University of Staffordshire

COMP50016

Server Side Programming - II

Module Learning Outcomes (LO)

On completion of this module, you will be able to:

1. Demonstrate a critical understanding of the functionality that is used in server-side web application frameworks.
2. Design, implement, test and demonstrate a flexible, robust and secure server-side web application solution.
3. Apply appropriate web application testing strategies and explain the importance of their use.
4. Demonstrate a critical understanding of the security issues that affect web applications and implement an appropriate strategy to counter potential risks.

Deadline	Weighting
Dates available on SAIS	80%

University Regulations

- University Regulations regarding exceptional circumstances and academic misconduct will apply.
- Please ensure that you are familiar with these regulations.
- https://www.staffs.ac.uk/students/course-administration/academic-policies-and-regulations/home

Submission rules

- late submissions attract 0 marks for that section
- failure to submit on Blackboard may forfeit your opportunity to present or demonstrate your work
- failure to attend the presentation or demonstration on time may result in 0 marks for that component of assessed work
- during the presentation/demonstration, if asked, you must be able to explain in detail your work, otherwise this may result in 0 marks for that component of assessed work

Any questions about the assessment should be directed to the module leader

Assignment Specification

Assessment Requirements:

You need to submit the following via LMS:

- A zip file containing your code (the database, the PHP theme files(if used), the PHP files and the database for PHP).
- You need to attend a 10-minute demonstration.
- The Github link to your project
- The Hosted Link

Scenario

This coursework will focus on **continuing the development of your SaaS application from SSP1**, now using the **Laravel 12 Framework**.

Assume that your system is being migrated from its **legacy PHP implementation** into a modern Laravel-based application. You are required to identify which core requirements from SSP1 can be **carried over**, and which parts need to be **refactored, extended, or newly created** in Laravel.

You must demonstrate an understanding of the **core features of server-side applications**, including:

- Security and data protection
- Encryption and secure storage of sensitive information
- Token-based authentication and authorization (e.g., Sanctum or Passport)
- Usage of **APIs** to expose data endpoints for integration or consumption

You will then apply this knowledge to **expand your own SaaS application**. This should include both:

1. A Laravel implementation of the SSP1 system's **core features** (CRUD, authentication, landing page).
2. Additional Laravel-based functionality such as **API routes, middleware, role management, or integrations** that demonstrate scalability.

Task

You are required to build a web application using **Laravel 12**, **SQL**, and **Tailwind** that meet the requirements stated in the table below.

Marking Criteria

Criteria	Marks
Is built using Laravel **12**	10
Has a SQL database connection	10
Meets specific criteria given by the business scenario:	**40**
• Use of external libraries (Livewire/Volt) for certain functional requirements (10) • Use of Laravel's Eloquent Model (10) • Use of Laravel's authentication package (Laravel Jetstream) to protect and authenticate routes (10) • Use of Laravel Sanctum to authenticate the API (10)	
Security Documentation and Implementation	15
API Extension / Integration	10
Use of a hosting service provider to host the application (Optional)	15
TOTAL	100

Mark allocations (Out of 10 marks)

