubectl apply -f k8s/redis.yaml
kubectl wait --for=condition=ready pod -l app=mongo --timeout=120s
kubectl wait --for=condition=ready pod -l app=redis --timeout=120s
kubectl apply -f k8s/app.yaml
kubectl apply -f k8s/ingress.yaml
```

5. Verifica statusul

```
kubectl get pods -A
kubectl get ingress
```

6. Acces aplicatia

Aplicatia este disponibila la adresa `http://localhost`. Pentru administrare foloseste `admin / admin123` (creat automat la pornire).

9.3 Actualizari dupa modificari

1. Ruleaza comanda de configurare Docker pentru Minikube in fiecare terminal nou.
2. Reconstruieste imaginea serviciului modificat cu `-no-cache` daca este nevoie.
3. Ruleaza `kubectl rollout restart deployment <nume>` pentru a relansa podurile.

10 Utilizare

10.1 Flux administrator

1. Autentificare cu contul administrativ.
2. Creare licitatii noi din butonul **Adauga Licitatie**, alegand titlu, pret de start si durata.
3. Monitorizare ofertelor si, la nevoie, stergerea licitatiilor direct din card.

10.2 Flux utilizator standard

1. Creare cont prin formularul de inregistrare sau logare cu date existente.
2. Vizualizare licitatii active si plasare de oferte cu suma dorita (respectand incrementul minim).
3. Urmarirea in timp real a statutului licitatiei si consultarea sectiunilor "Licitatiile mele" si "Castigate".

11 Structura proiectului

```
Notificari/
|-- README.md
|-- documentatie_licitatii.tex
|-- auction-service/
|   |-- Dockerfile
|   |-- index.js
|   |-- worker.js
|   |-- package.json
|   |-- config/
|   |   `-- redisClient.js
|   |-- middleware/
|   |   `-- auth.js
|   |-- models/
|   |   |-- Auction.js
|   |   |-- Bid.js
|   |   `-- User.js
|   `-- queue/
|       `-- auctionQueue.js
|-- frontend/
|   |-- Dockerfile
|   |-- package.json
|   `-- src/
|       |-- App.jsx
|       |-- main.jsx
|       |-- context/
|       |   `-- AuthContext.jsx
|       `-- pages/
|           `-- LoginPage.jsx
|-- notification-service/
|   |-- Dockerfile
|   |-- index.js
```

```
|   `-- package.json
`-- k8s/
    |-- app.yaml
    |-- config-secrets.yaml
    |-- ingress.yaml
    |-- mongo.yaml
    |-- redis.yaml
    `-- notification-sevice.yaml
```

Listing 4: Structura directoarelor

12 Concluzii

12.1 Rezultate obtinute

- Implementare completa a unei platforme de licitatii cu arhitectura distribuita si scalabila.
- Fluxuri de evenimente robuste bazate pe Redis si Socket.io, cu tratament al erorilor prin dead-letter list.
- Experienta utilizator moderna, reactive UI si sincronizare corecta a timpului de expirare.

12.2 Provocari intampinate

1. Managementul coerentei licitatiilor in scenarii cu multe oferte simultane a necesitat implementarea controlului optimist al concurentei.
2. Asigurarea compatibilitatii Socket.io in spatele Ingress-ului Kubernetes a impus configurarea cookie affinity si a path-ului dedicat.
3. Sincronizarea joburilor intarziate cu starea reala a bazei a fost rezolvata prin reconciliere la pornire.

12.3 Directii viitoare

- Integrare cu servicii de e-mail sau push notifications pentru alertarea castigatorilor.
- Persistenta datelor de audit in Azure Application Insights sau ELK pentru analiza istorica.
- Implementarea de licitatii cu preturi dinamice si suport multi-valuta.
- Adaugarea de teste automate (unitare si end-to-end) si pipeline CI/CD.

13 Bibliografie

1. Documentatia Docker – https://docs.docker.com/
2. Kubernetes Documentation – https://kubernetes.io/docs/home/
3. Socket.io Documentation – https://socket.io/docs/v4/
4. Redis Documentation – https://redis.io/docs/latest/
5. BullMQ Guide – https://docs.bullmq.io/
6. MongoDB Manual – https://www.mongodb.com/docs/manual/
7. React Documentation – https://react.dev/
8. Material UI – https://mui.com/