GovStack

Architecture and Nonfunctional Requirements

Developed by: Max Carlson, Kristo Vaher, Steve Conrad and Dr. Ramkumar Permachanahalli in cooperation with GIZ, ITU DIAL, and the Government of Estonia

Table of Contents

1 Version History

Version	Authors	Comment
1.0.0	Max Carlson, Steve Conrad	Initial revision
1.0.1	Max Carlson, Steve Conrad, Dr. Ramkumar, Trevor Kinsey from the Architecture Group	Added Key Decision Log
1.0.2	Architecture Team	Added GovStack System Architecture section
1..1.0	Max Carlson	Applied comments, added Future Considerations section

2 Description

This document is intended for building block working groups and developers. It provides guidelines and principles that should be considered by all building blocks. It also provides cross-cutting requirements that must be considered

GovStack aims to provide a reference architecture for digital governance software to support sustainable development goals.

This will accelerate the collaborative development of best-of-breed digital public goods, enhancing efficiency and transparency across the world - especially in low-resource settings.

Once an initial reference architecture is available, it can be deployed in a sandbox environment to demonstrate its utility.

2.1 Considerations for Low-Resource Settings

Low-resource settings have unique constraints that must be considered.

Here is a list of indicators with high-level descriptions:

Indicator	Description
ICT Governance	Poor or non-existent National ICT governance structure that makes decisions and ensures accountability framework to encourage desirable behavior in the use of ICT in the country. However, this may be described in documents but the implementation is suboptimal or not enforced.
Government ICT policy or Framework	No strategic policy framework for the acquisition and use of IT for social and economic growth in the country. The policy might be at the development stage and where the policy exists, the policy implementation is lagging or non-existent.
ICT infrastructure	The development of IT infrastructure in the country is lagging behind or sub optimal because of poor policies and insufficient investments in the ICT sector. Low coverage of power or the national grid and little penetration of alternative sources of energy especially in the rural.

Indicator	Description
Financial Resources and Investments in ICT	Limited funding for ICT projects and initiatives. ICT intervention may not be prioritized. No institutionalized or routinized support for ICT projects/ interventions by the government.
ICT projects/ Initiatives	ICT projects and intervention are implemented in a silo, none standard approaches and most of the ICT interventions are proprietary and high cost ventures from private institutions. No national standard architecture for interoperability/ integration of systems
Capacity development and social instruments	Low ICT literacy level among user, None or little research and development done by the national institutions/ academia on the use and scale up ICT in the country. Very few ICT professionals to support large scale ICT projects at national level
Connectivity/ Internet access	Lack of or minimum network coverage by GSM and or broadband technologies. Low cellular subscribers per capita and very low internet subscribers per capita. The percentage fibre connectivity in the country is low. A greater percentage of the population do not have computers, laptops or smart phones.
Access to information	Number of household with internet connectivity is concentrated in the urban areas as opposed to rural areas.
Cost competitiveness	Technologies, which are not always ready-for market, are often more expensive than incumbent technologies, without the necessary supportive infrastructure. Competition from existing technologies, including unsustainable technologies
Knowledge and skills	New technologies require specialized knowledge and skills, which are often lacking in host countries where education levels in science, engineering and technology can be low, and emerging areas. ICT specialists is low
Social Legitimacy	New technologies treated with suspicion in local communities especially if prior experience of job losses or unintended social consequences
Cultural barriers	New technologies are seen as a challenge to cultural traditions and communal activities. Technology can also face barriers such as language, role of women in the society, lack of entrepreneurs or dependencies created by decades of development aid

Source: https://docs.google.com/document/d/1r-rKhtFfCVE7MLNhbK6TdmtJT71NaPdo/edit

Additionally, the Principles for Digital Development are especially relevant when designing for low resource setting: https://digitalprinciples.org/

Each building block specification SHOULD specify mitigations for these issues.

We've seen the many benefits digitalization of governance can bring. We believe the democratization of key technologies can help accelerate this process around the world.

