~ ФІНАНСИ, БАНКІВСЬКА СПРАВА ТА СТРАХУВАННЯ ~

УДК 336 **DOI:10.32680/2409-9260-2023-9-310-59-71**

RUBY ON RAILS - A POPULAR CHOICE FOR FINANCIAL TECHNOLOGIES

Suprunenko Valeriy, Ruby on Rails Developer, MSc Banking and Finance Program applicant ZHAW School of Management and Law, Zurich, Switzerland
e-mail: swift.x@yahoo.com

Abstract. *This article examines the advantages of using the Ruby on Rails framework in the development of financial technologies, such as web applications for banks and financial management applications. The main advantages of using Ruby on Rails in the financial industry are discussed, including its benefits for full-stack development, the use of RubyMine IDE, application testing with Rspec, the advantages of using design patterns with ActiveRecord, transaction mechanisms, threads, fibers, clousers, Single Table Inheritance, UUID primary keys, encryption for storing sensitive account data, error handling and exceptions, as well as additional Ruby on Rails libraries ActiveMerchant, Finance, Money. The conclusion of the article summarizes the applicability of Ruby on Rails in the development of financial technologies.*
Key words: *Ruby on Rails; financial technologies; web development; banking applications; fintech applications; security; flexibility; modularity; performance; scalability; gem; STI; UUID; transactions; ActiveRecord.*

РУБІН НА РЕЙКАХ - ПОПУЛЯРНИЙ ВИБІР ДЛЯ ФІНАНСОВИХ ТЕХНОЛОГІЙ

Супруненко Валерій, розробник Ruby on Rails, аплікант програми MSc Banking and Finance Program ZHAW School of Management and Law, Цюріх, Швейцарія
e-mail: swift.x@yahoo.com

Анотація. *У цій статті розглядаються переваги використання фреймворку Ruby on Rails у розробці фінансових технологій, таких як веб-додатки для банків і додатки для управління фінансами. Обговорюються основні переваги використання Ruby on Rails у фінансовій індустрії, включаючи його переваги для розробки повного стека, використання RubyMine IDE, тестування додатків за допомогою Rspec, переваги використання шаблонів проектування з ActiveRecord, механізми транзакцій, потоки, волокна , clousers, Single Table Inheritance, первинні ключі UUID, шифрування для зберігання конфіденційних даних облікового запису, обробки помилок і винятків, а також додаткові бібліотеки Ruby on Rails ActiveMerchant, Finance, Money. Підсумок статті підсумовує застосовність Ruby on Rails у розробці фінансових технологій.*
Ключові слова: *Ruby on Rails; фінансові технології; веб-розробка; банківські програми; фінтех додатки; безпека; гнучкість; модульність; продуктивність; масштабованість; дорогоцінний камінь; ІПСШ; UUID; транзакції; ActiveRecord.*

JEL Classification: L860.

Formulation of the problem. Ruby on Rails is a popular web development framework that is used in many financial technologies, such as Chime, Emarchantpay, Coinbase, Kickstarter, Bloomberg, Shopify, and more. Its flexibility, modularity, and high development speed make it an ideal choice for quickly creating iterative prototypes and scaling projects.

The advantages of Ruby on Rails include fast development, ease of creation, high performance, and convenient database handling. Additionally, the framework has a high level of security and reliability, which is crucial in the financial industry.

But what enables fast application development with Ruby on Rails? Unfortunately, there is a lot of generalized information about the advantages of Ruby on Rails available on the Internet, but specific arguments are often found only in technical publications that may not be accessible to a wide range of readers.

You can find a complete list of the recomended Ruby on Rails books in Table 1. The most popular books about Ruby on Rails are below:

1. "Agile Web Development with Rails" (Dave Thomas, David Heinemeier Hansson, Sam Ruby) - This book is one of the classic sources for beginners in Ruby on Rails. It provides an introduction to the framework and teaches web application development using Rails.

2. "The Rails 5 Way" (Obie Fernandez) - This book is a detailed guide for understanding and

using Ruby on Rails. It covers a wide range of topics, including application architecture, testing, security, and optimization.

3. "Ruby on Rails Tutorial: Learn Web Development with Rails" (Michael Hartl) - This book is a popular resource for self-learning Ruby on Rails. It offers practical steps for creating a fully functional web application using Rails. Based on this, the author of the article has formulated the advantages of Ruby on Rails development based on his own experience working on projects such as Chime.com, myHomeIQ.com. and hope that this material will be helpful in deciding to use Ruby on Rails for application development in the banking and financial sector.

