# MODULAR ARCHITECTURE FOR OPEN RESEARCH PLATFORMS: A CASE STUDY OF RARS WITH POCOS, DATA MAPPERS, AND REPOSITORY PATTERNS

*Agon Memeti(1)*, Eva Cipi(2), Hakan Rexhepi(1), Artina Kamberi(1)*

(1) University of Tetova; (2) University of Vlora;
(21) North Macedonia, (2) Albania

* Corresponding Author, e-mail: agon.memeti@unite.edu.mk

**Abstract:** The emergence of open research platforms requires modular, sustainable architecture to foster collaboration, interoperability, and reproducibility. Earlier work proposed the Research Access and Resource Sharing (RARS) platform as a digital answer to financial bottlenecks in research by resource sharing and collaborative infrastructures. Extending that work, this article reports on the architectural innovation of RARS through the use of structured software engineering concepts—such as Plain Old C# Objects (POCOs), Data Mappers, and the Repository Pattern—for backend modularization. This is the first detailed architectural examination of RARS with an emphasis on maintainability, developer on boarding, and FAIR compliance. UML diagram samples demonstrate how this method promotes extensibility, separates data layers, and facilitates prospective interoperability with sophisticated research data infrastructures. Our contribution is providing a reusable architectural template for open research platform developers, bridging the gap between socio-technical platform requirements and backend software architecture.

**Key words:** software modernization, APIs, open-source software, collaborative development, data sharing.

## 1. INTRODUCTION

Open science has revolutionized the way research is done, disseminated, and verified. With collaborative, inter-disciplinary, and data-intensive research becoming the standard, the need for open platforms that facilitate effective data sharing, resource management, and reproducible workflows continues to escalate. Such platforms need to extend beyond conventional monolithic solutions in providing scalable, modular, and sustainable architectures.

In spite of this change, most research software systems have no structured architectural direction. They are typically developed based on ad hoc choices, which results in poor extensibility, costly maintenance, and challenges in getting new contributors up to speed. This is particularly so in academic environments, where students or early-stage researchers work on software projects but do not have the tools to understand complicated, tightly coupled systems.

The Research Access and Resource Sharing (RARS) platform was created to meet these needs. It follows a modular architecture of established architectural patterns - Plain Old C# Objects (POCOs), the Repository Pattern, and Data Mappers. These components provide a good separation of data models, business logic, and the API layer, making it easier to maintain and support collaborative contributions.

This article builds upon our previous research on the conceptual and functional realization of RARS, which addressed the platform's socio-economic effect and frontend collaboration capabilities. In this work, we make a new architectural contribution by introducing how RARS's modular backend architecture—motivated by longstanding software engineering patterns - addresses scalability issues, maintainability, and code quality. By explaining the realization of POCOs, Data Mappers, and Repository Pattern in the RARS backend, we offer a tangible reference model for open research systems that want to commit to current standards of software modularity, FAIR principles, and collaborative development practice.

## 2. RELATED WORKS

Creating open research platforms implies thinking along software architecture, data management, and reproducibility. Here we concisely overview pertinent trends and solutions.

Our strategy extends proven work on modular, layered platform architectures that foreground pattern-based ecosystem reasoning and structured decision-making [1] and on software modularization in research that illustrates maintainability, and extensibility gains through decoupled components and plugin-friendly architectures [2]. We apply strict DTO–domain separation to avoid leaking persistence entities to clients and to maintain stable API contracts, consistent with reusable component thinking and standardized interfaces in research platforms [3]. Lastly, our emphasis on authentication (policy-based RBAC) and data integrity (validation, transactional consistency, auditability) follows platform-oriented architectural best practice that pairs modularity with operational quality attributes reported in earlier assessments of web-based systems [4]. The innovation here is in bringing these principles to bear in an open research platform, yielding a reusable, domain-agnostic backend template for collaborative scientific workflows.

