SecureBlog

A Secure Blog Hosting solution

1

Our approach

Architecture + SSDLC + Testing methodology

“

Cybersecurity is like an Onion.

There's layers, and at some point you start to cry.

Architecture

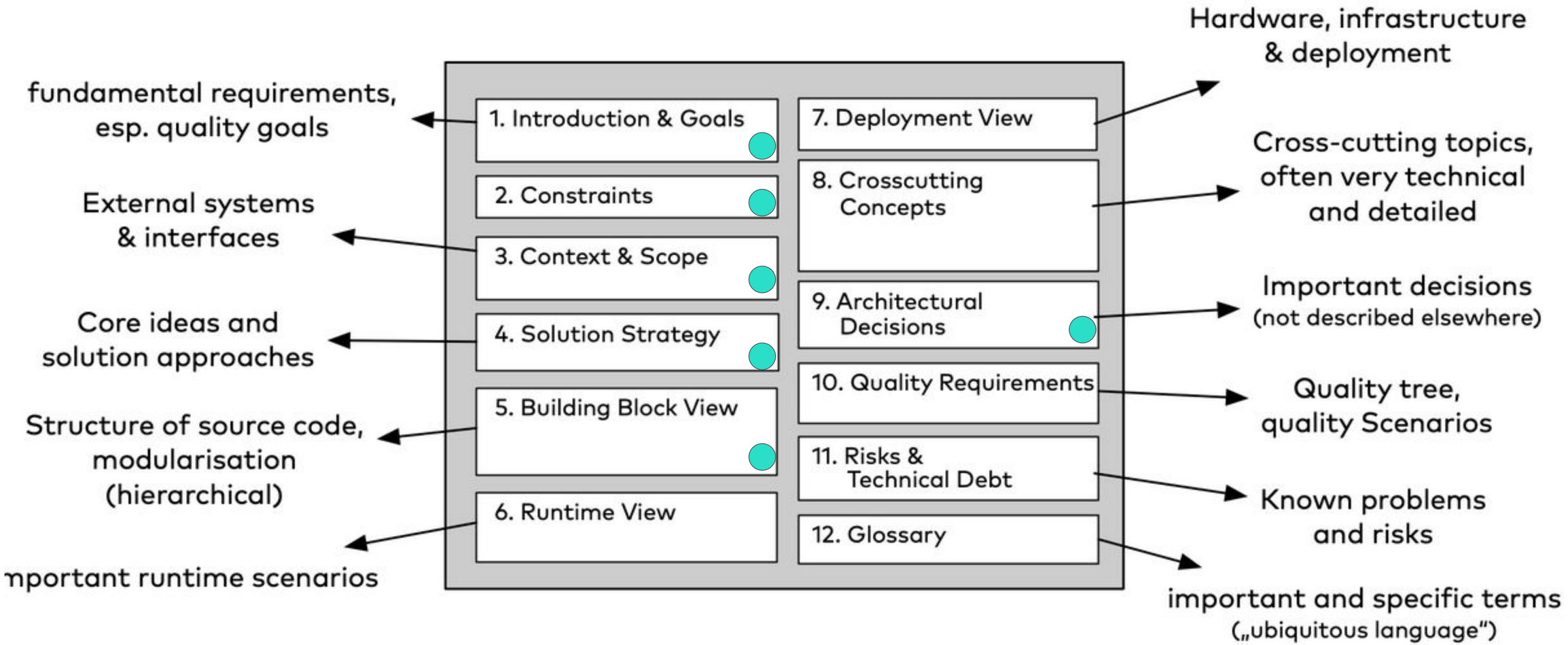
Motivation

- Clear, concise documentation to support implementation and testing activities
- Identify constraints, clear requirements, quality goals and design decisions
- Facilitate communication, decision making and limit scope

How

- Arc42 method and template
- C4 Model for visualising software architecture
- Architectural decision records

- *What* should you document/communicate about your architecture?
- *How* should you document/communicate?



Key Quality Goals

Integrity

Security is a major concern and proper authorisation should be implemented

Understandability

Requirements and design should be clear, understandable based on consensus.

