Solution Planning

1. Planning Stages

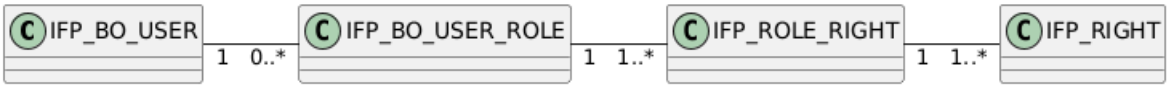
- **Phase 1**: Technical Implementation Idea (independent of any ubiquitous language)
- **Phase 2**: Completed Solution Planning (Business Requirements + Technical Implementation Details + Test Analysis)
- **Phase 3**: Development
- **Phase 4**: Testing & CI/CD
- **Phase 5**: Post-Implementation Documentation & Knowledge Sharing

2. Technical Implementation Idea

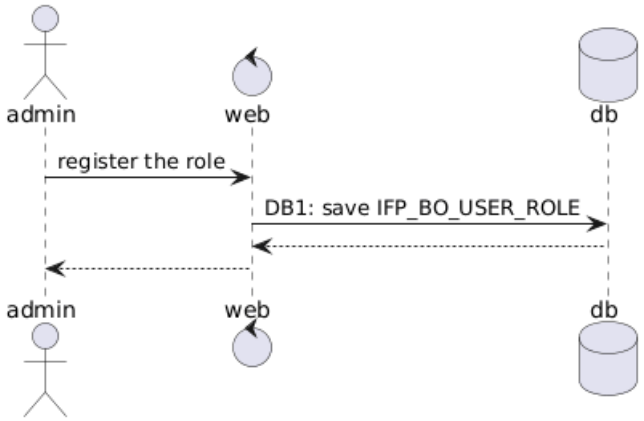
Terms	Explanation
BO	Backoffice
IFP	IFastPay

2.1. Roles-Right Relationship

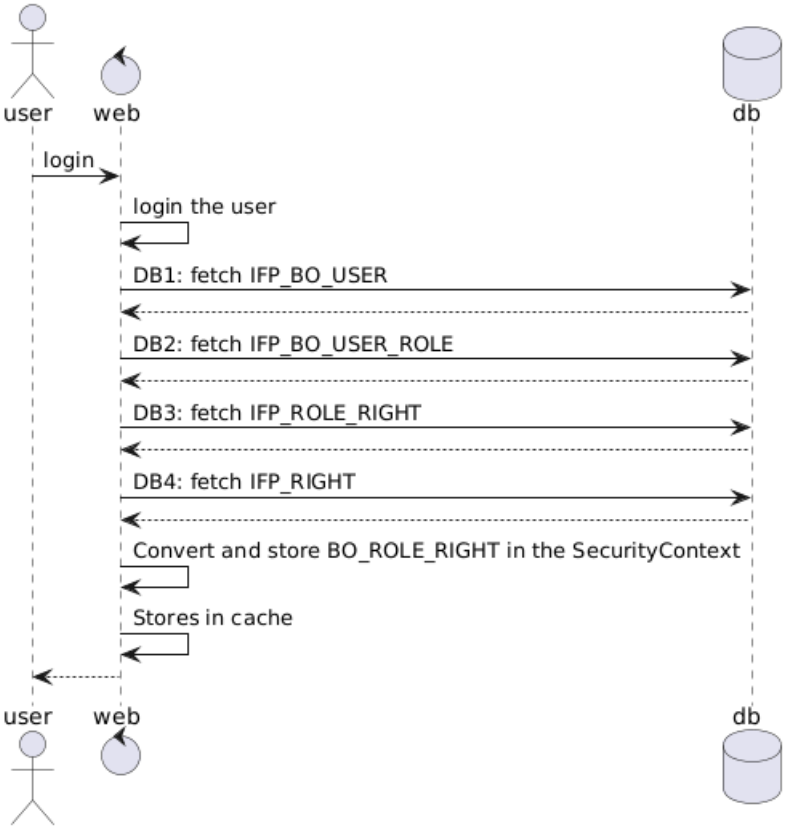
2.1.1. Design-1 (Role–Centric Model, used in iGB)