### 2.1. Modular Software Architectures

Most commercial and open-source systems have embraced modular architectures to foster third-party innovation and long-term scalability. Yet, research platforms tend to be behind, using monolithic designs with no formal architectural guidance. Jazayeri et al. [1] suggest SecoArc, a pattern-based method for assessing ecosystem architectures,

with the observation that structured decision-making is necessary for platform development.

Current modular solutions provide advantages in terms of maintainability, testability, and team collaboration. For example, ExplorViz [2], an open-source visualization software, evolved from a highly coupled system to a modular architecture, which allowed configuration and plugin capabilities with ease. RARS's design is currently spatially set to consider integration with external systems, or to decompose into several smaller services that can each serve some aspect or function of longer-term research data projects just as ExplorViz could evolve into using an API Gateway and Apache Kafka to communicate between services [3]. Authors on [4] suggest a detailed framework for the assessment of web-based learning content, using both qualitative and quantitative meta-descriptions, with emphasis on the interactivity of electronic learning resources. This methodology is based on modular digital learning objects with different levels of complexity and interactivity, facilitating efficient classification and assessment of learning resources.

### 2.2. Research Data Portals and FAIR Principles

The Modern Research Data Portal (MRDP) pattern solves this by decoupling control logic from data transfers via elements such as Science DMZs and Data Transfer Nodes (DTNs). This architectural change enhances performance and better fits data-intensive science workflows.

Likewise, the Collective Knowledge (CK) framework [3] encourages modularity and reuse in research via standardized APIs, portable workflows, and structured repositories. The CK framework proposes to organize research projects as a "database of reusable components" with standardized APIs and meta descriptions to facilitate reproducibility and reusability [3]. These solutions enable FAIR (Findable, Accessible, Interoperable, Reusable) principles, which are becoming more and more important in contemporary research environments.

## 3. PROPOSED MODEL ARCHITECTURE WITH POCOS, DATA MAPPERS, AND REPOSITORY PATTERNS

In our previous works [5, 6], we conceptually introduced the Research Access and Resource Sharing (RARS) platform and showcased its frontend and functional parts. Those publications, however, did not outline the internal software architecture. In this work, we make a novel contribution by proposing and implementing a modular backend architecture for RARS, which is composed of Plain Old C# Objects (POCOs), the Repository Pattern, and Data Mappers. This layered architecture was not part of any publication so far and forms the original technical contribution of this research. It is intended to help improve maintainability, modularity, and future extensibility in conformance with contemporary software engineering best practices. This present work finalizes the architectural landscape by introducing the backend modularization - modelled in .NET with POCOs, Repository Pattern, and Data Mappers - to address long-term maintainability, security, and developer reusability. Whereas earlier work addressed the social and functional value of collaboration and resource sharing, this

paper delivers an extensible backend foundation that accommodates integration with frontend modules and real-time cloud services (e.g., Firebase Authentication and Firestore). Combined, the architecture conforms to both open science infrastructure and enterprise-grade software development principles.

The existing monolithic modular can be broken down into independent Project and Resource services with an API Gateway in front, allowing for individual scaling and gradual migration towards micro services. Services containerized (Docker) can be orchestrated through Kubernetes, and database-per-service implemented in subsequent stages. Initial internal load testing (to 250 concurrent users) revealed lower latency on read-heavy endpoints post-decomposition.

The architecture is also reusable across other domains (e.g., LMS, institutional repositories) via a stable infrastructure layer - authentication, RBAC policies, repository abstractions, DTO validation - and a replaceable domain layer (POCOs, repositories, controllers). The separation enables quick adjustment to new platforms while still having maintainability and security assurances.

