Security In Software Engineering

M:SI 2022 / 2023

Secure Blog Hosting

David Pontes | up202209310@fc.up.pt

Lourenço Antunes | up202203893@fc.up.pt

Luís Oliveira | up201108115@fc.up.pt

arc42 Template

January 2023

About arc42

arc42, the template for documentation of software and system architecture.

Template Version 8.2 EN, January 2023

Introduction and Goals

SecureBlog is a blogging platform that allows developers to advertise their *SecDevOps* services and report their security analysis achievements to the world.

In addition, SecureBlog is also a case-study on *Secure Software Development* and should be a secure, upholding the company's good reputation.

This document is a lighter version of the arc42 template for software architecture and will not implement all recommendations from the template. The decision to produce a software architecture aligns with the project's goals, as software design and documentation are key factors for decision making, communication and correctness.

Requirements Overview

Essential features

The following is an outline of the blog's key features:

- Personal content management system for users to manage blog posts.
- Support user interaction via comments on blog posts.
- Rigorous access control policies.
- Demonstrable security and resilience against potential attacks from malicious users or client-side agents.

The blogging platform shall be able to store multiple blog posts for each author and should provide authors with the ability to easily manage their content.

In addition, the solution shall uphold good secure coding practices and shall be resilient against attempted exploits.

Requirements

The following tables represent the Functional Requirements for SecureBlog, split into different feature groups.

As this project's implementation is a case-study for *Secure Software Development*, security requirements will be considered functional requirements of the solution.

The security requirements presented in RF.2 – Security Requirements are further refined and modelled in our Secure Testing Guide, provided as a separate project deliverable.

RF.1 – Content Management

ID	REQUIREMENT	DESCRIPTION
A	Personal Content Management Service	The system shall provide users with a personal content management service that allows them to manage their blog posts and comments.
B	Blog Post Management	The system shall allow users to create, edit and delete their own blog posts
B.1	Create Blog Post	The system shall provide input screens for creating blog posts.
B.1.1.	Blog post structure	The system shall support blog posts according to the following structure: Title – A title for the blog post Slug – A shorthand name for site navigation Abstract – A summary description of the blog post Content – The content of the blog post in markdown format Categories – An association of blog categories for filtering Date of creation – The date the blog post was first created. Author – A reference to the user who authored the blog post. Status – The visibility status of the blog post.
B.2.	Edit Blog Post	The system shall provide input screens for editing existing blog

		posts. The system shall not allow users to modify a blogpost's author or date of creation.
B.2.1.	Manage Blog Post visibility	The system shall allow users to change the blog post status between the following states: 1. Draft – Only visible to the author 2. Published – Publicly visible
B.3.	Delete Blog Post	The system shall provide input screens for removing blog posts.
C	Comments on Blog Posts	The system shall support both the adding and moderation of comments on blog posts
C.1.	Add comments	The system shall allow site visitors and authenticated users to comment on blog posts.
C.2.	Content moderation	Authors shall have the ability to moderate comments before they are publicly displayed (Moderation)
C.2.1	Comment visibility	Authors shall have the ability to approve or reject comments to their blog posts.
C.2.2	Add comments	The system shall provide site users with input screens for adding new comments. Both authenticated and anonymous site visitors shall be able to comment on blog posts.
C2.3	Remove comments	Authors shall have the ability to remove comments on their blog posts.

RF.2 – Security Requirements

ID	REQUIREMENT	DESCRIPTION
A	Secure communication	The system shall engage in secure communication with external entities and service providers.
A.1.	HTTPS	All communication between web clients and the web application should be secure and enforce HTTPS.
B	Secure data storage	The system shall protect user data and apply security mechanisms to prevent unauthorized access to user data.
C	Content security	The system shall implement a set of measures to ensure all content is properly sanitized and correctly encoded.
C.1.	Input validation	The system shall implement proper input validation to prevent against injection attacks.
C.2.	Anti-spam	The system shall implement Captcha or anti-spam measures to prevent automated comment spam.
C.3.	XSS injection	The system shall implement measures to prevent, or otherwise mitigate, the occurrence of XSS vulnerabilities.
D	Dependencies	SecureBlog shall not depend on highly vulnerable, deprecated or otherwise unmaintained third-party libraries or services
E	Secure authentication	The system shall perform user authentication through secure mechanisms, including secure and tested code.
F	Logging and Error handling	The system shall ensure that logs and error messages do not convey sensitive system or user information.
G	Security Updates and Patch Management	The system shall be built such that it is possible to continuously upgrade the solution to the most recent versions of frameworks and libraries.
H	Credential Management	The system shall strive to secure and safely manage user credentials.
H.1.	Strong Password Policy	The system shall enforce password policies. Policies shall enforce a minimum length, usage of special characters and combination of alphanumeric characters
H.2.	Secure Password storing	The system shall store passwords using secure mechanisms, e.g., encryption or hashing with a salt.

