Publishing Your Project

- Student Project Lab: Student Publication Guide
- Student Project Lab: Application Security Requirements
- Student Project Lab: Requesting Security Review and Publication

Student Project Lab: Student Publication Guide

Overview

Student Project Lab provides hosting for approved student-developed software projects. The platform allows students to publish working applications that can be shared with employers, included in professional portfolios, or demonstrated as academic accomplishments.

Publication is available for eligible capstone projects, course projects, independent studies, research projects, and other approved student-developed applications.

Hosting is provided at the department's discretion.

1. How Publication Works

The publication process consists of six stages:

1. **Develop:** Complete your application in a Git repository.
2. **Prepare:** Configure your project for containerized deployment and document its requirements.
3. **Test:** Verify that your application builds, runs, and passes the required security checks.
4. **Submit:** Identify an exact release candidate and request a security review.
5. **Review:** Address any findings identified during faculty review.
6. **Publish:** Once approved, the department deploys the approved version to Student Project Lab.

Students are responsible for preparing their applications. Department administrators are responsible for production deployment, infrastructure management, and hosting authorization.

You will not receive SSH access to the production server or administrative access to the deployment platform.

2. Source Control Requirements

All applications must be maintained in a Git repository accessible to the department.

GitHub is the preferred hosting provider.

Your repository may be:

- A personal GitHub repository.
- A team-owned repository.
- A GitHub Classroom repository.
- Another Git repository approved by the department.

Repository requirements

Your repository must:

- Contain the complete source code needed to build the application.
- Include a README with installation and development instructions.
- Include deployment configuration and dependency information.
- Have no secrets, credentials, or private keys committed to Git history.
- Use versioned commits and release tags.
- Grant reviewers read access to source code and release information.
- Have an appropriate license or documented permission for third-party source code and assets.

Private repositories are permitted, provided the department can access them through an approved integration.

Recommended repository layout

A typical application might use:

```
my-project/
├── .github/
│   └── workflows/
│       └── ci.yml
├── frontend/
│   ├── Dockerfile
│   └── ...
├── backend/
```

```
│   ├── Dockerfile
│   └── ...
├── compose.yaml
├── .dockerignore
├── .gitignore
├── .env.example
├── README.md
└── DEPLOYMENT.md
```

The exact structure may differ by framework. Not every application requires a frontend, backend, or database.

3. Containerization Requirements

Student Project Lab uses Docker to host applications.

Your application must be deployable using Docker Compose unless the department explicitly approves another deployment method.

Dockerfile

Each custom application image must have a working Dockerfile.

Your Dockerfile should:

- Use an appropriate maintained base image.
- Install the dependencies required by the application.
- Configure a suitable production startup command.
- Exclude unnecessary build tools and development dependencies from the final image where practical.
- Run the application as a non-root user wherever feasible.
- Avoid including passwords, API keys, or other secrets.
- Build without requiring interactive user input.

Docker Compose

Your repository must include a `compose.yaml` file defining the services required to operate your application.

For a basic web application listening on port 8000, an example is:

```
services:
  web:
    build:
      context: .
      dockerfile: Dockerfile

    restart: unless-stopped

    environment:
      APP_ENV: production
      SECRET_KEY: ${SECRET_KEY:?Required}

    expose:
      - "8000"

    healthcheck:
      test:
        - CMD
        - python
        - -c
        - "import urllib.request; urllib.request.urlopen('http://127.0.0.1:8000/health',
timeout=3)"
      interval: 30s
      timeout: 5s
      retries: 3
      start_period: 30s
```

This example assumes a Python application with a `/health` endpoint. Adjust the health check and startup behavior for your framework.

Your application must listen on the configured port and bind to `0.0.0.0`, not only `127.0.0.1`.

Coolify handles public HTTPS and domain routing. Do not configure a separate public-facing reverse proxy unless it is required and explicitly approved.

Multiple services

Applications may use supporting services, such as:

- PostgreSQL or MariaDB.
- Redis.

- Background workers.
- Other approved components.

Internal services must communicate over private container networks. Databases and other internal services must not expose public host ports.

Use persistent Docker volumes for data that must survive container replacement.

