

Karlo Stjepanović¹
Alen Šimec, PhD²

METHODOLOGICAL APPROACH TO REST API DESIGN WITH THE IMPLEMENTATION OF AUTHENTICATION AND AUTHORIZATION USING JSON WEB TOKEN (JWT)

Stručni rad / Professional paper

UDK / UDC: 004.42

DOI: 10.51650/ezrvs.20.1-2.15

Primljeno / Received: 27/04/2026

Prihvaćeno / Accepted: 24/06/2026

Modern software development increasingly relies on the separation of client-side components and server-side services in order to improve modularity, scalability, and long-term maintainability. The paper connects the theoretical foundations of REST with a methodological approach to API development, with particular emphasis on interface design, URL endpoint structure, naming consistency, HTTP methods, status codes, and data exchange between client and server. The research is based on a review and synthesis of relevant literature on REST architecture, HTTP communication, resource-oriented design, JSON-based data exchange, and token-based security mechanisms. The main result is a structured set of design guidelines that support predictable API behavior, clearer communication between software components, and improved maintainability of network services. The paper concludes that high-quality REST API design requires the combined application of architectural principles, standardized communication semantics, and security mechanisms such as JSON Web Token authentication and authorization.

Keywords: REST API; HTTP; JSON; JWT; authentication.

1. Introduction

Modern software systems increasingly adopt a deliberate separation between client-facing components and server-side services in order to improve modularity, scalability, and long-term maintainability. Within such distributed architectures, interoperability becomes a central requirement because multiple applications and subsystems must exchange data and coordinate functionality across network boundaries. In this context, Representational State Transfer (REST) has become a dominant architectural style for exposing resources through web-based services, most commonly through an Application Programming Interface (API). REST APIs are especially valuable because a single service can support heterogeneous

¹ Student; University of Applied Sciences Zagreb, Vrbik 8, 10 000 Zagreb, Croatia; e-mail: karlo.stjepanovic@tvz.hr

² College professor; University of Applied Sciences Zagreb, Vrbik 8, 10 000 Zagreb, Croatia; e-mail: alen.simec@tvz.hr

consumers, including web applications, mobile applications, and third-party integrations, while remaining independent of a specific platform, operating system, or implementation technology.

The conceptual foundations of REST were formalized by Roy Thomas Fielding, who defined a set of constraints intended to yield scalable and evolvable networked systems (Fielding, 2000). Among these constraints, statelessness is particularly influential for API design because each client request must contain all information necessary for the server to interpret and process it without relying on previously stored session context. This principle directly motivates a disciplined approach to endpoint design, where resources are represented through clear and consistent URL structures and operations are expressed through standardized interaction semantics.

Because REST-based communication is typically implemented over the HTTP protocol, the correctness and clarity of an API are closely tied to the appropriate use of HTTP methods and status codes. HTTP provides a uniform request-response model in which methods encode intent and status codes convey processing outcomes. Accordingly, this paper presents the theoretical background, best practices, and a methodological framework for developing REST API services, with emphasis on API interface design, URL endpoint structuring, HTTP methods, status code usage, JSON-based data exchange, and authentication and authorization mechanisms.

2. HTTP Communication as the Operational Foundation of REST-Based Services

REST has become a prevailing architectural approach for developing contemporary networked systems, most commonly implemented through an API. By adopting the client-server model, a REST API enforces a clear division of responsibilities and enables individual parts of a system to be developed, tested, deployed, and maintained independently. This separation supports modular design and improves the ability of software solutions to evolve over time (Richardson & Ruby, 2007).

The theoretical foundations of REST were established by Fielding (2000), who defined key constraints for scalable web architecture. One of the most influential is the stateless constraint, which requires that each client request include all information needed for the server to understand and process it without relying on stored session context or previous interactions. As a result, REST API endpoints must be structured consistently and clearly so that they unambiguously represent the interface through which client and server communicate (Fielding, 2000). REST-based interaction is typically realized using the HTTP protocol. HTTP is a text-based communication protocol in which a client initiates a request and the server returns a response. Within this model, HTTP methods are essential because they define the intent of a request toward a specific resource, while status codes communicate the outcome of processing that request. Disciplined use of HTTP methods and status codes is therefore critical for building predictable, interoperable, and well-designed REST services (Gourley et al., 2002).