RF.3 – Access control

ID	REQUIREMENT	DESCRIPTION
A	Authorisation groups	Users shall be assigned to groups in accordance with their organizational function and intended use of the blog.
A.1.	Group assignment	Users shall be assigned groups on account registration.
A.2.	System groups	The system shall provide the following authorisation groups: - Reader: Identifies an authenticated blog user - Author: Identifies a staff member that writes blog posts - Moderator: Identifies users with moderation capabilities on all blog posts.
B	Authentication	The system shall provide input screens for user authentication.
B.1.	Authentication method	Users shall authenticate to the application using their username and password.
B.2.	Password policy	The system shall enforce a password policy.
C	Content Management	The system shall only allow authorized users' access to the content management features.
C.1.	Access to Content Management	The system shall ensure users are authenticated and properly authorized before granting them access to the personal content management service.
C.2.	Content Management Policies	Authorized users shall be able to access their personal content management service, to manage their blog posts and associated comments.
C.3.	Authentication to Content Management	The system shall provide a dedicated authentication form for accessing Content Management features
C.4.	Public content	The system shall ensure only *published* blogposts are publicly available. The system shall ensure all static resources required for the

		appropriate rendering of the web application are made available to web clients.
D	Authorisation	Access to specific features or data within the system should be restricted based on the user's role and permissions.
E	Account creation	Site visitors shall be able to create an account.
E.1.	User registration	On account registration, users shall provide the following details: - Username: User alias in the blog application - Email: User email Password: Authentication credential

Quality Goals

The following table describes SecureBlog's key quality objectives. Each is ranked to provide a rough estimate of their importance.

These are presented in accordance with **ISO 25010** - *System and software quality models*, a standard that provides consistent terminology for specifying, measuring, and evaluating system and software.

Priority	**ISO 25010**	**Quality Goal**	**Scenario / Description**
1	Security	Integrity	System components should enforce restrict authorization mechanisms to prevent unauthorized access and modification of data.
2	Functional Suitability	Understandability	The functional requirements should be presented clearly to allow the development of a simple, understandable solution while focusing on the security aspects of the development process.
3	Maintainability	Analysability	SecureBlog is a case-study on Secure Software Development and so it should be presented such that other students are able to quickly understand the application's design and implementation.
3	Maintainability and Security	Testability	Both the architecture and the code should be easy to test of all components.
5	Usability	Operability	System should be easy for users to use / operate.

Architecture Constraints

The few constraints for this project are reflected in the final solution. This section provides a clear reasoning for each constraint.

Technical constraints

	Constraint	**Motivation**
Software and programming constraints		
TC1	Use of the Django Framework	The project shall be implemented using Django
TC2	Third-party software	Third-party software must be available under a compatible open-source license and installable via pip. Used third-party software must be actively maintained and supported by Django.
TC3	Database access	Database queries and other operations shall be performed using Django's ORM and QuerySet API. Developers shall not create queries manually nor access the database directly, e.g. using raw invocations, or using any other external mechanism.
Deployment constraints		
TC3	Deployable to a Linux server	The system should be deployable to Linux based servers
TC4	Containerization	The system should be deployable in containerized form (e.g., using Docker /Compose)

Organisational constraints

	Constraint	Motivation
OC1	Team	The development team is predefined as the workgroup for this class
OC2	Requirements	Requirements shall adhere to the class project requirements and only minor, scarce, additions will be made.
OC3	Goals	The project's goals are set by the University staff and the team shall adhere to these during this project.
OC4	Security objectives	The project shall focus on the security aspects of the software development cycle.