Analysability

Design and implementation should be easy to analyse for deficiencies, flaws and changes.

Testability

Both the architecture and the code should be easy to test of all components.

Operability

System should be easy for users to use / operate.

Key feature overview

Personal CMS

Personal content management system for users to manage blog posts

User interaction

Support user interaction via comments on blog posts
Support content moderation

Access control

Rigorous access control policies and authorisation mechanisms

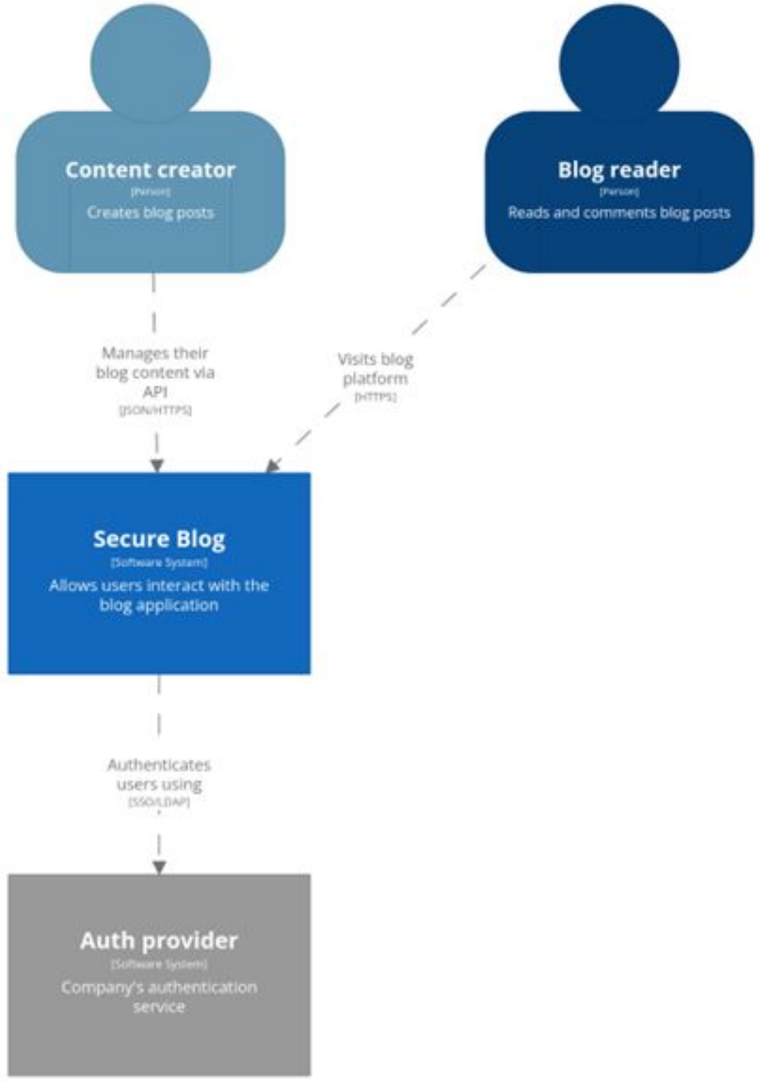
Testability

Demonstrable security and resilience against potential attacks from malicious users or client-side agents.

Business context

System boundaries, communication and external systems

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
Person	Content creator	A member of the company’s staff that creates blog entries



