COMPREHENSIVE REVIEWER

System Integration & Architecture
Systems Design
Laravel Fundamentals and Authentication

Prepared from the uploaded lecture files
Designed for fast review before quizzes, recitation, and exams

Included files	SIA Lecture 1, Lecture 2, Lecture 3, Systems Design, Laravel Introduction, Authentication Module
Style of reviewer	Simple explanations, key points, compare-and-contrast notes, and practical reminders
Best use	Read section summaries first, then memorize the checklists and quick-review tables

How to Use This Reviewer

- Start with the “Big idea” box of each file so you understand the lesson in one reading.
- Use the “What to remember” bullets for memorization and short-answer preparation.
- Read the comparison tables carefully because they are the most likely to appear in tests.
- For Laravel topics, focus on the sequence of commands and what each command actually does.
- For SIA topics, focus on definitions, classifications, phases, and how concepts connect to one another.

Tip: If you are reviewing the night before an exam, read only the file summaries, the “What to remember” lists, and the quick tables.

File 1 - SIA Lecture 1: Foundations of System Integration and Architecture

Big idea: This lecture introduces the course by explaining why systems fail when they are isolated, why architecture matters before integration, and why projects, stakeholders, and organizational context must all be considered together.

Main topics in simple words

- System integration means making a proposed system work smoothly with other existing or planned systems.
- Architecture is the high-level structure of a system; without it, systems may not interoperate well.
- Integration is valuable because it improves flexibility, speed, cost efficiency, standardization, data integrity, reliability, and robustness.
- A system is a set of parts working together for a goal, while systems thinking explains a part by looking at the larger whole it belongs to.
- A project is temporary and unique, and it exists inside a broader organization rather than in isolation.
- Stakeholders include everyone involved in or affected by the project, from sponsors and users to suppliers and even opponents.
- Project success is strongly influenced by executive support, user involvement, clear objectives, realistic scope, and good requirements.
- The lecture also introduces organizational frames, organizational structures, project life cycle phases, and SDLC models.

What to remember

- Definition of system integration, system, system architecture, project, and stakeholder.
- Three sources of IS projects: problems, opportunities, and directives.
- Four organizational frames: structural, human resources, political, symbolic.
- Four basic organizational structures: functional, divisional, matrix, and project-based.

- Project life cycle phases shown in the slides: concept, development, implementation, close-out.
- Two broad SDLC families: predictive and adaptive.

Concept	Meaning / quick reviewer note
System integration	Connecting systems so they work seamlessly together.
System architecture	The high-level structure that shows components and their interactions.
Systems thinking	Understanding a part by first understanding the larger system.
Predictive model	Works best when scope, schedule, and cost can be planned clearly.
Adaptive model	Works best when change is expected and short cycles are needed.

Item	Examples from the lecture
System integration applications	Business process integration, legacy integration, new system integration, B2B, COTS, interface control, testing, program management, BCP
Predictive models	Waterfall, Spiral, Incremental Release, Prototyping, RAD
Adaptive models	Extreme Programming (XP), Scrum

Exam angle: This file is definition-heavy. Expect identification questions, comparisons of organizational structures, and classification questions about project sources and SDLC types.

File 2 - SIA Lecture 2: Framework, Methods, and Approaches of System Development

Big idea: This lecture explains SDLC as the overall framework for software development, then shows multiple methods that fit inside SDLC, from traditional linear models to modern iterative approaches.

Main topics in simple words

- SDLC is the full process of developing software from conception to deployment and maintenance.
- The slides present seven major SDLC phases: identify problems/opportunities/objectives, determine requirements, analyze system needs, design the system, develop and document software, test and maintain, then implement and evaluate.
- Different methods are simply different ways to move through SDLC phases depending on the project situation.
- Waterfall is linear and orderly; Agile is iterative and flexible; Spiral adds strong risk management; V-Model pairs each development stage with testing.

