ISSN: 2317-0840

Developing a Robust Shiny Application Integrated with SQL Databases for Aquaculture

Daniel Pinto Siqueira[1†] Gabriel Rodrigues da Silva[1], Luan Patrick Moura de Souza[1], Carlos Antônio Zarzar[1] Paulo Roberto Brasil Santos[1]

[1]*Federal University of Western Pará (UFOPA), Monte Alegre campus*

Abstract: *This work presents the development of an application for managing collective feed purchases in aquaculture, aiming to reduce costs and optimize logistics. This study aims to validate the robustness of applications developed using the Shiny framework of the R programming language by integrating a Database Management System (DBMS) in the backend. This aims to demystify the predominant perception that Shiny is exclusively a prototyping tool, demonstrating its viability for robust production applications. The adopted methodology encompasses three stages of software development: Backend, Intermediate Infrastructure, and Frontend. In the Backend stage, database modeling was carried out, which included conceptual, logical, and physical modeling, ensuring the system's efficiency and security. The choice of PostgreSQL as DBMS was based on the need for a robust and secure system. The Intermediate Infrastructure was developed using the R language with the Golem framework, facilitating the application's development and deployment. Its folder and metadata structures promote efficient project organization. The Frontend was built with the Shiny framework, standing out for its modularity and code organization. The developed modules include the interface and system logic for each specific functionality, such as supplier registration and stock management. Overall, the work presents a systematic and organized approach to application development, encompassing database modeling to user interface implementation, aiming to meet the needs of collective feed purchase management in aquaculture.*

Keywords: *Web Application; Aquaculture; R software; PostgreSQL; Shiny.*

[†]Autor correspondente: daniel.pis@discente.ufopa.edu.br

Manuscrito recebido em: 31/07/2024
Manuscrito revisado em: 24/10/2024
Manuscrito aceito em: 29/10/2024

Introduction

The feed cost is one of the main factors in aquaculture production, representing a significant portion, usually between 50% and 80% of the total required during a production cycle in fish farming. This is due to the substantial costs associated with feed formulation by industries. To ensure successful fish production, it is crucial to have tools that cover a variety of parameters related to feed acquisition and use. Spreadsheets are a viable option for organizing essential production information, allowing for strategy formulation adaptable to specific farm requirements.

Although many producers use Excel spreadsheets for fish production planning, their effectiveness is limited when detailed technical information is needed. This is because utilizing these tools requires skills that not all producers have. An alternative approach to efficient fish farming planning, which does not rely on specialized skills, is the adoption of easy-to-use designer programs developed with a focus on user experience, thereby facilitating and expanding the user base of these software tools. The R programming language, initially created for a community of statisticians, has been growing in recent years, and it is now possible to develop web applications that can include data modeling functionality when appropriate.According to Batista e Oliveira (2022), the R environment is an integrated structure of software resources designed to manipulate data, perform calculations, and generate graphical representations. Together with the R language (RDEVELOPMENT, 2011), RStudio constitutes an integrated environment that allows for expansion through the inclusion of packages, enabling the creation of Shiny applications, the preparation of reports using RMarkdown, and the integration of other packages into the R language. Although the use of RStudio for the development of Shiny apps is not widely adopted among developers due to the stereotypical perception that R along with Shiny is intended only for prototyping applications, this conception is considered restrictive. In reality, RStudio can function as a primary platform for development, providing resources for the creation of robust applications and offering the opportunity to develop applications for free.

Database modeling plays a crucial role in the field of data science, being essential for the development of efficient and well-structured systems. This process involves the representation and organization of data handled by an application or system, aiming to understand, visualize, and manage information optimally. It consists of creating abstractions and representations that describe relationships and the structure of data storage and access. According to Costa, Resende e Silveira (2008), this practice is one of the fundamental activities in software development.