[System Context] Secure Blog

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

Key Architectural decisions

Django

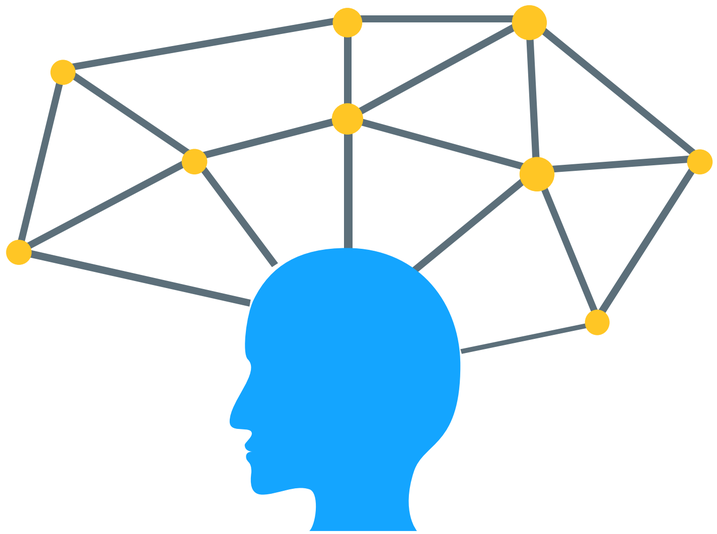
- Highly regarded as a productive web development framework
- Tailored for rapid development
- Built-in security mechanisms
- Extensive community support and extensions
- Lighter than the alternatives

Development constraints

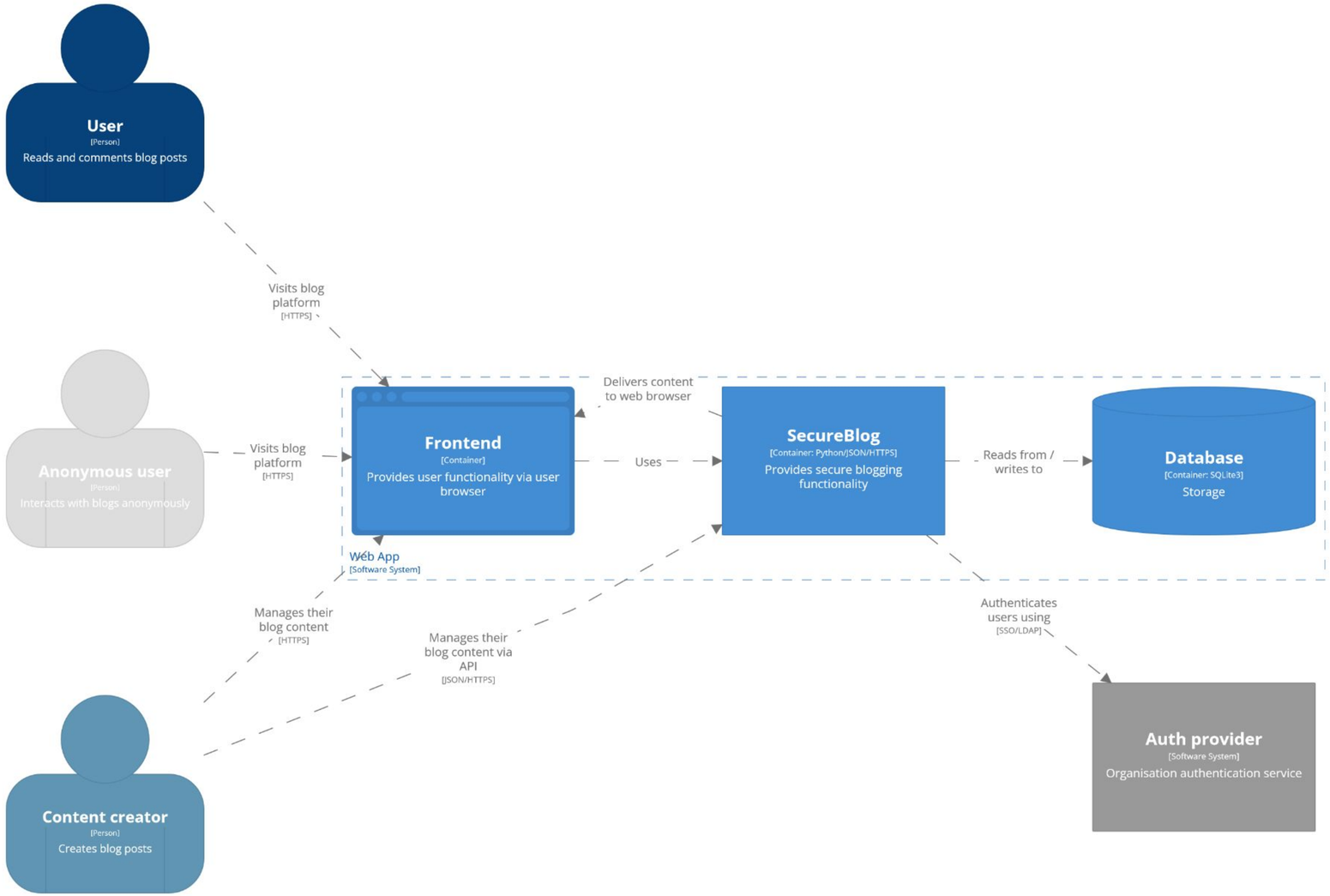
- Implement simple prototype from scratch
- Database access via ORM and built-in queryset mechanisms
- All third-party libraries and dependencies must be actively maintained in an open-source project
- Etc...