	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
Built using Laravel **12**	**0**: No Use of Laravel: The project does not utilize the Laravel framework at all. Another framework or technology stack is used instead. **1-2**: Limited Use of Laravel: There's some attempt to use Laravel, but it's minimal or incomplete. The project may lack fundamental Laravel features, such as routing, controllers, models, or migrations. **3**: Basic Laravel Implementation: Laravel is used, but there are significant deficiencies in implementation. Basic features such as routing and database management may be present, but they lack optimization, structure, or adherence to Laravel best practices.	Laravel Implementation : The project demonstrates a satisfactory level of Laravel implementation . Essential Laravel features are utilized, including routing, controllers, models, migrations, and views. The project follows Laravel conventions and best practices to some extent, but there's room for improvement in optimization and structure.	Implementation: The project showcases a good understanding of Laravel concepts and best practices. Laravel features are utilized effectively to build a robust and scalable application. The project follows Laravel conventions closely, with well-organized code structure and optimization.	Laravel Implementation: The project goes beyond basic requirements, demonstrating a high level of proficiency in Laravel development. Advanced Laravel features, such as middleware, relationships, authentication, and Eloquent ORM, are implemented effectively. The project follows Laravel conventions rigorously, with clear code structure and optimization.	7-8: Excellent Laravel Implementation: The project excels in Laravel development, showcasing mastery of the framework's capabilities. Advanced Laravel features are implemented seamlessly, demonstrating creativity and innovation in solution design. The project follows Laravel conventions meticulously, with exceptional code organization, optimization, and scalability. 9-10: Outstanding Laravel Implementation: The project sets a benchmark for Laravel development, demonstrating exceptional proficiency and innovation. It leverages Laravel's full potential to deliver a high-performance, scalable, and maintainable application. Advanced features, such as queues, caching, and event listeners, are implemented with excellence. The project serves as a testament to the developer's expertise in Laravel development.
Has a SQL database connection	**0: No SQL** Database Connection: The project lacks any implementation of a SQL database connection. There's no integration with a SQL database system. 1-2: Incomplete SQL Database Connection: There's some attempt to establish a SQL database	Adequate SQL Database Connection: The project demonstrates a satisfactory level of SQL database connection implementation . The connection is established securely, with appropriate settings for authentication and encryption. Basic CRUD	Good SQL Database Connection: The project showcases a good understanding of SQL database connection principles and best practices. The connection is established securely and efficiently, with optimized settings for performance and security. CRUD operations are implemented	Very Good SQL Database Connection: The project goes beyond basic requirements, demonstrating proficiency in SQL database connection management. Advanced features such as connection pooling or query optimization are implemented to enhance performance and scalability. CRUD operations are implemented with efficiency and robustness.	7-8: Excellent SQL Database Connection: The project excels in SQL database connection implementation, showcasing mastery of database management principles. The connection is established using industry best practices for security, scalability, and performance. Advanced features such as stored procedures, triggers, or transactions are implemented effectively to ensure data integrity and reliability. 9-10: Outstanding SQL Database Connection: The project sets a benchmark for SQL database connection implementation,

	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
	connection, but it's incomplete or poorly implemented. Connection settings may be incorrect or missing, leading to errors or inconsistencies. 3: Basic SQL Database Connection: A basic SQL database connection is established, but there are significant deficiencies in implementation. Connection settings may lack security measures, such as encryption or parameterized queries.	operations can be performed reliably.	effectively, with error handling and data validation.		demonstrating exceptional proficiency and innovation. The connection architecture is designed with scalability, fault tolerance, and high availability in mind. Advanced SQL features and optimizations are utilized to maximize database performance and efficiency. The project serves as a testament to the developer's expertise in SQL database management
Use of external libraries (Livewire/Volt) for certain functional requirements	0: No Use of External Libraries: The project lacks any implementation of external libraries such as Livewire or Volt for fulfilling functional requirements. The use of third-party libraries is absent. 1-2: Limited Use of External Libraries: There's some attempt to use external libraries for fulfilling functional requirements, but it's minimal or ineffective. The chosen libraries may not be suitable for the requirements, or their implementation may be incomplete or	Adequate Use of External Libraries: The project demonstrates a satisfactory level of use of external libraries such as Livewire or Volt for fulfilling functional requirements. The chosen libraries are effectively integrated into the project, providing value-added features and enhancing user experience. However, there may be room for improvement in optimization or customization.	The project showcases a good understanding of leveraging external libraries such as Livewire or Volt for fulfilling functional requirements. The chosen libraries are integrated seamlessly, providing enhanced functionality and user interaction. The implementation is well-optimized and customized to meet specific project needs.	The project goes beyond basic requirements, demonstrating proficiency in leveraging external libraries such as Livewire or Volt effectively. The chosen libraries are utilized innovatively to address complex functional requirements, resulting in a seamless user experience. The implementation reflects careful optimization and customization to align with project goals.	7-8: Excellent Use of External Libraries: The project excels in leveraging external libraries such as Livewire or Volt for fulfilling functional requirements, showcasing mastery of library integration and customization. The chosen libraries are utilized strategically to achieve project objectives, with a focus on scalability, performance, and user satisfaction. The implementation sets a high standard for innovation and excellence. 9-10: Outstanding Use of External Libraries: The project sets a benchmark for the use of external libraries, demonstrating exceptional proficiency and innovation. The chosen libraries such as Livewire or Volt are seamlessly integrated and customized to deliver transformative solutions for complex functional requirements. The implementation reflects a deep understanding of library capabilities and project requirements, resulting in a highly optimized and user-centric application.