Table 1

The Popular Books about Ruby on Rails

№	Name	Author	Description
1	Soon to be Ruby on Rails --API programmer	SaKKo Sama	Building API could be fund with Ruby on Rails. Let Rails help you with easy and fast coding while you only need to worry about requirements and features. If you want to start building your new web application and you picked to completely separate Client side and Server side codes completely. Then it's a high chance that you need to build API regardless of what you picked for your frontend. I've used Ruby on Rails in my software house since 2011 and personally think that it is more than enough for most of my customers. Now that Ruby on Rails has this "--api" option where you only use Rails to build API. No HTML, CSS, JS rendered from backend (except for emails). I only use this "--api" option for newer projects and moved my frontend to use NuxtJS, NextJS or AngularJS. My life is now much simpler because I can now separate task between frontend and backend. My team work more effectively, and even better unit test. This book is to show you how to write API on Rails. It's purely based on code and building prototype application. This book will not cover Unit Test, I think it's better to start with the fun part and continue with the boring part in the next book.
2	Beginning Rails 6	Brady Somerville, Adam Gamble, Cloves Carneiro Jr, Rida Al Barazi	Springboard your journey into web application development and discover how much fun building web applications with Ruby on Rails can be. This book has been revised to cover what's new in Rails 6 including features such as WebPack, advanced JavaScript integration, Action Mailbox, Action Text, system and parallel testing, Action Cable testing, and more. Beginning Rails 6 gently guides you through designing your application, writing tests for the application, and then writing the code to make your application work as expected. It is a book that will guide you from never having programmed with Ruby, to having a Rails 6 application built and deployed to the web. After reading and using this book, you'll have the know-how and the freely available source code to get started with your own Rails-based web development in days. What You Will Learn □ Create Ruby on Rails 6 web applications from the bottom up □ Gain the basics of the Ruby programming language □ Combine all the components of Rails to develop your own web applications □ Apply TDD to make sure your application works exactly as you expect □ Use Git source control and best practice techniques to create applications like a pro
3	Advanced Database Programming with Rails and Postgres		Learn about subqueries, materialized views, and custom data types in Postgres and Rails. We walk through realistic real-life examples, translating first into SQL, and then into Rails code. Every example comes with source code to follow along.
4	Efficient Rails	Andrew Allen	Efficient Rails gives you actionable upgrades to your workflow you can put into practice right now. Learn all the best shortcuts and tools for crafting elegant Rails apps. Whether you're just learning Rails or have been using it for years, Efficient Rails will save you time and make you a happier developer. 100+ Workflow Upgrades. Each can be learned in as little as 5 minutes, but will save you hours over time. A Workflow Upgrade is a way to do something you're already doing, but better. Usually, faster and with fewer steps. These upgrades will make you look feel like a superhero when that person looking over your shoulder asks "how did you do that?!"

5	API on Rails 6	Alexandre Rousseau	Learn best practice to build an API using Ruby on Rails 6. The intention with this book it's not only to teach you how to build an API with Rails. The purpose is also to teach you how to build scalable and maintainable API with Rails which means improve your current Rails knowledge. In this book you will learn to: Use Git for version control Building JSON responses Test your end-points with unit and functional tests Set up authentication with JSON Web Tokens (JWT) Use JSON:API specification Optimize and cache the API
6	Rails on Docker	Chris Blunt	Rails on Docker is the complete course showing you how to use Docker in the development, testing and deployment of your Ruby on Rails apps. Through the Rails on Docker course, you'll learn: What Docker is, why you should use it, and how to get started with Docker and Docker Compose. How to use Docker to simplify your development workflow (and make development across teams easy) by building a complete Rails application from scratch. How you can use Docker to split your app's various services into multiple containers and services. How Docker simplifies Continuous Integration and Deployment (CI/CD), and helps ensure you have development, test and production environment parity. How you can use Github Actions to build a CI/CD pipeline. How to deploy your Docker-based app into a production environment (Digital Ocean App Platform), with scheduled (cron) jobs, asset hosting, and more. How you can design your apps around "disposable infrastructure" to let you quickly run your apps anywhere, and scale to any size. After working through this course, you'll be ready to migrate your own Rails apps to Docker, consolidate your development environment and workflows, and make deployment easy and predictable.
7	Componentbased Rails Applications	Stephan Hagemann	Component-Based Rails is a proven method to manage the complexity of large applications. It ensures that we can maintain development speed, quality, and fun throughout the life cycle of applications. Components do this by enabling conversations about high-level structures that lead to improved communication of intent, more collaboration opportunities, and better maintenance capabilities. As Rails applications grow, even experienced developers find it difficult to navigate code bases, implement new features, and keep tests fast. Components are the solution, and Component-Based Rails Applications shows how to make the most of them. Writing for programmers and software team leads who are comfortable with Ruby and Rails, Stephan Hagemann introduces a practical, start-to-finish methodology for modernizing and restructuring existing Rails applications. One step at a time, Hagemann demonstrates how to revamp Rails applications to exhibit visible, provably independent, and explicitly connected parts– thereby simplifying them and making them far easier for teams to manage, change, and test. Throughout, he introduces design concepts and techniques you can use to improve applications of many kinds, even if they weren't built with Rails or Ruby. Learn how components clarify intent, improve collaboration, and simplify innovation and maintenance Create a full Rails application within a component, from first steps to migrations and dependency management Test component-based applications, manage assets and dependencies, and deploy your application to production Identify the seams in an existing Rails application, and refactor it to extract components Master a scripted, repeatable approach for refactoring Rails applications of any size Use component-based Rails with two popular structural patterns: hexagonal and DCI architecture Leverage your new component skills with other frameworks and languages Overcome the unique challenges that arise as you componentize Rails applications If you're ready to simplify and revitalize your complex Rails systems, you're ready for Component-Based Rails Applications.