Additional requirements

- Anti-spam protection:
  - Implement captcha
- Password policy:
  - Enforce secure password policy on user registration

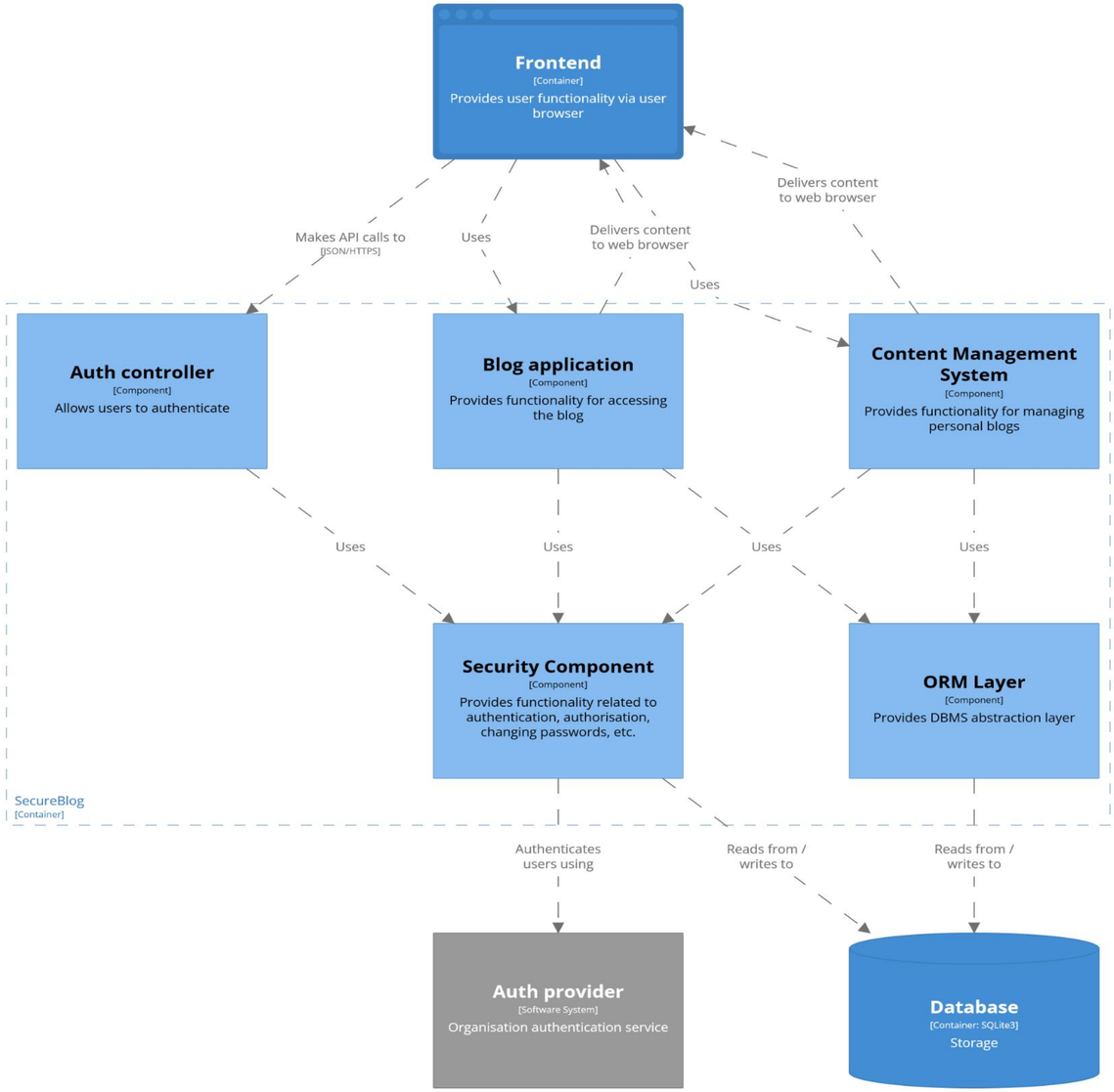


Building Blocks



[Container] Web App