Role Registration Flow

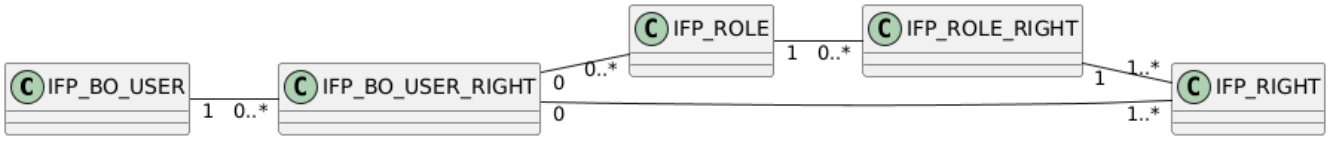


User Login Flow

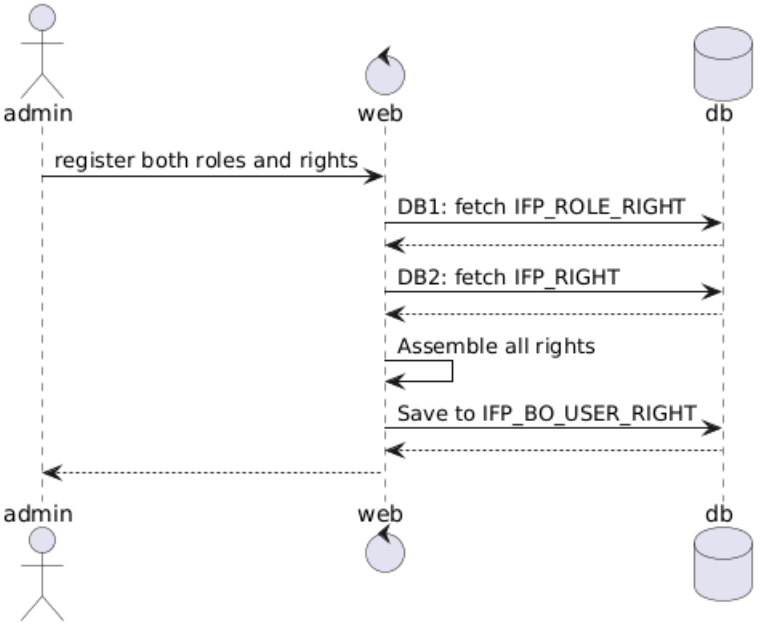


- **Characteristics:**
  - Simpler to model and manage, but requires more DB queries.
  - Every right must belong to a role, limiting flexibility.

