MASTER’S DEGREE IN COMPUTER ENGINEERING

DISTRIBUTED SYSTEMS AND MIDDLEWARE TECHNOLOGIES

PROJECT DISCUSSION

WHENLY

DISTRIBUTED EVENT SCHEDULING

PROFESSORS

ALESSIO **BECHINI**
JOSÉ LUIS CORCUERA **BÁRCENA**

STUDENTS

ALESSANDRO **ASCANI**
GIOVANNI **LIGATO**
GIOVANNI ENRICO **LONI**

IN SUPREMÆ DIGNITATIS
1343

UNIVERSITY OF PISA

INTRODUCTION

- Scheduling events with conflicting participant availability is challenging.
- Existing tools often lack scalability, robustness, and effective constraint handling.
- **Whenly** addresses these issues with a distributed architecture that ensures high availability and efficient computation.

A SCALABLE, FAULT-TOLERANT SOLUTION FOR MANAGING GROUP EVENT SCHEDULES

HOW WHENLY WORKS?

EVENT ORGANIZERS
create events with deadlines.

PARTICIPANTS
submit their availability constraints.

THE SYSTEM
computes partial solutions dynamically.

AT THE DEADLINE
a final schedule is derived by reconciling all constraints.

FUNCTIONAL REQUIREMENTS

USER-LEVEL

USER AUTHENTICATION:

- Secure sign-up.
- Log-in, and log-out.

EVENT MANAGEMENT:

- Organizers create events with deadlines and specific constraints.
- Participants join events via shared IDs and add their constraints.

CONSTRAINT SUBMISSION & SCHEDULING:

- Submit preferred time slots.
- Final schedule computed when the deadline expires.
- Visual feedback provided for successful scheduling or errors.

FUNCTIONAL REQUIREMENTS

SYSTEM-LEVEL

DISTRIBUTED TASK ALLOCATION

Backend distributes constraints to Event Server Nodes in a circular manner.

INCREMENTAL UPDATES

Each node updates partial solutions as new constraints arrive.

FINAL SCHEDULE COMPUTATION

The node managing the deadline aggregates partial solutions and computes the final schedule.

PERSISTENT STORAGE

User data, event metadata, constraints, and final schedules are stored reliably.

NON-FUNCTIONAL REQUIREMENTS

PERFORMANCE

- <1 second response time for WebApp requests.
- Final schedules available within 1 minute of the deadline.

SECURITY

- Secure storage of user credentials with hashing.

SCALABILITY

- Horizontally scalable by adding more Event Server Nodes.

AVAILABILITY

- 99.9% uptime.