- Prototyping uses early mock-ups to refine requirements, while RAD compresses development through rapid, user-centered iteration.
- DevOps strengthens implementation, deployment, and maintenance through automation, CI/CD, monitoring, and collaboration.
- Lean development removes waste and focuses on value; object-oriented development models systems as interacting objects.

What to remember

- Waterfall = linear and best for stable requirements.
- Agile = flexible, incremental, and best for changing requirements.
- Spiral = iterative plus risk analysis, usually for large or high-risk projects.
- V-Model = development and testing mapped together.
- Prototyping = early model for feedback and clarification.
- RAD = rapid delivery through reusable parts and compressed cycles.
- DevOps = collaboration + automation + CI/CD + monitoring.
- Traditional approaches in the slides mostly include Waterfall, Spiral, V-Model, Prototyping, RAD, and Incremental; modern approaches include Agile, DevOps, OOP, and Lean.

Method	Best-known strength	Good fit
Waterfall	Orderly sequence	Projects with fixed and clear requirements
Agile	Flexibility and continuous feedback	Projects with changing requirements
Spiral	Risk management	Large, complex, high-risk systems
V-Model	Strong testing focus	Projects where validation is critical
Prototyping	Requirement clarification	Projects with unclear user needs
RAD	Speed	Fast delivery with active user involvement
DevOps	Automation and operations support	Continuous delivery and maintenance-heavy systems

Additional phase	Typical focus
Feasibility study	Project viability before full commitment
Risk management	Identify and reduce project risks early
Prototyping and validation	Refine requirements before full build
CI/CD	Automate integration and deployment
Monitoring and optimization	Track performance and improve after release
Compliance and security auditing	Meet standards, laws, and controls

Exam angle: Most likely questions are compare-and-contrast questions: Waterfall vs Agile, Spiral vs V-Model, traditional vs modern approaches, and SDLC vs specific methodologies.

File 3 - SIA Lecture 3: Requirements

Big idea: This lecture focuses on how software teams discover, document, organize, model,

manage, and test requirements so the system being built actually matches stakeholder needs.

Main topics in simple words

- Requirements elicitation is the process of identifying, gathering, and understanding stakeholder needs, expectations, constraints, and objectives.
- The lecture lists several elicitation techniques: interviews and surveys, user stories and use cases, prototyping, requirement workshops, observation and job shadowing, document analysis, and brainstorming.
- Requirements are classified into functional requirements and non-functional requirements.
- Functional requirements describe what the system should do, such as login, CRUD, validation, payment processing, and reporting.
- Non-functional requirements describe how the system should perform, such as speed, scalability, security, usability, reliability, maintainability, and portability.
- Requirements should be documented using tools such as SRS, BRD, FRD, user stories, use case diagrams, and a traceability matrix.
- Requirement management includes change management, version control, impact analysis, and conflict handling.
- The lecture also introduces UML and related visual tools, then connects requirements with testing through validation, verification, TDD, BDD, test case design, and UAT.

Functional requirements	Non-functional requirements
Describe what the system does	Describe how the system performs
Examples: authentication, CRUD, search, reports	Examples: response time, security, scalability, reliability
Usually tested through functional testing	Usually tested through performance, security, or usability testing
Drive features and operations	Drive constraints, quality, and architecture

What to remember

- Know at least five elicitation techniques and their purpose.
- Be able to distinguish functional requirements from non-functional requirements quickly.
- Remember key requirement documents: SRS, BRD, FRD, user stories, use case diagrams, traceability matrix.
- UML-related modeling tools in the slides include use case, class, sequence, activity, and state machine diagrams, plus DFD, ERD, BPMN, and Agile modeling tools.
- Sample user stories in the lecture are for an online library system: registration, borrowing, and librarian book management.

Requirement activity	Simple purpose
Elicitation	Gather needs from stakeholders
Documentation	Write requirements in clear and consistent form
Maintenance and management	Control change and analyze impact

Requirement activity	Simple purpose
Modeling	Visualize processes, actors, modules, and interactions
Testing of requirements	Check that requirements are correct, testable, and covered

Exam angle: This lesson is ideal for identification, scenario-based classification, and "give examples" questions. If a question asks whether something is FR or NFR, focus on whether it is a feature or a quality attribute.