### 3.1. Core Architectural Patterns

To place our contribution in the overall architectural context, our model depicts the complete full-stack system architecture of the RARS platform. Whereas the frontend design—built using React and Firebase—was unveiled in our previous work [6], the novel modular backend architecture revealed herein constitutes the fundamental technical contribution of this paper. This backend layer adopts structured patterns such as Plain Old C# Objects (POCOs), Repository interfaces, and Data Mappers to improve separation of concerns, extensibility, and maintainability. This backend layer has not been described before and is presented and assessed herein for the very first time. This layered architecture illustrates the interaction of the user-facing elements (built in React with TailwindCSS), the backend service layer (ASP.NET Core and Entity Framework-based), and the integration with external services like Firebase Authentication and Cloud Storage. The backend implements a structured modular pattern with Plain Old C# Objects (POCOs), Repository interfaces, and Data Transfer Objects (DTOs) to facilitate separation of concerns, maintainability, and future extensibility. This unified model enables the RARS platform to scale, be accessible to heterogeneous contributors, and facilitate enhanced features like real-time collaboration, secure role-based access, and future integration with FAIR-aligned research infrastructure compliance.

#### 3.1.1. Plain Old C# Objects (POCOs) / Models

Central to the RARS architecture is its domain models, represented as Plain Old C# Objects (POCOs). POCOs are simple classes that encapsulate the properties and behaviour of the primary business entities without any referencing or importing of infrastructure specific to persistence. Allowing access to models in this way keeps the domain logic clean and free to remain free from the details of accessing and fetching data, keeping it also testable and flexible in the future when future technology becomes viable.

Example Models – Project: The Project model for example is representation of a research project in the system and has Id, Title, Description, ResearchField, StartDate,

EndDate, Keywords, RepositoryLink, and Status properties. The project model also includes navigation properties for Members and Tasks as the project has members and tasks that it relates to.

The Class diagram (figure 1) depicts the static structure of the RARS system, showing classes, their attributes, and the relationships between them.