Communication between the server and the client in REST-oriented architectures is typically carried out using HTTP. This standardized request-response model provides a uniform foundation for exchanging data and invoking operations across networked applications.

A central element of HTTP communication is the use of HTTP methods, which explicitly define the intent of a request with respect to a particular resource. Rather than encoding actions directly into the URL, the method indicates what operation the client wants to perform, such as retrieving, creating, updating, or deleting a resource. This separation contributes to clarity, consistency, and predictable behavior across API endpoints.

In addition to methods, HTTP status codes play a critical role in expressing the outcome of each request. They provide a compact and standardized signal that allows clients to interpret whether processing was successful, whether redirection occurred, or whether an error was encountered on the client or server side. Gourley et al. (2002) emphasize that correct and meaningful use of both HTTP methods and status codes is essential for building robust and interoperable web services.

Table 1. Most common HTTP methods overview

HTTP method	Meaning
GET	Retrieve data for a resource
POST	Create a new resource
PUT	Create a new resource, depending on context, or update an existing resource
PATCH	Partially update a resource
DELETE	Delete an existing resource

Source: Authors

Table 1 lists the most commonly used HTTP methods and their primary roles. In addition to these widely used methods, HTTP also defines several less frequently used methods, including HEAD for checking whether a resource exists or for obtaining response metadata such as content length, OPTIONS for retrieving the set of methods supported by a specific endpoint, TRACE for diagnostic purposes, and CONNECT for establishing a secure communication tunnel using TLS or SSL.

When using GET and DELETE methods, the client generally does not send data in the request body. Instead, additional input is typically provided through query parameters within the URL. Code snippet 1 illustrates an HTTP GET request that includes an additional query parameter.

Code 1. Use of an HTTP method with a query parameter

```
GET http://localhost/users?id=21
```

Source: Authors

On the other hand, POST, PUT, and PATCH methods allow the client to send data in the request body so that the server can process that data. Code snippet 2 shows an HTTP POST request used to create a new user, where the request body contains the data the server uses to create an entity that is subsequently stored in the database.

Code 2. Use of an HTTP method with data in the request body

```
POST http://localhost/users
{
  "firstName": "Ivan",
  "lastName": "Horvat",
  "email": "ivan.horvat@tvz.hr"
}
```

Source: Authors

After processing each request, the server returns a response along with a status code that indicates the outcome of the request. Status codes consist of three digits, and the first digit defines the response category:

- 1XX informational response
- 2XX successful request processing
- 3XX redirection
- 4XX client error
- 5XX server error.

3. Data Exchange Between Client and Server

JavaScript Object Notation (JSON) is a data format that uses plain text to represent information. Richardson and Ruby (2007) explain that JSON is commonly used in client-server communication because it enables the straightforward transfer of complex data structures, such as arrays and objects, in a human-readable textual form. It is typically used when the client sends data in the request body and when the server returns a response. Since HTTP supports textual content exchange, JSON has become one of the most widespread and practical formats for communication between clients and servers.

To send objects to the client in textual form, the server performs data serialization. Serialization is the process of converting objects into a text-based format suitable for transmission or storage.

When the client sends data to the server in JSON format, the server uses deserialization. Deserialization is the reverse process of serialization and involves converting the textual JSON representation back into objects or arrays that the application can process. In this way, data sent from the client becomes available to the server in a form that can be stored, validated, or used within the system's business logic.

4. URL Endpoints

Massé (2012), as well as Richardson and Ruby (2007), emphasize that REST is a resource-oriented architecture in which resources are exposed to clients through URL endpoints combined with the appropriate HTTP method. Resources represent entities or data managed by the API,

such as users, orders, or products, and each resource is identified by a unique Uniform Resource Identifier (URI). The URI provides an unambiguous way to address a specific resource, typically by including its identifier within the URL endpoint (Richardson & Ruby, 2007; Massé, 2012).

Because a resource represents an entity within the system that can be retrieved, created, updated, or deleted through the API, each route defines which resource is the subject of the request, while the HTTP method determines the action to be performed. Massé (2012) highlights several guidelines for proper URL design, including the need to reflect hierarchical relationships between resources. In such cases, nesting is used to express dependency relationships by separating path segments with a forward slash. URL routes should also be written in lowercase, and multiword terms should use a hyphen instead of an underscore to improve readability. Finally, common conventions recommend using plural nouns to represent resource collections (Massé, 2012).

