Architectural Style Overall System: Service-Oriented Architecture (SOA)

By Nicolas Ceron

Frontend

- **Platform**: Android
- **Language**: Kotlin
- **Architecture Pattern**: MVC (Model-View-Controller)

Backend

- **Platform**: JakartaEE
- **Language**: Java
- **Architecture Pattern**: Layered Architecture

Integration

- **Protocol**: SOAP
- **Server**: Glassfish

Database

- **System**: Oracle

Service-Oriented Architecture (SOA)

1.1 Definition

An architectural style using discrete, reusable software "services" communicating over a network via standardized interfaces, often to integrate larger, enterprise-level functions (AWS, n.d.; IBM, n.d.).

1.2 Core Characteristics

- Emphasizes loose coupling through service contracts, abstraction, reusability, and interoperability using standards (AWS, n.d.; GeeksforGeeks, n.d.-a).

1.3 History and Evolution

Emerged late 90s from distributed object concepts, popularized by web services (SOAP/WSDL), often using ESBs, and influenced microservices (IBM, n.d.; Graca, 2017a; DZone, n.d.).

1.4 Advantages and Disadvantages

- Offers increased agility and maintainability but faces challenges with initial investment, complexity, and potential performance overhead (AWS, n.d.; GeeksforGeeks, n.d.-a; Graca, 2017a).

1.5 Use Cases (Situations/Problems Solved)

- Primarily used for Enterprise Application Integration (EAI), legacy system modernization, business process automation, and promoting reuse across large organizations (IBM, n.d.; AWS, n.d.).

1.6 Application Cases (Industry Examples)

- Implemented by large organizations like Cisco (ordering), Independence Blue Cross (patient data), and in military systems for integration and consistency (IBM, n.d.; GeeksforGeeks, n.d.-a).

Section 2: Android Platform (Frontend)

2.1 Definition

A mobile OS based on a modified Linux kernel and AOSP open-source software, led by Google, primarily for touchscreen devices (Wikipedia, n.d.-a).

2.2 Core Characteristics

- Features open-source core (AOSP), Linux kernel, ART runtime for app execution, extensive Java/Kotlin API framework, and high customizability (Wikipedia, n.d.-a; GeeksforGeeks, n.d.-b; Android Developers, n.d.-a).

2.3 History and Evolution

Acquired by Google (2005), launched 2008, evolved rapidly through versions adding key features, unifying UI (ICS), improving performance/design (Material), and now focuses on privacy/security with numerical versions (Wikipedia, n.d.-a; Velvetech, n.d.).

2.4 Advantages and Disadvantages

- Benefits from openness, huge user base, and hardware choice, but suffers from fragmentation, inconsistent/delayed updates, and related security concerns (Wikipedia, n.d.-a; SearchMyExpert, n.d.).

2.5 Use Cases (Situations/Problems Solved)

- Dominant platform for smartphones/tablets; also used in wearables, automotive, TV, IoT, enterprise mobility, and specialized electronics (Wikipedia, n.d.-a).

2.6 Application Cases (Industry Examples)

- Hosts nearly all major mobile applications across social media, communication,

entertainment, productivity, e-commerce, finance, etc. (e.g., Facebook, Netflix, Office, banking apps) (Wikipedia, n.d.-a).

Section 3: Kotlin Language (for Android)

3.1 Definition

A modern, statically typed, cross-platform language from JetBrains, fully interoperable with Java/JVM, also targeting JS/Native (Wikipedia, n.d.-b).

3.2 Core Characteristics

- Known for conciseness, null safety, seamless Java interoperability, coroutines for async programming, and functional features (Wikipedia, n.d.-b; Kotlin language specification, n.d.; KotlinLang.org, n.d.-a).

3.3 History and Evolution

Created by JetBrains (unveiled 2011), gained massive traction after Google's 2017 Android support announcement, becoming the preferred language in 2019 (Wikipedia, n.d.-b; Linares-Vásquez et al., 2019).

3.4 Advantages and Disadvantages (vs. Java for Android)

- Offers improved safety, conciseness, and async handling (coroutines) over Java for Android, with minor trade-offs in learning curve and historically compile times (Wikipedia, n.d.-b; KotlinLang.org, n.d.-a).

3.5 Use Cases (Where Kotlin Excels)

- Primary choice for Android development; also strong for server-side JVM applications, multiplatform projects (KMP), and building libraries/DSLs (Wikipedia, n.d.-b).

3.6 Application Cases (Industry Examples)

- Used extensively in Android apps (Google, Airbnb, Netflix), KMP projects (McDonald's, Philips), server-side systems (AWS, Adobe), and JetBrains' own tools (Daily.dev, n.d.; KotlinLang.org, n.d.-a).

Section 4: Model-View-Controller (MVC) Pattern (for Android/Kotlin)

4.1 Definition

Architectural pattern separating Model (data/logic), View (UI), and Controller (input/mediation) (StudySmarter, n.d.). On Android, Activity/Fragment often problematically acts as Controller+View (GeeksforGeeks, n.d.-e).

4.2 Core Characteristics

- Aims for separation of concerns, but common Android interpretation leads to tight coupling

between View and Controller (AngularMinds, n.d.; GeeksforGeeks, n.d.-e).

4.3 History and Evolution

Originated in Smalltalk, widely used for GUI/web, but limitations on mobile (tight coupling, testability issues) led to MVP and MVVM patterns (Wikipedia, n.d.-c; Daily.dev, n.d.).

4.4 Advantages and Disadvantages (in Android/Kotlin Context)

- Conceptually simple but severely hampered on Android by tight coupling and poor testability, making maintenance difficult; largely superseded by MVVM (Daily.dev, n.d.; GeeksforGeeks, n.d.-e; Miquido, n.d.).

4.5 Use Cases (Suitable Situations)

- Generally discouraged for modern Android; potentially acceptable only for trivial apps, prototypes, or basic educational examples (Daily.dev, n.d.).

4.6 Application Cases (Implementation Examples in Android/Kotlin)

- Represents an early, now outdated, approach where Android Activities handled UI, input, and model interaction directly; modern apps use MVVM/MVI (Daily.dev, n.d.).

Section 5: Jakarta EE Platform (Backend)

5.1 Definition

A set of Java specifications extending Java SE for enterprise applications, providing APIs for persistence, web services, messaging, etc., running on compatible servers (Wikipedia, n.d.-d). Now managed by Eclipse Foundation.

5.2 Core Characteristics / Key Specifications

- Specification-driven, uses component-container model, offers profiles (Platform, Web, Core), key APIs include JAX-RS, CDI, JPA, JMS (Wikipedia, n.d.-d; Jakarta.ee, n.d.-a; Payara Blog, n.d.).

5.3 History and Evolution

Evolved from complex J2EE through simpler Java EE versions (EE5+) to community-driven Jakarta EE, involving a major namespace change (EE9) and focus on modernization (EE10+) (Wikipedia, n.d.-d; Jakarta.ee, n.d.-a; The Eclipse Foundation, n.d.; Payara Blog, n.d.).

5.4 Advantages and Disadvantages

- Provides standardization, comprehensive features, and mature ecosystem, but can be perceived as complex/heavyweight with potential runtime overhead compared to alternatives (Wikipedia, n.d.-d; The Enterprisers Project, n.d.; OmniFish, n.d.-a).

5.5 Use Cases (Situations/Problems Solved)

- Suitable for large-scale, transactional, secure enterprise systems, integration tasks, web services, and increasingly microservices (using specific profiles/runtimes) (Wikipedia, n.d.-d; The Enterprisers Project, n.d.).

5.6 Application Cases (Industry Examples)

- Underpins backend systems in finance, telecommunications, retail, government, and healthcare, often running on servers like WebSphere, JBoss EAP, WildFly (The Enterprisers Project, n.d.; Devart, n.d.; TheirStack.com, n.d.).

Section 6: Java Language (for Backend/Jakarta EE)

6.1 Definition

Mature, object-oriented, platform-independent ("WORA") language using the JVM, foundational for Jakarta EE (Wikipedia, n.d.-e; Oracle, n.d.-a).

6.2 Core Characteristics

- Features static typing, robustness (GC, exceptions), security focus, high performance (JIT), multithreading, and a vast ecosystem/standard library (Wikipedia, n.d.-e; Oracle, n.d.-a; The Server Side, n.d.).

6.3 History and Evolution

Released 1995, evolved significantly with key versions (Java 5, 8 adding major features), open-sourced (OpenJDK), now follows faster release cadence with LTS versions (Unstop, n.d.; The Server Side, n.d.).

6.4 Advantages and Disadvantages (for Backend/Jakarta EE)

- Strengths include JVM maturity, huge ecosystem, performance, robustness, and backward compatibility; weaknesses include verbosity, memory footprint, and startup time vs. some alternatives (Oracle, n.d.-a; Softjourn, n.d.; AltexSoft, n.d.).

6.5 Use Cases (in Server-Side Development)

- Dominant in enterprise backends, APIs, microservices, big data frameworks (Hadoop/Spark/Kafka), FinTech, and e-commerce platforms (Softjourn, n.d.; AltexSoft, n.d.).

6.6 Application Cases (Large-Scale Systems Using Java)

- Powers significant parts of major internet services (Google, Amazon, Netflix), financial institutions, retail systems, Android OS, and numerous open-source projects.

Section 7: Layered Architecture Pattern (for Backend)

7.1 Definition

Fundamental pattern structuring applications into horizontal layers (e.g., Presentation, Business, Persistence) based on technical responsibility (O'Reilly, n.d.; DEV Community, n.d.-a).

7.2 Core Characteristics

- Focuses on separation of concerns and isolation between layers, typically with downward dependencies (Exatosoftware, n.d.; O'Reilly, n.d.; Graca, 2017b).

7.3 History and Evolution

An early software design concept, standardized in client-server/web eras, influenced more modern domain-centric patterns like Clean/Onion Architecture (ResearchGate, n.d.; O'Reilly, n.d.; Exatosoftware, n.d.).

7.4 Advantages and Disadvantages

- Enhances maintainability and testability but can introduce performance overhead and complexity, potentially leading to monolithic deployments or tight coupling if not managed well (Exatosoftware, n.d.; O'Reilly, n.d.).

7.5 Use Cases (Suitable Situations)

- Common for standard enterprise applications, systems prioritizing maintainability, and often used as a default starting architecture (DEV Community, n.d.-a; O'Reilly, n.d.).

7.6 Application Cases (Implementation Examples)

- Forms the structural basis for countless backends using Jakarta EE, .NET, Python/Django, mapping framework components to layers (O'Reilly, n.d.; Baeldung, n.d.).