	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
	poorly integrated. 3: Basic Use of External Libraries: Basic use of external libraries such as Livewire or Volt is demonstrated for fulfilling certain functional requirements. However, the implementation may lack optimization, customization, or adherence to best practices				
Use of Laravel's Eloquent Model	**0**: No Use of Eloquent Model: The project lacks any implementation of Laravel's Eloquent Model for database interactions. Direct SQL queries or alternative methods are used instead. **1-2**: Limited Use of Eloquent Model: There's some attempt to use Laravel's Eloquent Model, but it's minimal or ineffective. The implementation may be incomplete, inconsistent, or poorly integrated with the project. **3**: Basic Use of Eloquent Model: Basic use of Laravel's Eloquent Model is demonstrated for database interactions. However, the implementation may lack optimization, scalability, or	The project demonstrates a satisfactory level of use of Laravel's Eloquent Model for database interactions. CRUD operations are performed effectively using Eloquent's ORM functionalities. Relationships between models are established correctly, but there may be room for improvement in optimization or customization.	The project showcases a good understanding of leveraging Laravel's Eloquent Model for database interactions. Eloquent's ORM functionalities are utilized effectively to perform CRUD operations and manage relationships between models. The implementation is well-optimized and customized to meet specific project needs.	The project goes beyond basic requirements, demonstrating proficiency in leveraging Laravel's Eloquent Model effectively. Advanced Eloquent features such as query scopes, accessors, and mutators are utilized to enhance database interactions and data manipulation. The implementation reflects careful optimization and customization to align with project goals.	7-8: Excellent Use of Eloquent Model: The project excels in leveraging Laravel's Eloquent Model for database interactions, showcasing mastery of ORM principles and best practices. Eloquent functionalities are utilized strategically to achieve project objectives, with a focus on performance, scalability, and maintainability. The implementation sets a high standard for efficiency and effectiveness. 9-10: Outstanding Use of Eloquent Model: The project sets a benchmark for the use of Laravel's Eloquent Model, demonstrating exceptional proficiency and innovation. The Eloquent Model is seamlessly integrated and customized to deliver transformative solutions for database interactions. Advanced Eloquent features are leveraged to optimize performance, simplify complex queries, and ensure data consistency. The implementation reflects a deep understanding of ORM principles and project requirements, resulting in a highly efficient and scalable application.

	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
	adherence to best practices.				
Use of Laravel's authenticati on package (Laravel Jetstream) to protect and authenticat e routes	0: No Use of Laravel's Authentication Package: The project lacks any implementation of Laravel's authentication package, such as Laravel Jetstream, for route protection and authentication. Custom authentication methods or no authentication at all are used instead. 1-2: Limited Use of Authentication Package: There's some attempt to use Laravel's authentication package, but it's minimal or ineffective. The implementation may be incomplete, inconsistent, or poorly integrated with the project. Basic authentication functionalities are implemented, but there are deficiencies in security or user experience. 3: Basic Use of Authentication Package: Basic use of Laravel's authentication package, such as Laravel Jetstream, is demonstrated for route protection and authentication. However, the implementation may lack customization,	The project demonstrates a satisfactory level of use of Laravel's authentication package, such as Laravel Jetstream, for route protection and authentication. Basic authentication functionalities are implemented effectively, providing user registration, login, and password reset features. However, there may be room for improvement in customization or integration with the project's frontend.	The project showcases a good understanding of leveraging Laravel's authentication package, such as Laravel Jetstream, for route protection and authentication. Authentication functionalities are implemented seamlessly, providing a secure and user-friendly experience for registration, login, and password management. The implementation is well-integrated with the project's frontend and follows best practices for security and usability.	The project goes beyond basic requirements, demonstrating proficiency in leveraging Laravel's authentication package, such as Laravel Jetstream. Advanced authentication features such as two-factor authentication (2FA) or OAuth integration may be implemented to enhance security and user experience. The implementation reflects careful customization and optimization to align with project goals.	7-8: Excellent Use of Authentication Package: The project excels in leveraging Laravel's authentication package, such as Laravel Jetstream, for route protection and authentication, showcasing mastery of authentication principles and best practices. Authentication functionalities are implemented strategically to achieve project objectives, with a focus on security, scalability, and user satisfaction. The implementation sets a high standard for efficiency, effectiveness, and usability. 9-10: Outstanding Use of Authentication Package: The project sets a benchmark for the use of Laravel's authentication package, demonstrating exceptional proficiency and innovation. The authentication package, such as Laravel Jetstream, is seamlessly integrated and customized to deliver transformative solutions for route protection and authentication. Advanced authentication features and customization options are leveraged to optimize security, simplify user management, and enhance user experience. The implementation reflects a deep understanding of authentication principles and project requirements, resulting in a highly secure, scalable, and user-centric application.