Conventions

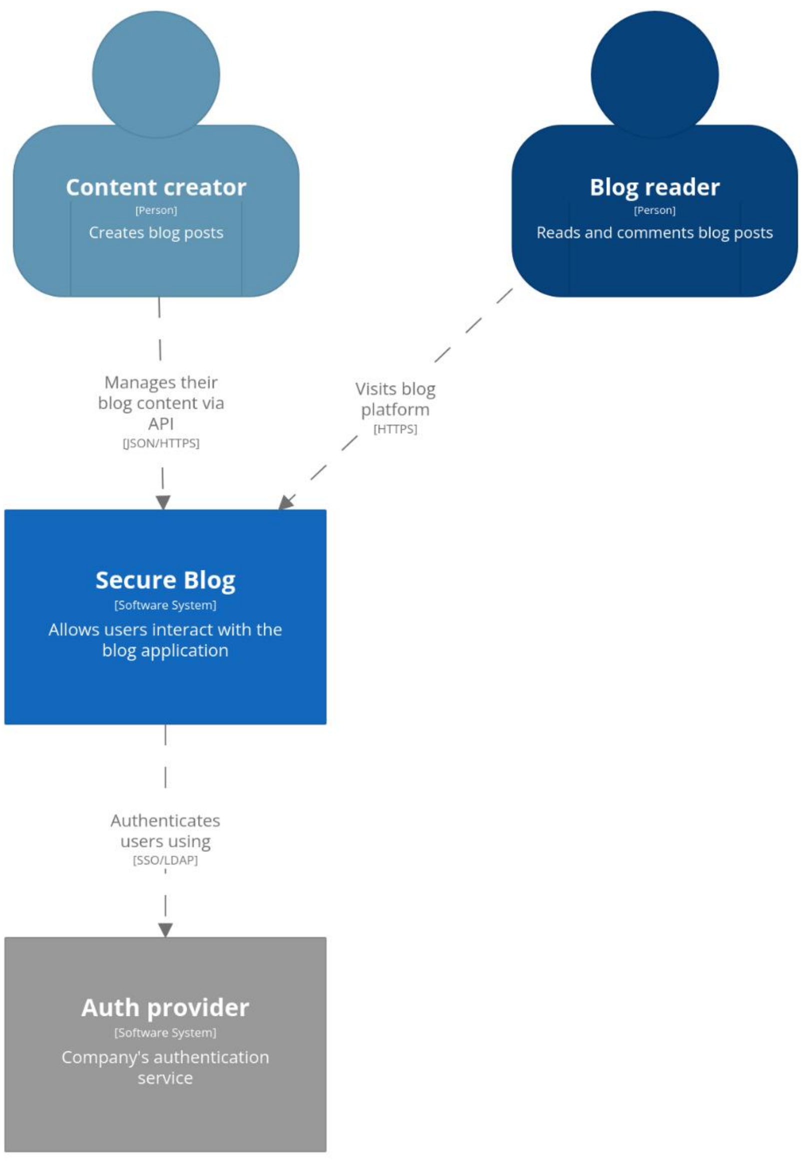
	Convention	Motivation
C1	Modelling	Software modelling for this architecture is done using the C4 Model for visualising software architecture.
C2	Architecture	Software architecture based on a lightweight version of the arc42 template.

System Scope and Context

This section describes the environment of the SecureBlog application, who its users are, and with which other systems it interacts with.

Presented diagrams are accompanied by a description table, describing each component of the diagram in further detail. Diagrams and software views were made using the C4 model for visualising software architecture domain specific language (DSL).

Business Context



Content creator
[Person]
Creates blog posts
Blog reader
[Person]
Reads and comments blog posts
Manages their blog content via API
[JSON/HTTPS]
Visits blog platform
[HTTPS]
Secure Blog
[Software System]
Allows users interact with the blog application
Authenticates users using
[SSO/LDAP]
Auth provider
[Software System]
Company's authentication service

[System Context] Secure Blog

The system context diagram for the Secure Blog Hosting.
Wednesday, 24 May 2023 at 11:23 British Summer Time

Type	**Name**	**Description**
Person	Content creator	A member of the company’s staff that creates blog entries
Person	Reader	A site visitor that can read and comment on blog posts
System	Secure Blog	The SecureBlog application
System	Auth Provider	The company’s centralized authentication services

Additional system decomposition is provided in Building blocks view.

Solution Strategy

This section contains an overview of the system's architecture, describing the most important goals, decisions, and the overall approach to the solution.