Table 2. Endpoints of a single resource

HTTP method	Endpoint	Meaning
GET	/users/{userId}/orders	Retrieve all orders for a specific user
GET	/users/{userId}/orders/{orderId}	Retrieve a specific order for a user
POST	/users/{userId}/orders	Create a new order for a user
PUT	/users/{userId}/orders/{orderId}	Update a specific order for a user
DELETE	/users/{userId}/orders/{orderId}	Delete a specific order for a user

Source: Authors

Table 2 illustrates endpoint design for managing a user's orders. The example demonstrates that each resource must be unambiguously accessible through a URI identifier, while the purpose of the request is defined by the HTTP method (Massé, 2012).

The REST design philosophy assumes that the URL route structure itself should clearly describe which resource is being accessed and include all information required to locate that resource. When combined with appropriate HTTP methods, this approach produces consistent and predictable API behavior, where resources are uniquely defined through URL routes (Gourley et al., 2002).

It is also important to recognize that, although URL routes may be publicly reachable, the actual content of an HTTP response depends on the permissions and context of the authenticated user. For example, two users can call the same endpoint yet receive different results depending on their roles, access rights, and operational context.

5. Stateless Security Through Token-Based Verification

In REST API services, reliable authentication and authorization are essential. Dalimunthe, Putra, and Ridha (2023) describe the use of JWT (JSON Web Token) in REST-based systems as a text-based token issued to the client after a successful login using credentials such as a username and password. Unlike traditional web applications that rely on server-side sessions to store information about an authenticated user, REST APIs typically follow a stateless approach (Dalimunthe et al, 2023).

A JWT is a digitally signed, text-based token generated by the server after the user successfully authenticates. Its digital signature provides integrity protection, making the token suitable for secure authentication workflows in stateless REST architectures.

```

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3MiOiJodHRwczov-
L2FwaS56YXZyc25pLXJhZC5jb20iLCJhdWQiOiJodHRwczovL2FwcC56YXZyc-
25pLXJhZC5jb20iLCJzdWIiOiIwMTk4YTA3NC03MWIyLTcyNDYtYjE1Zi1mNjFmYjZkdzWI1MmIiLCJpYXQiOi-
jE3NTUxMTk3NzAsImV4cCI6MTc1NTEyMDA3M3wianRpIjoiMmVlZDBkZTUwZjFmMjE0OTB-
jZDhkZW51YWNhYWJkaGUifQ.10E7e_BzLzFNhwAmz3-
RJAeqmuwq8_rpuVS7xDo77MHOa_bUQcAWfTog3xGzPh5iArfMST-
T59eUGgskV5d1P8Q

```

Code snippet 3 shows an example of a JWT token. It consists of three parts separated by dots: the header, which contains information about the token type and signing algorithm; the payload, which contains metadata about the token and the user for whom it is issued; and the signature, which is used to verify the token's authenticity and integrity.

```
sequenceDiagram
    participant Client
    participant Credentials as Credentials (Login Request)
    participant REST_API_Service as REST API Service
    REST_API_Service->>Credentials: 2
    Credentials->>Client: 1
    Client->>JWT_Token: 4
    REST_API_Service->>JWT_Token: 3
    participant JWT_Token as JWT Token
```

256

Figure 1 illustrates that, during the login process, the client sends the user's credentials to the server, and after successful authentication the server returns a JWT token. The client must then include this token in subsequent communication with the network service.

A JWT token should have a clearly defined validity period. Its payload typically contains claims indicating the time of issuance and expiration time. A recommended practice is to set a reasonable lifetime to reduce the risk of misuse if the token is compromised. When a token expires, a new one must be issued so the user can continue using the application without repeatedly re-entering credentials (Dalimunthe et al., 2023). For this purpose, access tokens and refresh tokens are commonly used. A refresh token has a longer lifetime and is usually issued together with the access token after a successful login. The renewal process works by having the client send the refresh token to the server when the access token expires, after which the server issues a new access token for continued communication.

Because refresh tokens are long-lived and therefore carry higher security risk, they should be transmitted and stored with particular care. When possible, refresh tokens should be stored in HTTP-only cookies. If the client environment does not support cookies, the refresh token must be stored locally in a secure manner.