Local testing

Before submission, confirm that your application runs using Docker Compose with a local environment configuration:

```
docker compose config
docker compose up --build -d
docker compose ps
```

Inspect application logs:

```
docker compose logs --tail=100
```

Stop the application when finished:

```
docker compose down
```

Do not use `docker compose down -v` if you need to preserve local database or application data.

A locally successful deployment does not guarantee approval, but applications that cannot build and run locally will not be accepted for review.

4. Environment Variables and Secrets

Applications must support production configuration without modifying source code.

Use environment variables or another approved secrets mechanism for values such as:

- Database credentials.
- Application encryption keys.
- Session signing secrets.

- Third-party API credentials.
- SMTP passwords.
- Production configuration settings.

Include an `.env.example` file showing the required variable names.

Example:

```
APP_ENV=production
SECRET_KEY=
DATABASE_URL=
```

Never commit a real `.env` file containing credentials.

The department will configure approved production secrets using the hosting platform.

Do not include production passwords or other secrets in documentation, screenshots, review requests, or GitHub issues.

5. Application Requirements

Your application must:

- Start without interactive input.
- Use a suitable production server rather than a development server.
- Run with debugging disabled.
- Provide a reliable HTTP health-check endpoint.
- Handle application restarts without manual intervention.
- Store persistent data outside replaceable application containers.
- Record operational errors without logging sensitive information.
- Use the externally configured production domain correctly.
- Work behind an HTTPS-terminating reverse proxy.
- Fail safely when required configuration is missing.

Applications that require special infrastructure, hardware, privileged containers, or unusually large resources must receive prior approval.

6. Required Documentation

Include a `README.md` and a `DEPLOYMENT.md` file.

README.md

Describe:

- Project name and purpose.
- Student contributors.
- Major technologies and frameworks.
- Principal application features.
- Local installation and development instructions.
- Build and test instructions.
- Any third-party software, assets, or services.

DEPLOYMENT.md

Describe:

1. Required containers and services.
2. Application ports and HTTP endpoints.
3. Required environment variables, without secret values.
4. Database setup and migration procedures.
5. Required persistent storage.
6. Startup and health-check behavior.
7. External API or network access requirements.
8. Initial configuration or administrative setup.
9. Backup and restoration considerations.
10. Known limitations and deployment risks.

A reviewer or administrator should be able to understand how the application runs without contacting you for basic setup instructions.

7. Preparing a Release Candidate

Publication is based on a specific Git commit rather than whichever code happens to be in your repository.

Before requesting review:

1. Commit all changes.
2. Push the intended version to your remote repository.
3. Verify tests and local deployment.
4. Create an annotated release-candidate tag.

For your first release:

```
git tag -a v1.0.0-rc.1 -m "Publication review candidate"
git push origin v1.0.0-rc.1
```

Find the associated commit:

```
git rev-parse v1.0.0-rc.1^{commit}
```

Submit the full commit SHA and tag in your review request.

Do not move or replace a submitted tag. If changes are required, create a new commit and a new candidate tag, such as `v1.0.0-rc.2`.

Do not assume that pushing a tag will deploy the application. Only approved releases are authorized for production.

8. Publication and Updates

After a release passes security review, the department may approve the commit as a production release, such as `v1.0.0`, and publish it at an assigned Student Project Lab subdomain.

Example:

`inventory.studentprojectlab.com`

Updates to published applications require a new release candidate and approval. The level of review may depend on the changes being introduced.

Minor documentation updates may need less review than changes to authentication, database handling, file uploads, or application dependencies.

The department may suspend, remove, or decline an application if it introduces security, legal, operational, or resource concerns.

9. Hosting Responsibilities

Students are responsible for the correctness, security, and maintainability of their application code.

The department manages the production infrastructure, domains, HTTPS, deployment authorization, and approved hosting configurations.

Students must not assume indefinite hosting. Applications may be retired after the approved hosting period, and extensions require departmental authorization.

Students should maintain independent copies of their source code and any project data they are authorized to retain.