8	Gradual Modularization for Ruby and Rails	Stephan Hagemann	For the longest time, Ruby and Rails developers had gems and engines as their main tools for creating structure to manage large-scale structures within their applications. This book is about a new tool in their toolbelt: packages. Based on the work on packwerk by Shopify packages allow a much more fluid move to modularization then components ever did. The effects are astounding: discussions about where to draw boundaries can be far less technical and focus more on the business because the underlying technology gets out of the way. The concept underlying this is gradual modularization, which the author expects we will see spread into other languages and frameworks over the coming years. Why? Because gradual modularization allows for a not-before seen level of approachability and flexibility to modularization work. Work that required difficult decisions that were hard to reverse changes. Those decisions are now the extreme points on a spectrum of options where the right thing for the team can be somewhere in between.
9	Rebuilding Rails	Noah Gibbs	With Rebuilding Rails you'll build a complete Ruby MVC framework from an empty directory. Your framework will be structured like Rails, using the same architecture and the same metaprogramming tricks. You'll learn the magic behind Rails. You'll finish each system and solidify your knowledge with structured exercises. And you'll get the gut-level understanding that only a framework builder has. To be an expert, learn the fundamentals. Then you can pick up not just Rails, but any framework you want. Rebuilding Rails will show you the tricks and where to start reading the source code. When you've built these systems, you get a sort of X-ray vision into the framework. You've had that before with code that you built for yourself. Wouldn't you like the same thing for Rails, Sinatra or your framework of choice?
10	The Ruby on Rails Performance Apocrypha	Nate Berkopec	The world's foremost Ruby on Rails performance expert and maintainer of the popular Puma webserver, Nate Berkopec, has compiled 4 years of short-form writing into a single volume: The Ruby on Rails Performance Apocrypha. Appropriate for both beginners and experts, this short book is an introduction to performance science and engineering, frontend performance, Ruby performance, and scaling. The Apocrypha is a fun ramble with a lot of tidbits and useful information scattered about. This book covers useful and important topics in Ruby and Rails performance, such as: □ Benchmarks for Rails Apps □ Reading Flamegraphs □ Microservices and Trends □ Why is Ruby Slow? □ Popularity □ Page Weights and Frontend Load Times □ What is the GVL? □ Reproducing Issues Locally □ Worker Killers □ Multi-threading □ Read Replicas □ .and much more.
11	Practicing Rails	Justin Weiss	In this ebook, you'll learn: □ How to start your own Rails apps today, and learn as you build them. □ The best ways to understand the most about Rails in the tiny amount of free time you have. □ When to pay attention to new gems, libraries, and programming techniques, and when you can ignore them. □ What error messages mean, where they come from, and how to fix them on your own. □ Simple processes you can follow to build even the largest featuresand apps. And most importantly, Practicing Rails will keep you motivated and on the right track to finally mastering Rails.