File 4 - Systems Design: Modern, Framework-Agnostic View

Big idea: This file explains systems design in a modern way: not tied to one framework, centered on users and business goals, and supported by frameworks that give structure, security, and speed during implementation.

Main topics in simple words

- Framework-agnostic means the design ideas can be applied no matter what framework or programming language is used.
- Traditional systems design asked what components must be designed and how they are combined.
- Modern systems design adds human-centered concerns: who the users are, how they experience the system, how the system evolves, and how it is continuously improved.
- Modern software and systems design integrates engineering, UX, data modeling, security, authentication, and continuous feedback.
- Frameworks support design by giving reusable structures, conventions, architectural patterns, security features, routing, data access, and API support.
- The lecture maps design ideas into phases: requirements analysis, conceptual and architectural design, detailed design and data modeling, validation and security, agile implementation, testing/deployment/DevOps, and maintenance/improvement.

What to remember

- Framework-agnostic = principle-based, not tool-bound.
- Modern systems design is iterative, collaborative, and user-driven.
- Framework contributions include MVC or layered structure, separation of concerns, persistence, UI layers, authentication, routing, API support, and DevOps/CI/CD integration.
- Each design phase has a practical goal: translate user stories, design architecture, model data, secure inputs, deliver incrementally, test and deploy, then improve continuously.

Traditional focus	Modern added focus
What components do we design?	For whom are we designing?
How do we combine components?	How do users experience the system?
Technical assembly of parts	Continuous evolution and improvement

Phase	Design concern
Requirements analysis	Map stories and requirements to components, workflows, and rules
Conceptual and architectural design	Choose patterns, layers, and system boundaries
Detailed design and data modeling	Create schemas, business logic organization, and UI consistency
Validation and security	Input checks, sanitization, auth, vulnerability protection, logging
Agile implementation	Build incrementally with reusable framework tools
Testing, deployment, DevOps	Use automated tests, deployment, cloud/container support
Maintenance and improvement	Refactor, monitor, update dependencies, enhance continuously

Exam angle: This file connects theory and practice. If asked why frameworks matter in design, answer in terms of structure, maintainability, security, reusability, and faster delivery.

File 5 - Laravel Introduction

Big idea: This file introduces Laravel as a full-stack PHP framework, explains why frameworks are useful, lists Laravel's major features and advantages, and shows the basic installation and run process on Windows.

Main topics in simple words

- A framework is a pre-built structure with reusable code, libraries, and tools that make development faster and more organized.
- Compared with development from scratch, frameworks improve speed, code reuse, security, maintainability, and scalability, though they require learning framework-specific concepts.
- Laravel is a PHP framework that follows the MVC architectural pattern and includes tools for authentication, routing, caching, and database management.
- The slides describe Laravel as often being considered full-stack because it supports both backend and frontend concerns for web applications.
- Core Laravel features listed in the file include MVC architecture, Eloquent ORM, Blade templating, routing, security features, authentication and authorization, Artisan CLI, queues, events/broadcasting, and RESTful API support.
- Advantages emphasized in the slides include faster development, easier authentication/authorization, reduced technical vulnerabilities, MVC-based organization, and traffic handling support such as queues and load balancing.
- The file also shows how to install Composer, create a Laravel project, run php artisan serve, and view the default Laravel pages in the browser.

Framework	Development from scratch
Faster due to pre-built components	Slower because everything is built manually
Reusable modules and libraries	Write all functionality manually
Built-in security measures	Security must be implemented manually
Easier maintenance and updates	Maintenance is heavier and more manual
Good structure through MVC and conventions	Structure depends entirely on the developer

What to remember

- Laravel = PHP + MVC + built-in tools for common web tasks.
- Blade = templating engine for views.
- Eloquent ORM = easier database interaction.
- Artisan = command-line tool for recurring tasks like migrations and serving the app.
- Queues defer time-consuming tasks such as email sending.
- Running the local server is done with php artisan serve.