2.1.2. Design-2

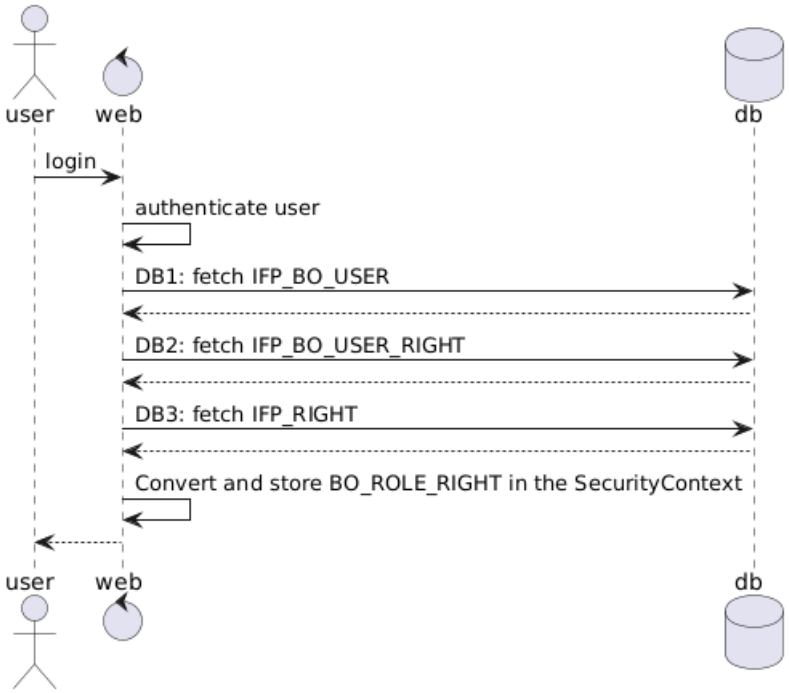


Role Registration Flow



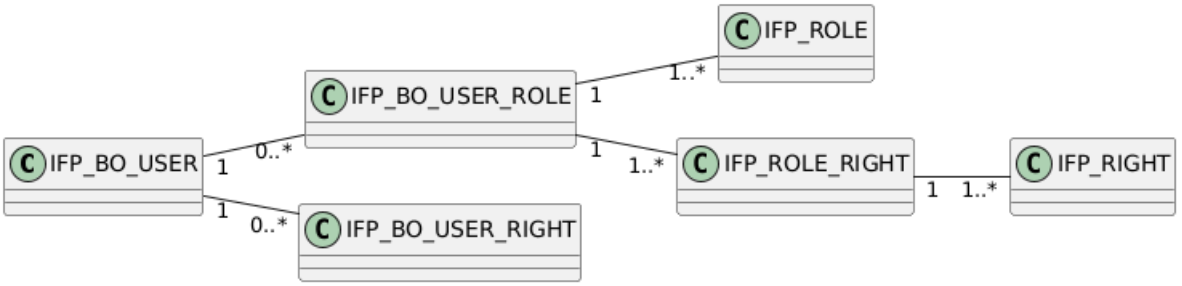
*Can be optimized using cache

User Login Flow



- **Characteristics**:
  - More flexible, as users can have either full roles or standalone rights.
  - More efficient querying but harder to model and maintain (a lot of edge cases)

2.1.2. Design-3

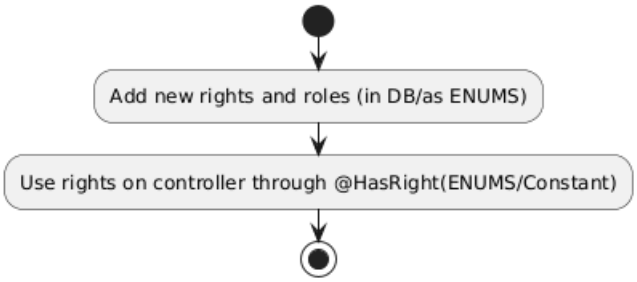


- Querying becomes more complex & take times, mitigated through caching.

2.2. DB/ Enums/ Constants

- Use **Enums** if roles and rights are fixed and cannot be added dynamically by BO users (developer-managed only).
- Use **DB tables** if BO users must be able to create/manage roles and rights dynamically.

2.3. Developer Side



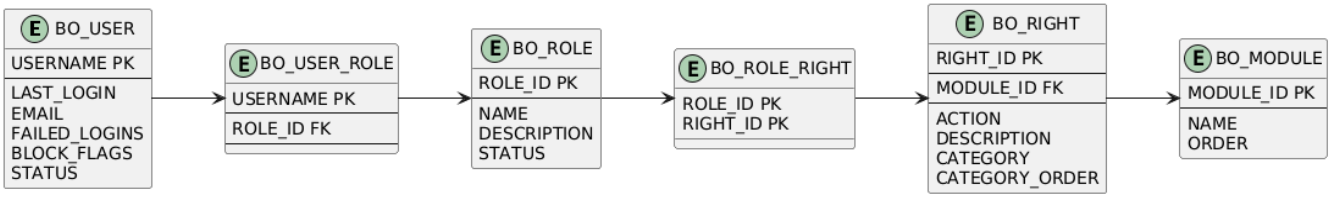

```
@RequiredArgConstructor
public enum BoRight {
    USERS_EDIT("bo:users:edit"),
    USERS_VIEW("bo:users:view");
    private final String authority;

    public String authority() {
        return authority;
    }
}

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@PreAuthorize("hasAuthority('{value}')")
@PreAuthorize("@authService.authenticate(authentication, '{value}')")
public @interface RequiresRight {
    BoRight value();
}

@Service
public class AuthService {

    public boolean authenticate(Authentication authentication, BoRight
boRight) {
        return authentication.getAuthorities().stream()
            .anyMatch(val -> val.getAuthority().equals(boRight.authority()));
    }
}

@Configuration
public class MethodSecurityTemplatesConfig {
    @Bean
    static AnnotationTemplateExpressionDefaults templateExpressionDefaults() {
        return new AnnotationTemplateExpressionDefaults();
    }
}
```

```
@GetMapping
@RequiresRight(BoRight.USERS_EDIT)
public ResponseEntity<UserDetails> updateUserDetails() { ... }
```

3. Design Details (Draft)

1. What should the API and domain model for BO users look like? (To refine further.)
2. Should we adopt a ubiquitous language (consistent naming conventions across code, DB, and docs)?
3. How should rights be named (namespace convention + granularity)?

References:




```
@EnableWebSecurity
@EnableMethodSecurity
@Configuration
public class SecurityConfig {
    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws
Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .anyRequest().authenticated();
            );
        return http.build();
    }
}
```

Others

1. Use Cache
2. Use Auditing, Testing, Observability
3. Use ABAC later
4. Testing Strategies
5. MakerChecker → next stage