The example demonstrates two basic operations: generating a JWT after successful user authentication and validating the token when the client requests access to a protected end-point. The implementation is conceptually aligned with the REST principle of stateless communication, because the server does not store session data between requests. Instead, each protected request contains the token in the HTTP Authorization header.

Code 4. Simplified JWT generation and validation in PHP

```
<?php
use Firebase\JWT\JWT;
use Firebase\JWT\Key;
function generateJwtToken(string $userId, array $roles): string
{
    $secretKey = $_ENV['JWT_SECRET'];
    $issuedAt = time();
    $payload = [
        'iss' => 'https://api.example.com',
        'aud' => 'https://app.example.com',
        'sub' => $userId,
        'roles' => $roles,
        'iat' => $issuedAt,
        'exp' => $issuedAt + 900,
        'jti' => bin2hex(random_bytes(16))
    ];
    return JWT::encode($payload, $secretKey, 'HS256');
}
function validateJwtToken(string $authorizationHeader): ?object
{

```

```

    $secretKey = $_ENV['JWT_SECRET'];
    if (!preg_match('/^Bearer\s+(.+)$/ ', $authorizationHeader,
        $matches)) {
        return null;
    }
    try {
        $decodedToken = JWT::decode($matches[1], new Key
($secretKey, 'HS256'));
        if ($decodedToken->iss !== 'https://api.example.com') {
            return null;
        }
        if ($decodedToken->aud !== 'https://app.example.com') {
            return null;
        }
        return $decodedToken;
    } catch (Throwable $exception) {
        return null;
    }
}

```

Source: Authors

In the first function, the server generates a JWT token after successful authentication. The token payload contains standard claims such as issuer (iss), audience (aud), subject (sub), issued-at time (iat), expiration time (exp), and unique token identifier (jti). The sub claim identifies the authenticated user, while the roles claim can be used by the server to determine authorization rules for protected resources. The expiration time is intentionally short in order to reduce the risk of token misuse if the access token is exposed.

The second function demonstrates token validation during access to a protected REST endpoint. The client sends the JWT in the Authorization header using the Bearer scheme. The server extracts the token, verifies its signature using the same secret key and algorithm, and then checks whether the issuer and audience claims match the expected values. If the token is missing, invalid, expired, or issued for a different context, the request should be rejected with an appropriate HTTP status code, most commonly 401 Unauthorized or 403 Forbidden, depending on whether the problem is related to authentication or authorization.

This example shows how JWT supports stateless REST API security. Since all required identity information is contained in the signed token, the server can validate each request independently without relying on server-side session storage. However, the secret key must be stored securely, and sensitive configuration values should never be hardcoded in the source code. In production systems, additional security measures should be applied, including HTTPS communication, short access-token lifetime, refresh-token rotation, and careful handling of token storage on the client side.

6. Conclusion

This paper presents the fundamental principles of developing a REST API network service and emphasizes the importance of a consistent approach to API interface design. REST methodology is highlighted as a dominant standard for communication between client and server systems due to its simplicity, scalability, and broad adoption in modern software solutions.

Particular emphasis is placed on the correct modeling of REST resources and URL endpoints, the appropriate use of HTTP methods and status codes, and data exchange using a standardized format such as JSON. Such an approach enables predictable API behavior, clearer communication, and easier integration with different types of client applications.

In addition, the paper addresses the security aspects of REST API services through user authentication and authorization, underscoring the importance of resource protection and access control as essential elements of any modern networked system. By applying established best practices and standards, a REST API service can be designed to ensure reliability, stability, and long-term sustainability.

In addition to the theoretical and methodological contribution, this paper is accompanied by a practical implementation available in the public GitHub repository *Methodological-Approach-to-REST-API*: <https://github.com/asimec1/Methodological-Approach-to-REST-API>. The repository represents a practical extension of the concepts discussed in the paper, particularly in relation to REST API structure, server-side organization, authentication logic, configuration, migrations, and deployment support. By connecting the written methodological framework with an implementation example, the work demonstrates that REST API design should not be observed only as a theoretical architectural concept, but also as a practical development process that requires consistent organization of source code, configuration files, database structure, and security mechanisms.