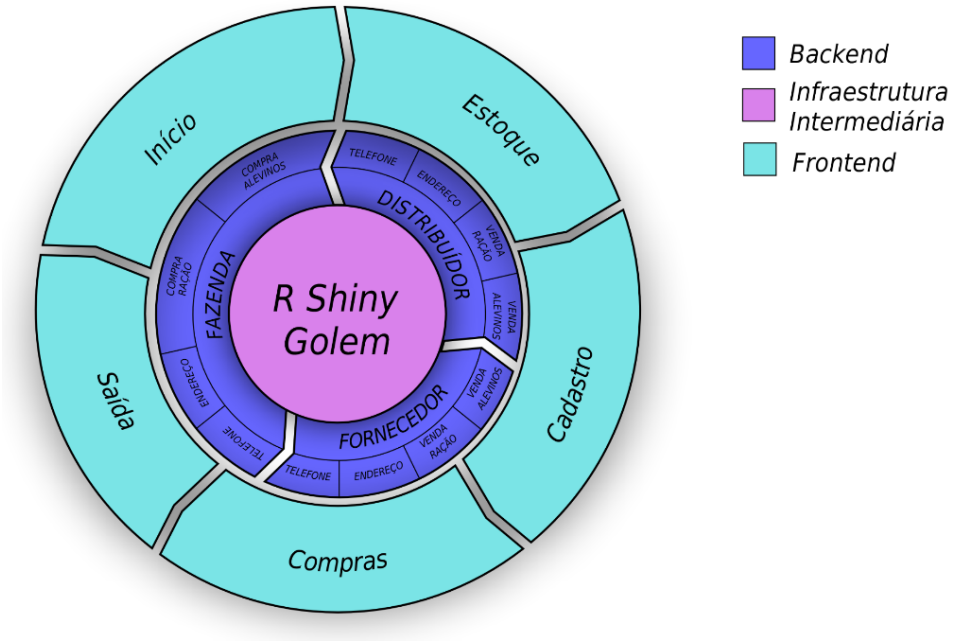
The resulting models act as a link between the understanding of the problem domain and its technical implementation, facilitating clearer communication between business professionals and software developers. Thus, this work proposed the integration of the R programming language with a web application based on the Shiny framework, also using the SQL programming language (relational database) for manipulating relational databases, in addition to applying database modeling concepts. The objective was to develop a robust application aimed at aquaculture for managing collective feed purchases in aquaculture to reduce production costs.

Material e Methods

The application aims to facilitate collective feed purchases among fish farming producers, aiming to reduce costs and optimize logistics and storage. For this, it records feed purchase transactions, including details such as manufacturing date, batch, manufacturer, supplier, and distribution to producers associated with the cooperative.

For the preparation and construction of the application, we adopted a methodological approach divided into three distinct stages: Backend, Intermediate Infrastructure, and Frontend, as illustrated in the diagram presented in Figure 1. This division allowed for a clear and systematic organization of the process of creating, designing, and building the application, facilitating resource management and progress monitoring in each development phase.

Figure 1: Diagram representing the methodological approach divided into three stages (Backend, Frontend, and Intermediate Infrastructure) of the Software Architecture for application development.



Source: from the authors (2024).

Backend

The software development area is widely addressed with architectures categorized into backend, frontend, and fullstack. The backend plays a crucial role by connecting the database and managing the infrastructure required for the application to function, in addition to implementing the business logic.

In this project phase, the structural model of the database is developed, defining entities, attributes, and relationships according to the software application, and selecting the Database Management System (DBMS). Database modeling is divided into three main levels: conceptual, logical, and physical, each intended to be applied in a DBMS.

Conceptual modeling

The conceptual model visualizes entities, relationships, and key attributes of the problem, being an abstraction close to the users' reality. In this phase, understanding the requirements was prioritized, observing relationships, and identifying data integrity issues.

Technological issues were not considered in this phase because some model components are not compatible with relational database resources. This requires transforming the Entity-Relationship Model into a suitable notation for this implementation. The resulting logical model depends on the characteristics of the chosen DBMS.

Logical modeling

The logical model was based on the conceptual model, which logically describes how data is stored and related to each other, obtaining a graphical representation of the database structure. In the logical model, physical implementation details were not considered, but the conceptual form of the data was described, allowing for flexible system projection.

At this stage, the data to be stored in the system was presented, focusing on concepts and relationships between tables (entities). Additionally, normalization methods were applied to ensure data integrity. More details about the method can be found in (CODD, 1970).

Physical modeling

The physical data modeling was derived from the specific elements of the logical model that directly influence the system's development and efficiency without modifying its essential functionalities. This model described how the data is actually structured.

This structuring involves defining the DBMS to be used in the project, identifying the attributes of the entities relevant to the business logic of the application or system, culminating in the creation of tables (entities), columns (attributes), and their respective primary keys and foreign keys. This is essential to establish the relationships between entities.

Intermediary Infrastructure

The application's intermediate infrastructure was developed using the R programming language in the RStudio environment. We chose R because it is open source, has an active community, graphical capabilities, and is effective in computational analyses. It acts as communication between the backend and frontend of the software.

The workflow was integrated with the Golem package (FAY et al., 2023), facilitating the rapid and robust development of the application. The toolkit simplifies the creation, development, and deployment of Shiny applications, with results showing the final structure and discussing its advantages and disadvantages.

Frontend

The application's frontend was built with the R Shiny framework (CHANG et al., 2022) using Shiny Modules to divide the application into independent parts. We used the shinydashboard package for the interface and followed Shiny's reactivity paradigm, prioritizing which outputs should be updated when an input is triggered. This includes reactive values, reactive expressions, and observers.