Given these constraints, how can we best reach Sustainable Development Goals?

2.2 SDG Digital Framework Criteria

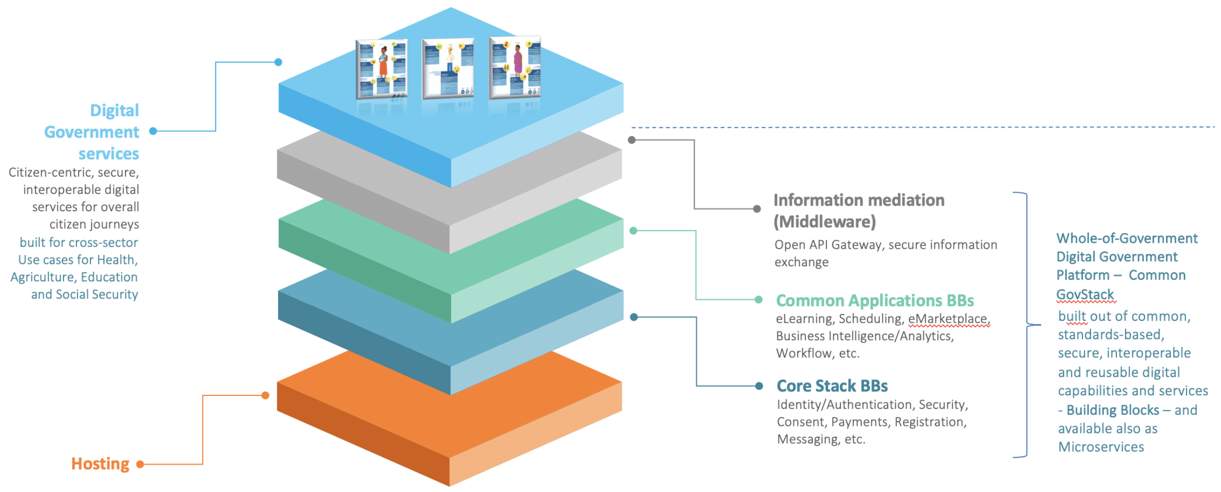
The SDG digital framework has formally defined criteria. Building blocks MUST meet the following criteria:

- *Reusable* software components
- *Licensed* as open source, proprietary, or freely available with Open Access to data
- Facilitates one or more *generic Workflows*
- Applicable to *multiple SDG Use Cases* across multiple sectors
- *Interoperable* with other ICT Building Blocks
- Designed for *Scalability*
- Designed for *Extensibility*
- *Standards* Based Conformance or Compliance

See https://drive.google.com/file/d/1Ix2A3W_vpvaRvKyek524kCuB5MKcRFd3/view?usp=sharing for details, including a checklist.

2.3 Building Blocks

Building blocks are software modules that can be deployed and combined in a standardized manner. Each building block is capable of working independently, but they can be combined to do much more:


Digital Government services
Citizen-centric, secure, interoperable digital services for overall citizen journeys
built for cross-sector Use cases for Health, Agriculture, Education and Social Security
Hosting
Information mediation (Middleware)
Open API Gateway, secure information exchange
Common Applications BBs
eLearning, Scheduling, eMarketplace, Business Intelligence/Analytics, Workflow, etc.
Core Stack BBs
Identity/Authentication, Security, Consent, Payments, Registration, Messaging, etc.
Whole-of-Government Digital Government Platform – Common GovStack
built out of common, standards-based, secure, interoperable and reusable digital capabilities and services - Building Blocks – and available also as Microservices
5

A WoG Digital Government Platform is a "platform of platforms" that can be used by any government agency, department across different sectors to build new government digital services without having to design, test and operate the underlying systems and infrastructure themselves.