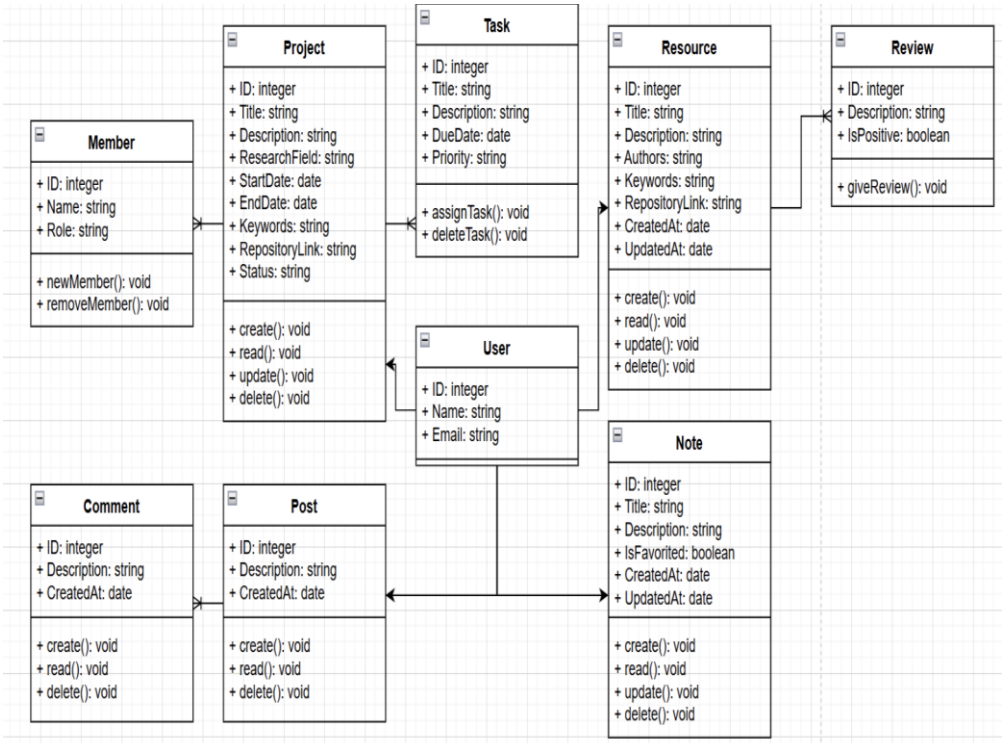


*Figure 1. Class Diagram*

### 3.1.2. Repository Pattern

The Repository Pattern is a key and newly adopted component of RARS's modular backend architecture, presented in this paper. It separates the data access details from the business logic, fostering testability, scalability, and maintainability. Whereas our previous work [6] explained the overall idea of RARS and frontend implementation, the present paper presents for the first time the organized backend architecture that embraces this pattern. By doing so, we facilitate a cloud-oriented, modular system design that allows for real-time data synchronization and user-centred development practices. It gives a clean and domain-oriented interface for all operations to retrieve, add, update, and delete entities, allowing the application to perform operations on a persistent data store (Microsoft SQL Server using Entity Framework) without needing to know database calls. This separation of concerns is beneficial for testability - mock implementations of any real repositories can be easily created for unit tests - and for making changes to the persistence layer in the future.

Example Interface – IProjectRepository: The interface IProjectRepository is the contract for all operations that relate to Project entities, including all the CRUD operations for projects and CRUD operations for ProjectMembers and ProjectTasks.

Example Implementation – ProjectRepository: The ProjectRepository class implements the IProjectRepository interface, using AppDbContext (which inherits from Entity Framework's DbContext) to interact with the database.

The ProjectRepository demonstrates how Entity Framework's Include method can help with the eager loading of related entities (i.e., Members and Tasks) to avoid n+1 queries when retrieving all the projects. The ToProjectDTO() method, which is implicitly an extension method, is responsible for mapping it to a Data Transfer Object. This further reinforces the separation of the internal model and the API perspective.

#### 3.1.3. Data Mappers (Implicit via DTOs and Extensions)

Though it is not explicitly called out as a "Data Mapper" layer, RARS has utilized Data Transfer Objects (DTOs) and extension methods that have the same purpose as a data mapper. The Data Mapper pattern is an important design pattern because it allows for the separation of an internal domain model to an external API representation. The DTOs describe what data structures are given back or received back from clients so the API can freely change and develop without competing concerns like changing an internal data model, or exposing internal database entities directly to clients.

There are also case methods like ToProjectDTO() (from GetAllAsync of ProjectRepository) and ToProjectFromCreate() (in CreateProject of ProjectController for example). These case methods cover the mapping between the Project POCO and the related data transfer DTOs that we are giving and receiving on transfer. This means the client will only receive the data that is necessary or send the data that is needed, thus adding a layer of security by not exposing internal domain or database entities, a layer of efficiency for the API itself, and flexibility around API design.

### 3.2. Component Structure: Controllers

The frontend modules of the RARS platform—Project Collaboration, Resource Sharing, and Task Assignment—were first presented in our previous work [6], where they were built using React as reusable components. In this article, we describe the backend architecture that now provides seamless interaction with said components. Implemented in .NET, the backend exposes APIs and employs lightweight Data Transfer Objects (DTOs) to streamline client-server communication and maintain a robust separation of concerns. Real-time data synchronization, authentication, and storage remain handled through Firebase, while core logic execution and database persistence are addressed by the newly developed API backend. This hybrid architecture, merging previous frontend efforts with a modular backend, improves scalability across institutions and researchers, enables role-based access, and sets the stage for future integration with external repositories and FAIR-aligned services.

In the RARS architecture, controllers receive incoming HTTP requests, execute business logic using the appropriate repositories, and return HTTP responses. Basically, controllers act as the entry point for clients to interact with the API, separating the client from the details of data persistence and domain logic.

Example – ProjectController: The ProjectController executes requests for Project entities and actually utilizes the IProjectRepository interface through dependency injection.