Results and discussion

Backend

For the backend development of the proposed software applied to aquaculture, a comprehensive survey of all potentially involved variables in the application's functional process was necessary, understanding their inherent relationships to the phenomenon, and pointing out the main attributes of the entities for backend construction.

This survey aimed to structure the tables according to the business development of the application in the most appropriate way in terms of database modeling. For the modeling, three main stages were divided: conceptual modeling, logical modeling, and physical modeling. Below are the results for each stage of development in this phase of building a robust application.

Conceptual modeling

At this stage, the primary interaction occurs between specialists in the application's area of interest and project developers. Through interviews with these specialists, the main entities related to the application's business logic were identified, as well as the practical needs in the field where the application is intended to operate, aiming to contribute to solving specific problems in aquaculture.

For the application of collective feed purchase among farmers in aquaculture, the main entities identified were: Manufacturer (feed suppliers), Distributor (sellers who distribute feed), Feed (physical and nutritional characteristics), Fingerling (morphological characteristics and species), Purchase (purchase orders), Fingerling Purchase (relation between purchase and fingerling type), Feed Purchase (relation between purchase and feed type), Inventory (quantity of feed stored), Output (total quantity

withdrawn), Output Feed (quantity of feed withdrawn), and Fingerling Output (quantity of fingerlings withdrawn).

In the survey of the main entities for the database, we also included: Telephone (registers mobile phone numbers), Address (registers addresses), Owner (registers owners), and Farm (registers farms of the respective owners).

Logical modeling

The logical model represents graphical information requirements from conceptual model data descriptions. While simplifying conceptual modeling for greater flexibility and clarity, the lack of context can make tables difficult to interpret, confusing developers during the transition from modeling to data system deployment.

In this database modeling, the results clarify the entities essential to the functioning of the application, defining their relationships to build the backend efficiently and securely. All stages of development have been documented to guide future developers. In this phase of database modeling, the relationships between the entities identified in the conceptual model (Figure 2) and the discussion about the backend structure for optimization are highlighted. In Figure 2, it is clear that the entities Feed and Fingerling are related to the Distributor which, in turn, relates to the Manufacturer.

Feed and fingerlings must be registered in the system, with data such as addresses, telephone numbers, and owners. The Manufacturer, Distributor, Farm, and Owner entities are linked to the Address and Phone tables. Initially, Address and Phone were considered attributes of these entities, but due to data duplication and possible multiple phone numbers per owner or vendor, they are treated as separate entities to ensure the robustness of the system as data grows.

In the system, the registered feeds provide information on physical, chemical, nutritional characteristics, suppliers and, local manufacturers. They are purchased in large quantities for distribution to local fish farms, reducing logistics costs. Records of feed entry and exit are necessary for tracking, generating the Purchase and Exit entities, related to Feed, Fingerlings, Farm and Stock.

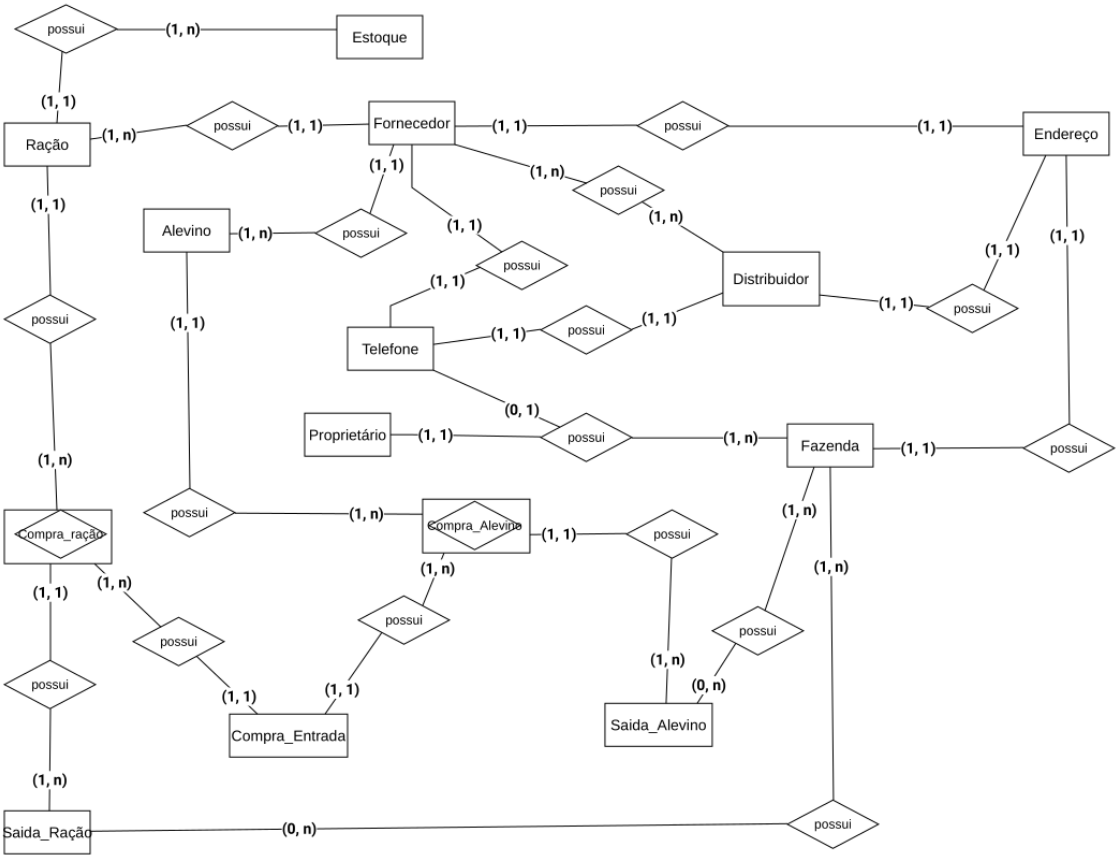
Based on the table normalization technique, the Purchase and Output entities were derived into two relational entities (Purchase-Feed and Output-Feed), to reduce their relational cardinalities (N:N), improving the efficiency of information requests (query) between the intermediate program and the backend. These intermediate tables facilitate the exchange of information between R and the backend and minimize domain conflicts.