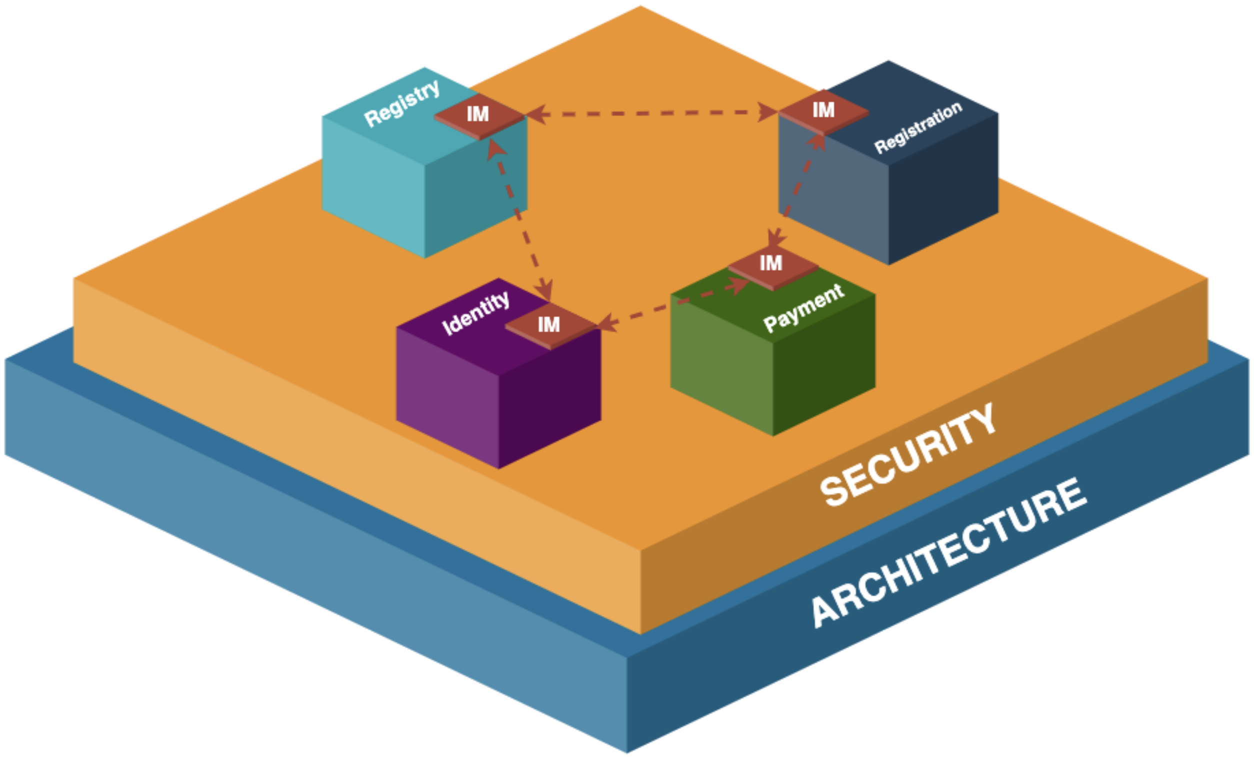
Building blocks are composable, interoperable software modules that can be used across a variety of use cases. They are standards-based, open source and designed for scale.

Each block represents, as much as possible, the minimum required functionality (MVP) to do its job. This ensures each block is usable and useful on its own, and easily extensible to support a variety of use cases.

A block is composed of domain-driven microservices, modeled as closely as possible on existing human roles and processes. This helps ensure each block is as useful as possible in the real world, per Conway's law.

Blocks exchange data using lightweight, human-readable data that can easily be extended where needed. Data models and APIs are described in a lightweight manner that's human-readable, allowing them to be easily and quickly understood and validated.

Building blocks should meet the criteria of the Digital Square Global Goods maturity model for Global Utility, Community and Software Quality. See https://www.google.com/url?q=https://wiki.digitalsquare.io/index.php/Global_Goods_Maturity&sa=D&source=editors&ust=1614177377366000&usg=AOvVaw3rKLBBw5qDajPOetkDdmTv for details.