Section 8: Supporting Technologies (Integration and Database)

8.1 SOAP Protocol

- **Definition:** Standardized XML-based messaging protocol for web services, transport-neutral but often uses HTTP(S) (CIO Wiki, n.d.; PhoenixNAP, n.d.).
- **Core Characteristics:** Uses XML (Envelope/Header/Body/Fault), standardized (W3C), extensible via WS-* standards (security, reliability) (PhoenixNAP, n.d.; W3C, n.d.-b).
- **History:** Developed late 90s, standardized early 2000s, WS-* added enterprise features; less favored than REST for new public APIs (CIO Wiki, n.d.; Raygun, n.d.).
- **Adv/Disadv:** Strong standardization, extensibility, built-in error handling vs. complexity, verbosity, performance overhead (PhoenixNAP, n.d.; Raygun, n.d.).
- **Use Cases:** EAI, B2B integration, financial services, legacy systems needing formal contracts or high reliability/security via WS-* extensions (PhoenixNAP, n.d.).
- **Application Cases:** Used in payment gateways, legacy booking systems, enterprise CRM/ERP APIs (Salesforce, SAP) (Raygun, n.d.; PhoenixNAP, n.d.).

8.2 Glassfish Server

- **Definition:** Open-source Jakarta EE application server, serves as the official Reference Implementation (RI) (Wikipedia, n.d.-g; eG Innovations, n.d.).
- **Core Characteristics:** Full Jakarta EE compliance, open source, comprehensive admin tools, clustering/HA support (Redress Compliance, n.d.-a; eG Innovations, n.d.).
- **History:** Originated as Sun's RI, moved to Eclipse, implements successive Jakarta EE versions (Wikipedia, n.d.-g; Oracle, n.d.-b).
- **Adv/Disadv:** Guarantees RI compliance, free/open source vs. complexity, resource usage, and often lower production adoption than competitors (Redress Compliance, n.d.-a, n.d.-b; OmniFish, n.d.-b).
- **Use Cases:** Primarily for development/testing against Jakarta EE standards, deploying full-platform EE apps, educational purposes (Redress Compliance, n.d.-a; eG Innovations, n.d.).
- **Application Cases:** Used in official tutorials, some SMEs, and potentially specific company contexts or product demos (Oracle Help Center, n.d.-c; TheirStack.com, n.d.; cdcgroup.com, n.d.).

8.3 Oracle Database System

- **Definition:** Leading enterprise multi-model Relational (and Object-Relational) DBMS known for performance, scalability, HA, and security (The Knowledge Academy, n.d.; Oracle Help Center, n.d.-a).
- **Core Characteristics:** Robust SQL/PL/SQL support, advanced scalability/HA (RAC, Data Guard), strong security, ACID compliance, manageability tools, Multitenant architecture (The Knowledge Academy, n.d.; Devart, n.d.).
- **History:** Long history since 1979, continuously adding major features like RAC, partitioning, In-Memory, Multitenant, and Autonomous capabilities (The Knowledge Academy, n.d.; Oracle Help Center, n.d.-b).
- **Adv/Disadv:** Top-tier performance, reliability, features vs. high cost, complexity, and potential vendor lock-in (The NineHertz, n.d.; HashMicro, n.d.).
- **Use Cases:** Mission-critical OLTP and data warehousing, large-scale enterprise systems requiring highest levels of performance, availability, and security (The Knowledge Academy, n.d.).
- **Application Cases:** Underpins core operations for countless major global corporations across finance, retail, telecom, etc.

Section 9: Relationship Between Topics

This stack integrates a modern Android/Kotlin frontend (using the dated MVC pattern) with an enterprise Java backend (Jakarta EE/Layered Arch) via SOAP web services hosted on Glassfish, all persisted in an Oracle database. SOA principles likely inform the backend service design.

Section 10: How Common is the Designated Stack?

While individual components like Android/Kotlin, Java/Jakarta EE, Layered Architecture, and Oracle DB are very common in their domains, this *specific combination* is **less common for new projects**. MVC on Android is outdated, SOAP is less preferred for new mobile APIs than REST, and Glassfish has lower production usage than alternatives. It's plausible mainly when integrating a modern frontend with existing legacy Java EE/SOAP backend infrastructure.

Matrices

A. SOLID Principles Analysis Matrix

Component/Pattern	SRP (Single Resp.)	OCP (Open/Closed)	LSP (Liskov Sub.)	ISP (Interface Seg.)	DIP (Dependency Inv.)
SOA (Principles)	Supports (Service=Capability)	Supports (Extend via services)	Contextual (Contract adherence)	Supports (Contract=Interface, maybe large)	Supports (Depends on contracts)
Android (Kotlin)	Facilitated (Kotlin features)	Facilitated (Kotlin extensions, IFs)	Facilitated (Kotlin null safety)	Facilitated (Kotlin interfaces)	Facilitated (Kotlin IFs, DI)
MVC (on Android)	Challenged (Controller overload)	Challenged (Controller modification)	Contextual	Challenged (Large implicit IFs)	Challenged (Direct C->V, C->M)
Jakarta EE (Platform)	Promotes (Components, CDI)	Promotes (Extensibility, Interceptors)	Supports (Std. interfaces, e.g., JPA)	Supports (Specific APIs)	Strongly Supports (CDI)
Java (Language)	Enables (OO, access mods)	Enables (IFs, Abstract Classes)	Enables (Inheritance, IFs)	Enables (Interfaces)	Enables (IFs, DI frameworks)
Layered Architecture	Supports (Layer=Responsibility)	Contextual (Extension vs Mod.)	Contextual (Depends on IFs)	Contextual (IFs between layers maybe large)	Challenged (Traditional Top-Down)
SOAP (Protocol)	N/A (Service design issue)	Supports (Extend via Headers/WS-*)	N/A (Service design issue)	N/A (Service design issue, WSDL=Interface)	N/A (Service design issue, WSDL=Abstract.)

Glassfish (Runtime)	N/A (Provides env.)	N/A (Provides env.)	N/A (Provides env.)	N/A (Provides env.)	N/A (Provides env. via CDI)
Oracle DB (System)	N/A (Applies to Stored Procs/Interact.)	N/A	N/A	N/A	N/A

B. Quality Attributes Impact Matrix

Component	**Performance**	**Security**	**Maintainability**	**Scalability**	**Reliability/Avail.**	**Testability**	**Interoperability**	**Usability (Dev/End)**
SOA (Principles)	Negative	Mixed/ Contextual	Positive	Mixed/ Contextual	Mixed/ Contextual	Mixed/ Contextual	Very Positive	Mixed (Dev) / NA (End)
Android OS	Positive	Positive (Platform)	Mixed (Framework)	Positive (Platform)	Positive (Platform)	Mixed (Framework)	Positive (Ecosystem)	Very Pos (End) / Pos (Dev)
Kotlin (Language)	Positive	Positive (Null safety)	Very Positive	Positive (Coroutines)	Positive	Positive	Very Pos (w/ Java)	Very Pos (Dev) / NA (End)
MVC (on Android)	Mixed/ Contextual	Neutral	Very Negative (Complex)	Very Negative (Complex)	Negative	Very Negative	Neutral	Pos (Initial)/ Neg (Dev)
Jakarta EE (Plat.)	Mixed/ Contextual	Very Positive	Positive	Positive	Positive	Positive	Very Positive	Positive (Dev) / NA (End)
Java (Langu	Positive	Positive	Positive	Positive	Positive	Positive	Very Positive	Positive (Dev) /

age)								NA (End)
Layered Arch.	Negative	Positive	Positive (Intra-layer)	Mixed (Monolith)	Mixed/ Contextual	Positive	Neutral	Positive (Dev) / NA (End)
SOAP (Protocol)	Very Negative	Positive (with WS-Sec)	Mixed/ Contextual	Mixed/ Contextual	Positive (with WS-Rel)	Mixed/ Contextual	Very Positive	Negative (Dev) / NA (End)
Glassfish (Runtime)	Mixed/ Contextual	Positive	Mixed/ Contextual	Positive (Cluster)	Positive (Cluster)	N/A	Very Positive	Positive (Dev) / NA (End)
Oracle DB (System)	Very Positive	Very Positive	Very Negative (Complex)	Very Positive (RAC/Shard)	Very Positive (RAC/ DG)	N/A	Very Positive	Negative (Dev) / NA (End)

C. Architectural Tactics Support Matrix

Component	**Availability (Redundancy, Failover, Isolate)**	**Performance (Cache, Pool, Concurrency)**	**Modifiability (Encaps., Interfaces, DI)**	**Security (AuthN/Z, Encrypt)**	**Testability (Mocking, Isolate)**
SOA (Principles)	Supports (Isolate), Contextual (Redund./Failover)	Hinders (Latency), Supports (Async)	Supports (Interfaces, Encaps.)	Supports (Per-service)	Contextual (Isolate vs Int.)
Android OS	Managed by OS	Supports (ART, Threads)	Framework (Components)	Supports (Perms, Sandbox)	Hinders (Framework Dep.)
Kotlin (Language)	Supports (Coroutines)	Supports (Coroutines, Inline)	Supports (OO, Functional)	Supports (Null Safety)	Supports (Testing Frameworks)

MVC (on Android)	Hinders	Hinders	Hinders (Coupling)	Hinders	Hinders (Controller)
Jakarta EE (Plat.)	Supports (Cluster, Tx)	Supports (Pool, JPA Cache, Concurr.)	Strongly Supports (CDI, Interfaces)	Strongly Supports (Security)	Supports (CDI Mocks)
Java (Language)	Supports (Threads, Exceptions)	Supports (Threads, JIT, GC)	Supports (OO, Interfaces)	Supports (Security Mgr)	Supports (JUnit, Mocking)
Layered Arch.	Supports (Isolation)	Hinders (Latency)	Supports (Separation of Concerns)	Supports (Layer Boundaries)	Supports (Layer Isolation)
SOAP (Protocol)	Supports (WS-Rel, Faults)	Hinders (XML), Supports (MTOM)	Supports (WSDL Contract, Extend)	Strongly Supports (WS-Sec)	Contextual (Requires Tools)
Glassfish (Runtime)	Strongly Supports (Cluster, Failover)	Supports (Pools, Tuning)	Supports (Modularity, Hot Deploy)	Supports (Jakarta EE Sec.)	Facilitates (Test Env.)
Oracle DB (System)	Strongly Supports (RAC, DG, RMAN)	Strongly Supports (Cache, Index, IM)	Supports (PL/SQL, Part.)	Strongly Supports (TDE, RBAC)	N/A

D. Patterns Matrix

Component	**Microservices**	**EDA**	**API Gateway / ESB**	**Observer**	**Factory/ Singleton**	**Repository / DAO**	**DI / IoC**	**ORM (JPA)**	**Others**
SOA (Principles)	Precursor	Complementary	ESB (Common)	Possible	N/A	N/A	Via Contracts	N/A	Service Contract, Discovery