Physical modeling

The graphical representation of logical data modeling was used to build the database, outlining segments relevant to reality based on the requirements consolidated by previous models. During the data modeling process, the elements were designed at schematic levels, defined by sets that relate to the database structure (Figure 2).

Based on the Relational Database Management System (RDBMS), the SQL programming language was determined for this project, as well as PostgreSQL, which was chosen as the relational database management system, which used tables and columns from the logical modeling to structure the database. Recognized for its open source nature, PostgreSQL stands out for its affordability, flexibility in text indexing, comprehensive technical support and efficient code maintenance, as well as natively providing access to large volumes of data.

Figure 2: Relational structure defined in the logical modeling of the database for the construction of collective feed purchase software applied in aquaculture.



Source: from the authors (2024).

The types of data to be used were carefully considered, ensuring adequate protection for the storage of data and the operations that will be carried out on it. In this context of complex data systems, the selection of this platform for data storage was of fundamental importance for the development of the web application that required such reliability attributes, as is the case with PostgreSQL.

The attributes outlined the specific characteristics of each entity proposed in the project, such as the Manufacturer entity, which specifies the name of the manufacturer (Varchar 20), the type of product manufactured (Varchar 15), the foreign key to the address ID (Foreign Key - FK), the foreign key for the phone ID (Foreign Key - FK) and the primary key for the manufacturer ID (Primary Key - PK). The Distributor entity describes the distributor ID (PK), the distributor name (Varchar 20), the foreign key to the phone ID (FK), the foreign key to the address ID (FK), the type of product distributed (Varchar 15) and the foreign key to the manufacturer ID (FK). The Feed entity details the feed ID (PK), the name of the feed desired by the buyer (Varchar 20), the size of the feed pellet (Integer), the type of feed desired considering the animal's phase (Varchar 20), the Manufacturer ID (PK), the percentage of feed protein (Real), the minimum ether extract in g/Kg (Real), the maximum moisture presented in percentage (Real), the maximum amount in g/Kg of minerals, fiber and calcium, with all attribute values ??defined as Real, in addition to the minimum quantities established for calcium, phosphorus and vitamin C in g/Kg, with values ?? defined as Real.

Additionally, there are other entities with distinct attributes, such as the Fry entity, which includes the following attributes: Fry ID (PK), Manufacturer ID (FK), Fry Production (Varchar 20), Fry Nickname (i.e. the popular name of the fish), (Varchar 20) and the sex of the desired fry (Varchar 10). The Purchase entity has attributes such as purchase ID (PK), quantity of selected items (Integer), total quantity purchased in kg of feed or thousands of fingerlings (Numeric), total purchase value (Numeric), date and time of purchase (Date), date and time of arrival of the purchased product (Date) and the type of purchase, whether feed or fry (Varchar 10). The Fry Purchase entity has attributes such as fry purchase ID (PK), purchase ID (FK), fry ID (FK), supplier ID (FK), distributor ID (FK), unit value of the millet sold by the manufacturer (Numeric), quantity per million of specific fry sold (Real), entry value (Numeric), average weight of individuals in grams during the purchase of fry (Real), days of life of individuals at the time of purchase (Date) and the code corresponding to the purchased batch and the manufacturer's code for identification (Varchar 30).