2.4 Building Blocks Working Together

A building block is only so useful on its own. In practice, building blocks MUST be connected together in a secure, robust, trusted manner that facilitates distributed deployments and communications with existing services.

2.4.1 Federation and Data Exchange Requirements

Each building block deployment MUST use a security server to federate and communicate with other data consumers and providers. This ensures the confidentiality, integrity, and interoperability between data exchange parties. A security server MUST provide the following capabilities:

- address management
- message routing
- access rights management

- organization-level authentication
- machine-level authentication
- transport-level encryption
- time-stamping
- digital signature of messages
- logging
- error handling
- monitoring and alerting

2.4.2 Organizational Model
Some policies and processes need to be applied to support this methodology.

First, a central operator needs to be created. This organization will be responsible for the overall operation of the system, including operations and onboarding new members. Policies and contractual agreements for onboarding need to be created.

Trust services need to be set up internally or procured from third parties, including timestamp and certificate authorities. This provides the necessary infrastructure to support distributed deployments.

Finally, members can be onboarded and provided with access to the security server.

Once agreements are in place, members can deploy new services in a decentralized, distributed manner. Before deploying a new service, the central operator must be notified of any changes to access-rights, including organization and machine-level authentication before it can publish or consume data.

2.4.3 Technical Architecture
A Central Operator is responsible for maintaining a registry of members, the security policies for building blocks and other member instances, a list of trusted certification authorities and a list of trusted time-stamping authorities. The member registry and security policies MUST be exposed to security servers over HTTP.

Certificate authorities are responsible for issuing and revoking certificates used for securing and ensuring the integrity of federated information systems. Certificate authorities MUST support the Online Certificate Status Protocol (OCSP) so security servers can check certificate validity.

Time-stamping authorities securely facilitate time stamping of messages. Time stamping authorities MUST support batched time stamping.

Security servers MUST be used by members to publish or consume data and services. Building blocks and existing information services that use REST MUST work seamlessly with the security server.

2.4.4 Support for Rooms
Ideally, publish/subscribe SHOULD allow blocks to be connected together in rooms to collaboratively solve problems. Once available, blocks should support rooms, e.g.
- everything is connected through a room
- requests are made in the room
- blocks process data
- answer(s) are posted to the room

2.5 System Architecture

It's vital that GovStack be able to connect to existing applications. Likewise, existing applications should be able to connect to and utilize GovStack resources as they see fit. This must be done without compromising the easy and secure interoperability provided by the GovStack system.