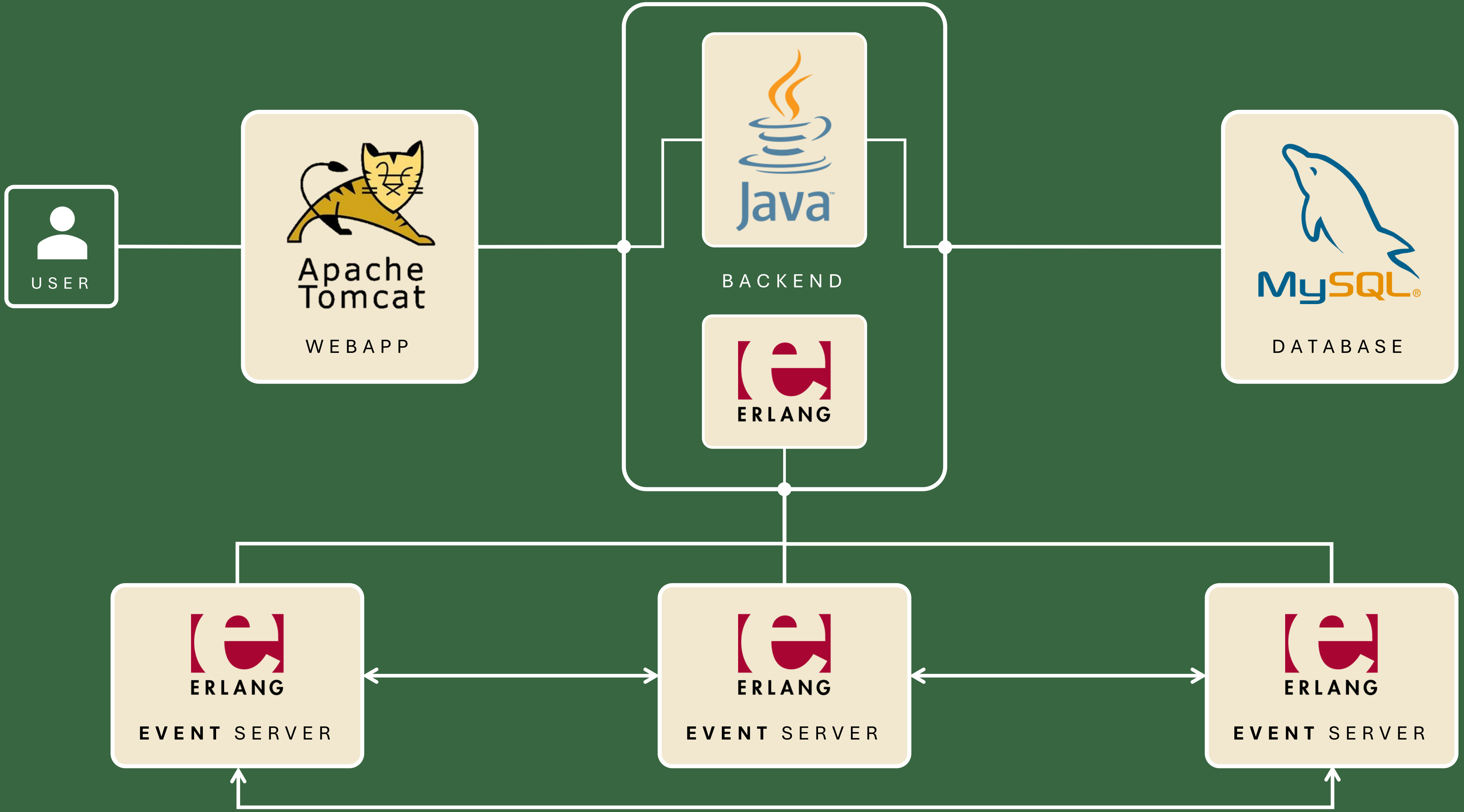
FAULT TOLERANCE

- No data loss in case of node failure; automatic task reassignment.
- Consistent backups via the Database Node.

USABILITY

- Intuitive, responsive web interface across devices.

ARCHITECTURE OVERVIEW



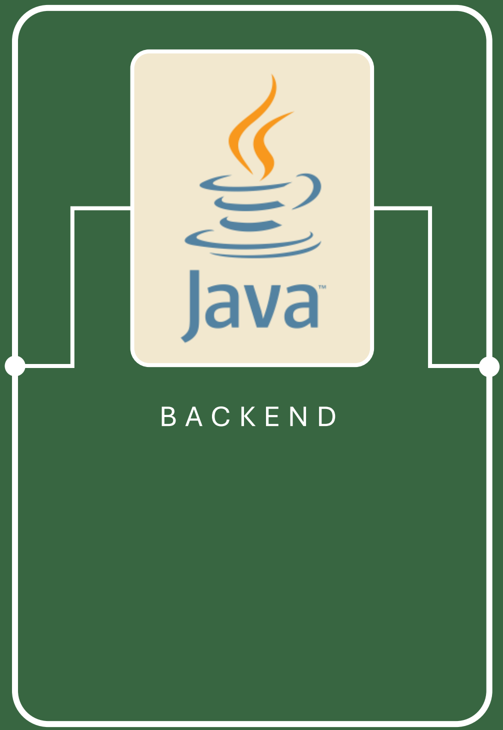
WEBAPP NODE

WEBAPP

- **Frontend Implementation:**
  - Java Server Pages (JSP) on Apache Tomcat.
- **Communication:**
  - Uses servlets to send HTTP requests to the Backend REST APIs.
- **Key Functions:**
  - Homepage: Login/registration handling.
  - User Area: Display events in tabular format.
  - Event Management: Create events & submit constraints (timestamps: `YYYY-MM-DD HH:mm`).

BACKEND NODE

JAVA BACKEND COMPONENT



- **Framework:**
  - Spring Boot (REST APIs).

- **Responsibilities:**
  - User authentication & registration.
  - Event creation & constraint management.
  - Distributes tasks in a circular manner to Event Server Nodes.
  - Uses JInterface for Java ↔ Erlang communication.

BACKEND NODE

ERLANG BACKEND COMPONENT

- **Role:**
  - Intermediary between Java Backend and distributed Event Server Nodes.
- **Features:**
  - Exposes APIs for event & constraint requests.
  - Monitors Event Server Nodes for dynamic load balancing & recovery.
  - Receives final scheduling results for forwarding to Java Backend.
- **Implementation:**
  - OTP application (rebar3).

DATABASE NODE

- **Technology:**
  - MySQL Database.
- **Stores:**
  - User credentials.
  - Event metadata (ID, creator, deadline, designated Erlang node, event name, final result) .
  - Constraint records (event_id, username, assigned Erlang node, constraints list).
- **Purpose:**
  - Ensures data integrity & supports recovery processes.