Future research could further expand this work in several directions. One relevant direction would be a comparative analysis of JWT-based authentication and OAuth 2.0 authorization flows. Such research could evaluate the advantages, limitations, and appropriate use cases of both approaches, especially in systems that require third-party integrations, delegated access, and more complex authorization scenarios. While JWT is suitable for stateless authentication in many REST API services, OAuth 2.0 provides a broader authorization framework that may be more appropriate in distributed environments where external applications need controlled access to protected resources.

Another important direction for further research would be the implementation and evaluation of refresh token mechanisms in single-page applications. Since single-page applications frequently communicate with REST APIs through asynchronous client-side requests, token storage, expiration handling, and session renewal become critical security concerns. Future work could therefore examine different strategies for storing access tokens and refresh tokens, including HTTP-only cookies, secure storage approaches, token rotation, and protection against cross-site scripting and cross-site request forgery attacks.

Further research could also include a performance analysis of REST API services under high load. Such an analysis could measure response time, throughput, error rate, scalability, and resource consumption when a large number of concurrent clients access protected

endpoints. This would make it possible to evaluate how authentication and authorization mechanisms, especially JWT validation, influence API performance in real-world usage scenarios. The results could be used to define optimization guidelines for caching, database access, request validation, token verification, and deployment architecture.

The paper concludes that high-quality REST API design requires an understanding of theoretical principles, industry guidelines, and development best practices in order to build a modern and secure API interface aligned with the needs of contemporary applications and users.

LITERATURE

1. Dalimunthe, S., Putra, E. H., & Ridha, M. A. F. (2023). Restful api security using json web token (jwt) with hmac-sha512 algorithm in session management. *IT Journal Research and Development*, 8(1), 81-94.
2. Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. Doctoral dissertation. Irvine, CA: University of California, Irvine.
3. Gourley, D., Totty, B., Sayer, M., Reddy, S., & Aggarwal, A. (2002). *HTTP: The definitive guide*. Sebastopol, CA: O'Reilly Media.
4. Massé, M. (2012). REST API design rulebook. Sebastopol, CA: O'Reilly Media.
5. Richardson, L., & Ruby, S. (2007). RESTful web services. Sebastopol, CA: O'Reilly Media.



Autorska prava: ©2026 Autori. Ovaj rad objavljen je u otvorenom pristupu pod licencom Creative Commons Imenovanje – Nekomercijalno 4.0 međunarodna.

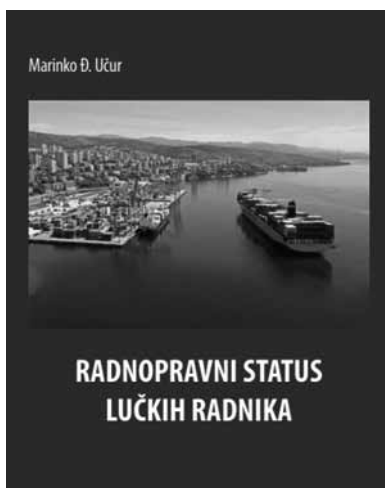
Copyright: ©2026 The Authors. Open Access: The work is licensed under a Creative Commons Attribution- NonCommercial 4.0 International License.

Dr. sc. socio. Drago Pupavac¹

Prikaz knjige

RADNOPRAVNI STATUS LUČKIH RADNIKA

ISBN 978-953-7722-71-5



U prosincu godine 2025. u izdanju ZIRS d.o.o. u Zagrebu izašla je knjiga (priručnik) profesora dr.sc. Marinka Đ. Učura s naslovom „Radnopravni status lučkih radnika“. Ukupan opseg teksta iznosi 217 stranica B-5 formata. Osim predgovora, uvoda, poveće bibliografije, popisa priloga i zahvale autora, knjiga sadrži i pet poglavlja u kojima autor izlaže specifičnosti radnih odnosa lučkih radnika.

Brojne promjene u svim područjima društvenog života, uključujući i pravne odnose općenito, odražavaju se i na sadržaj radnih odnosa lučkih radnika, pri čemu se oni jasno diferenciraju u odnosu na druge oblike radnih odnosa. Tu diferencijaciju određuju posebna obilježja luke kao poslodavca, specifičnosti u zaključivanju ugovora o radu, način kretanja lučkih radnika tijekom trajanja radnog odnosa, sadržaj samog radnog odnosa te sredstva i način organizacije lučkog rada. Lučki rad odvija se uz primjenu suvremenih lučkih tehnologija u tehnološki opremljenim pogonima luka, uključujući lučka skladišta, dizaličnu mehanizaciju i kontejnerske terminale, pri čemu su povećani rizici rada, složenost radnih procesa i uporaba teške mehanizacije ključni čimbenici koji zahtijevaju strogo poštovanje sigurnosnih standarda, sustavnu primjenu mjera zaštite na radu te obveznu uporabu odgovarajuće osobne i kolektivne zaštitne opreme.