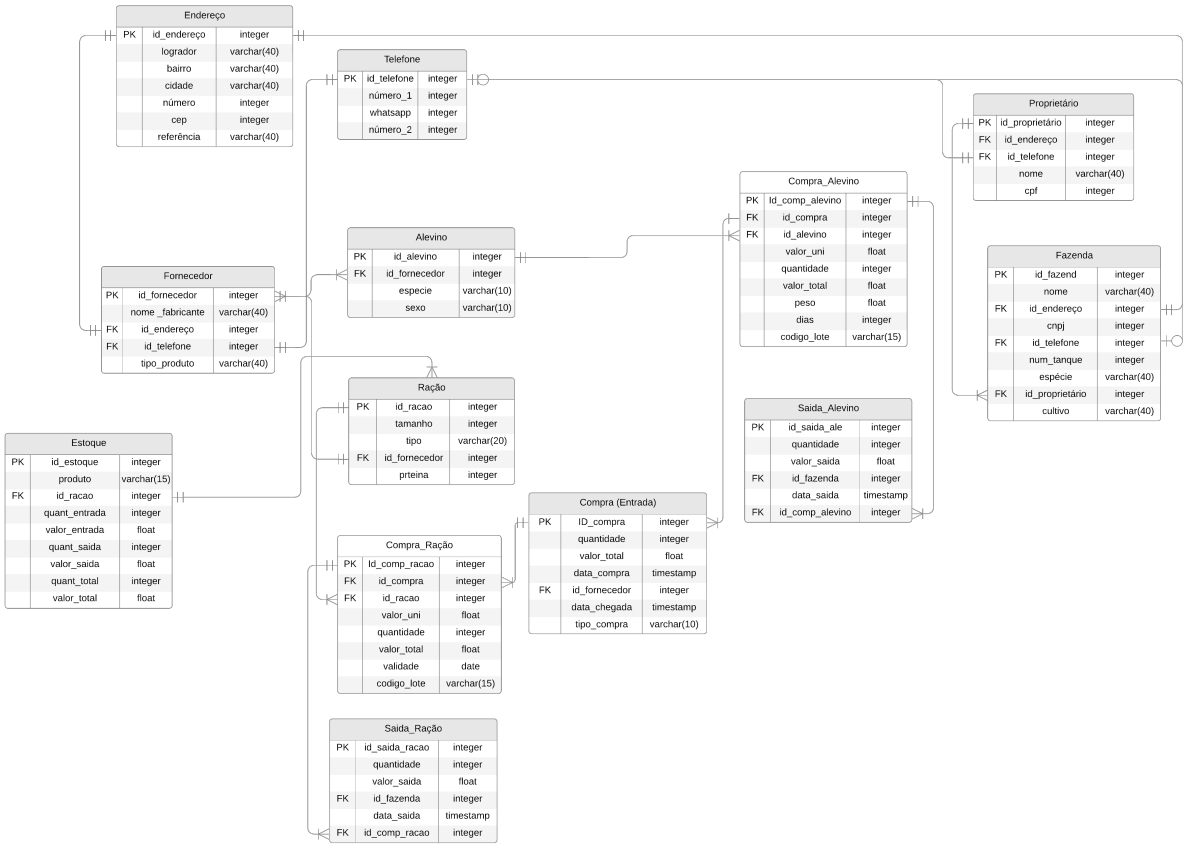
The attributes of the Feed Purchase entity are: feed purchase ID (PK), purchase ID (FK), feed ID (FK), supplier ID (FK), distributor ID (FK), unit value in reais per kg of feed (Numeric), purchased quantity of the specific feed (Real), input value of the specific feed (Numeric), feed storage expiration date (Varchar 10) and batch and manufacturer code for identification of the feed properties together to the manufacturer (Varchar 30). The attributes of the Stock entity are: stock ID (PK), product available in stock (Varchar 15), feed ID (FK), quantity of feed that entered (Integer), monetary value of the feed that entered (Float), quantity of feed left in stock (Integer), monetary value of feed left in stock (Float), total amount of feed in stock (Integer) and monetary value of feed in stock (Float).