EVENT SERVER NODES

- **Deployment:**
  - 3 distinct nodes (sharing cookie `whenly`).
- **Built Using:**
  - OTP and rebar3.
- **Key Modules:**
  - `storage` **Module:**
    - Uses Mnesia DB (`local_content`) to store `event_record` and `event_deadline`.
  - `base` **Module:**
    - Entry point for event creation & constraint requests.
  - `coordinator` **Module:**
    - Manages deadlines and aggregates partial solutions via RPC calls.
  - **Utility Modules:**
    - `calculator`.
    - `config_reader`.

SYSTEM WORKFLOW

1

USER INTERACTION

Users log in, create events, and submit constraints via the WebApp (JSP & servlets).

2

BACKEND PROCESSING

Backend authenticates users, creates events, distributes tasks to Event Server Nodes in a circular fashion, and persists data in MySQL.

3

CONSTRAINT PROCESSING

Event Server Nodes update partial solutions incrementally as the constraints arrive.

4

FINAL SCHEDULING

At the event deadline, the designated node aggregates partial solutions and computes the final schedule.

5

RESULT STORAGE

The final schedule is sent to the Backend and stored in MySQL for user access via the User Area.

SYNCHRONIZATION, COORDINATION & COMMUNICATION ISSUES

CHALLENGES

DEADLINE EXPIRY OVERLAP

Ongoing updates must halt when the event deadline is reached.

CONCURRENT ACCESS TO SHARED ERLANG NODE LIST

Simultaneous read/write operations by different services may cause race conditions.

SYNCHRONIZATION SOLUTIONS

ATOMIC UPDATES VIA MNESIA DB

Use transactions to fully apply or rollback updates.

DEADLINE EXPIRATION MECHANISM

Initiate RPC calls at deadline; mark events as `expired` to ignore late constraints.

SHARED LIST CONTROL

Use the `sharedStringList` module with `ReentrantLock` to serialize modifications.

CHALLENGES

DEADLINE EXECUTION OWNERSHIP

The node receiving the event creation request must coordinate scheduling.

FAILURE HANDLING DURING COORDINATION

Reassign responsibilities quickly if the deadline owner fails.

COORDINATION SOLUTIONS

PREDEFINED DEADLINE OWNERSHIP

Designate the originating node as the deadline owner.

DISTRIBUTED SCHEDULING

Aggregate partial solutions from all nodes for final computation.

FAILURE RECOVERY VIA MONITORING

Use `net_kernel:monitor_nodes(true)` and `base` Module connections to trigger recovery.

CHALLENGES

NETWORK LATENCY

Delays can impact system responsiveness.

MESSAGE DUPLICATION/LOSS

Risk of unreliable data exchange.

SCALABILITY

Increasing nodes can complicate message management.

COMMUNICATION

SOLUTIONS

RELIABLE MESSAGING PROTOCOL

Utilize Erlang's built-in messaging system.

ASYNCHRONOUS COMMUNICATION

Handle Java–Erlang messaging asynchronously using JInterface.

DEFINED PROTOCOLS

- **WebApp ↔ Backend**: HTTP/REST
- **Java Backend ↔ Erlang Backend**: JInterface
- **Erlang Backend ↔ Event Servers**: RPC calls
- **Event Server ↔ Event Server**: RPC calls

RECOVERY MECHANISMS

Two main recovery paths:

1. **Erlang Node** Failure Recovery
2. **Intra-Node** Process Recovery

ERLANG NODE FAILURE RECOVERY

NODE FAILURE DETECTION

- `ErlangBackendAPI` notifies `EventService` via `ErlangMessageHandler`.
- Remove failed node from `sharedStringList`.

CONSTRAINT RECOVERY

- Retrieve constraints assigned to the failed node.
- Reassign constraints to an active node.
- Update the database.

+

EVENT RECOVERY

- Retrieve events without final results assigned to failed node.
- Reassign events to an active node.
- Update the database.

ERROR HANDLING & SCALABILITY

- Throw an exception if no nodes are available.
- Process is transparent to users and other event server nodes.

INTRA-NODE PROCESS RECOVERY

OTP SUPERVISION

A `one_for_one` strategy auto-restarts failed processes.

COORDINATOR PROCESS

- In `init`, retrieve stored deadlines via the `storage` Module.
- Reschedule expiration events using `send_after`.
- At the event deadline, if partial solutions from other nodes are `expired`, skip final computation.

CRITICAL RECOVERY

ENSURES

Continuous recovery of event state even after process crashes.

CONCLUSION

Integrates Java and Erlang for scalability and fault tolerance.

Employs strong synchronization, coordination, and recovery mechanisms.

Achieves high availability, ensuring a seamless user experience.

WHENLY DELIVERS A ROBUST SOLUTION FOR DISTRIBUTED EVENT SCHEDULING.

TOOLS

PALETTE

coolors.co/palette/386641-6a994e-a7c957-f2e8cf-bc4749

ICONS

www.flaticon.com

THANKS

FOR YOUR ATTENTION