10. Publication Checklist

Before requesting a security review, confirm:

- My repository is accessible to reviewers.
- The intended version is committed and tagged.
- My Dockerfile builds successfully.
- My Docker Compose configuration starts the application.
- My application can run with production-style settings.
- All required configuration variables are documented.
- No credentials or sensitive information are committed.
- Databases and internal services are not publicly exposed.
- The application has a functioning health check.
- Persistent data uses appropriate storage.
- My README and deployment instructions are complete.
- I have reviewed the Application Security Requirements.
- I have tested my application and documented known issues.

Next: Read the Application Security Requirements, then follow the procedure in Requesting Security Review and Publication.

Student Project Lab: Application Security Requirements

Purpose

All applications published through Student Project Lab are publicly accessible software running on shared department-managed infrastructure.

To protect users, project developers, other hosted applications, and the hosting environment, every application must satisfy the security requirements described here.

All applications must pass security review before publication.

The requirements apply to initial releases and subsequent updates. Approval of one release does not automatically authorize future releases.

These requirements are informed by the OWASP Application Security Verification Standard (ASVS), with a baseline adapted to the risks and resources of student-developed applications. Passing the Student Project Lab review does not constitute OWASP certification or guarantee that an application is free of vulnerabilities.

1. Secure Development

Applications must follow recognized secure software development practices.

Requirements

- Use maintained frameworks, libraries, and dependencies.
- Correct known vulnerabilities that present unacceptable risk before publication.
- Use established, maintained security libraries instead of implementing custom cryptographic algorithms or authentication mechanisms without approval.

- Validate untrusted inputs on the server.
- Use parameterized database queries or a secure ORM to prevent SQL injection.
- Apply context-appropriate output encoding to prevent cross-site scripting.
- Protect state-changing requests against cross-site request forgery where applicable.
- Avoid insecure deserialization and unsafe execution of user-controlled input.
- Handle errors without exposing internal implementation details.

Students must disclose unresolved security findings and known vulnerabilities during review.

2. Authentication and Authorization

Applications with user accounts must implement appropriate authentication and authorization controls.

Requirements

- Passwords must never be stored in plaintext or using fast, general-purpose hashes.
- Password authentication must use an established password-hashing mechanism supported by the framework, such as Argon2id or an appropriately configured alternative.
- Authentication credentials must be transmitted securely.
- Sessions must use appropriately configured secure cookies or another reviewed mechanism.
- Authentication endpoints must have appropriate protections against brute-force attacks.
- Sensitive operations must require authentication.
- Authorization must be enforced on the server for every protected operation.
- Users must not be able to access another user's private information merely by changing identifiers or URLs.
- Administrative functionality must be restricted to authorized administrators.
- Default passwords and demonstration credentials must not remain active in production.

Students must test access control using accounts with different permission levels.

Applications using third-party identity providers must document the integration and its security assumptions.

3. Secrets and Sensitive Information

Secrets must never be embedded in source code, container images, or repository history.

This includes:

- Passwords.
- Database connection credentials.
- API keys.
- Authentication tokens.
- Application encryption and signing keys.
- Private cryptographic keys.

Use the deployment platform's approved secret configuration.

Applications must not log passwords, session tokens, authorization headers, or other sensitive credentials.

Publicly hosted applications must not collect or store actual student records, protected educational information, medical information, financial account information, or other sensitive personal data without explicit prior authorization and an approved protection plan.

Synthetic or fictional demonstration data is preferred.

Do not populate demonstration databases with real information copied from university systems, workplaces, or other individuals.

4. Production Configuration

Applications must be configured for production operation.

Requirements

- Debug mode must be disabled.
- Detailed stack traces must not be exposed to visitors.
- Development-only endpoints and debugging tools must be disabled.
- Unnecessary services and ports must not be publicly accessible.
- HTTPS must be used for public access.
- Session cookies must use appropriate security attributes.

- Cross-origin requests must be restricted to approved origins.
- Security-related HTTP headers must be configured appropriately.
- Upload limits, request limits, and resource constraints must be reasonable for the application.
- Configuration failures must not silently enable insecure defaults.

The hosting platform provides HTTPS termination. Applications must still correctly handle forwarded protocol information and securely configure authentication and sessions.

5. Container Security

Student applications share infrastructure with other hosted projects. Container security is therefore mandatory.