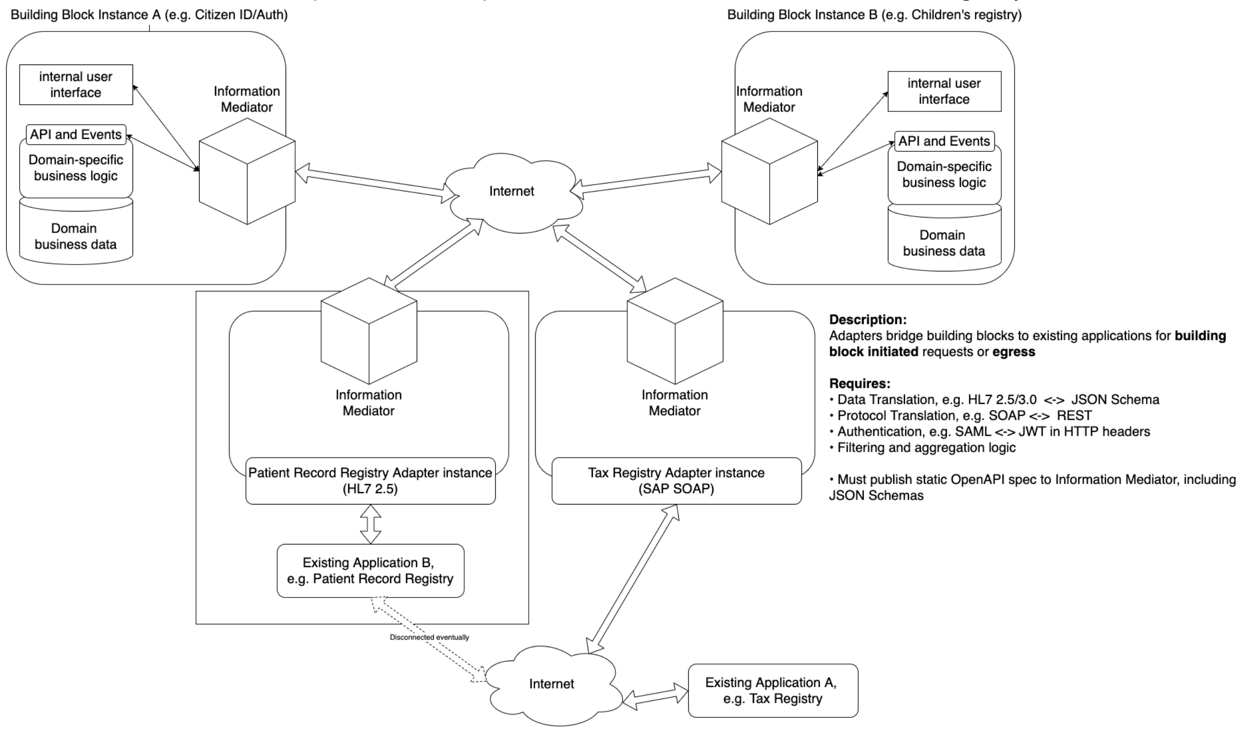
This can be accomplished with Adapters that connect existing applications for use by GovStack and API Gateways that provide secure access to GovStack services for citizens and other applications, including existing applications.

2.5.1 Adapters

Adapters connect GovStack building blocks to existing applications. There are many possible flavors of adapter, e.g. HL7 2.5, HL7 3.0, FHIR, SAP SOAP, SQL and many others. Each flavor can be quickly configured to give GovStack access to existing resources.

Adapters are responsible for data and protocol translation, authentication, and filtering and aggregation logic. They publish an OpenAPI specification via Information Mediator so they can be used by any building block.

Here we can see two adapters, one for patient records and another for a tax registry:


Building Block Instance A (e.g. Citizen ID/Auth)
internal user interface
API and Events
Domain-specific business logic
Domain business data
Information Mediator
Internet
Building Block Instance B (e.g. Children's registry)
Information Mediator
internal user interface
API and Events
Domain-specific business logic
Domain business data
Information Mediator
Patient Record Registry Adapter instance (HL7 2.5)
Existing Application B, e.g. Patient Record Registry
Information Mediator
Tax Registry Adapter instance (SAP SOAP)
Disconnected eventually
Internet
Existing Application A, e.g. Tax Registry

Description:
Adapters bridge building blocks to existing applications for **building block initiated** requests or **egress**

Requires:
- Data Translation, e.g. HL7 2.5/3.0 <-> JSON Schema
- Protocol Translation, e.g. SOAP <-> REST
- Authentication, e.g. SAML <-> JWT in HTTP headers
- Filtering and aggregation logic

- Must publish static OpenAPI spec to Information Mediator, including JSON Schemas

Editable version (github repo / image - link)