Android (Kotlin/MVC)	N/A (Client)	Possible (Event Bus)	N/A	Key (UI)	Common	Common (Repo)	Key (Hilt)	Room (ORM-like)	MVP, MVVM, MVI (MVC Alts)
Jakarta EE (Java/Layered)	Enables	Supports (JMS/MDB)	Supports (JAX-RS)	CDI Events	CDI (Managed)	Common (DAO/Repo)	Fundamental (CDI)	Key (JPA)	EJB Patterns, Front Ctlr
SOAP (Protocol)	Used in (rare)	Via JMS	Used with	N/A	N/A	N/A	N/A	N/A	RPC, WSDL
Glassfish (Runtime)	Deploys	Deploys (MDB)	Deploys	N/A	N/A	N/A	Provides (CDI)	Provides (JPA)	Implements Jakarta EE
Oracle DB (System)	Backend for	Backend for	Backend for	N/A	N/A	Accessed by	N/A	Accessed by	RAC (Cluster), DG (Standby)

E. Job Market Demand Matrix (Bogotá Focus)

Skill/Component	General Demand Trend	Bogotá/CO Demand (Level)	Key Skills Mentioned	Typical Experience	Indicative Salary (Bogotá)
SOA (Concepts)	Stable (Implicit)	Medium (Implicit)	Service Design, Integration, API (SOAP/REST), Microservices	Senior/Architect	High
Android Dev	Very High	High	Kotlin, Android SDK,	All levels	Variable (High for

			Jetpack (Compose, MVVM), REST, Git		Senior/Remote USD)
Kotlin	Very High (Growing)	High	Kotlin Lang, Coroutines, Java Interop, Functional	All levels	Variable (Tied to Role)
MVC (Android Pattern)	Low (Replaced)	Low	(Usually MVP/MVVM/ MVI requested instead)	N/A	N/A
Jakarta EE	Stable/Growin g	Medium/High (Implicit)	APIs (JPA, REST, CDI, JMS), App Servers (WildFly, etc.)	Mid/Senior	Variable (Tied to Java role)
Java	Very High (Stable)	High	Core Java, Spring Boot, Jakarta EE APIs, SQL, Cloud	All levels	Variable (High for Senior)
SOAP	Decreasing (New Proj.)	Low/Medium (Niche)	XML, WSDL, WS-Security, JAX-WS, Legacy Integration	Mid/Senior	Variable (Tied to Integration role)
Glassfish Admin/Dev	Very Low / Decreasing	Very Low / Non-existent	(General Jakarta EE Admin Skills)	N/A	N/A
Oracle DBA/Dev	Stable (Evolving)	High	Oracle Admin, SQL, PL/SQL, Tuning, RAC, DG, OCI	Senior	High (e.g., 12M+ COP/month Senior)

References

AltexSoft. (n.d.). *Pros and Cons of Java Development*. Retrieved April 21, 2025, from https://www.altexsoft.com/blog/pros-and-cons-of-java-programming/

Android Developers. (n.d.-a). *Platform architecture*. Retrieved April 21, 2025, from https://developer.android.com/guide/platform

Android Developers. (n.d.-b). *Guide to app architecture*. Retrieved April 21, 2025, from https://developer.android.com/topic/architecture

AngularMinds. (n.d.). *MVC Pattern Explained in Angular*. Retrieved April 21, 2025, from https://www.angularminds.com/blog/mvc-pattern-explained-in-angularjs

AWS. (n.d.). *What is SOA (Service-Oriented Architecture)*. Retrieved April 21, 2025, from https://aws.amazon.com/what-is/service-oriented-architecture/

Baeldung. (n.d.). *Layered Architecture*. Retrieved April 21, 2025, from https://www.baeldung.com/cs/layered-architecture

Built In. (n.d.). *What Is Service-Oriented Architecture (SOA)?* Retrieved April 21, 2025, from https://builtin.com/software-engineering-perspectives/service-oriented-architecture

cdcgroup.com. (n.d.). *Glassfish Server Open Source Edition*. Retrieved April 21, 2025, from https://cdcgroup.com/index.jsp/libweb/389686/GlassfishServerOpenSourceEdition.pdf

CIO Wiki. (n.d.). *Simple Object Access Protocol (SOAP)*. Retrieved April 21, 2025, from https://cio-wiki.org/wiki/Simple_Object_Access_Protocol_(SOAP

Cortex. (n.d.). *The Ultimate Guide to Service-Oriented Architectures*. Retrieved April 21, 2025, from https://www.cortex.io/post/the-ultimate-guide-to-service-oriented-architectures

Daily.dev. (n.d.). *Android Architecture Patterns: MVC vs MVVM vs MVP*. Retrieved April 21, 2025, from https://daily.dev/blog/android-architecture-patterns-mvc-vs-mvvm-vs-mvp

Devart. (n.d.). *All about Oracle Database - Definition, Features, Benefits*. Retrieved April 21, 2025, from https://www.devart.com/dbforge/oracle/all-about-oracle-database/

DEV Community. (n.d.-a). *Software Architecture Patterns: Layered Architecture*. Retrieved April 21, 2025, from https://dev.to/alexr/5-common-software-architecture-patterns-28a7

DEV Community. (n.d.-b). *Understanding the Layered Architecture Pattern: A Comprehensive Guide*. Retrieved April 21, 2025, from https://dev.to/yasmine_ddec94f4d4/understanding-the-layered-architecture-pattern-a-comprehensive-guide-1e2j

Devzery. (n.d.). *SOAP Protocol: Understanding Web Services Communication*. Retrieved April 21, 2025, from https://www.devzery.com/post/soap-communication-protocol

DigitalDefynd. (n.d.). *5 Ways Oracle is Using AI*. Retrieved April 21, 2025, from https://digitaldefynd.com/IQ/ways-oracle-use-ai/

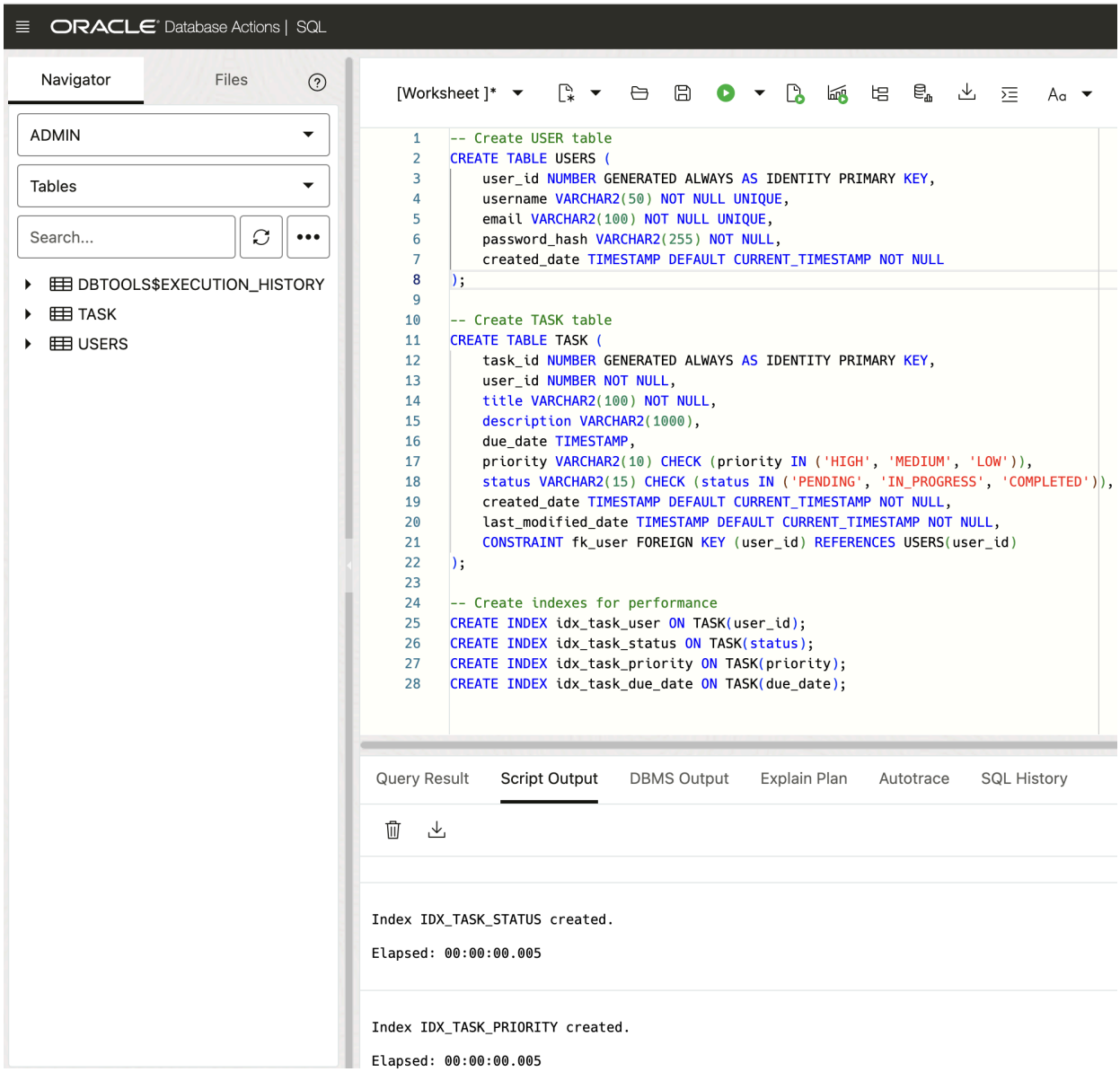
DZone. (n.d.). *Service Oriented Architecture: A Dead Simple Explanation*. Retrieved April 21, 2025, from https://dzone.com/articles/service-oriented-architecture-a-dead-simple-explan

Eclipse News. (n.d.). *A Developer's Guide to Jakarta EE 11*. Retrieved April 21, 2025, from https://newsroom.eclipse.org/eclipse-newsletter/2024/july/developer%E2%80%99s-guide-jakarta-ee-11

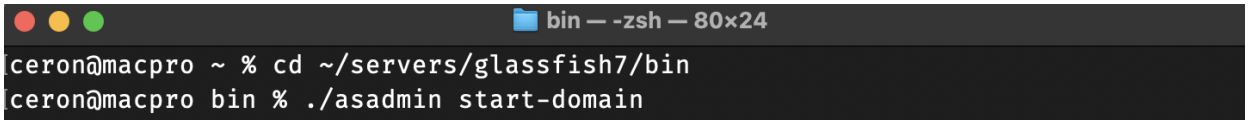
eG Innovations. (n.d.). *What is the GlassFish application server? - IT Glossary*. Retrieved April 21, 2025, from https://www.eginnovations.com/glossary/glassfish

Process:

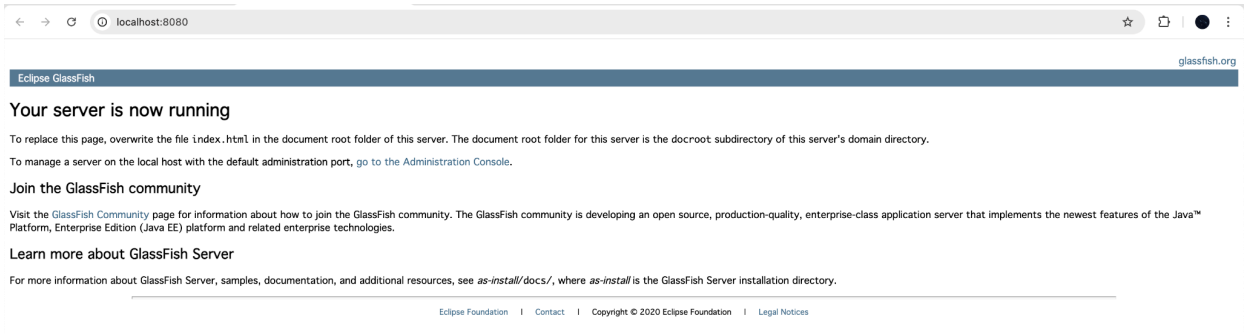
Set-up database:



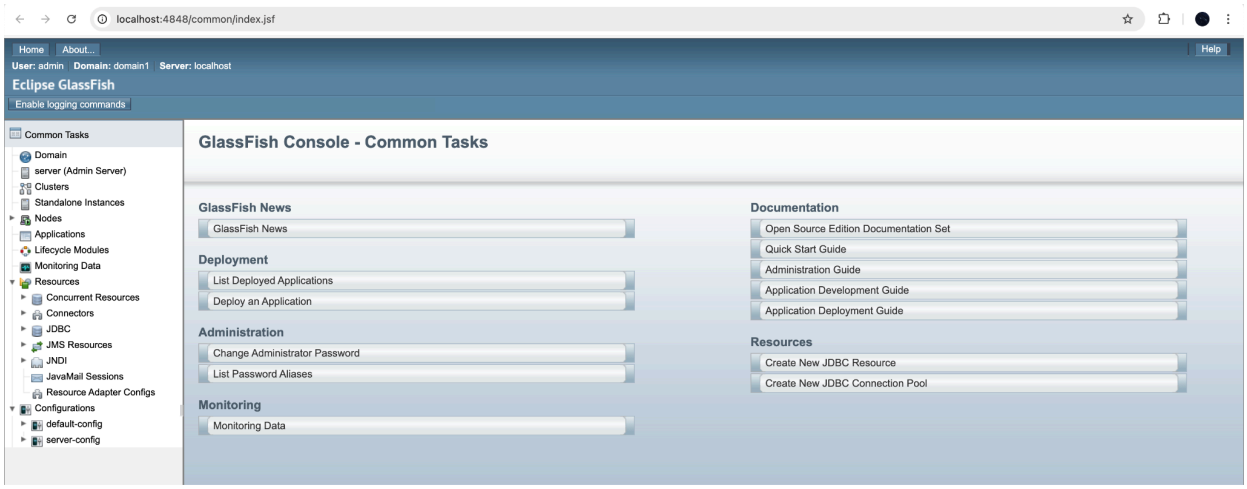
GlassFish



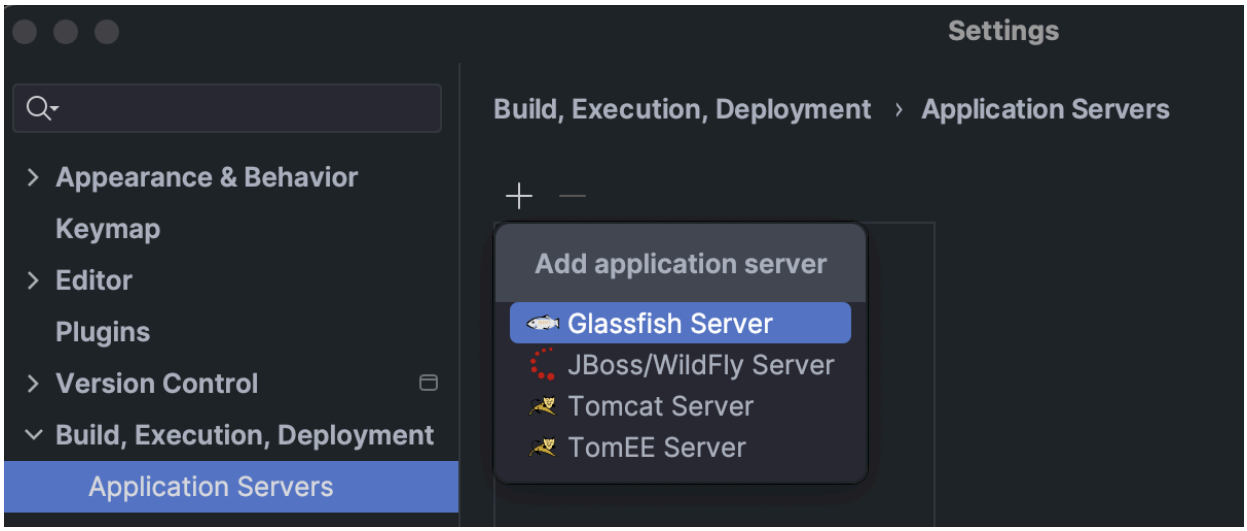
Server started

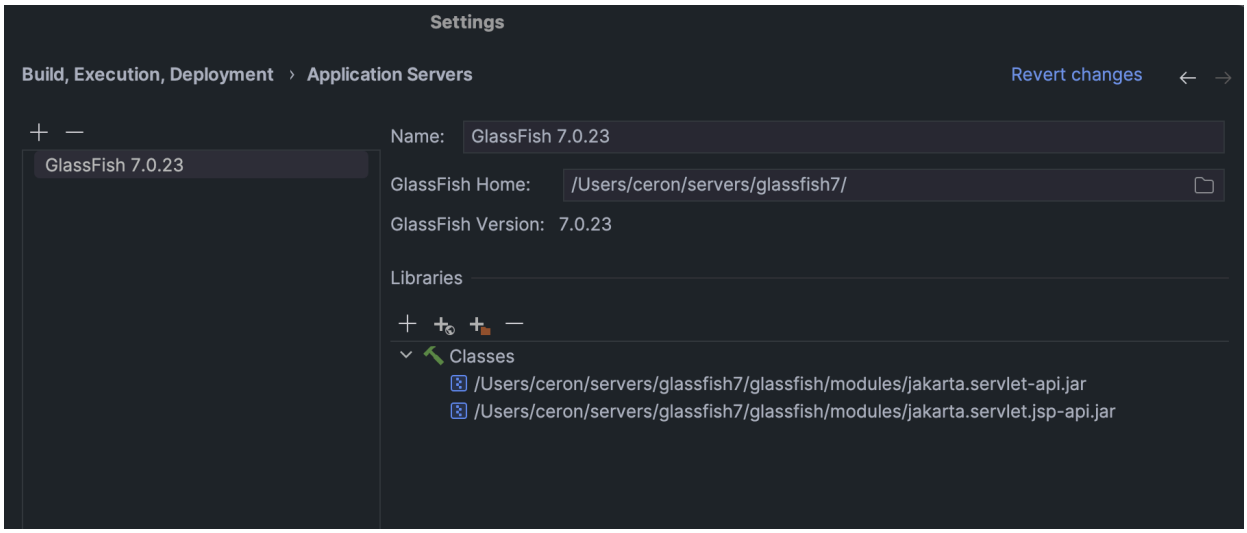


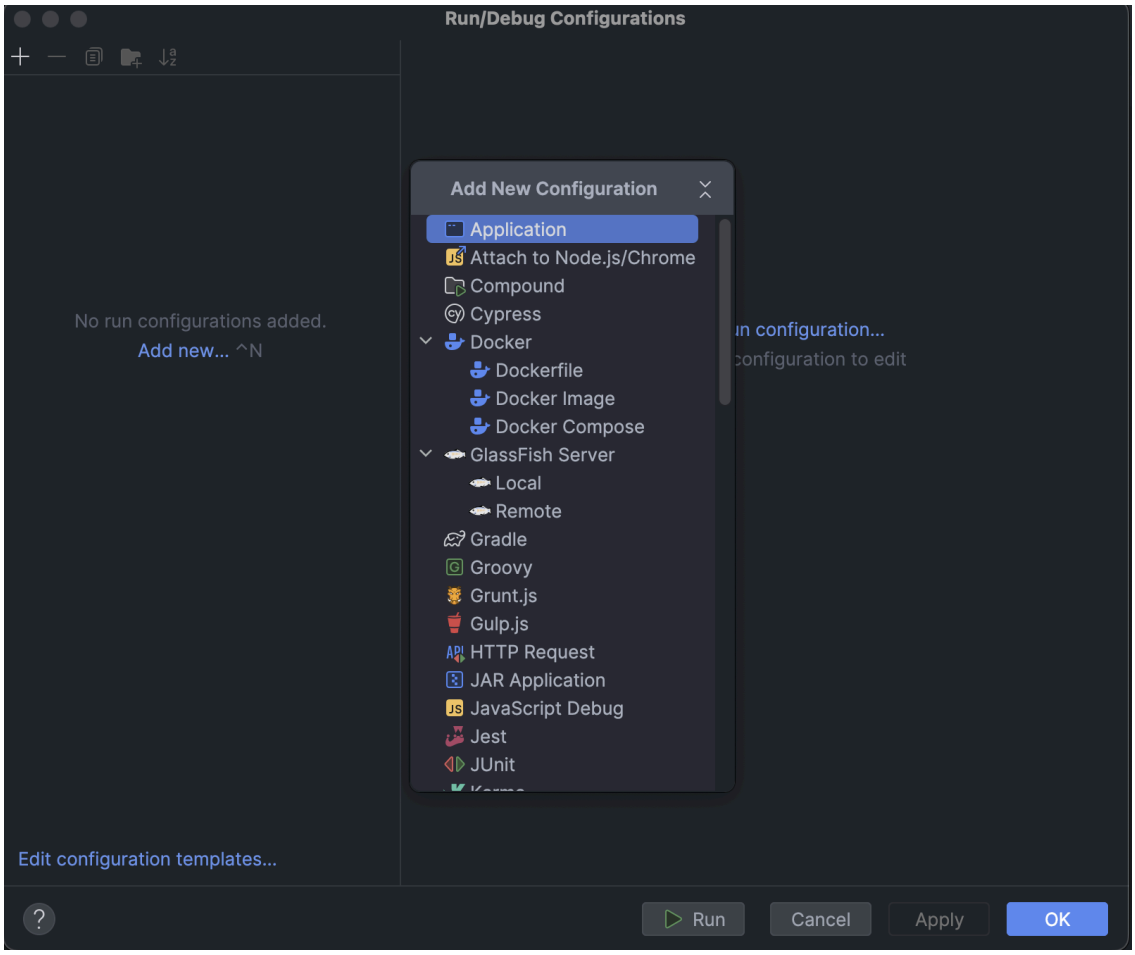
Admin page

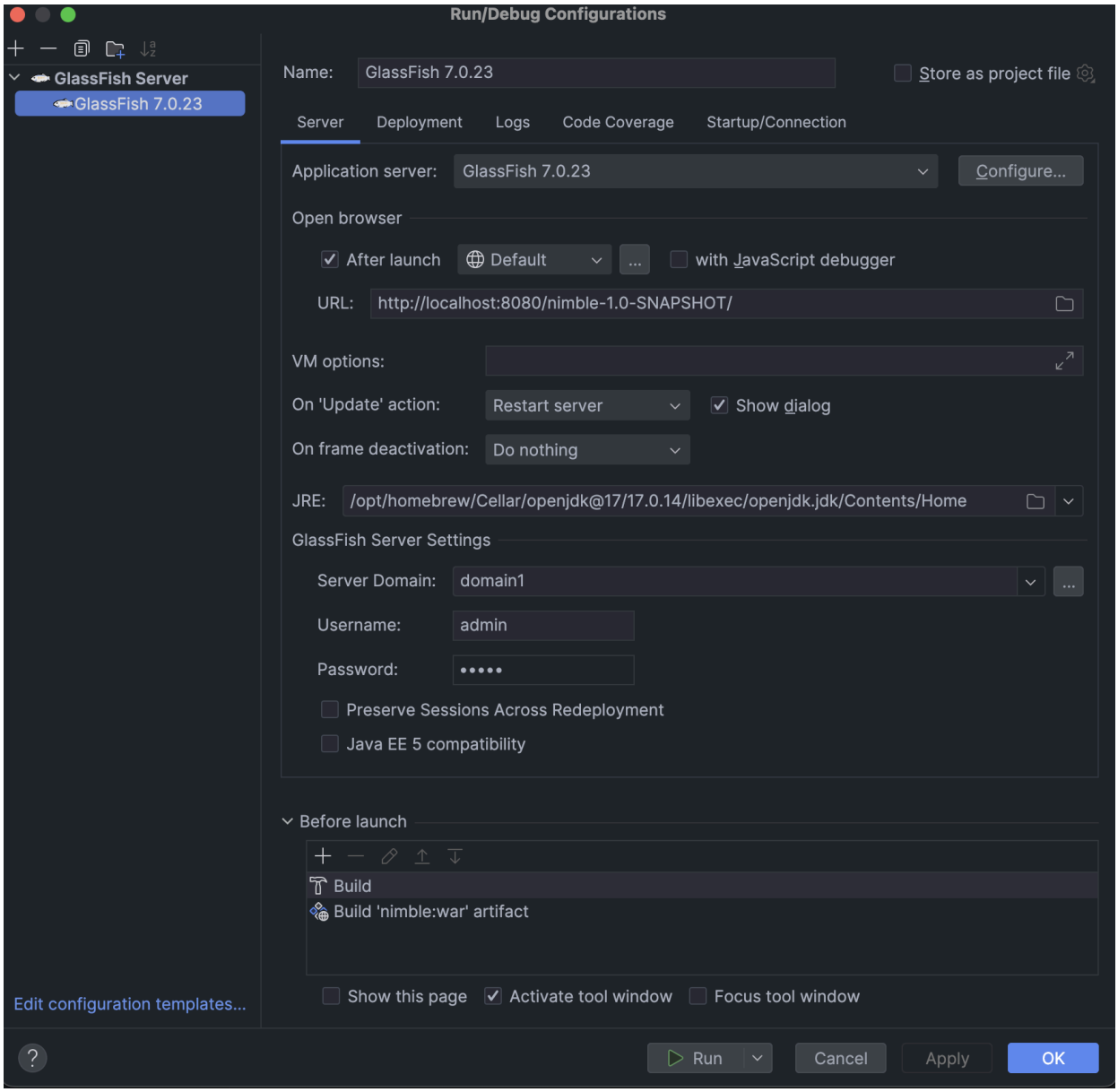


Set up Java with GlassFish

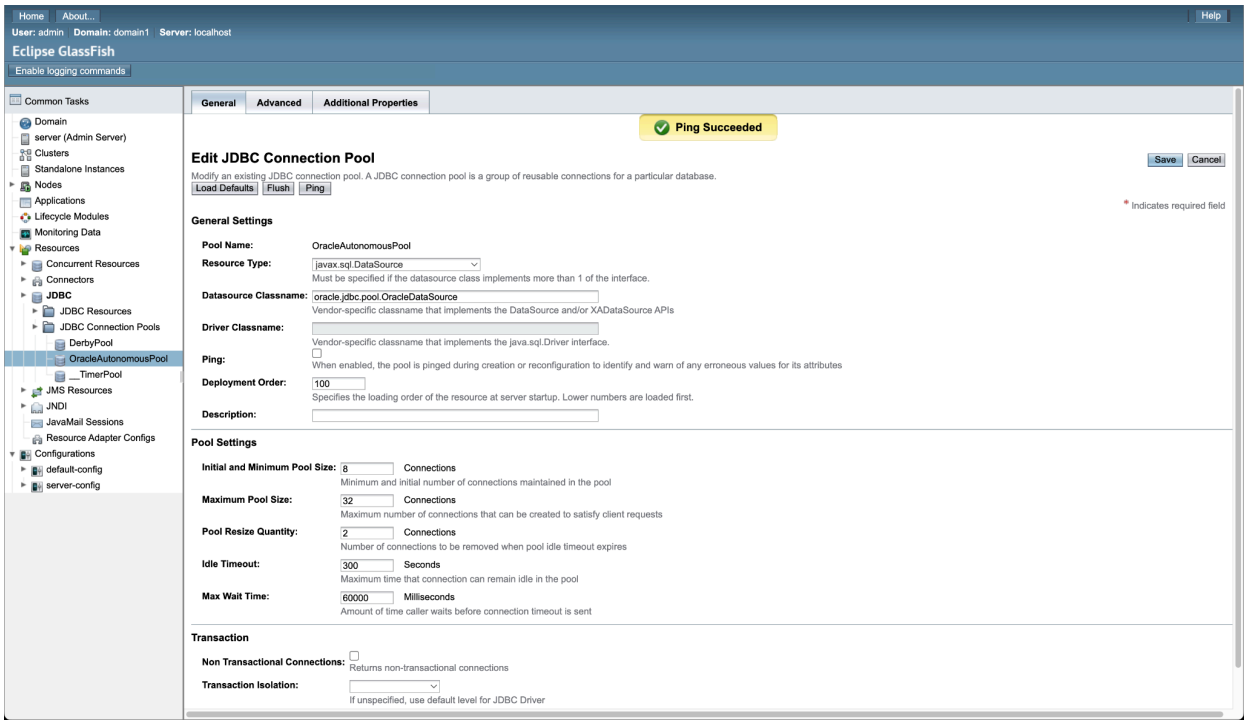


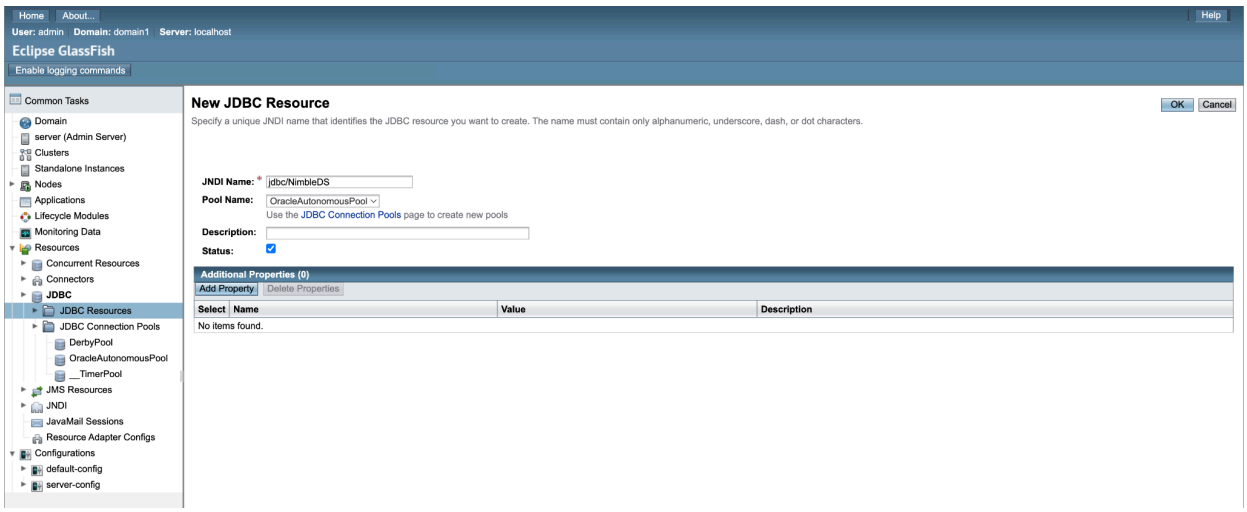

Settings
Build, Execution, Deployment › Application Servers
Revert changes
GlassFish 7.0.23
Name: GlassFish 7.0.23
GlassFish Home: /Users/ceron/servers/glassfish7/
GlassFish Version: 7.0.23
Libraries
Classes
/Users/ceron/servers/glassfish7/glassfish/modules/jakarta.servlet-api.jar
/Users/ceron/servers/glassfish7/glassfish/modules/jakarta.servlet.jsp-api.jar


Run/Debug Configurations
No run configurations added.
Add new... ^N
Add New Configuration
Application
Attach to Node.js/Chrome
Compound
Cypress
Docker
Dockerfile
Docker Image
Docker Compose
GlassFish Server
Local
Remote
Gradle
Groovy
Grunt.js
Gulp.js
HTTP Request
JAR Application
JavaScript Debug
Jest
JUnit
Edit configuration templates...
Run
Cancel
Apply
OK