Prohibited configurations

The following are prohibited unless the department explicitly authorizes an exception:

- Privileged containers.
- Access to the Docker socket.
- Host filesystem mounts outside explicitly approved storage locations.
- Host networking.
- Access to host devices.
- Excessive Linux capabilities.
- Running unnecessary services as root.
- Publicly exposed database or cache ports.
- Container configurations intended to bypass platform restrictions.

Applications should use non-root container users wherever feasible.

Images must come from trusted sources and must not contain unnecessary software or embedded credentials.

The department may impose CPU, memory, disk, network, or process limits.

6. Files and Persistent Storage

Applications that accept file uploads must:

- Restrict allowed file types and sizes.
- Validate file content where appropriate, rather than trusting file extensions alone.

- Store uploaded files safely.
- Prevent directory traversal.
- Prevent uploaded files from executing as application code.
- Enforce authorization when accessing private files.
- Avoid exposing secrets or internal server files through download functionality.

Applications must not rely on the container's temporary filesystem for persistent application data.

Persistent storage must be declared and documented.

7. Database Security

Applications using databases must:

- Use separate, appropriately limited database credentials.
- Avoid using database administrator accounts for normal application requests.
- Avoid publishing database ports to the internet.
- Use parameterized queries or safe ORM operations.
- Protect sensitive records through appropriate authentication and authorization.
- Handle migrations and schema changes predictably.
- Document database initialization and restoration procedures.

Development and production databases must remain separate.

8. Third-Party Services and Network Access

All external dependencies must be documented.

Examples include payment services, mapping APIs, email delivery, external databases, cloud storage, and AI APIs.

Applications may only use external services for which the student or department has appropriate authorization.

The following activities require specific approval and may be prohibited on the shared platform:

- Network scanning or penetration-testing services.
- Automated vulnerability exploitation.
- Proxies or relays accessible to arbitrary internet users.

- Automated scraping or high-volume external requests.
- Cryptocurrency mining.
- Hosting malware or other potentially dangerous executable content.
- Unrestricted remote command execution.
- Hosting infrastructure that requires privileged operating-system access.

Some cybersecurity research projects may be legitimate academic work but unsuitable for shared public hosting. The department may require a separate isolated environment instead.

9. Security Testing

Students must perform reasonable security testing before requesting review.

At minimum:

- Run the application's automated tests.
- Perform dependency vulnerability scanning.
- Check for committed secrets.
- Review container and deployment configuration.
- Test authentication and authorization, where applicable.
- Test the application's input validation.
- Review error handling and logging.
- Confirm that debugging features are disabled.
- Verify that only intended HTTP services are externally accessible.

Recommended tools include:

- GitHub Dependabot or an equivalent dependency scanner.
- GitHub CodeQL or another suitable static analyzer.
- Gitleaks or another secret scanner.
- Trivy or another container vulnerability scanner.
- OWASP ZAP for application security testing.

Students may use alternative tools if they provide comparable coverage.

Testing must be limited to applications and systems the student owns or is explicitly authorized to assess.

10. Requirements for Higher-Risk Features

Some application features require additional review because they introduce greater security or privacy risk.

Students must identify these features in their publication request.

Examples include:

- Public account registration.
- User-uploaded files.
- User-generated HTML or executable content.
- Administrative dashboards.
- Payment processing or financial transactions.
- Third-party OAuth or SSO.
- Email sending or other externally triggered communication.
- Sensitive personal information.
- Public APIs allowing data modification.
- AI agents capable of using external tools or executing code.
- Integration with physical devices or external systems.
- Security testing, network analysis, or similar cybersecurity functionality.

The department may require additional testing, architectural changes, isolation, or external services before publication.

11. Vulnerability Severity and Approval

Security findings will be evaluated based on exploitability, impact, exposure, and the application's purpose.

The department may use CVSS alongside contextual risk analysis when classifying vulnerabilities.

Severity	Publication requirement
Critical	Must be resolved before publication
High	Must be resolved before publication
Medium	Must be resolved or receive documented risk acceptance
Low	May be accepted with documented remediation recommendations
Informational	Document where appropriate

Severity classifications are determined by the reviewer, not solely by automated scanning tools.