Overview of Quality Goals

The following table contrasts the quality goals for SecureBlog, described in Quality Goals with the matching architecture approaches.

Quality Goal	**Matching approach in the solution**
Integrity	System components should enforce restrict authorization mechanisms to prevent unauthorized access and modification of data.
Understandability	Clear and concise software architecture structured in arc42 template.
Testability	Both the architecture and the code should be easy to test of all components. Development and testing methodologies are provided in separate to this architecture.
Analysability	It should be possible to diagnose and assess the impact of systems failures, misconfigurations, and modifications.
Operability	System should be easy for users to use / operate.

Structure of SecureBlog

SecureBlog is implemented in **Django**, a highly tested, productive, *Pyhton* framework for web development. The decision to build the application with *Django* arose from the need to extensively test, document, and demonstrate the security qualities of the blog application. As such, implementing a *simple prototype from scratch*, as noted by the project's assignment, was deemed the most viable solution, as it allowed us to limit our exposure to large, existing, codebases, third-party libraries and potential misconfigurations arising from the complexity of other CMS frameworks. This upholds the *Minimize attack surface area principle*, in which every feature added to an application adds a certain amount of risk

Furthermore, a custom implementation also allowed us to focus on a particular set of features and tailor our dependencies and code to our specific needs.

The solution is, roughly, split into the following parts:

- An implementation of a Content Management System
- An administrative backend
- A blog web application accessible to browsers

As *Django* implements the **MVT** pattern (Model-View-Template), the blog web application, served to browser clients, is not a standalone frontend (or single-page application), but a combination of functionality provided by *Models* and *Templates.*

The CMS and the blogging application are to be developed as separate Django applications.

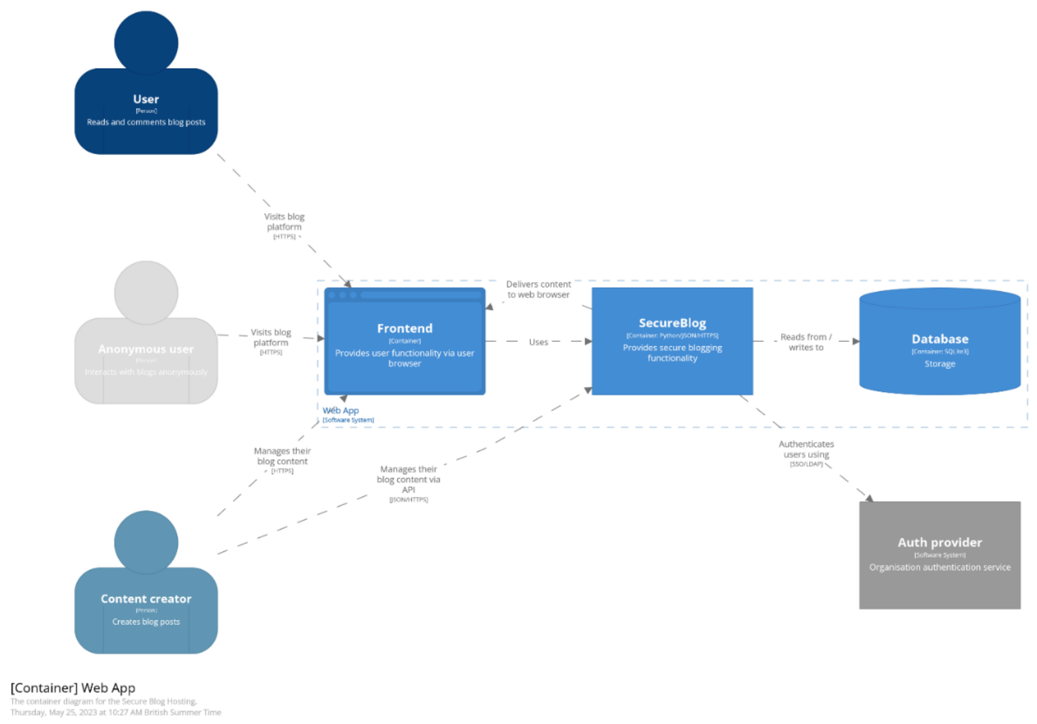
In the case of security requirements, some of the concerns lead us to stipulate strict code design rules, as documented in Technical Constraints - Database Access.

SecureBlog shall provide registration and authentication mechanisms for both company staff and blog visitors, who may want to register themselves with the blog. Registration or authentication is not mandatory for viewing or commenting on blog posts.