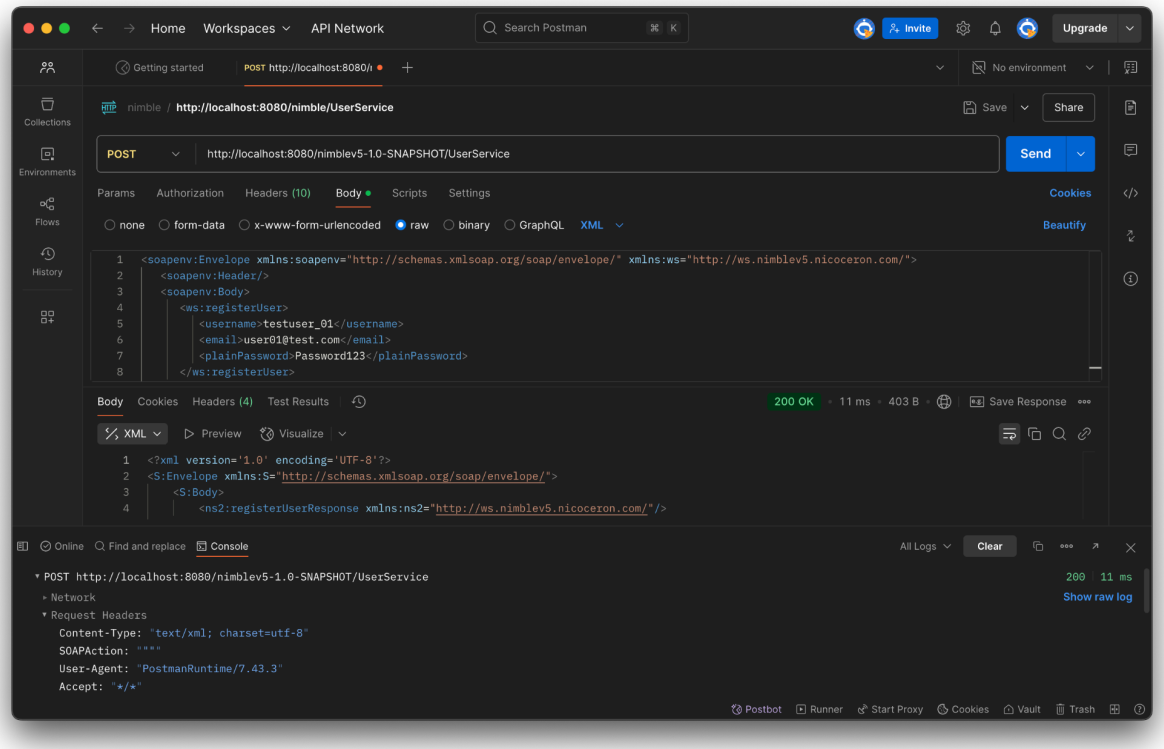


Setup glassfish connection to Oracle

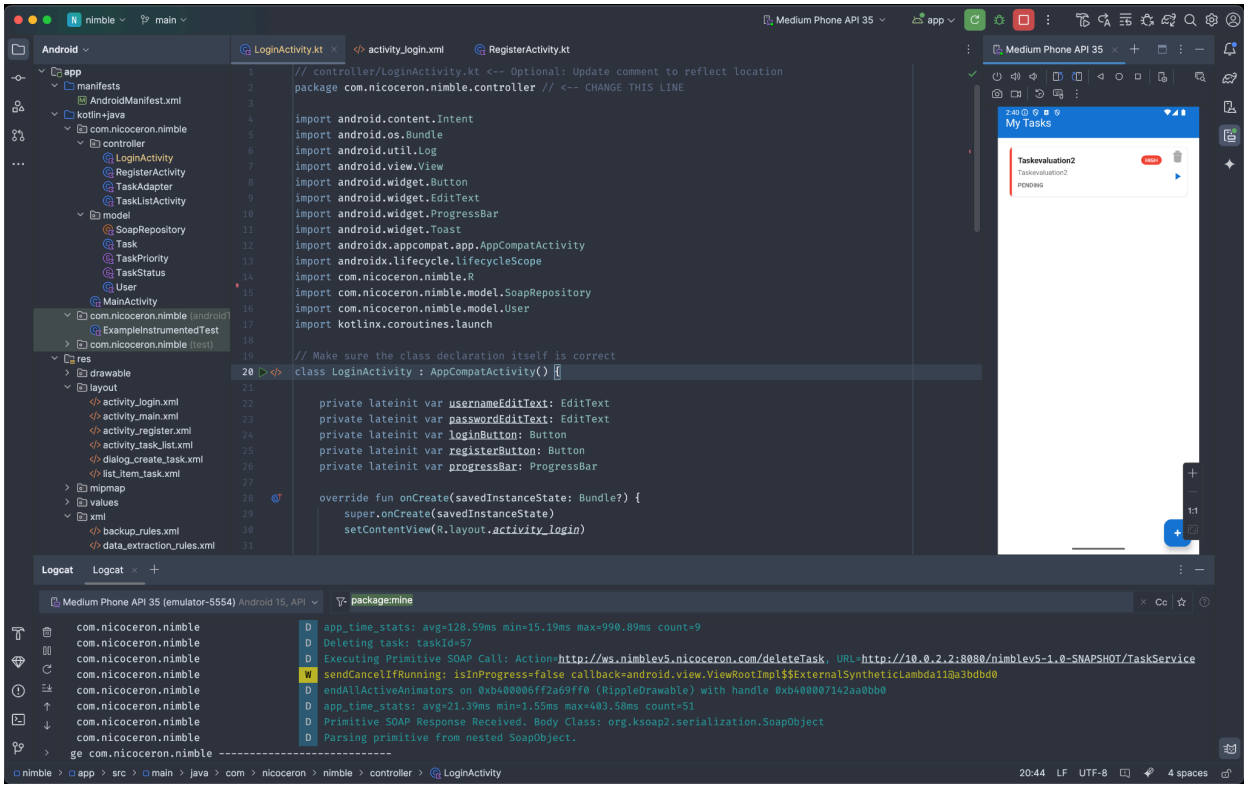




Response with Postman

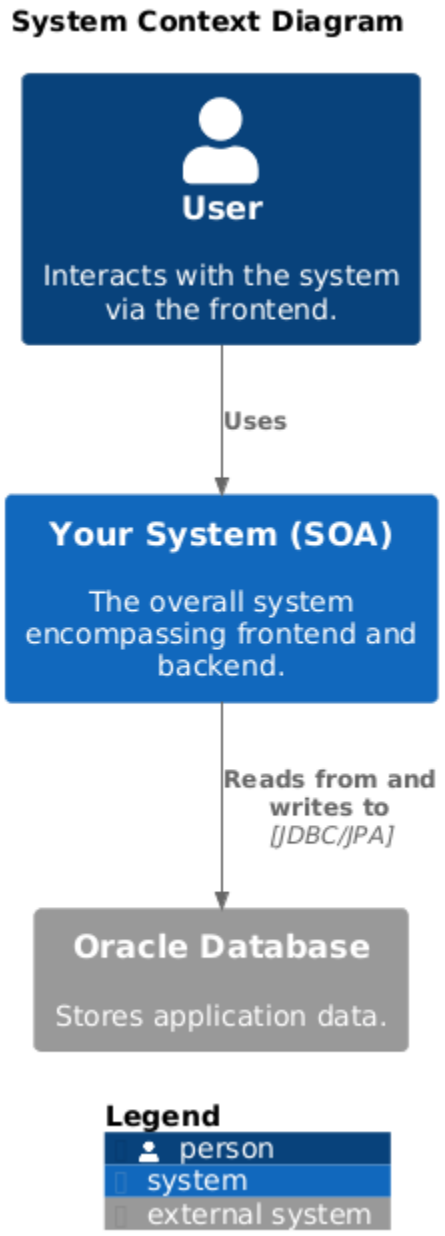


Android making requests



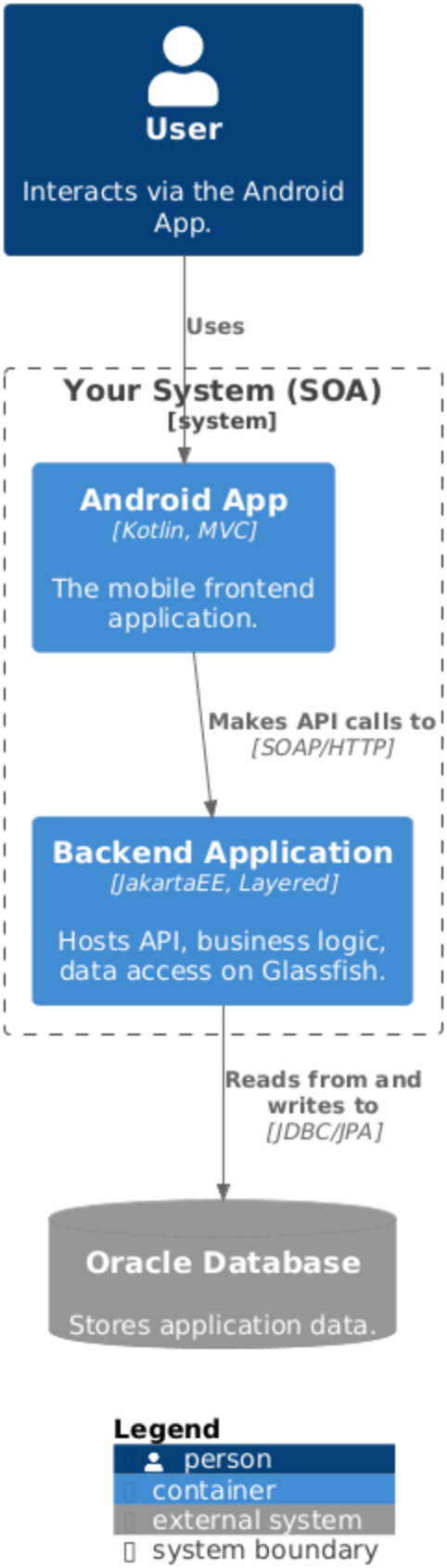
Diagrams