The container diagram for the Secure Blog Hosting.
Thursday, May 25, 2023 at 10:27 AM British Summer Time



[Component] Web App - SecureBlog

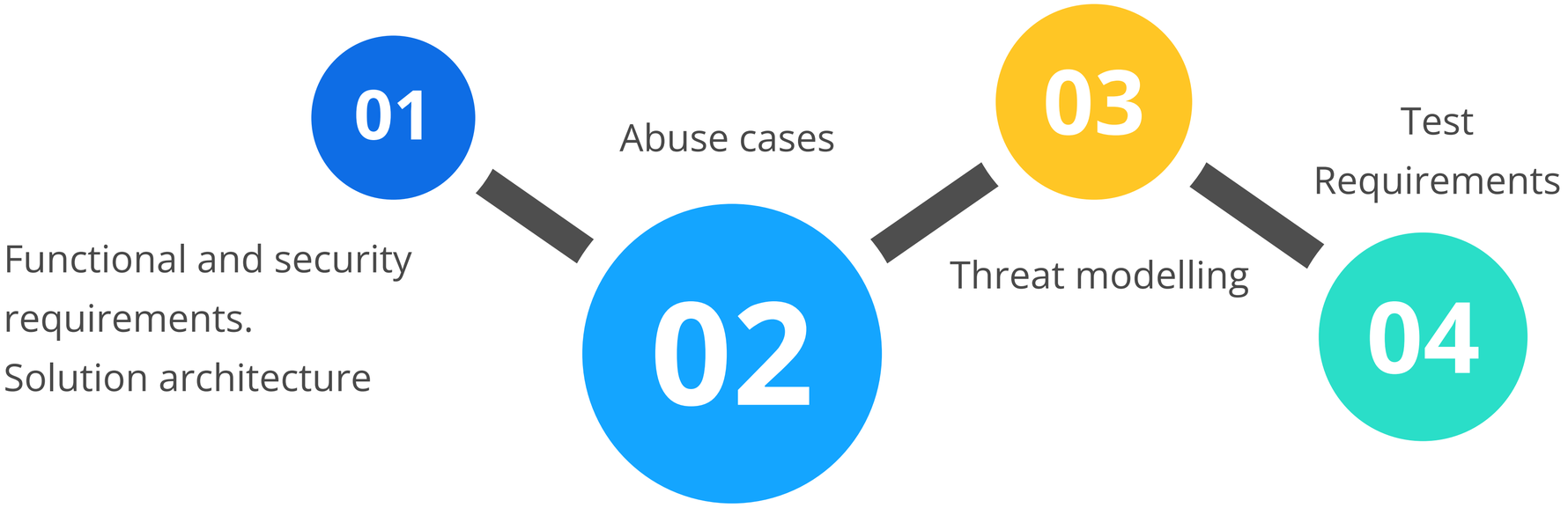
The component diagram for the API Application.