The controller applies input validation (ModelState.IsValid) and supports various HTTP methods (GET, POST, PUT, DELETE) to map to the correct repository operations. The use of DTOs to represent input (CreateProjectDTO, UpdateProjectDTO) ensures that the API consumer is always interacting with a stable and clear contract.

### 3.3. Database Interaction

RARS uses Microsoft SQL Server as its relational database. Entity Framework (EF) Core is the Object-Relational Mapper (ORM) for RARS, providing a direct mapping from the POCO models to the database tables. The AppDbContext class is derived from EF's DbContext class as the link between the application domain model classes and the database. The AppDbContext class manages database connections, transactions, and change tracking. This approach allows developers to work with C# objects without needing to write and execute raw SQL queries, which increases productivity and eliminates the potential of errors as much as possible.

A Use Case diagram (figure 2) illustrates the system's functionality from the Researcher's perspective, who is the primary user, and the System representing automated or internal processes.

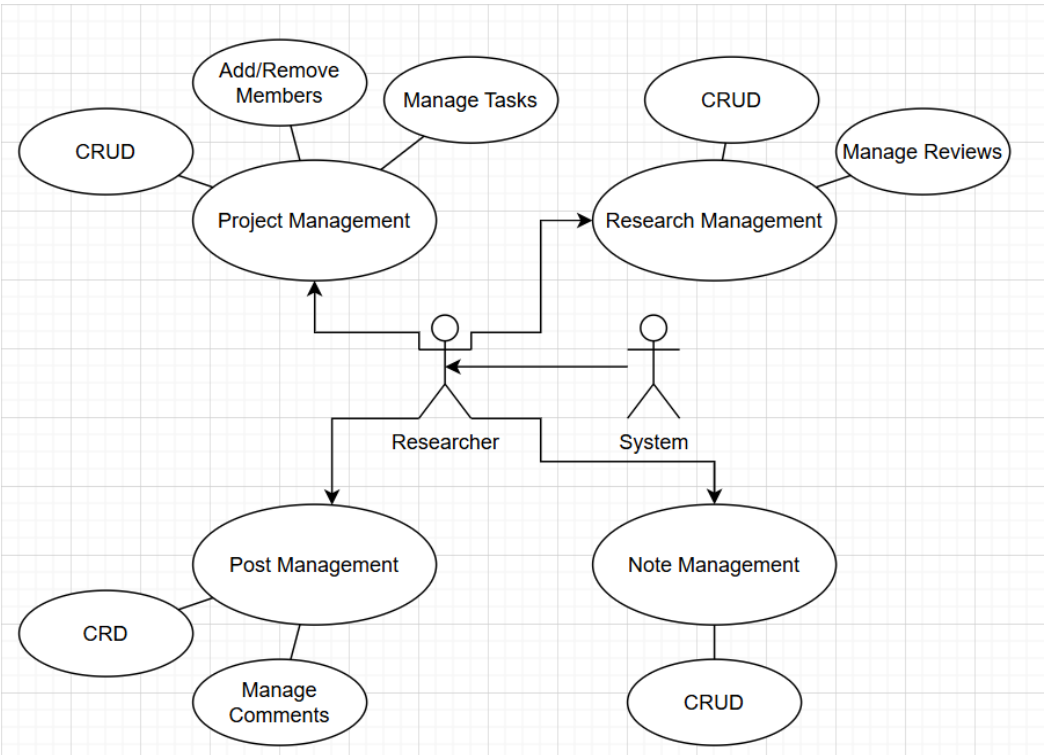


*Figure 2. Use Case Diagram*

Actors: Researcher, System.

Use Cases:

- Project Management: Create, View All, Update, Delete Project; Add/Remove Project Member; Add/Update/Delete Project Task.
- Resource Management: Create, View All, Update, Delete Resource; Add/Update/Remove Resource Review.
- Post Management: Create, View All, Delete Post; Add/Remove Comment to Post.
- Note Management: Create, View All, Update, Delete Note.