Building blocks view

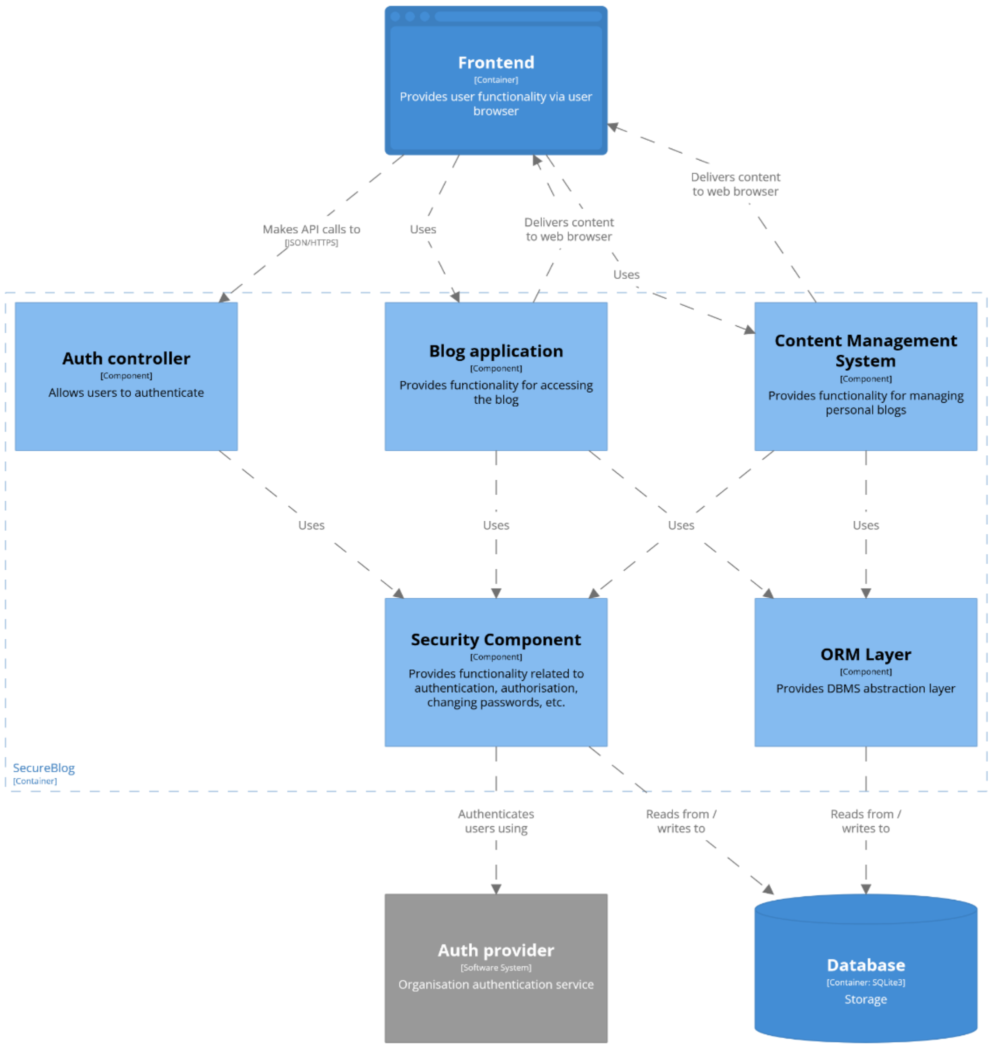
This section describes SecureBlog decomposition into containers and components. Level 1 is an overview of the web application, corresponding to a C4 Model - Container diagram. Level 2 details Components of the web application from Level 1.

Level 1


User
[Person]
Reads and comments blog posts
Visits blog platform
[HTTPS]
Anonymous user
[Person]
Interacts with blogs anonymously
Visits blog platform
[HTTPS]
Frontend
[Container]
Provides user functionality via user browser
Delivers content to web browser
Uses
SecureBlog
[Container: Python/JSON/HTTPS]
Provides secure blogging functionality
Reads from / writes to
Database
[Container: SQLite3]
Storage
Web App
[Software System]
Authenticates users using
[SSO/LDAP]
Auth provider
[Software System]
Organisation authentication service
Manages their blog content
[HTTPS]
Manages their blog content via API
[JSON/HTTPS]
Content creator
[Person]
Creates blog posts
[Container] Web App
The container diagram for the Secure Blog Hosting.
Thursday, May 25, 2023 at 10:27 AM British Summer Time

Type	Name	Description
Person	Content creator	A member of the company’s staff that creates blog entries
Person	Reader	A site visitor that can read and comment on blog posts
Person	Anonymous	An anonymous site reader
Container	Frontend	Provides user functionality via web browser
Container	SecureBlog	Backend application that provides secure blogging features.