12	Growing Rails Applications in Practice	Henning Koch Thomas Eisenbarth	We'd like to show you one path to write Rails apps that are a joy to understand and change, even as your team and codebase grows. This book describes a complete toolbox that has served us well for all requirements that we have encountered. But before we do that, we need to let you in on an inconvenient secret: Large applications are large. The optimal implementation of a large application will always be more complex than the optimal representation of a smaller app. We cannot make this go away. What we can do is to organize a codebase in a way that "scales logarithmically". Twice as many models should not mean twice as many problems. To accomplish this, you don't necessarily need to change the entire way your application is built. You don't necessarily need to introduce revolutionary architectures to your code. You can probably make it with the tools built into Rails, if you use them in a smarter way. Compare this to sorting algorithms. When a sorting function is too slow, our first thought is not "install a Hadoop cluster". Instead we simply look for an algorithm that scales better. In a similar fashion this book is not about revolutionary design patterns or magic gems that make all your problems go away. Instead we will show how to use discipline, consistency and organization to make your application grow more gently.
13	Vue on Rails	Richard LaFranchi Bryan Lim	This book explores how to manage and understand priorities when working with Vue on Rails and how to determine the best configuration for your project. You'll see how to reuse your Vue components in a Rails project with less coding and harness the component options, as well as how to use props, events and slots of Vue components. You'll also use webpacker to set up your project and pass data from your Vue component to a Rails controller and back. Finally, you'll learn which is the best configuration for the router and how to test and deploy your application. What You'll Learn □ Use Vue and Rails together to improve products □ Build a web app with a focus on developer happiness □ Take a pragmatic approach to web app development with practical code examples □ Save time configuring Vue and Rails □ Control and manage states in a Vue on Rails project
14	Rails: Novice to Ninja	Patrick Lenz	Unlike other Rails books, this book doesn't assume that you are an experienced web developer, or that you've used Ruby before. An entire chapter is devoted to learning Ruby in a fun way, using the interactive Ruby console, so you can follow along at home. You'll be an accomplished Ruby programmer in no time! You'll then start using Rails to build a practical, working project: a Reddit-like social news application. As you'll build the app, you'll gain valuable experience of using Rails features such as user authentication, session cookies, and automated testing. The book finishes with chapters on debugging, benchmarking and deployment to a live web server.
15	Learning Rails 5	Mark Locklear	If you're a web developer or designer ready to learn Ruby on Rails, this hands-on guide is the ideal way to get started. Rather than toss you into the middle of the framework's Model-View-Controller architecture, as many books do, Learning Rails 5 begins with the foundations of the Web you already know. You'll learn how to create something visible with Rails' view layer before diving into the more difficult inner layers: the database models and controller code. All you need to begin your Rails journey is HTML experience. Each chapter includes exercises and review questions to test your understanding as you go. □ Present content by building an application with a basic view and a simple controller □ Build forms and process their results, progressing from simple to more complex □ Use Rails scaffolding and REST to build effective applications quickly □ Connect forms to models and create code that maps directly to database structures □ Build applications that combine data from multiple tables □ Use migrations to track changes to your database over time □ Add common elements such as sessions, cookies, and authentication

16	Hotwired ATS: Modern, fullstack Rails 7 development	David Colby	Hotwired ATS is a step-by-step tutorial for Ruby on Rails developers interested in learning more about the modern Rails 7 stack: Turbo, Stimulus, CableReady, and StimulusReflex. In this book we will build a simple applicant tracking system (ATS) with job postings, job applications, two-way email communication, in-app notifications, and multi-user accounts. From rails new to the final code change, we will walk through each step together, creating an application using the latest in Ruby on Rails techniques.

Analysis of recent research and publications. When developing with Ruby on Rails, developers write code that handles the functionality of both the server-side and client-side of the application. The communication between the server and the client is automatically established, taking into account all security requirements. To expedite development, the standard Bootstrap style library can be utilized, providing pre-designed interface elements such as buttons, tables, input forms, and other necessary page layout components. This saves time on designing the page and creating layouts. The client doesn't need to maintain separate development teams for the server-side and client-side. One development team handles all the work.

Additionally, if needed, a mobile application can be added to the server-side of the application. Instead of using the standard Bootstrap styles, alternative JavaScript frameworks like Vue.js or React can be employed. Recommended Level of Technical Expertise for a Ruby on Rails Developer is listed in Table 2.

Table 2

Recommended Level of Technical Expertise for a Ruby on Rails Developer

Technology	Ecosystem
Ruby Frameworks	Rails, Sinatra, Hamani, Crepe, Roda Ruby, Cuba
Database	MySQL, MongoDB, PostgreSQL, SQLite, MariaDB
Gems	RSpec, Devise, PRY, RuboCop, Pagy, Rolify, cancancan, Braintree, Capistrano
CMS	Refinery CMS, Locomotive CMS, Camaleon CMS, Radiant CMS, Comfortable Mexican Sofa, Alchemy CMS, Spina CMS, Scrivito CMS
IDE	Ruby Mine, Atom Editor, VIM, Sublime Text, Aptana Studio, Emacs, Cloud9, Visual Studio Code
Integration	PHP, All Javascript Tech, .Net, Python
Testing	Unit, Cucumber, Timecop, Fabrication
TDD Tools	Rspec-Rails, Capybara, Minitest, Factory_girl_rails, Spork, Database_Cleaner, Simplecov, Faker, Launchy
Rest API	SwaggerDoc, APIPie, Grape, Active Model, Serializers
Deployment Tools	Capistrano, Mina, Heroku
Server	AWS, Digital Ocean, Heroku, Liquid Web, DreamHost, Bluehost, HostGator, RackSpace
App Server	Puma, Passenger, Unicorn, WEBrick
Web Server	Nginx, Apache
Scheduled/ Recurrence Jobs	Whenever, Delayed Job, SideKiq, Rufus-Scheduler, Resque, Sucker Punch, Sidekiq Cron
Searching	Elasticsearch-Rails, Ransack
Coding Style	RuboCop, Rails Best Practice

Catching	Redis
Version Control	GIT (GitFLow)
Process	Agile Scrum and Lean
PM Tools	Jira, Trello, Slack, Linear
Deployment Process	CI/CD