Laravel feature	Why it matters
MVC architecture	Separates logic, interface, and data for easier maintenance
Security features	Helps defend against SQL injection, CSRF, and other common risks
Authentication	Provides built-in support for login, registration, and authorization
Routing	Organizes request handling and clean URLs
Queues	Improves responsiveness by moving heavy tasks to the background

Exam angle: This file is likely to produce “define and compare” questions, plus practical questions about what Laravel components do and why using a framework is better than coding from scratch.

File 6 - Laravel Authentication Module

Big idea: This file shows a practical workflow for adding authentication to a Laravel project using Laravel Breeze, connecting the app to SQLite, running migrations, and verifying that registered users are saved in the database.

Main topics in simple words

- The process begins with creating or opening a Laravel project and running the development server so the app can be viewed in the browser.
- Laravel Breeze is installed as a development dependency using composer require laravel/breeze --dev.
- Breeze is then used to generate authentication scaffolding with php artisan breeze:install.

- Scaffolding means Laravel automatically creates the common files needed for login, registration, password reset, dashboard, routes, controllers, and views instead of forcing the developer to code everything manually.
- The lecture then shows database connection selection, specifically SQLite, where DB_CONNECTION=sqlite is used and server-based values like DB_HOST or DB_PORT are not needed.
- SQLite is a lightweight, file-based relational database stored in one file, making it convenient for small projects, local development, and testing.
- After configuration, migrations are run to create the necessary database tables automatically.
- The final practical goal is to register a user and verify that the record exists in the SQLite database.

Command / setting	Purpose
composer require laravel/breeze --dev	Install Laravel Breeze package as a development dependency
php artisan breeze:install	Generate authentication scaffolding
php artisan serve	Run the local development server
DB_CONNECTION=sqlite	Tell Laravel to use SQLite instead of a server-based database
php artisan migrate	Create required database tables automatically

What to remember

- Breeze provides login, registration, password reset, dashboard, routes, controllers, and Blade views.
- Scaffolding = auto-generated code structure for common features.
- SQLite does not need a separate database server; it stores data in one local file.
- Migrations create tables such as users and other auth-related tables.
- Verification means checking that the registered user was actually saved in the database.

SQLite	Traditional server database (e.g., MySQL)
File-based and lightweight	Runs as a separate database server
No DB host, port, username, or password needed	Requires host, port, database name, username, and password
Good for local development and testing	Better for larger multi-user production systems

Exam angle: This file is process-oriented. Be ready to explain what each command does, what scaffolding means, why SQLite is simple to set up, and how migrations connect the app to the database structure.

Final Synthesis Across All Files

Big idea: Taken together, the files move from theory to implementation: first understand systems, integration, projects, and SDLC; then learn requirements and systems design; finally apply those ideas using Laravel and an authentication module.

Area	What the reviewer suggests you focus on
System Integration & Architecture	Definitions, classifications, project sources, stakeholders, organizational structures, SDLC families
System Development Methods	When to use Waterfall, Agile, Spiral, V-Model, Prototyping, RAD, DevOps, Lean, OOP
Requirements	FR vs NFR, elicitation techniques, requirement documents, UML and testing links
Systems Design	Framework-agnostic principles, design phases, role of frameworks
Laravel	MVC, Blade, Eloquent, Artisan, security, authentication, queues
Authentication Module	Breeze install, scaffolding, SQLite setup, migrations, user verification

Fastest last-minute checklist

- Memorize the definitions from Lecture 1.
- Know the strengths of each SDLC method from Lecture 2.
- Separate functional from non-functional requirements instantly.
- Explain why modern systems design is user-centered and framework-supported.
- Know what Laravel components do and why MVC matters.
- Memorize the purpose of the key authentication commands and SQLite settings.