	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
	scalability, or adherence to best practices.				
Use of Laravel Sanctum to authenticate the API	0: No Use of Laravel Sanctum: The project lacks any implementation of Laravel Sanctum for API authentication. No authentication mechanism is in place for the API. 1-2: Limited Use of Laravel Sanctum: There's some attempt to use Laravel Sanctum for API authentication, but it's minimal or ineffective. The implementation may be incomplete, insecure, or poorly integrated with the project. Basic token generation or validation might be present but lacks robustness. 3: Basic Use of Laravel Sanctum: Basic use of Laravel Sanctum is demonstrated for API authentication. Tokens are generated and validated, but the implementation may lack customization, security measures, or adherence to best practices.	Adequate Use of Laravel Sanctum: The project demonstrates a satisfactory level of use of Laravel Sanctum for API authentication. Tokens are generated and validated effectively, providing a basic level of security for API endpoints. However, there may be room for improvement in customization, security, or integration with the project's overall architecture.	Good Use of Laravel Sanctum: The project showcases a good understanding of leveraging Laravel Sanctum for API authentication. Token-based authentication is implemented seamlessly, providing secure access to API endpoints. The implementation includes essential security measures, such as token expiration and revocation, and is well-integrated with the project's overall architecture.	Very Good Use of Laravel Sanctum: The project goes beyond basic requirements, demonstrating proficiency in leveraging Laravel Sanctum for API authentication. Advanced features such as token scopes and personal access tokens are utilized to enhance security and flexibility. The implementation reflects careful customization and optimization to align with project goals and best practices.	7-8: Excellent Use of Laravel Sanctum: The project excels in leveraging Laravel Sanctum for API authentication, showcasing mastery of API security principles and best practices. The implementation is highly secure, with advanced features such as token scopes, expiration, revocation, and multi-device support. The integration is seamless, providing a robust and scalable solution for API authentication. 9-10: Outstanding Use of Laravel Sanctum: The project sets a benchmark for the use of Laravel Sanctum, demonstrating exceptional proficiency and innovation. Laravel Sanctum is seamlessly integrated and customized to deliver a secure and user-friendly API authentication solution. Advanced features and security measures are leveraged to ensure optimal performance and protection against threats. The implementation reflects a deep understanding of API security and project requirements, resulting in a highly secure, scalable, and maintainable application.
	0–5: Unsatisfactory	6–7: Satisfactory	8–9: Good attempt	10–11: Very good attempt	12–15: Excellent attempt
Security Documentation and Implementation	No evidence of security practices. Sensitive data (e.g., passwords)	Basic security measures implemented (hashed passwords,	Secure authentication, input validation, and session handling in place. Basic encryption	Strong security practices (encryption, CSRF tokens, role-based access). Clear documentation of threats, mitigations, and testing.	Exemplary security implementation. Covers multiple layers (validation, XSS/SQLi prevention, HTTPS, secure APIs). Documentation is thorough, well-structured, and maps directly to the system.