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

2

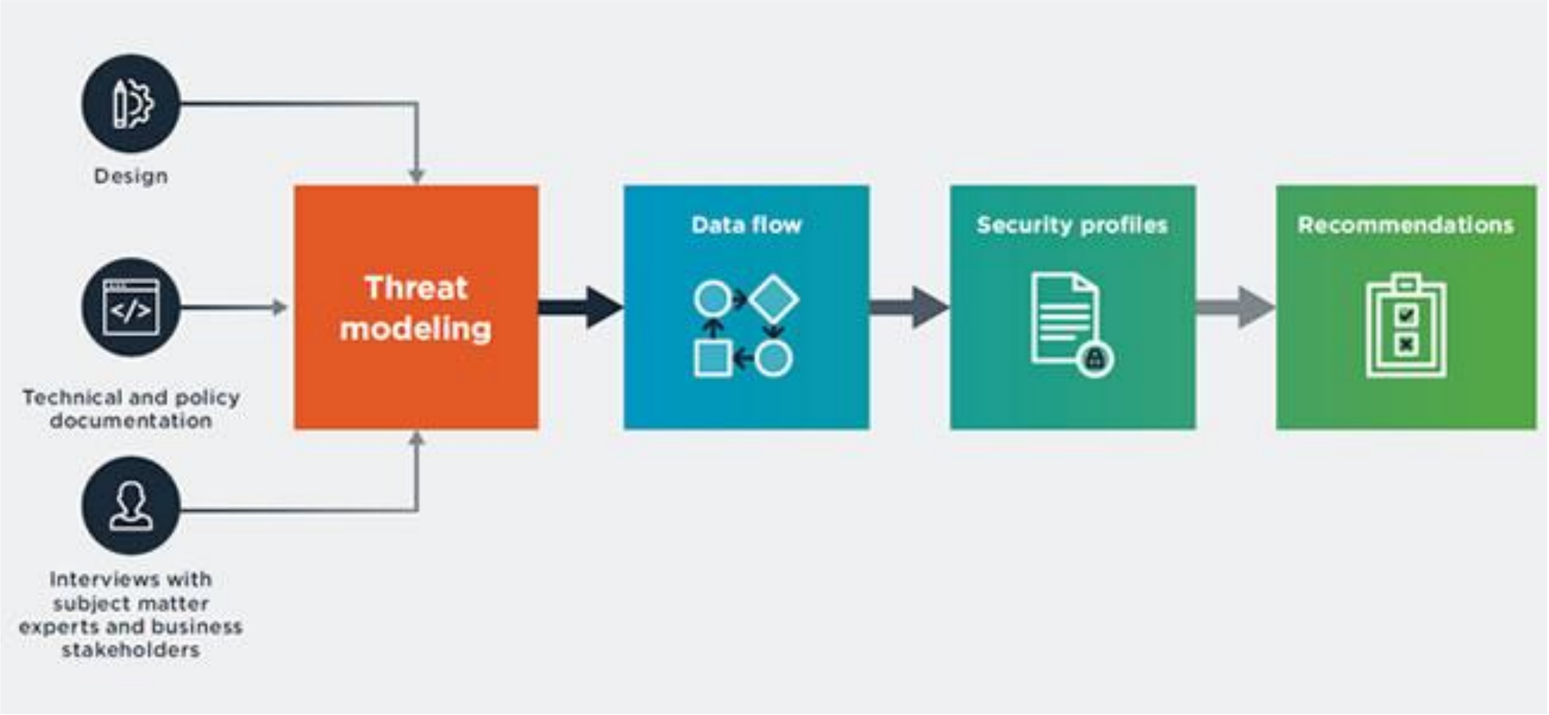
SSDLC

Initial modelling

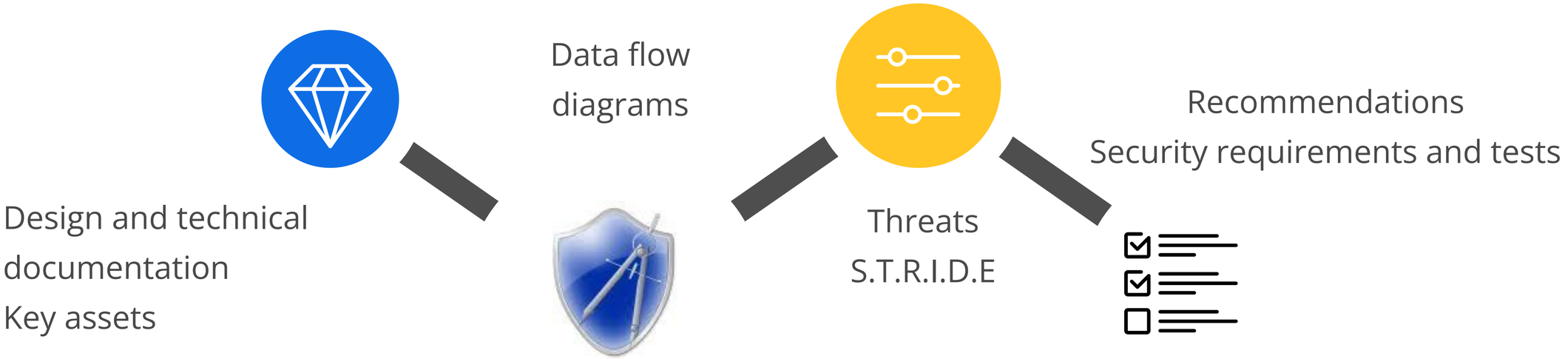


Functional requirements	Abuse case	Countermeasures
RF.1 - A Content management service (CMS)		
RF.1 - C Comments on Blog Posts	Adversaries may attempt Client-Side Injection attacks to introduce malicious content in the database	Use Django’s built-in ORM. Access database only using QuerySets and secure mechanisms provided by the development framework.
RF.1 - C.1 Add comments		