Separation of previously unresolved parts of the overall problem. The RubyMine IDE helps developers maximize productivity in every aspect of project development, from writing and debugging code to testing and deploying the final program in production. RubyMine offers an interactive code editor with features such as code completion, refactoring options, code style corrections, and formatting. It provides a wide range of code templates and performs code inspections and fixes. RubyMine includes a fast search and interactive code navigation feature, allowing developers to navigate between files, classes, superclasses, and definitions in external gems. RubyMine recognizes Ruby on Rails concepts, provides autocompletion for database fields, associations, and methods defined by named routes and resourceful routes. It also offers syntax highlighting and a convenient interface for application internationalization, as well as visual model dependency diagrams. RubyMine has tight integration with popular Ruby tools such as RuboCop,

Bundler, Rake, and many others. Interaction with the application can occur through the built-in interactive IRB and Rails console without the need to exit the application. This is just a fraction of the complete list of all the functional capabilities of this IDE, but it is evident that using RubyMine significantly enhances a developer's productivity compared to other development tools..

Purpose of the study. The RSpec tool, which is written in Ruby just like Ruby on Rails, provides extensive capabilities for writing unit and integration tests that can test both individual components of an application and their interdependencies. Having a high level of test coverage in an application helps avoid the time-consuming efforts of developers and testers during further development. Often, when developing new features for an application, necessary changes are made to the existing code, which may affect the application's logic. To automatically verify the functionality of all application features, tests need to be executed since the manual testing effort increases as the complexity of the application grows. RSpec allows testing various use cases of the application and verifying the correctness of data processing. In case of any issues in the application's behavior, RSpec tests will report errors, enabling the developer to quickly identify the cause. This significantly saves the client's resources in terms of development costs.

Basic material. ActiveRecord, which allows the integration of object classes with their corresponding database tables, significantly speeds up application development. Time savings are achieved because the associations between object attributes and table columns, as well as the rendering of data in web pages, are automatically handled by ActiveRecord. Otherwise, the developer would have to perform these tasks manually.

For example, if an object has 10 attributes, the create, read, update, and delete forms for that object would have a total of 40 fields (4 forms * 10 fields), which would need to be created on the client-side of the application. To store all the object's parameters, 10 columns would need to be created in the table, and the application controller would need to handle 10 incoming parameters. ActiveRecord takes care of all this work, and the developer can generate the necessary model, controller, view files, test templates, database migration files, and other required files with a single command using Ruby on Rails' generator, specifying the desired parameters. This provides a significant time savings by reducing the need for developers to perform repetitive operations. It's worth noting that all the necessary view files for the model are automatically created for the create, read, update, and delete operations.

Additionally, when needed, ActiveRecord performs validation of the entered data before saving it to the database.

To execute specific actions before or after data saving, ActiveRecord provides callback mechanisms. Callbacks allow developers to define additional functions that will be automatically called when a specific event occurs.

To manage the structure of an existing database, Rails provides a migration mechanism. Each

migration is stored in a separate file and represents the structure changes to be applied. This allows developers to quickly make or revert necessary changes to the data structure without directly accessing the database.

This helps avoid errors when working with data structures and saves time on debugging and fixing issues.

In Ruby on Rails, transactions provide a way to ensure that a set of database operations will be executed only if all of them succeed. Otherwise, they will be rolled back to the previous state.

Transactions are particularly useful when creating scenarios involving accounting entries or payment records. If an error occurs within a transaction block, all operations within that block are rolled back.

By using transactions, you can group multiple database operations together and treat them as a single unit of work. This ensures data consistency and integrity, as well as the atomicity of the operations. If any part of the transaction fails, the entire transaction is rolled back, and the database is left unchanged.

Here's an example of how transactions are used in Ruby on Rails:

```
def transfer_funds(sender, recipient, amount)
ActiveRecord::Base.transaction do
sender.balance -= amount
sender.save!
recipient.balance += amount
recipient.save!
end
rescue ActiveRecord::RecordInvalid => e
# Handle the exception or log the error
end
```

In this example, the transfer_funds method performs a funds transfer between two accounts. The transaction ensures that both account balances are updated atomically. If an error occurs during the saving process, the transaction is rolled back, and any changes made to the database are reverted.

Transactions help maintain data integrity and provide a safety net when working with complex database operations. They ensure that the database remains in a consistent state even if errors occur during the execution of a set of operations.

The use of threads in Ruby on Rails for developing financial and banking sector applications can have several advantages:

Parallel execution: Threads allow for executing different parts of the program in parallel, which improves the utilization of processor resources and can speed up computational tasks. In the financial and banking sector, where complex and time-consuming computations are common, the use of threads can enhance program performance.

Asynchronous programming: Threads can be used to implement asynchronous task processing. This is beneficial for financial applications where there may be multiple concurrent requests or a need to handle different financial events in parallel.

Distr buted operations: Threads can be utilized for distributed processing of financial data or connecting to various services asynchronously. This can help improve scalability and performance of financial applications, especially when there's a need to interact with multiple data sources or perform calculations on large datasets.