The attributes of the Output entity are: output ID (PK), quantity of items that came out (Integer), total quantity that came out in kg of feed or millet of fingerlings (Numeric), total monetary value that came out (Numeric), date and time the fry or feed left stock (Date), product that left stock, fry or feed (Varchar 10) and farm ID (FK). The attributes of the Feed Output entity are: ID of feed output (PK), quantity of feed in kg or in the future tons that left stock (Real), total value that left stock for the farm (Numeric), ID of feed purchase (FK), farm ID (FK), feed ID (FK), output ID (FK), date and time of feed leaving stock for the farm (Timestamp) and the batch and manufacturer identification code (Varchar 30). The Fry Output entity also has attributes such as: fry exit ID (PK), quantity of fry per number of fingerlings that left the stock (Integer), total value of fingerlings per number of fingerlings that left the stock for the farm (Numeric), Farm ID (FK), date and time of fry departure from stock to the farm (Timestamp) and fry purchase ID (FK), batch and manufacturer identification code (Varchar 30) and exit ID (FK).

Entities related to users' personal data also have attributes. The Phone entity includes attributes such as Phone ID (PK), Phone Number (Varchar 15) and the WhatsApp (Integer). The attributes of the Owner entity are: owner ID (PK), owner's full name (Varchar 40), phone ID (FK) and owner's CPF (Varchar 11). The attributes of the Address entity are: address ID (PK), street (Varchar 40), neighborhood (Varchar 30), city (Varchar 30), state (Varchar 30), address number (Varchar 10), zip code of the street address (Varchar 15) and reference point to assist with delivery (Text). The attributes of the Farm entity are: farm ID (PK), farm name (Varchar 40), address ID (FK), farm CNPJ (Varchar 14), telephone ID (FK), number of tanks registered on the farm (Integer), species produced on the farm (Varchar 20), owner ID (FK) and farm production system (Varchar 20).

The entities, that is, the tables that make up the database, were structured based on the attributes mentioned previously. This structuring was represented visually through a diagram, as exemplified in Figure 3. In addition to the arrangement of entities and their attributes, the diagram also included the relationships established between these entities, indicating the connections and interactions between them and other tables in the system.

Figure 3: Representative diagram of the relational structure of the database, highlighting attributes and entities, elaborated during the physical modeling phase for the development of the collective feed purchase software for aquaculture.



Source: from the authors (2024).

Intermediary infrastructure

Golem's structure organizes the application into folders and metadata, similar to the development of R packages, which benefits programmers familiar with the language. Each directory has a specific purpose, aiming at efficiency in software development (Figure 4).

The application follows a structure organized into different directories. In the "dev" folder, there are the scripts recommended by the Golem workflow, which are essential for the development of the application within an R package "01_star. R" starts the project and establishes the basic settings. The "02_dev. R" contains the main code for the Shiny app, while the "03_deploy. R" is used for deployment configurations. The "run_dev. R" allows local execution of the application during development.

In the "inst" directory, there is the "golem-config.yml" file, which contains metadata such as the application's name, version, and working directory. Based on YAML, it is used for metadata serialization. In the "app/www" subfolder, external resources such as images, CSS, JavaScript, and text fonts are stored, as well as important files such as report templates and Markdown. The "www" directory is accessible via the browser while the application is running, not via R.

Figure 4: Structure of the organization of the application based on the Golem template. (A) Directory structures in the development of the feed purchase application applied to aquaculture. (B) Modules in Shiny that make up the application.

Source: from the authors (2024).

The "man" folder is reserved for package documentation, usually generated by the roxygen2 package. This documentation provides detailed information about all functions created in the application, including inputs, outputs, parameters, and purposes, guiding developers and future contributors in understanding the structure and functionality of the code for efficient production and future updates.

In the "R" directory are all the functions of the application, following the standard of the R language for package development. These files contain Shiny modules and other functions that make up the business logic. Some modules created make connections with the database described in Table 1, for these connections requests for table data are made, for example, for the registration of feed and/or fry.

The “SQL” directory contains intermediary commands between R and the backend, performing CRUD operations on a PostgreSQL database.

In the main directory are crucial files: “Rbuildignore” excludes items in version control, “DESCRIPTION” contains metadata, “NAMESPACE” stores data for selective loading, “LICENSE” has the licensing terms, and “README” offers installation and usage instructions.

The Golem framework was chosen for its standardization in the organization of scripts and files, integration with packages to speed up development, facilitation of code documentation, and sharing between projects and collaborators. In addition, it automatically generates the files needed for building the user interface and server, simplifying Shiny app development.

Table 1: Modules connecting to the database.