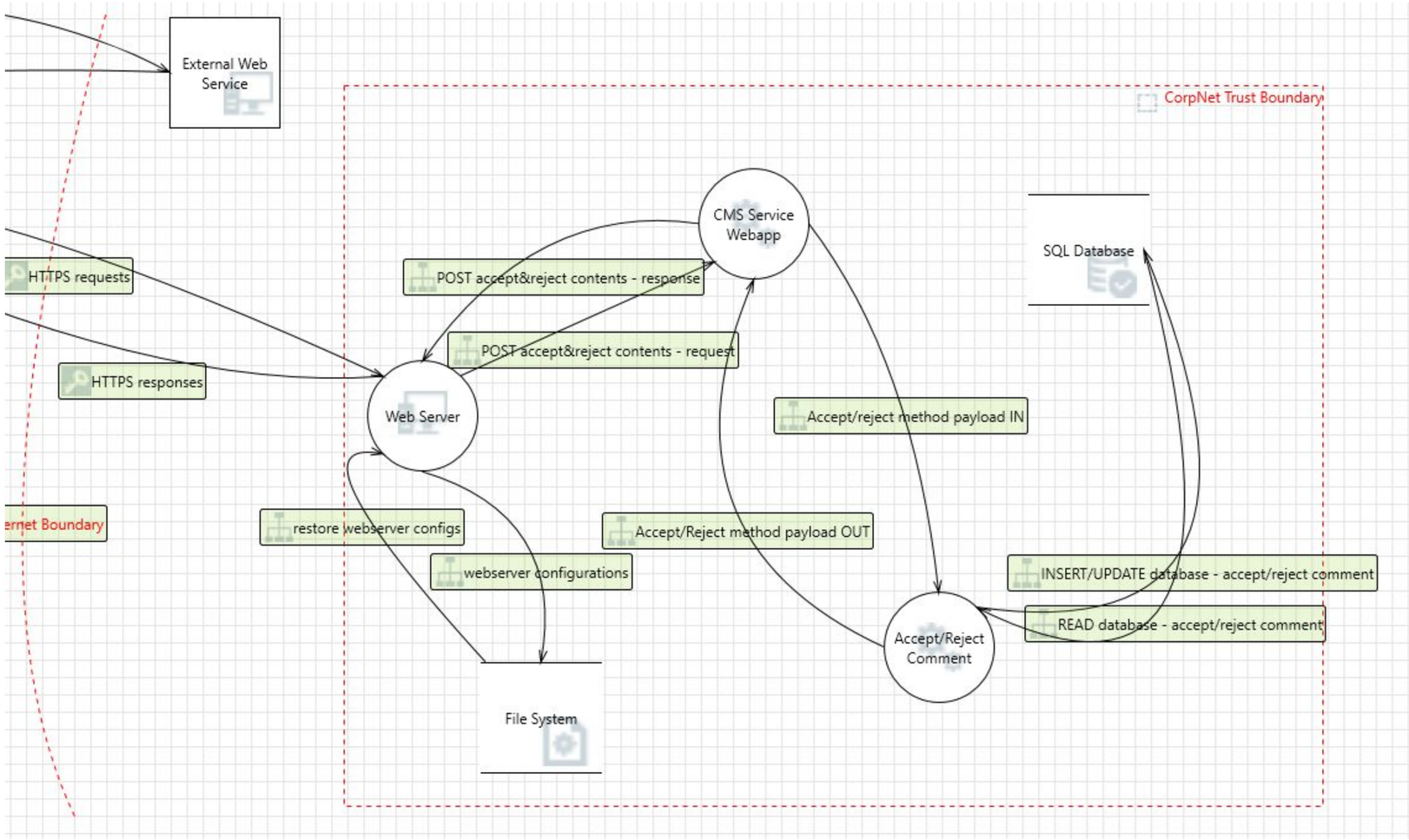
Threat modeling



Threat modelling



Threat modelling



Threat modelling

Title	Category	Interaction
Spoofing the Browser Client Process	Spoofing	HTTPS requests
Add/Remove Comment Post Process Memory Tampered	Tampering	POST add/remove comment (request/r
Spoofing of Destination Data Store File System	Spoofing	webserver configurations
Potential Excessive Resource Consumption for Web Server o	Denial Of Service	webserver configurations
Spoofing of Source Data Store File System	Spoofing	restore webserver configs
Spoofing of Destination Data Store Cookies	Spoofing	Saved cookies-sessions data
Spoofing of Source Data Store Cookies	Spoofing	cookies-sessions reading
Elevation by Changing the Execution Flow in Browser Client	Elevation Of Privilege	HTTPS responses
Potential Lack of Input Validation for Browser Client	Tampering	webapp output from browser
Weak Access Control for a Resource	Information Disclosure	READ database - accept/reject commen
Authorization Bypass	Information Disclosure	webserver configurations
Elevation Using Impersonation	Elevation Of Privilege	webapp output from browser

3

Testing

Security testing

Manual code reviews

Review and refactor code

Automated code inspections

SonarCloud, code-linters, python bandit

Dependency analysis

Python safety
Python bandit
Manually validate all django extensions according to architecture constraints

Test for common vulnerabilities

OWAS ZAP, SonarCloud
Manual tests

Pentesting

BurpSuite, OWASP ZAP
Manual tests

Finding details

Finding details

Replay Attack on Comment submissions

Severity: Critical

Location: Any blog post that currently accepting comments