Multithreaded programming: With threads, it is possible to create multithreaded programs where each thread is responsible for executing a separate task. This can be useful for concurrent processing of different financial operations such as trading on an exchange, payment processing, market analysis, and more.

It's important to consider that working with threads can be complex and potentially lead to issues related to concurrent access to shared resources (e.g., shared memory). Therefore, when using threads, careful planning, synchronization, and resource management are necessary to avoid potential problems.

In addition to threads, it's also worth considering alternative approaches such as asynchronous

programming using event-driven or fibers, which can be efficient for handling asynchronous financial operations and building scalable applications.

Fibers are a powerful tool in Ruby on Rails for developing financial and banking sector applications, offering the following benefits:

State management: Fibers allow for maintaining their own state and can be suspended and resumed at any point. This enables convenient management of the state of financial operations, including the ability to pause and resume them as needed.

Asynchronicity: Fibers enable performing operations asynchronously without blocking the main execution thread. This is particularly useful for financial applications where there may be multiple financial operations that require non-blocking execution.

Event handling: Fibers can be used for handling various financial events or data streams. They provide a convenient way to manage events and respond to them according to the program's needs.

Resource efficiency: Fibers consume less memory and resources compared to threads since they don't require separate call stacks. This can be beneficial in financial programs where efficient resource utilization is an important factor.

Ease of use: Fibers have a simple and understandable syntax, making them easy to use in program development. They can be easily created, suspended, and resumed, simplifying work with financial operations.

Overall, fibers in Ruby are a powerful tool for developing financial applications, allowing for state management, asynchronous processing, and efficient resource utilization. They complement threads and other mechanisms of parallel execution, opening up new possibilities in program development for the financial and banking sector.

Closures have several advantages and benefits for financial and banking sector programs:

Anonymous functions: Closures allow for creating anonymous functions that can be passed as arguments to other functions or stored in variables. This enables the creation of dynamic event handlers or generic functions that can be used in various financial operation contexts.

Functional composition: Closures enable function composition by chaining functions together.

This is particularly useful for processing financial data, where a series of operations need to be applied sequentially to a dataset.

State preservation: Closures can capture and access variables from their surrounding context, allowing functions to remember previous states and perform calculations based on historical data.

Advantages of functional programming: Closures promote the use of functional programming approaches such as data immutability, absence of side effects, and pure functions. This can make programs more predictable, easier to test, and maintain. Increased flexibility and extensibility: Closures enable the dynamic creation of functions on the fly and the ability to change their behavior according to the current context or conditions. This can be useful for responding to market condition changes or accommodating different variations of financial operations.

Overall, closures offer broad possibilities for developing flexible, expressive, and powerful programs in the financial and banking sector. They help make code more understandable, modular, and easy to maintain in changing financial environments.

STI (Single Table Inheritance) is a mechanism provided by Ruby on Rails that allows storing objects with different classes in a single database table. It enables the inheritance of attributes and behavior from a parent class to multiple child classes while storing them in the same table. Each record in the table has a type column that indicates the class of the object.

STI is useful when you have a set of related models that share common attributes but also have some unique attributes or behavior. Instead of creating separate tables for each subclass, you can store them all in a single table, which saves resources and simplifies database structure.

Here's an example to illustrate how STI works in Ruby on Rails:

```
class Transaction < ActiveRecord::Base
# Common attributes and behavior for all transaction types
end
class ApprovedTransaction < Transaction
# Additional attributes and behavior specific to approved transactions
end
```

```
class CapturedTransaction < Transaction
# Additional attributes and behavior specific to captured transactions
end
class VoidedTransaction < Transaction
# Additional attributes and behavior specific to voided transactions
end
class RefundedTransaction < Transaction
# Additional attributes and behavior specific to refunded transactions
end
class ErrorTransaction < Transaction
# Additional attributes and behavior specific to error transactions
end
```

In this example, the Transaction class is the parent class, and the ApprovedTransaction, CapturedTransaction, VoidedTransaction, RefundedTransaction, and ErrorTransaction classes are the child classes. All the transactions are stored in the same transactions table, and the type column differentiates between the transaction types.

Using STI simplifies the database schema, as you only need one table instead of multiple tables for each transaction type. It also allows you to leverage the common functionality defined in the parent class while extending it with specific behavior in each subclass.

UUID, also known as GUID, is an alternative type of primary key for SQL databases. It offers some non-obvious advantages compared to standard integer-based keys. The beta version of Rails 6 introduces a new feature in ActiveRecord that makes working with UUID primary keys easier. UUID is a random string in a predefined format.

The value of a primary key is usually accessible in URLs and API network logs, which can potentially allow an attacker to estimate approximate information, such as the number of registered users or the number of orders placed, as well as assess the system's growth rate. A public address might look like "/orders/160/checkout" Transitioning to UUIDs results in the use of internet addresses that do not reveal any personalized information, such as "/orders/aa22-ee33-44ff-55tt/checkout."