A finding marked as a false positive must be explained and accepted by the reviewer.

A successful automated security scan does not replace manual review.

12. Review Outcomes

Security reviews may result in:

Approved: The identified commit meets the publication requirements.

Approved with conditions: Publication is permitted only after specified conditions are satisfied and verified.

Changes required: The student must correct identified deficiencies and submit a new release candidate.

Rejected: The application is unsuitable for the shared hosting environment or presents unacceptable risk.

The department reserves the right to reject, suspend, or remove applications that violate these requirements.

13. Changes After Approval

Security approval is associated with an exact source-code revision and deployment configuration.

Any code change after approval may require a new review.

Changes involving authentication, permissions, dependencies, API behavior, database access, infrastructure configuration, or sensitive data handling will generally require renewed security assessment.

Students may not bypass this process by pushing new tags, altering deployment files, or attempting to access production directly.

14. Security Review Checklist

Students must complete the following checklist before requesting publication.

Source code

- Dependencies are reasonably current and supported.
- No known critical or high-risk vulnerabilities remain unresolved.
- Untrusted input is validated.
- Database queries are protected against injection.
- Application output is appropriately encoded.
- Error handling does not reveal sensitive information.

Authentication and authorization

- Passwords are securely hashed, if applicable.
- Session management is secure.
- Authorization is enforced server-side.
- Administrative functionality is protected.
- Default credentials are removed or disabled.

Secrets and privacy

- No credentials are committed to the repository.
- Production secrets are externally configured.
- Logs do not expose sensitive credentials.
- Demonstration data is synthetic or otherwise authorized.
- Any personal data collection has been disclosed.

Deployment

- Containers do not require privileged access.
- The application uses production settings.
- Debug features are disabled.
- Databases and internal services are not publicly exposed.
- Persistent storage is configured appropriately.
- Health checks and startup procedures work.
- Required external network access is documented.

Verification

- Automated tests pass.
- Dependency and secret scans have been performed.
- Security-relevant functionality has been tested.

- Known risks and limitations are disclosed.
- The exact release candidate has been prepared.

Security review is a publication requirement, not a guarantee of security. Students remain responsible for promptly reporting newly discovered vulnerabilities in their applications.

Student Project Lab: Requesting Security Review and Publication

Overview

Before an application can be published on Student Project Lab, the student or project team must submit a publication request and pass a security review.

The review establishes whether the proposed application meets the technical, security, and operational requirements for public hosting.

Publication requests must identify an exact version of the application.

No application may be published without departmental approval.

1. Before Requesting a Review

Complete the following tasks:

1. Finish the application version you intend to publish.
2. Push all relevant changes to your Git repository.
3. Verify that Docker Compose can build and run the project.
4. Complete the Student Project Lab security checklist.
5. Run applicable automated tests and security scans.
6. Update `README.md` and `DEPLOYMENT.md`.
7. Create a release-candidate Git tag.
8. Record the full commit SHA.

For example:

```
git tag -a v1.0.0-rc.1 -m "Publication candidate"
git push origin v1.0.0-rc.1
```

```
git rev-parse v1.0.0-rc.1^{commit}
```

If the repository is private, confirm that the designated reviewers have access.

2. Submitting a Publication Request

Submit your request through the department-designated **Student Project Lab publication request board**.

The department will provide the submission link.

The request must include:

Project information

- Project name.
- Student name or team members.
- Course or project affiliation, if applicable.
- Brief description and intended audience.
- Repository URL.
- Requested project subdomain.

Release information

- Release-candidate tag.
- Full Git commit SHA.
- Link to the repository documentation.
- Build and startup instructions.
- Confirmation that the submitted revision runs successfully.

Architecture and operations

- Programming language and framework.
- Required containers and services.
- Database technology, if applicable.
- Persistent storage requirements.
- Required environment variable names.

- External API and network dependencies.
- Estimated resource requirements.
- Initial setup or database migration instructions.

Do not include actual passwords or secret values.

Security information

- Confirmation that the security checklist has been completed.
- Summary of automated testing and scans performed.
- Known vulnerabilities or unresolved issues.
- Authentication and authorization approach.
- Description of any stored personal information.
- Disclosure of higher-risk application features.
- Description of any unusual container privileges, networking, or filesystem access.