Level 2


Frontend
[Container]
Provides user functionality via user browser
Makes API calls to
[JSON/HTTPS]
Uses
Delivers content to web browser
Uses
Delivers content to web browser
Auth controller
[Component]
Allows users to authenticate
Blog application
[Component]
Provides functionality for accessing the blog
Content Management System
[Component]
Provides functionality for managing personal blogs
Uses
Uses
Uses
Uses
Security Component
[Component]
Provides functionality related to authentication, authorisation, changing passwords, etc.
ORM Layer
[Component]
Provides DBMS abstraction layer
SecureBlog
[Container]
Authenticates users using
Reads from / writes to
Reads from / writes to
Auth provider
[Software System]
Organisation authentication service
Database
[Container: SQLite3]
Storage

[Component] Web App - SecureBlog

The component diagram for the API Application.

Thursday, May 25, 2023 at 10:27 AM British Summer Time

Type	**Name**	**Description**
Container	Frontend	Provides user functionality via web browser
Component	Auth Controller	Allow users authentication. Deals with administrative or user authentication requests and delegates to the security components for processing
Component	Blog Application	A Django App for user interaction with blog posts and comments
Component	Content Management System	A Django App for managing the blog's content. The personal Content Management System
Component	Security	Provides authentication, authorisation, and other security features
Component	ORM Layer	An abstraction layer for uniform interaction with the DBMS
Container	Database	SQLite3 DMBS that supports the application
System	Auth Provider	The company's centralized authentication services

Architecture Decisions

ADR.1 – Use Django CMS	
Context	Decision is to be made on whether to develop the application from scratch or use an existing CMS or blogging solution for Django.
Decision	Reject in favour of building the solution from scratch. Although initial tests resulted in a working blog / CMS application, setup is rather complex, and the plugin ecosystem also introduces an excessive level of complexity.
Status	Rejected
Consequences	Implement a simple prototype from scratch. Limit requirements scope and number of features to implement.
Considered alternatives	Other CMS and blogging frameworks, build from scratch using Django.

ADR.2 – Database Management System	
Context	A decision is to be made on which storage engine to use for the project, mainly for the development efforts, as future production environments will be out-of-scope for this implementation.
Decision	Use SQLite3 for ease of usage and native integration with Django.
Status	Approved
Consequences	
Considered alternatives	PostgreSQL SQLite3 MongoDB

ADR.3 – Spam prevention	
Context	Following initial testing, comments are perceived as a source of potential spam. Initial implementation of the feature did not sufficiently block users from submitting multiple comments prior to approval of previous submitted comments. Initial threat modelling also identified this to be a medium-risk threat, as detailed in the Security requirements of our Security Testing Guide.
Decision	Decision is to implement a captcha mechanism and keep track of pending comments approvals.
Status	Approved
Consequences	Captcha extension shall be chosen following the rules and constraints applied to dependencies. Application shall require additional endpoints for requesting captchas from the backend. Additional functional requirements related to the captcha feature. Overall improved security and control over spam attempts.
Considered alternatives	

ADR.4 – System configuration via .env file	
Context	A configuration mechanism is required to enable proper application setup and launch across different environments, including docker containers. Additionally, secrets, user data and credentials must not be kept in the source code. Proposal is to make use of a *.env* file for initial setup of the application, including passing credentials.
Decision	In accordance with best practices, such as the Twelve Factor App and given the need for a lightweight, easy to use logic for multiple purposes, usage of an environment file is deemed appropriate.
Status	Approved
Consequences	Django's SECRET_KEY is moved from settings.py to the *.env* file. *.env* file must be protected. Ensure it is not exposed to the web nor made publicly available. Application configurations, including super user definitions, database credentials can be dynamically adequate to the target environment. Both local application setup and docker setup will require a properly configured environment file.
Considered alternatives	-