Module Name	Module Description	Database Connection Type
Mod_TabAle_est	Updates the status of fry purchases in the database and refreshes the table in the frontend.	dbExecute, dbGetQuery, dbDisconnect
mod_tabAlevino	Performs operations for deleting and inserting fry data, and updates the frontend.	dbExecute, dbSendQuery, dbDisconnect, dbClearResult
mod_tabCompAle	Inserts supplier data into the database and removes temporary tables after insertion.	dbSendQuery, dbClearResult, dbGetQuery, dbRemoveTable, dbDisconnect
mod_tabCompRac	Inserts and manages data related to feed purchases in the database, updates stock, and performs queries.	dbWriteTable, dbSendQuery, dbClearResult, dbGetQuery, dbRemoveTable, dbDisconnect
mod_tabProprietario	Manages owner data in the database: deleting, inserting, and updating records.	dbExecute, dbSendQuery, dbClearResult, dbGetQuery, dbDisconnect
mod_tabFazenda	Manages farm data in the database: inserting, updating, and reading records.	dbSendQuery, dbClearResult, dbGetQuery, dbDisconnect
mod_tabRac_est	Updates arrival and purchase status data in the database and manages the feed stock visualization.	dbExecute, dbGetQuery, dbDisconnect, glue::glue, read_sql_file, data.frame
mod_tabRacao	Manages data related to feed: deleting, inserting, editing, and updating records in the database.	dbExecute, dbSendQuery, dbClearResult, dbGetQuery, dbDisconnect, glue::glue
mod_tabSaidaAle	Inserts, deletes, and queries data from the fry output table.	Connection to PostgreSQL via DBI::dbWriteTable, DBI::dbSendQuery, DBI::dbGetQuery, DBI::dbExecute, and DBI::dbDisconnect
mod_tabSaidaRac	Inserts, queries, and deletes data from the feed output table.	Connection to PostgreSQL via DBI::dbWriteTable, DBI::dbSendQuery, DBI::dbGetQuery, DBI::dbExecute, DBI::dbRemoveTable, DBI::dbClearResult, and DBI::dbDisconnect

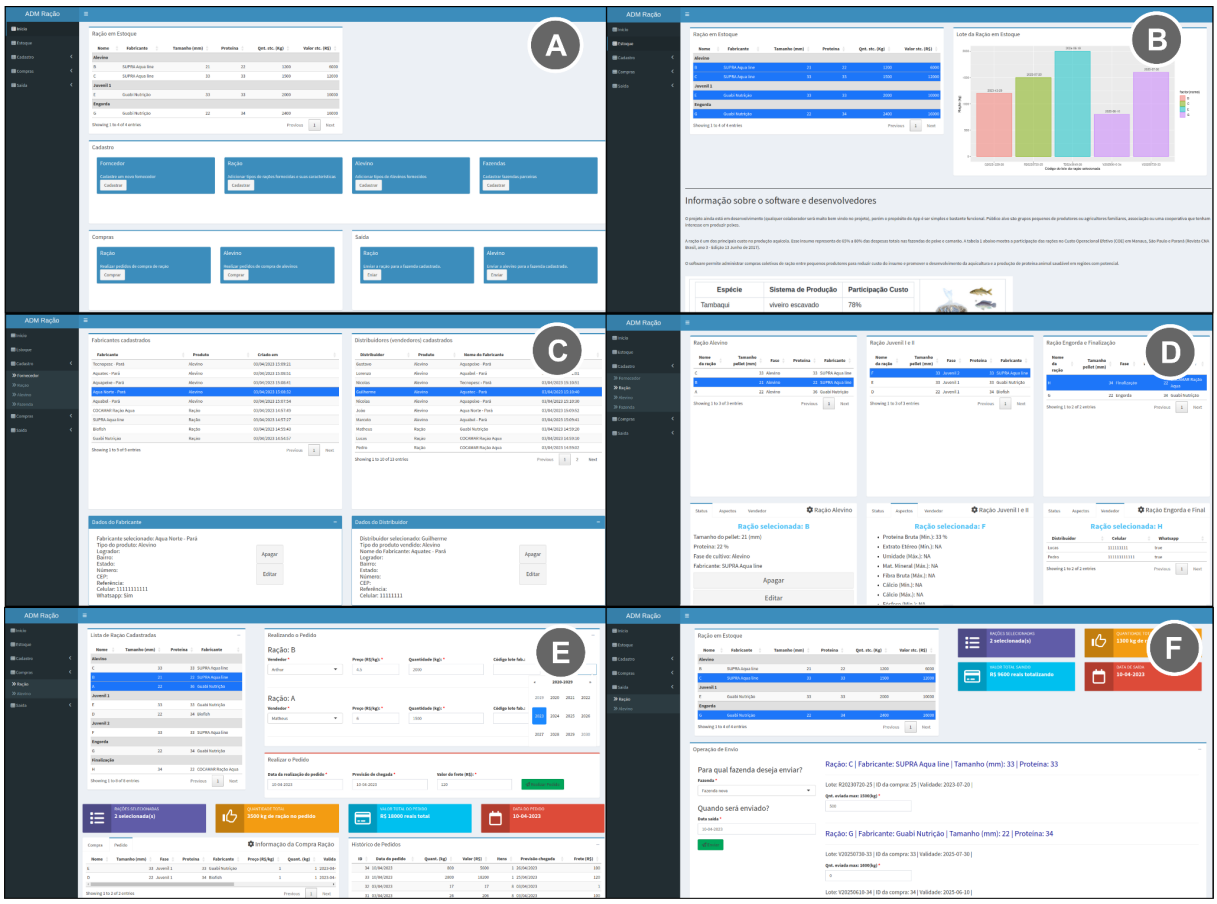
Source: from the authors (2024).