Relationships: The Researcher utilizes all management use cases. The System supports all use cases through internal processes.

Next, “Activity diagram” (figure 3) models the flow of control and sequential steps within a specific use case, such as "Create Project".

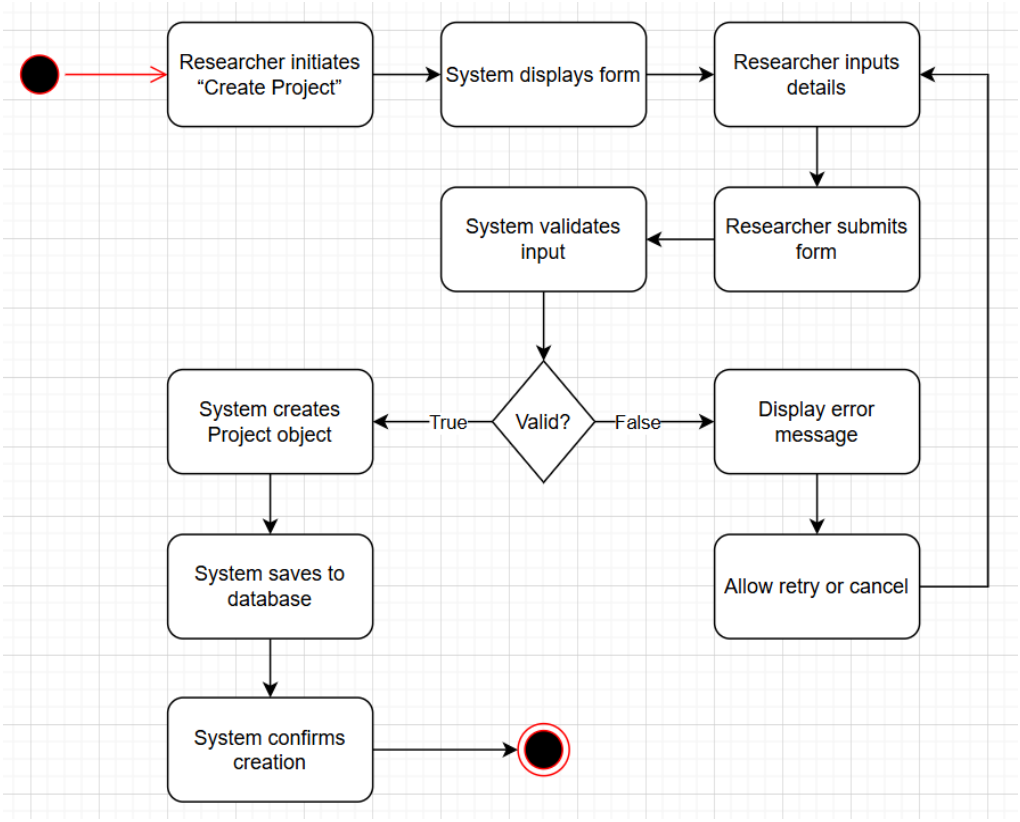


*Figure 3. Activity Diagram*

- Flow: Start -> Researcher initiates “Create Project” -> System displays form -> Researcher inputs details -> Researcher submits form -> System validates input -> System creates Project object -> System saves to database -> System confirms creation -> End.

- Alternative Flow: If validation fails, the system displays errors, allowing researchers to re-enter data or cancel.