Additionally, the generation of primary keys can be moved to the client-side programs, which can save time and resources by not relying on server calls.

Ruby on Rails stores secret keys and passwords for API access in a special file called config/credentials.yml.enc in an encrypted format. Since the file is encrypted, it can be stored in a version control system as long as the master key is kept secure.

Ruby on Rails provides a mechanism for handling errors and exceptions that helps ensure the reliability of applications. In case of an exceptional situation, the framework automatically generates an error report that can be sent to developers for further resolution.

When an exception occurs, Rails handles it using the middleware mechanism, which allows intercepting and handling errors at different levels of the application. The framework also provides the ability to create custom error handlers for specific situations.

When an exception is not handled in user code, Rails catches it and generates an error page.

This page can contain useful information such as a stack trace, request parameters, and other details that assist developers in identifying and fixing the problem.

Furthermore, Rails offers various methods for configuring and handling errors, including the use of specialized error handlers, debugging modes, and logging tools. This enables developers to effectively manage and track errors in the application.

Overall, the error and exception handling mechanism in Ruby on Rails contribute to the increased reliability and stability of applications, providing the ability to quickly detect and resolve issues.

Extending the functionality of a Ruby on Rails application is achieved by installing additional libraries, commonly known as gems. By integrating these popular gems into the application, various tasks can be addressed, such as user authentication and authorization, internationalization of the application, interacting with third-party services like Google, Telegram, Facebook, Twitter, Stripe, and many others, integrating an administration panel, role-based access control, interaction with cloud services, and much more.

These gems provide pre-built functionality and modules that can be easily integrated into the Rails application, saving development time and effort. They are developed and maintained by the community, ensuring their reliability and compatibility with the Rails framework.

By leveraging these additional libraries, developers can enhance their applications with advanced features and streamline the development process, focusing on the core functionality of their application rather than reinventing the wheel for common tasks. It promotes code reuse, and extensibility, and accelerates the overall development process.

Finance is a Ruby gem that provides advanced mathematical functions and algorithms for financial modeling. It includes functions for calculating interest rates, discounts, amortization, dividends, and much more. This gem can be useful for developing financial analytical tools and calculations.

Money is a gem that facilitates convenient handling of monetary values and currency operations. It allows for currency conversion, formatting of monetary values, precise calculations down to the penny, and more. This gem helps ensure accuracy and reliability in financial operations within your application.

ActiveMerchant is a library that provides an interface for working with payment gateways and processing payments. It has support for many popular payment providers and enables convenient and secure handling of financial transactions within your application.

Active Merchant is a simple payment abstraction library used in and sponsored by Shopify. It is written by Tobias Luetke, Cody Fauser, and contributors. The aim of the project is to feel natural to Ruby users and to abstract as many parts as possible away from the user to offer a consistent interface across all supported gateways. The full list of the supported payment operations of each gateway supported by ActiveMerchant you can find in Table 3.

Table 3

The supported payment operations of each gateway supported by ActiveMerchant

Name	Purchase	Authorize	Capture	Void	Credit	Recurring	Card Store
Authorize.net	X	X	X	X	X	X	-
Authorize.Net CIM	X	X	X	X	X	-	X
Balanced	X	X	X	X	X	-	X
Beanstream.com	X	X	X	-	-	-	X
Braintree	X	X	X	X	-	-	X
CardStream	X	-	-	X	X	-	-
CyberSource	X	X	X	X	X	-	-
Efsnet	X	X	X	X	X	-	-
eWAY	X	X	X	X	-	-	X
E-xact	X	X	X	-	X	-	-
Fat Zebra	X	-	-	-	X	-	X
LinkPoint	X	X	X	X	X	X	-
Mastercard	X	X	X	X	X	-	-
MONEI	X	X	X	X	-	-	-
Moneris	X	X	X	X	X	-	X
NETBANX by Optimal Payments	X	X	X	X	X	X	X
NETbilling	X	X	X	-	-	-	-
NetRegistry	X	X	X	-	X	-	-
PayJunction	X	X	X	X	X	X	-
PaymentExpress	X	X	X	-	X	-	X