Demonstration access

If the application requires authentication, explain how reviewers can test it.

Provide temporary demonstration credentials through a department-approved private communication channel, not a public GitHub issue.

3. Review Process

After submission, the request follows this process:

Initial screening: The department checks whether the submission is complete and suitable for the hosting environment.

Technical review: The reviewer verifies that the application can build, start, and operate using the documented deployment configuration.

Security review: The reviewer examines the applicable security requirements, including deployment settings, dependencies, authentication, authorization, and other relevant controls.

Review decision: The reviewer approves the release, requests changes, or rejects the publication request.

Deployment: An authorized department administrator deploys the approved revision.

Publication verification: The deployed application is checked for basic functionality, correct routing, HTTPS, and operational health.

Reviews may involve source-code inspection, automated scanning, application testing, and requests for clarification.

4. Responding to Review Findings

If changes are required, the reviewer will describe the findings and expected corrections.

Students must:

1. Correct the identified problems.
2. Run relevant tests again.
3. Commit and push the corrections.
4. Create a new release-candidate tag.
5. Update the publication request with the new tag and commit SHA.
6. Explain how the findings were addressed.

Example:

```
git tag -a v1.0.0-rc.2 -m "Security review corrections"
git push origin v1.0.0-rc.2
```

Do not overwrite or move the original release-candidate tag.

The reviewer will examine the revised version and determine whether additional changes are necessary.

5. Approval and Publication

Once the review is approved, the department records the approved commit and authorizes a production release.

For example:

Item	Example
Candidate	`v1.0.0-rc.2`
Approved release	`v1.0.0`
Repository	`student/project-name`

Deployment	`project-name.studentprojectlab.com`

An authorized department process selects the approved Git commit or immutable build artifact for deployment.

Students do not need production server credentials or direct access to Coolify.

After publication, the department verifies the application and provides its public URL.

6. Updating a Published Application

To publish an updated version:

1. Complete the updated code.
2. Test the changes.
3. Create a new release-candidate tag, such as `v1.1.0-rc.1`.
4. Submit an update request referencing the existing application.
5. Describe the changes and their security implications.
6. Complete any required review.
7. Wait for the department to deploy the approved release.

The department may use a reduced review process for low-risk changes, but all production updates require authorization.

If the update changes the database schema, the student must document migration requirements and potential rollback limitations.

7. Problems After Publication

Students must report discovered security vulnerabilities, exposed credentials, data loss, or serious application failures promptly through the designated private faculty or administrator contact.

Do not post active credentials, sensitive logs, vulnerability exploit details, or private user information in a public GitHub issue.

The department may temporarily disable the application while a problem is investigated.

Routine code corrections must follow the approved update procedure.

8. Publication Request Template

Copy this information into the designated publication request form if a structured form is not yet available.

Project

Project name:

Student(s):

Course or affiliation:

Project description:

Repository URL:

Requested subdomain:

Release

Release candidate tag:

Full commit SHA:

README URL:

Deployment instructions URL:

Tests completed and results:

Deployment

Framework/technology stack:

Required containers:

Application's internal HTTP port:

Database and storage requirements:

Environment variable names:

External service dependencies:

Estimated resource requirements:

Special initialization or migration instructions:

Security

Authentication and authorization approach:

Type of data stored:

Higher-risk features:

Security scans performed and results:

Known vulnerabilities or limitations:

Special permissions or infrastructure requirements:

Student Certification

- The submitted release represents the exact code intended for publication.
- The application runs successfully using the documented configuration.
- I have read and followed the Student Project Lab security requirements.
- I have disclosed known security issues and relevant operational risks.
- I have not committed production credentials or secrets.
- I understand that changes to an approved release require additional authorization.

9. After Your Application Is Published

Once your application is live, you may include its public URL in your portfolio, resume, or project documentation.

Keep the corresponding source-code repository available, subject to any course, research, or intellectual-property restrictions.

Students should monitor project issues and remain available to address significant security or operational problems during the approved hosting period.

Publication on Student Project Lab indicates that a specific version was accepted for hosting. It does not imply that the department endorses every statement, feature, or use of the application.