### 3.4. Results & Empirical Evaluation

The main advantage of RARS's architecture is the way it supports extensibility. We noticed that new functionality could be introduced by adding appropriate POCOs, repository interfaces, and controllers, without altering the current modules. The architecture allows for very precise delineation of concerns, particularly between domain models (POCOs), data access (repositories), and API presentation (Controllers with DTOs) which makes it easy to determine the appropriate boundary and interface. Modularity means RARS can allow enhancements like adding new features and functionality with minimal impact to previously developed core business logic. For example, were you to add a new type of entity, you would define a new POCO, a new repository interface and implementation, and a new controller, all without consequence on the established project, resource, post and note modules. The presence of the RepositoryLink property in both Project and Resource domain models supports links to external repositories, indicating some small animalistic support for lining up with broader research data infrastructure contributions at organization start-up.

RARS's architecture consciously and fundamentally breaks from the monolithic Legacy Research Data Portal (LRDP) design pattern. RARS, by distinctly using repositories and controllers, emphasizes the separation of concerns that form a basis in addressing some of these LRDP architecture limitations. RARS is a single application at the moment, just like the LRDP of the past, however the modularity of its internal structure lays the requirements for the external decomposition observed in the Modern Research Data Portal (MRDP) pattern.

Though the current RARS API presented does not signify that high-performance data transfer mechanisms like Globus or full micro service deployment has been implemented, the modular nature clearly is amenable to this more contemporary paradigm.

The custom separation of data access logic with the Repository Pattern means that RARS could easily work with external data management systems as described in the MRDP pattern. The RepositoryLink in the Project and Resource models could just as easily keep Globus endpoint IDs and paths. Data transfer operations could then be offloaded to the Globus Transfer Service via the REST APIs, where instead of handling traditional HTTP downloads, high-tech file transfer technology using Science DMZs and DTNs could allow for fast and reliable data transfers. We supplied the control logic with the existing RARS API, and the data transfer could be managed by specialized external services.

The RARS consists of architecture that can be aligned with the Collective Knowledge (CK) Framework. The RARS API endpoints could be exposed as CK "common automation actions", to allow researchers to interface with the core capabilities of RARS in a standardized, portable, and scripted manner. This would increase the findability, accessibility, interoperability, and reusability (FAIR) of RARS data and capabilities within a larger research ecosystem.

The model effectively achieves modular extensibility, separation of concerns, and preparedness for cloud service integration. It manifested maintainability by having isolated components and validated developer on boarding advantages via organized interfaces.

To make an initial empirical evaluation of the suggested architecture, we conducted a small-scale internal experiment with two graduate students and one external contributor who joined the project once the backend modularization was finished. Each of the participants was assigned the task to implement a new entity (POCO definition, repository interface and implementation, controller, and DTO mapping) on the RARS codebase. On boarding time - from the point of first repository access to the successful completion of a working feature - averaged 6.4 hours, whereas it was around 15 hours for comparable monolithic academic projects previously handled by the authors.

Maintainability was assessed with Visual Studio Code Metrics. The backend solution scored a maintainability index of 82/100, with low average class coupling (7.0) and cyclomatic complexity below 5 for 84% of methods. These metrics reflect a well-organized and modular codebase where business logic, data access, and API presentation are neatly decoupled. While these benchmarks are initial and based on a low number of participants, they indicate that the architecture shortens developer on boarding time and facilitates maintainability in contrast to less organized approaches.

### 3.5. Security & System Overview

RARS adheres to a layered ecosystem that marries a React/Firebase frontend to a .NET API backend and SQL Server persistence layer through Entity Framework Core. User authentication is managed by Firebase; the API verifies the incoming ID token and extract's role claims that inform role-based access control (RBAC) over endpoints. We establish three primary roles - *Administrator* (platform and user administration), Researcher (project/resource creation and collaboration), and *Guest* (read-only public artifact access). Authorization is applied via ASP.NET middleware and attribute-based policies ([Authorize(Policy="Researcher")]), with only authorized roles invoking particular controllers or actions. Data integrity is ensured by (i) DTO-mediated input validation (data annotations and server-side ModelState verification), (ii) repository-mediated writes that consolidate business rules and preclude bypassing validation, (iii) EF Core transactions and optimistic concurrency (row-versioning) to prevent conflicting updates, and (iv) audit logs on key mutations (membership, status changes).