	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
	stored in plain text. No documentation on threats or mitigation.	form validation). Limited documentation on risks. Some vulnerabilities remain.	and error handling. Documentation identifies some risks and countermeasures.		
	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
API Extension / Integration (Optional)	**0:** No extension or integration. No API endpoints created, no third-party API used, and no attempt to extend the system beyond basic CRUD **1–2:** Limited extension. Some attempt at an API or integration, but incomplete or ineffective (e.g., broken endpoints, unstable third-party calls, or poorly configured NoSQL). **3:** Basic extension. A simple REST API or minimal integration is present (e.g., a few working endpoints, or a basic MongoDB CRUD). Lacks optimization, robust error handling, or clear connection to the system's business scenario.	Adequate extension. A working API or integration with stable data handling. CRUD operations or third-party consumption is functional. Error handling exists but is basic. Limited use of advanced features (e.g., authentication, request validation, indexing/aggregation in MongoDB)..	Good extension. The project demonstrates a solid understanding of API development/integration. Endpoints are well-structured, error handling is adequate, and some advanced features are included (e.g., role-based access, aggregations, meaningful third-party data usage).adequate.	Very good extension. The API or integration goes beyond basics, demonstrating proficiency in performance optimization, validation, and security. Advanced techniques are applied (e.g., token-based auth, middleware, indexing in MongoDB, or thoughtful third-party API workflows).	Outstanding extension. The project sets a benchmark for API design and integration. Endpoints are highly efficient, secure, and scalable. Advanced features are fully utilized (e.g., Sanctum for tokens, complex aggregations in MongoDB, robust error handling, caching, pagination, or external service integrations). The implementation follows best practices and is maintainable, professional, and innovative.
	0–5: Unsatisfactory	6–7: Satisfactory	8–9: Good attempt	10–11: Very good attempt	12–15: Excellent attempt
Use of a hosting service provider to host the application (Optional)	No use of hosting service provider. Application only runs locally (e.g., on	Basic deployment attempted (e.g., shared hosting, cPanel, or free-tier cloud).	Application successfully deployed and accessible with proper configuration.	Deployed on a proper cloud platform (e.g., Heroku, Railway, DigitalOcean, AWS free tier).	Hosting demonstrates proficiency with advanced deployment practices. Application uses robust configurations such as autoscaling, load balancing, managed databases, or CDNs.

	0 – 3 Unsatisfactory	4 satisfactory	5 good attempt	6 very good attempt	7-10 Excellent Attempt
	XAMPP/Artisan serve) without any attempt to deploy externally. No documentation provided on deployment steps.	Application may be accessible online, but with reliability issues (e.g., no HTTPS, frequent downtime, misconfigured environment variables).	Hosting includes HTTPS (SSL/TLS) and stable uptime. Basic use of hosting features like database connections, environment configuration, or error logs. *Note: Many hosting platforms (e.g., Heroku, Render, Railway) provide free HTTPS via Let's Encrypt. AWS EC2 does not provide free SSL by default; HTTPS can be configured via Let's Encrypt with a custom domain or via AWS Certificate Manager when used with services like Elastic Load Balancer or CloudFront*	Includes environment configuration (.env), secure database access, HTTPS enabled. Demonstrates optimization such as caching, automated backups, or use of a CDN. Shows understanding of pushing code updates (e.g., via Git or CI/CD pipeline).	CI/CD pipelines in place for automated deployment. Strong focus on security (firewalls, environment separation for dev/prod, encrypted secrets). Application is highly reliable, scalable, and documented with clear deployment steps..

Submission and Assessment

- Submission is via LMS. A zipped copy of your code (not a RAR file) should be submitted immediately following your demonstration.
- Submission will not be accepted by alternative means (such as email), so you should ensure your submission is made well before the deadline to avoid last-minute problems.
- There will be a scheduled demonstration slot for you to show your work. Failure to demonstrate the system during your allocated time slot will automatically give you zero marks regardless of whether the work has been submitted.
- The database you choose to build the application on must be based on SQL. **If SQLite is used, the DB file must be attached; if MySQL is used, an SQL dump must be attached to the ZIP submission file**. If you do need to include a NoSQL database as well, screenshots of your node collections should be included in the submission file under a folder labeled "NoSQL DB Nodes Collections."

Optional Marks

The marking scheme details 2 components that are optional, the use of a API and the ability to host your application. These 2 can be completed at your own discretion, as these topics will not be covered during lecture hours. You can use the below resources to learn more about these 2 topics.

Do note, the resources below require an active Linkdlin learning account to use.

https://www.linkedin.com/learning/aws-and-react-creating-full-stack-apps-23432036/full-stack-react-development-on-aws?u=180139196

https://www.linkedin.com/learning/aws-essential-training-for-developers-17237791/what-is-the-best-way-to-use-aws?u=180139196

https://www.linkedin.com/learning/learning-aws-amplify-22879744/build-apps-with-aws-using-amplify-cli?u=180139196

https://www.linkedin.com/learning/amazon-ec2-essential-training/learning-amazon-ec2?u=180139196

https://www.linkedin.com/learning/nosql-essential-training/get-to-know-nosql?u=180139196

https://www.linkedin.com/learning/mongodb-essential-training/an-introduction-to-mongodb-23754278?u=180139196