C4 Context Diagram

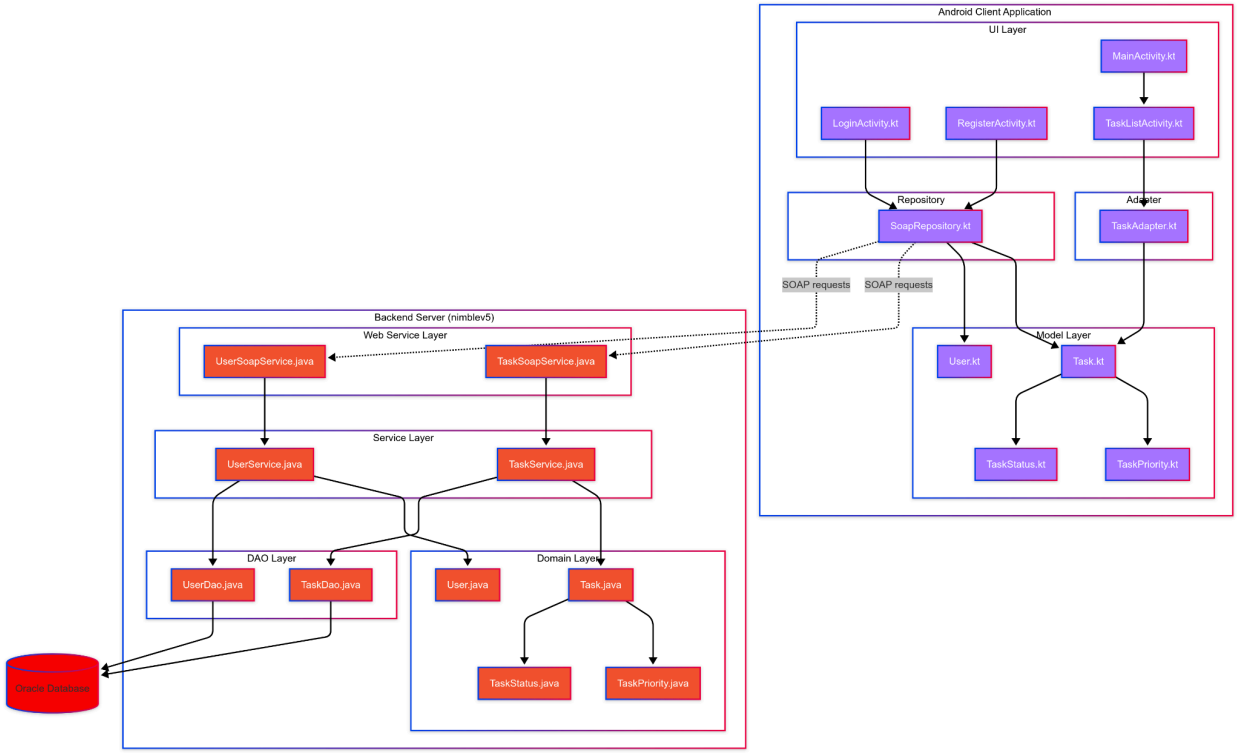


C4 Container

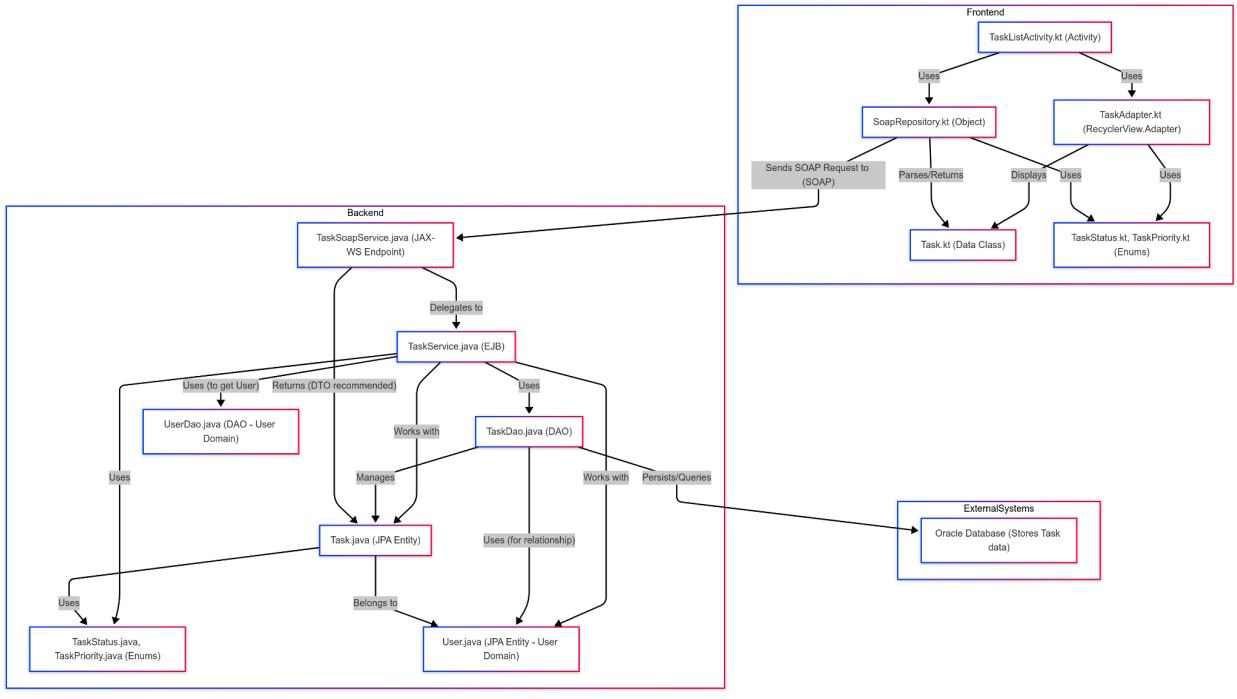
Container Diagram for Your System



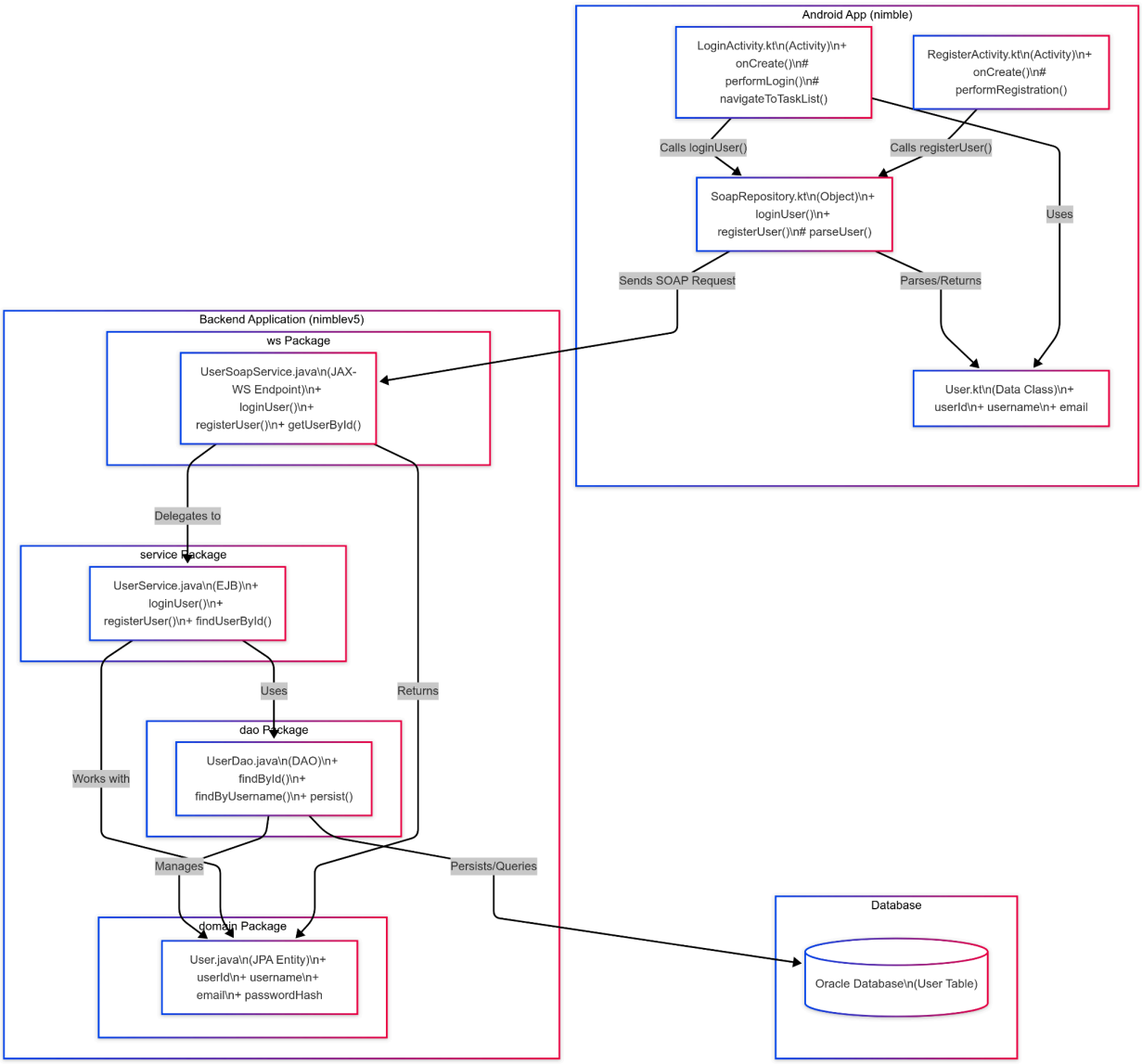
C4 Components



C4 Code (Task)

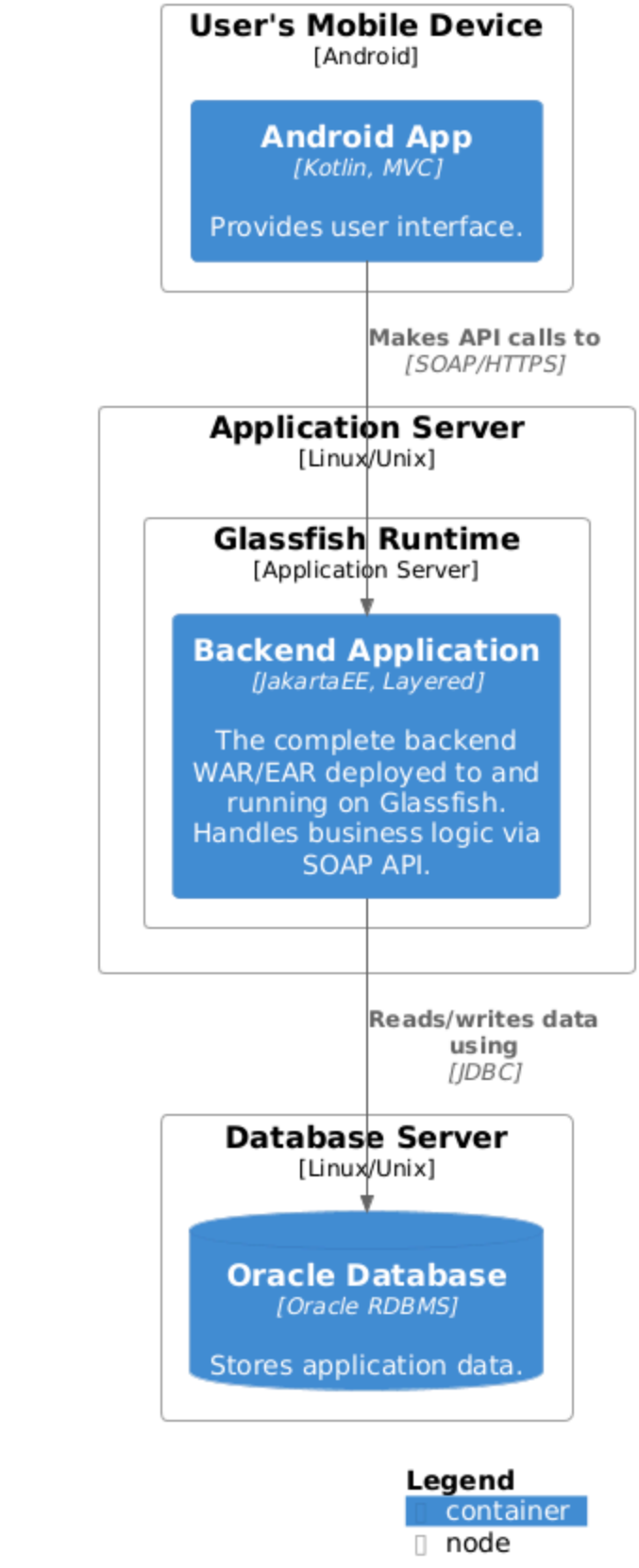


C4 Code (User)