¹ Profesor stručnog studija u trajnom izboru, Veleučilište u Rijeci, Vukovarska 58, 51000 Rijeka, Republika Hrvatska; e-mail: drago.pupavac@veleri.hr.

Cilj knjige je razjasniti normativni okvir radnih odnosa lučkih radnika, identificirati probleme u primjeni prava te ponuditi znanstveno utemeljena rješenja za njihovo unapređenje. Da bi se cilj knjige i ostvario, nametnula se potreba da se problematika radnopravnog statusa lučkih radnika istraži i obradi u šest zaokruženih, logično raspodijeljenih i tematski povezanih dijelova.

U prvome dijelu, „Uvodu“, autor postavlja temeljne hipoteze i ciljeve istraživanja, ističući važnost regulacije radnih odnosa lučkih radnika u morskim lukama. Posebno se osvrće na Zakon o radu kao opći zakon koji regulira radne odnose, ali ističe i potrebu za posebnim propisima koji bi uzeli u obzir specifičnosti lučkog rada.

Drugi dio s naslovom „Okvir i osnova projekta“ daje pregled propisa i subjekata koji uređuju radne odnose lučkih radnika. Time je postavljen pravni i organizacijski temelj za daljnju razradu specifičnih pitanja radnopravnog statusa lučkih radnika. Potom se pojašnjavaju temeljni pojmovi na kojima počiva cijeli projekt kao što su: mjesto rada, morske luke, lučke djelatnosti, prijevoz, promet, plovidba i drugi. Ovi se pojmovi ne razmatraju isključivo u pravnom smislu, već i u njihovoj funkcionalnoj i organizacijskoj ulozi unutar lučkog i prometnog sustava.

„Specifičnosti (posebnosti) lučkog rada i lučkih radnika – pojmovno“, naslov je trećeg dijela knjige. Posvećen je analizi posebnih obilježja rada lučko-transportnih radnika u morskim lukama. Autor lučki rad sagledava kroz prizmu lučkih operacija i organizacije radnih procesa, ali i šireg radnog okoliša morske luke, uključujući odnos prema moru kao prometnici i prirodnom resursu. Poseban naglasak stavljen je na organizaciju rada, sigurnosne mjere te zaštitu zdravlja na radu, pri čemu se ističe da su lučki radnici izloženi povećanim i specifičnim rizicima. Autor stoga naglašava nužnost osiguravanja odgovarajuće razine zaštite radnika, kako kroz normativni okvir, tako i kroz njegovu dosljednu provedbu u praksi.

Četvrti dio knjige, naslovljen „Kolektivni radnopravni i interesni sporovi“, naslov je četvrtog dijela. Ovaj dio posvećen je pitanjima mirenja i arbitraže u kolektivnim radnim sporovima te drugim oblicima konfliktnog rješavanja sporova između sindikata i poslodavaca, pri čemu se, ovisno o njihovoj pravnoj naravi i predmetu spora, razlikuju pravni i interesni sporovi. Poseban naglasak stavljen je na jačanje zaštite privatnosti i dostojanstva radnika, kao i na osiguravanje više razine pravne i socijalne sigurnosti u radnim odnosima. U tom se kontekstu analiziraju prava radnika i poslodavaca na udruživanje, sporazumi između radničkog vijeća i poslodavca te uloga skupova radnika kao mehanizama kolektivne zaštite i stabilizacije radnih odnosa.