Description: A replay attack is a form of network attack in which valid data transmission is maliciously or fraudulently repeated or delayed. Typically carried out either by the originator, or by an adversary who intercepts the data and re-transmits it, possibly as part of a spoofing attack.

The vulnerability derives from an implementation bug in the *view* responsible for handling new comment submissions. After successful submission, the server does not perform an internal redirection and, instead, serves new content in-place. This causes the browser to perform a resubmission on page refresh (POST on reload), which can lead users to post multiple comments unwillingly or an adversary to be able to spam comments, despite the measures already in place.

Recommendation: Perform code review to identify the issue and reimplement the portion of the code responsible for form processing and associated content moderation. The application should redirect the browser to *itself*, so that the browser refresh requests are not piggybacking on the previous operation. The following guide provides a detailed explanation on the solution: Prevent POST on reload.

Improper Neutralization of Special Elements

Severity: High

Location: Framework component

Description: A software flaw on an input escaping library/component was detected, which may lead to XSS vulnerabilities, if exploited. These were detected by code and dependency analysis and related to CWE-79: Improper Neutralization of Input During Web Page Generation and CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page.

Proof: Provided in relevant outputs - Bandit.

Recommendations: Perform code review to identify the issue and fix the software vulnerability. If fixing is possible, due to third party restrictions, recommendation is to upgrade to a safer library or a newer version of the component

Thank you!

Q&A