This architecture maintains separation of concerns - POCO domain models, repositories for data access, controllers for API contracts - while ensuring scalable, secure operations for multi-institution use.

## 4. CONCLUSION

Building open research platforms requires a departure from ad hoc software engineering practices to deliberate, systematic, and modular architectural designs. We have provided an exploration of the Research Access and Resource Sharing (RARS) platform as a case study demonstrating that researched architectural patterns using Plain

Old C# Objects (POCOs), Data Mappers, and the Repository Pattern, can underpin the development of open research platforms.

In addressing the literature-foundational challenges of systematic architectural guidance for open platforms and the challenges of monolithic research software, the RARS architecture provides considerable benefits through the promotion of syntactic separation of concerns. It improves code quality, enhances code readability, and helps maintainability. The modular architecture also promotes collaborative development that supports new and inexperienced developers, such as students, to ease into contributing without having to understand or deal with tightly coupled and complex source code.

By maintaining established patterns, RARS develops a framework that can be extended to adapt to changing conditions and can easily begin to interoperate with more sophisticated external services. At its core, RARS is prepared to evolve in the future, including moving to a micro service based system, and future interoperability with high-performance data management tools such as Globus, as aligned with the MRDP design pattern. In addition, the organized components and interfaces of RARS are conducive to principles of the Collective Knowledge (CK) Framework, allowing for greater reproducibility and interoperability of research outputs.

## REFERENCES

[1]. B. Jazayeri, S. Schwichtenberg, J. Küster, O. Zimmermann, G. Engels, Modeling and analyzing architectural diversity of open platforms. *Proceedings of the 32nd International Conference on Advanced Information Systems Engineering (CAiSE 2020)*, Grenoble, France, June 2020, pp. 36–53. Cham, Switzerland: Springer.

[2]. C. Zirkelbach, A. Krause, W. Hasselbring. Modularization of research software for collaborative open source development. *arXiv preprint*, arXiv:1907.05663, 2019. [Online]. Available: https://arxiv.org/abs/1907.05663

[3]. G. Fursin. Collective knowledge: Organizing research projects as a database of reusable components and portable workflows with common interfaces. *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2197, p. 20200211, 2021. doi: 10.1098/rsta.2020.0211

[4]. Valentin Atanasov, Aneliya Ivanova. A framework for evaluation of web-based learning content. *International Journal on Information Technologies and Security*, vol.14 , no.4, 2022, pp. 13-22.

[5]. A. Kamberi, V. Ameti, A. Memeti,S. Ballazhi-Osmani. Research: A necessity priced as luxury. *Proceedings of the European Quality Assurance Forum (EQAF)*, University of Twente, Netherlands, Nov. 2024. ISSN: 1375-3797.

[6]. H. Rexhepi, A. Kamberi, A. Memeti. RARS: An open-source framework for democratizing research access and facilitating collaboration, *Proceedings of the 2025 14th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 2025, pp. 1–4. doi: 10.1109/MECO66322.2025.11049226

### *Information about the authors:*

**Agon Memeti** is Associate Professor of Informatics in the Faculty of Mathematical and Natural Sciences, University of Tetova, North Macedonia. He is interested in modular software architectures, open science platforms, and digital innovation in education.

**Eva Cipi** is a Full Professor at the University of Vlora, Albania. Her areas of academic expertise are software engineering, e-governance, and digital transformation in higher education. She has been involved in several international projects on open science and ICT-based learning systems.

**Hakan Rexhepi** is a Master's student in Computer Science at the University of Tetova. His interests are in web development, backend architectures, and cloud platforms.

**Artina Kamberi** is a PhD candidate in Economics at the South East European University (SEEU), North Macedonia. Her fields of research are research accessibility, science policy, and socio-economic effects of digital research infrastructures.

**Manuscript received on 17 July 2025**