Frontend

The frontend was built with the Shiny framework, with modularization to divide the system into smaller, autonomous units. The interface is built with shinydashboard. The sidebar contains five menus: Home, Inventory, Registration, Purchasing, and Output, as shown in Figure 5. Each menu is fragmented into modules, such as “mod_tabInicio”, “mod_tabEstoque” and so on.

The Home menu summarizes the general information, while the Inventory menu shows the quantity, granulometry, shelf life, and batch of the feed. In the Registration menu, the submenus "Sup-

Figure 5: Presentation of the Application Interface. (A) Home Menu, with overview. (B) Inventory menu. (C) and (D) Registration Menu with the submenus Supplier and Feed, respectively. (E) Purchase menu, Feed submenu. (F) Output menu, Feed submenu.



Source: from the authors (2024).

plier", "Feed", "Fingerling" and "Farm" have specific modules (mod_tabFornecedor, mod_tabRacao, mod_tabAlevino and mod_tabFazenda) for registration and data editing. In the Purchasing menu, the "Feed" and "Fingerlings" (mod_tabCompRac and mod_tabCompAle) submenus allow collective purchases to reduce costs. In the Output menu, the "Feed" and "Fingerlings" submenus (mod_tabSaidaRac and mod_tabSaidaAle) manage the distribution of orders between farms.

The use of Shiny Modules, such as “mod_tabAlevino” and “mod_tabRacao”, fragments the code into independent components to facilitate collaborative development. This simplifies the organization of files, making the code more readable and efficient.

The UI (User interface) has an intuitive aesthetic, with side menus. Shiny server programming follows Shiny’s reactivity paradigm, which establishes a dependency graph to automatically manage updates between inputs (frontend) and outputs (server). This approach ensures a consistent and responsive user experience, as outputs are automatically updated to reflect changes in inputs (WICKHAM, 2021).

Conclusions

The integration between the backend, which establishes the connection with the database through the “db_connect” function, and the Shiny modules, which perform queries to the database through queries to request specific data, together with the Golem framework, is essential for the effectiveness and success of the application developed for the management of collective feed purchases in aquaculture. The descriptions and database connections of the modules are detailed in Table 1.

The PostgreSQL DBMS plays a central role in ensuring the integrity, security, and efficiency in handling application data. By adopting PostgreSQL, a robust and open-source solution, the project gains in reliability and scalability, which are key to handling the growing demands of the application.

Shiny modules offer a modular approach to frontend development, which makes it easy to organize, maintain, and scale the application. By breaking down the app into smaller, independent components, Shiny modules make the code more readable, making it easier for team members to develop and collaborate.

Finally, the Golem framework provides a standardized and cohesive organization of the application's source code. By following the standards set by Golem, the project benefits from a clear and easy-to-understand architecture, which is essential for the maintenance and ongoing development of the application over time.

Together, the integration between the backend, Shiny modules and the Golem framework allow the Shiny application to meet the complex demands of managing collective feed purchases in aquaculture, offering a robust, scalable and easy-to-maintain solution for fish farming producers.

References

BATISTA, B. D. O.; OLIVEIRA, A. B. J. *R básico*. Ouro Branco, MG: [s.n.], 2022. v. 1. (Estudando o Ambiente R, v. 1). ISBN 978-65-00-51600-5.

CHANG, W. et al. *Shiny: Web Application Framework for R*. [S.l.], 2022. R package version 1.7.4.

CODD, E. F. Further normalization of the data base relational model. *Data base systems*, v. 6, p. 33–64, 1970.

COSTA, H. A. X.; RESENDE, A. M. d.; SILVEIRA, F. F. Relato de experiência de ensino de modelagem e implementação de software em um curso de graduação em ciências da computação. *Fórum de Educação em Engenharia de Software*, p. 46, 2008.

FAY, C. et al. *Goelm: A Framework for Robust Shiny Applications*. [S.l.], 2023. R package version 0.4.0.

RDEVELOPMENT, C. R: A language and environment for statistical computing. *R Foundation for Statistical Computing Team*, 2011.

WICKHAM, H. *Mastering shiny*. [S.l.]: O'Reilly Media, Inc., 2021.