Android App (nimble)
LoginActivity.kt\n(Activity)\n+ onCreate()\n# performLogin()\n# navigateToTaskList()
RegisterActivity.kt\n(Activity)\n+ onCreate()\n# performRegistration()
Calls loginUser()
Calls registerUser()
SoapRepository.kt\n(Object)\n+ loginUser()\n+ registerUser()\n# parseUser()
Uses
Sends SOAP Request
Parses/Returns
User.kt\n(Data Class)\n+ userId\n+ username\n+ email
Backend Application (nimblev5)
ws Package
UserSoapService.java\n(JAX-WS Endpoint)\n+ loginUser()\n+ registerUser()\n+ getUserById()
Delegates to
service Package
UserService.java\n(EJB)\n+ loginUser()\n+ registerUser()\n+ findUserById()
Uses
Returns
dao Package
UserDao.java\n(DAO)\n+ findById()\n+ findByUsername()\n+ persist()
Works with
Manages
Persists/Queries
domain Package
User.java\n(JPA Entity)\n+ userId\n+ username\n+ email\n+ passwordHash
Database
Oracle Database\n(User Table)

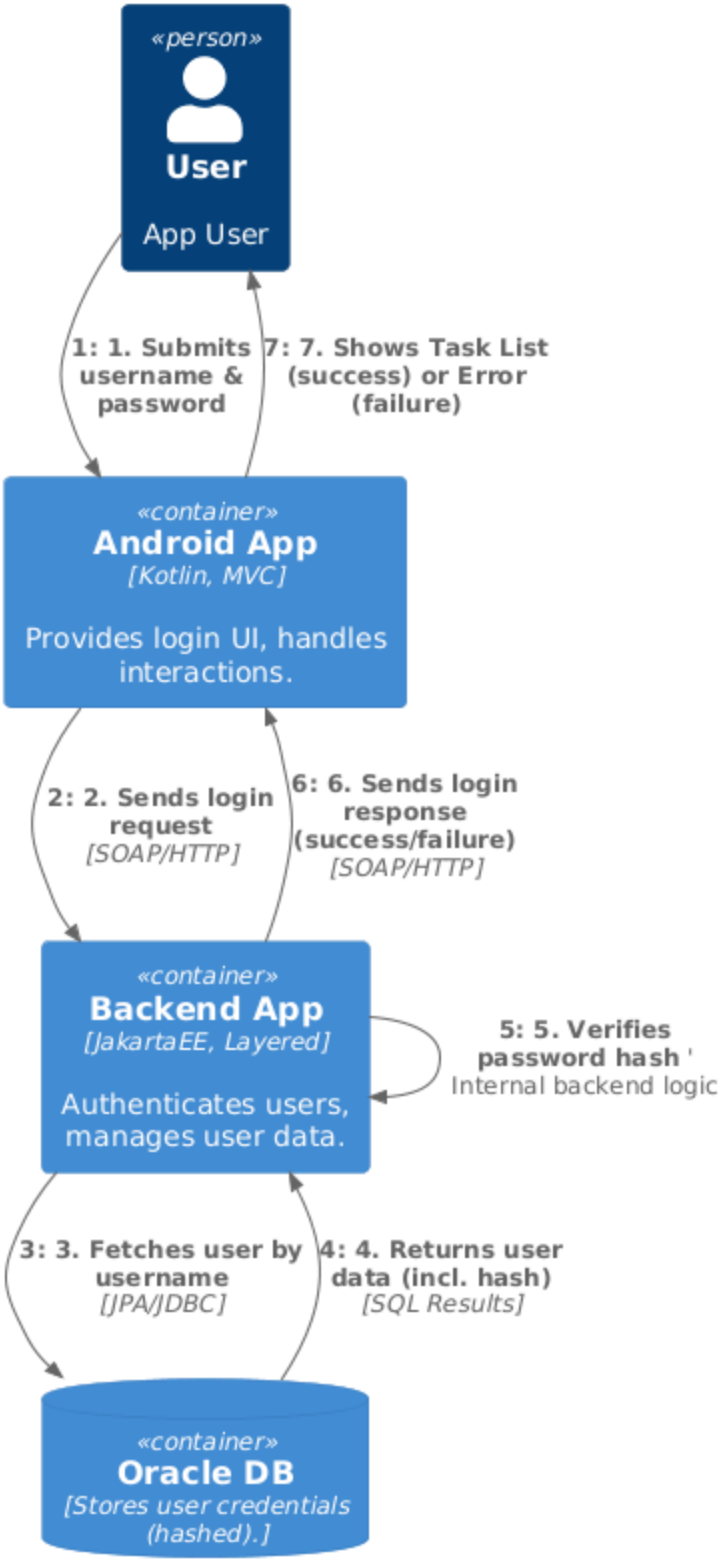
C4 Deployment

Deployment Diagram: Production Environment

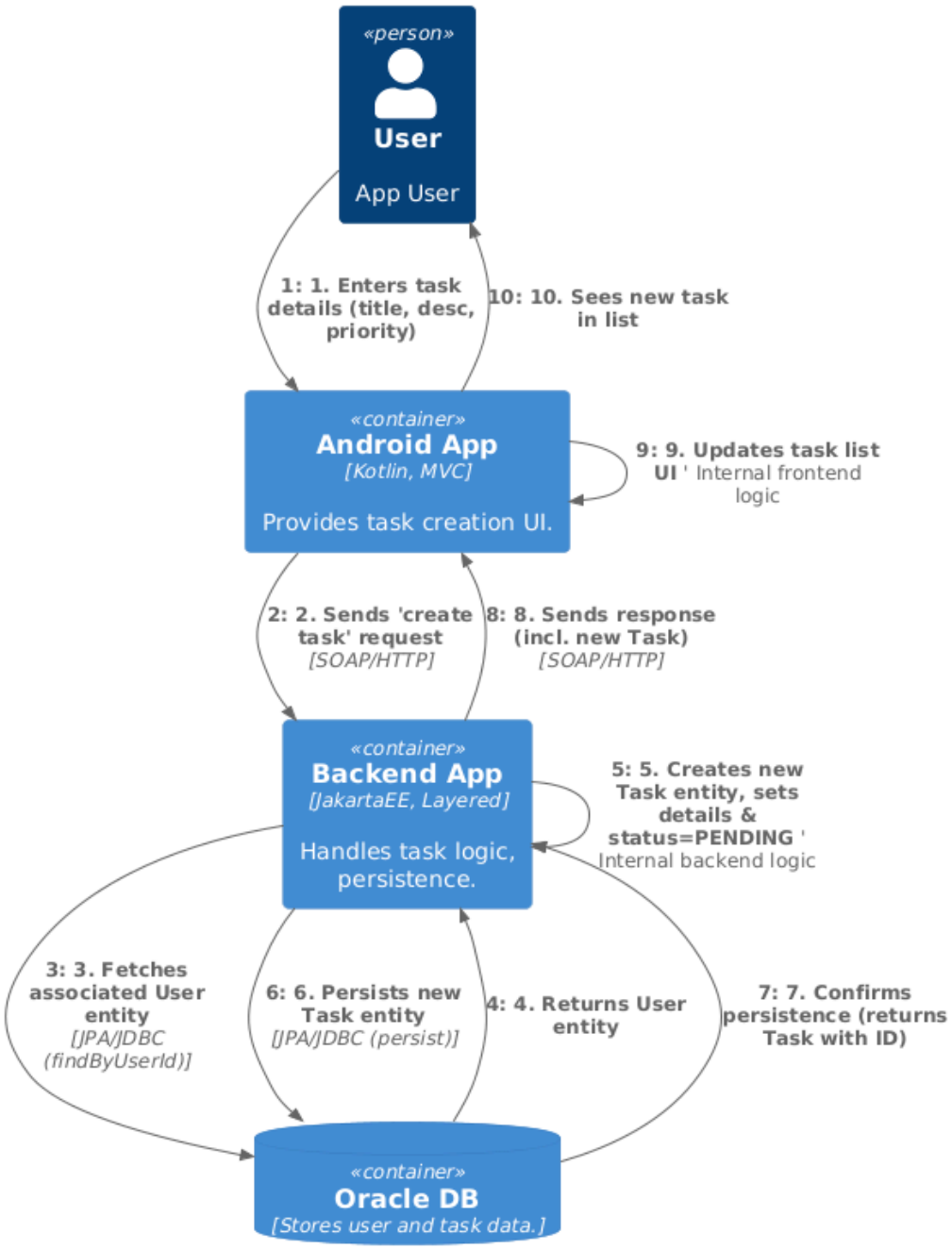


C4 Dynamic

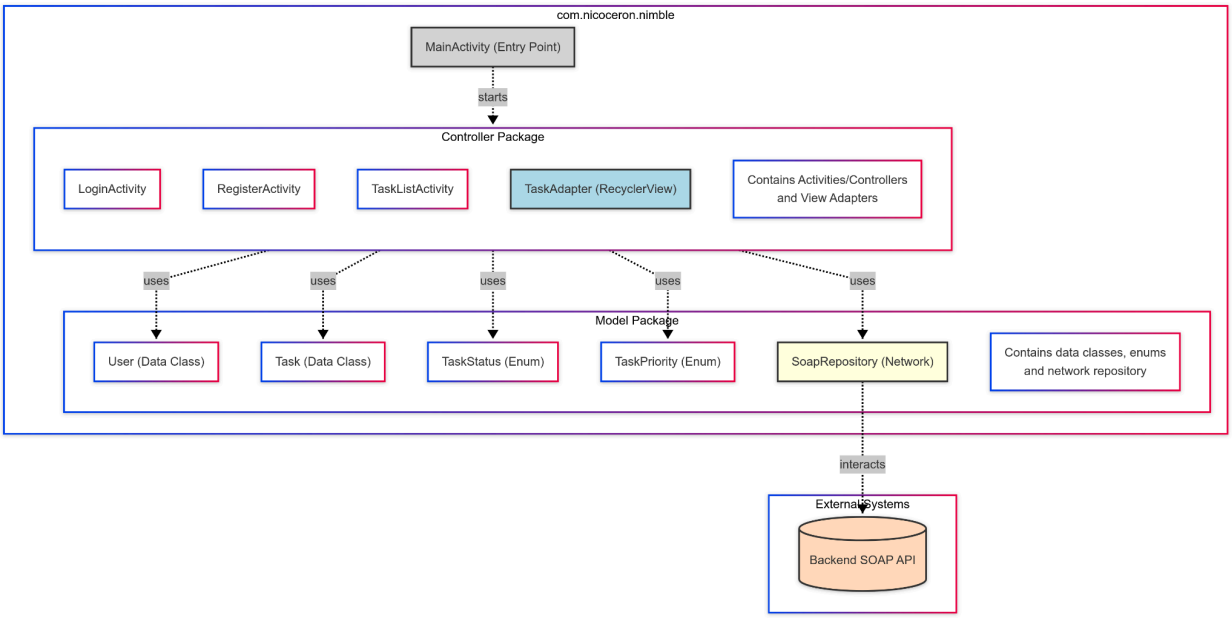
Dynamic Diagram: User Login Flow



Dynamic Diagram: Create New Task Flow



Package Diagram (Front)



Package Diagram (Back)