PayPal Express Checkout	X	X	X	X	X	-	-
PayPal Express Checkout \	X	X	X	X	X	-	-
PayPal Payflow Pro	X	X	X	X	X	X	-
PayPal Website Payments	X	X	X	X	X	X	-
Pro \ PayPal Website Payments	X	X	X	X	X	-	-
Pro \ PaySecure	X	-	-	-	-	-	-
Pin Payments	X	X	X	X	-	X	X
Plug'n Pay	X	X	X	X	X	-	-
Protx	X	X	X	X	X	-	-
Psigate	X	X	X	-	X	-	-
PSL Payment Solutions	X	X	X	-	-	-	-
Quickpay	X	X	X	X	X	-	X
Realex	X	-	-	-	-	-	-
Sage Payment Solutions	X	X	X	X	X	-	-
Samurai	X	X	X	X	-	-	X
SecurePay \	X	-	-	-	-	-	-
SecurePay \	X	-	-	-	-	X	-
SecurePayTech	X	-	-	-	-	-	-
SkipJack	X	X	X	X	X	-	-
Stripe	X	X	X	X	X	X	X
TransFirst	X	-	-	-	-	-	-
TrustCommerce	X	X	X	X	X	X	X
USA ePay	X	X	X	-	-	-	-
Verifi	X	X	X	X	X	-	-
ViaKLIX	X	-	-	-	X	-	-
Wirecard	X	X	X	-	-	-	-
Worldpay	X	X	X	X	X	X	-

Conclusions. The financial technology industry demands reliability, security, and efficiency, and Ruby on Rails possesses all of these characteristics. The conclusions regarding the applicability of Ruby on Rails in the financial sector can be summarized as follows:

Reliability: Ruby on Rails provides a set of tools and mechanisms that help create stable and reliable applications. Refined framework components such as database migration mechanisms, transactions, and error handling enable developers to build robust financial applications.

Security: The Ruby on Rails framework is actively supported by the community and regularly updated to address security vulnerabilities. Additionally, there are numerous additional gems and tools available that provide application protection against attacks and data breaches.

Performance: Ruby on Rails offers efficient development tools that enable rapid application development. With Ruby's simple and expressive syntax and the ready-made components of the framework, developers can accelerate the development process and reduce time to market.

Integration with third-party services: Ruby on Rails has a vast ecosystem of additional libraries and gems that facilitate integration with popular third-party services such as payment gateways, social networks, and other APIs. This simplifies the creation of applications that interact with

financial services and integrate with other systems.

Development convenience: Ruby on Rails features a straightforward and intuitive syntax that simplifies code development and maintenance. The multitude of built-in features and configuration conventions make the development of financial applications more efficient and less labor-intensive.

In conclusion, it can be stated that Ruby on Rails is a sought-after framework in the financial industry and offers numerous advantages that make it an ideal choice for developing financial technologies.

References

1.Active Record and PostgreSQL. How to use UUID primary keys. [Electronic resource]. Access mode: https://guides.rubyonrails.org/v5.0/active_record_postgresql.html (accessed 05/21/2023)

2.Active Record Transcations [Electronic resource]. Access mode: https://api.rubyonrails.org/classes/ActiveRecord/Transactions/ClassMethods.html (accessed 05/21/2023)

3. ActiveMerchant: gem ActiveMerchant https://github.com/activemerchant/active_merchant (accessed 05/21/2023)

4. ActiveRecord [Electronic resource]. Access mode: https://guides.rubyonrails.org/active_rcord_basics.html (accessed 05/21/2023)

5. Books about Ruby and Ruby on Rails [Electronic resource]. Access mode: https://rubyandrails.info/ (accessed 05/21/2023)

6. Error Reporting in Rails Applications [Electronic resource]. Access mode: https://edgeguides.rubyonrails.org/error_reporting.html (accessed 05/21/2023)

7. Fibers in Ruby [Electronic resource]. Access mode: https://ruby-doc.org/core-2.5.0/Fiber.html (accessed 05/21/2023)

8. gem Finace [Electronic resource]. Access mode: https://github.com/marksweston/finance (accessed 05/21/2023)

9. gem Money [Electronic resource]. Access mode: https://github.com/RubyMoney/money (accessed 05/21/2023)

10. gem RubyMine [Electronic resource]. Access mode: https://www.jetbrains.com/ru-ru/ruby/ (accessed 05/21/2023)

11. gem Rspec. [Electronic resource]. Access mode: https://rspec.info/ (accessed 05/21/2023)

12. gem ActiveMerchant https://github.com/activemerchant/active_merchant (accessed 05/21/2023)

13. Ruby Closures for Dummies [Electronic resource]. Access mode: https://medium.com/swlh/ruby-closures-for-dummiesfbf846720c1f (accessed 05/21/2023)

14. Ruby gems [Electronic resource]. Access mode: https://rubygems.org/ (accessed 05/21/2023)

15. Securing Rails Applications [Electronic resource]. Access mode: https://edgeguides.rubyonrails.org/security.html (accessed 05/21/2023)

16. Single table inheritance [Electronic resource]. Access mode: https://api.rubyonrails.org/classes/ActiveRecord/Inheritance.html (accessed 05/21/2023)

17. Threading and Code Execution in Rails [Electronic resource]. Access mode: https://guides.rubyonrails.org/threading_and_code_execution.html (accessed 05/21/2023)

18. 81 Key Fintech Statistics 2023: Market Share & Data Analysis [Electronic resource]. Access mode: https://financesonline.com/fint

Стаття надійшла до редакції 13.10.2023
Прийнята до публікації 19.10.2023