U petom dijelu knjige s naslovom „Kolektivni ugovori i drugi autonomni akti poslodavca“, analizira se kolektivni ugovor kao autonomni opći normativni akt kojim se uređuju uvjeti rada od zasnivanja do prestanka radnog odnosa. Posebno se analizira jedan kolektivni ugovor u morskoj luci, s osvrtom na odredbe kojima se uređuju sigurnost i zaštita zdravlja na radu u specifičnom i rizičnom lučkom okruženju. Predmet istraživanja u ovom dijelu jest i pravilnik o radu, odnosno pravilnik o unutarnjoj organizaciji luke, kao instrument unutarnjeg normiranja kojim se, uz organizaciju radnih procesa, propisuju sigurnosni postupci i preventivne mjere. Autor razmatra kolektivne ugovore i pravilnike o radu s posebnim naglaskom na prava i obveze lučkih radnika u području sigurnosti rada, ističući njihov doprinos unapređenju sigurnih uvjeta rada, prevenciji profesionalnih rizika i osiguravanju veće razine pravne sigurnosti zaposlenih.

Šesti dio knjige, naslovljen „Reguliranje i ostvarivanje zaštite na radu“, predstavlja ujedno i najobimniji dio monografije te je posvećen sustavnoj analizi normativnih, organizacijskih i provedbenih aspekata sigurnosti i zaštite zdravlja na radu u lučkom okruženju. Autor razmatra cjelokupni sustav zaštite na radu kroz tri međusobno povezana segmenta: reguliranje, ostvarivanje i nadzor. Posebna se pozornost posvećuje nacionalnim zakonskim i podzakonskim izvorima, kao i internim aktima poslodavaca kojima se konkretiziraju obveze u području sigurnosti rada. Ovaj dio knjige utemeljen je i na detaljnoj analizi pravilnika o zaštiti na radu na primjeru jedne morske luke, čime se teorijska razmatranja povezuju s konkretnom praksom. Takav pristup omogućuje uvid u stvarnu primjenu propisanih mjera, uključujući procjenu rizika, organizaciju rada, osposobljavanje radnika te uporabu tehničkih sredstava i osobne zaštitne opreme. Ujedno se razmatra usklađenost nacionalnog okvira s europskim i međunarodnim standardima zaštite zdravlja i sigurnosti na radu. Autor nadalje analizira način ostvarivanja propisanih mjera u praksi, uključujući organizaciju rada, osposobljavanje radnika, primjenu tehničkih sredstava i osobne zaštitne opreme te preventivne postupke usmjerene na smanjenje profesionalnih rizika. Završno se razmatra uloga nadzora nad provedbom mjera zaštite na radu, s naglaskom na odgovornost poslodavaca i nadležnih tijela, pri čemu se zaštita zdravlja i sigurnosti radnika ističe kao temeljni preduvjet održivog i sigurnog funkcioniranja lučkog sustava. Ovo poglavlje knjige završava pregledom ozljeda na radu u morskim lukama.

Knjiga (priručnik) Radnopravni status lučkih radnika prof. dr. sc. Marinka Učura predstavlja značajan i vrijedan doprinos hrvatskoj pravnoj literaturi, osobito u području radnog i pomorskog prava, ali i u širem interdisciplinarnom kontekstu koji obuhvaća zaštitu zdravlja i sigurnost na radu, organizaciju rada i prometne sustave. Autor na sustavan i metodološki utemeljen način analizira specifičnosti radnih odnosa lučkih radnika, uzimajući u obzir njihove pravne, ekonomske i tehničke dimenzije te posebnosti lučkog rada kao rizičnog i organizacijski složenog područja.

Iako je knjiga primarno usmjerena na hrvatski pravni sustav, njezini zaključci i preporuke nadilaze nacionalni okvir te mogu biti relevantni i primjenjivi u drugim državama s razvijenim pomorskim i lučkim sektorom. Posebna vrijednost djela ogleda se u obradi aktualnih i društveno važnih tema, poput zaštite zdravlja i sigurnosti na radu, kolektivnih radnih odnosa te primjene europskih i međunarodnih standarda, što je osobito značajno u kontekstu globalizacije i intenziviranja međunarodne trgovine.

Autor pritom ne ostaje na razini teorijske analize, već kroz konkretne primjere i praktične preporuke nudi smjernice za unapređenje radnopravnog statusa lučkih radnika i učinkovitiju primjenu propisa u praksi. Time knjiga ostvaruje dvojaku vrijednost – kao znanstveno relevantno djelo i kao praktično uporabljiv priručnik – te se može preporučiti kako znanstvenicima i studentima, tako i pravnim praktičarima, socijalnim partnerima i donositeljima odluka u području lučkog i pomorskog gospodarstva.