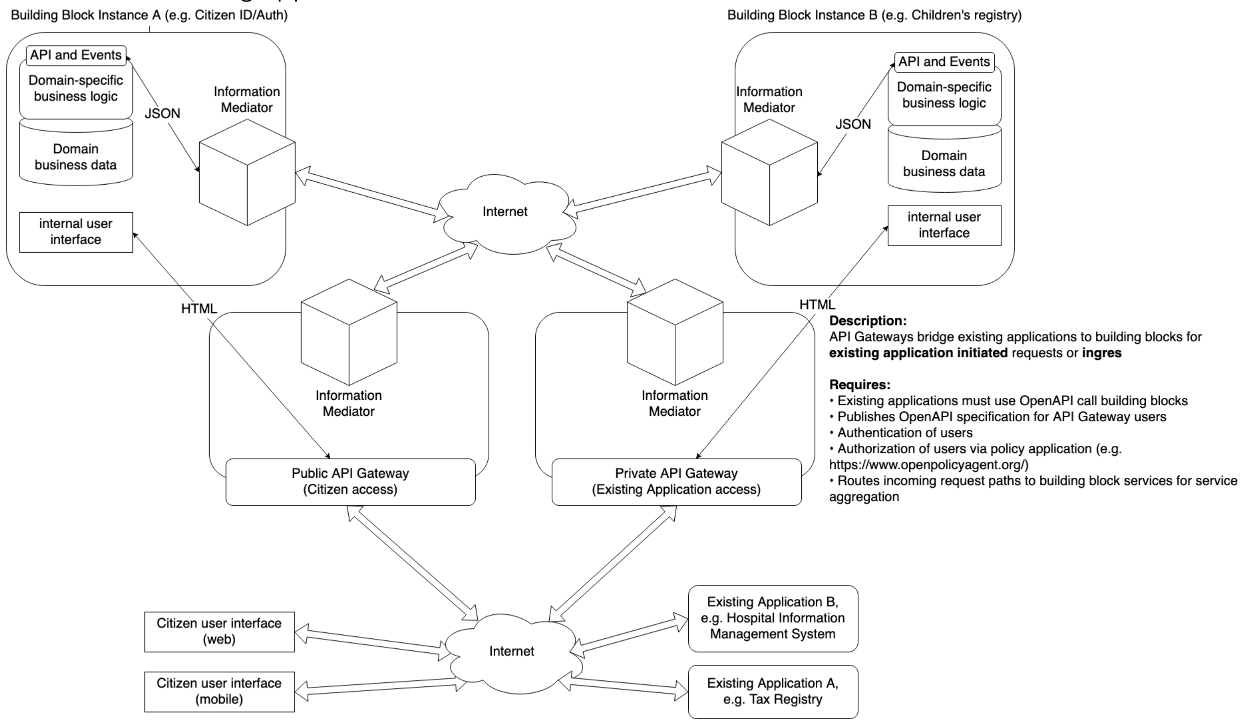
In this example, an HL7 2.5 adapter connects an existing application's patient record registry, and an SAP SOAP adapter connects an existing applications tax registry to GovStack. Both adapters provide services that are available for use by other building blocks.

If an existing application sends events, they are exposed as webhooks in the adapter's OpenAPI specification like other APIs. This allows any GovStack building block to be notified when the event occurs.

Talk about terminology management as part of workflow/adapters, e.g. https://openconceptlab.org/

2.5.2 API Gateways

API gateways connect citizens and existing applications to GovStack building blocks. Here, a public API gateway provides GovStack resources to citizens, while a private API gateway provides GovStack resources to existing applications:



Building Block Instance A (e.g. Citizen ID/Auth)
API and Events
Domain-specific business logic
Domain business data
internal user interface
JSON
Information Mediator
Building Block Instance B (e.g. Children's registry)
Information Mediator
JSON
API and Events
Domain-specific business logic
Domain business data
internal user interface
Internet
HTML
HTML
Information Mediator
Information Mediator
Public API Gateway (Citizen access)
Private API Gateway (Existing Application access)
Citizen user interface (web)
Citizen user interface (mobile)
Internet
Existing Application B, e.g. Hospital Information Management System
Existing Application A, e.g. Tax Registry

Description:
API Gateways bridge existing applications to building blocks for **existing application initiated** requests or **ingres**

Requires:
• Existing applications must use OpenAPI call building blocks
• Publishes OpenAPI specification for API Gateway users
• Authentication of users
• Authorization of users via policy application (e.g. https://www.openpolicyagent.org/)
• Routes incoming request paths to building block services for service aggregation

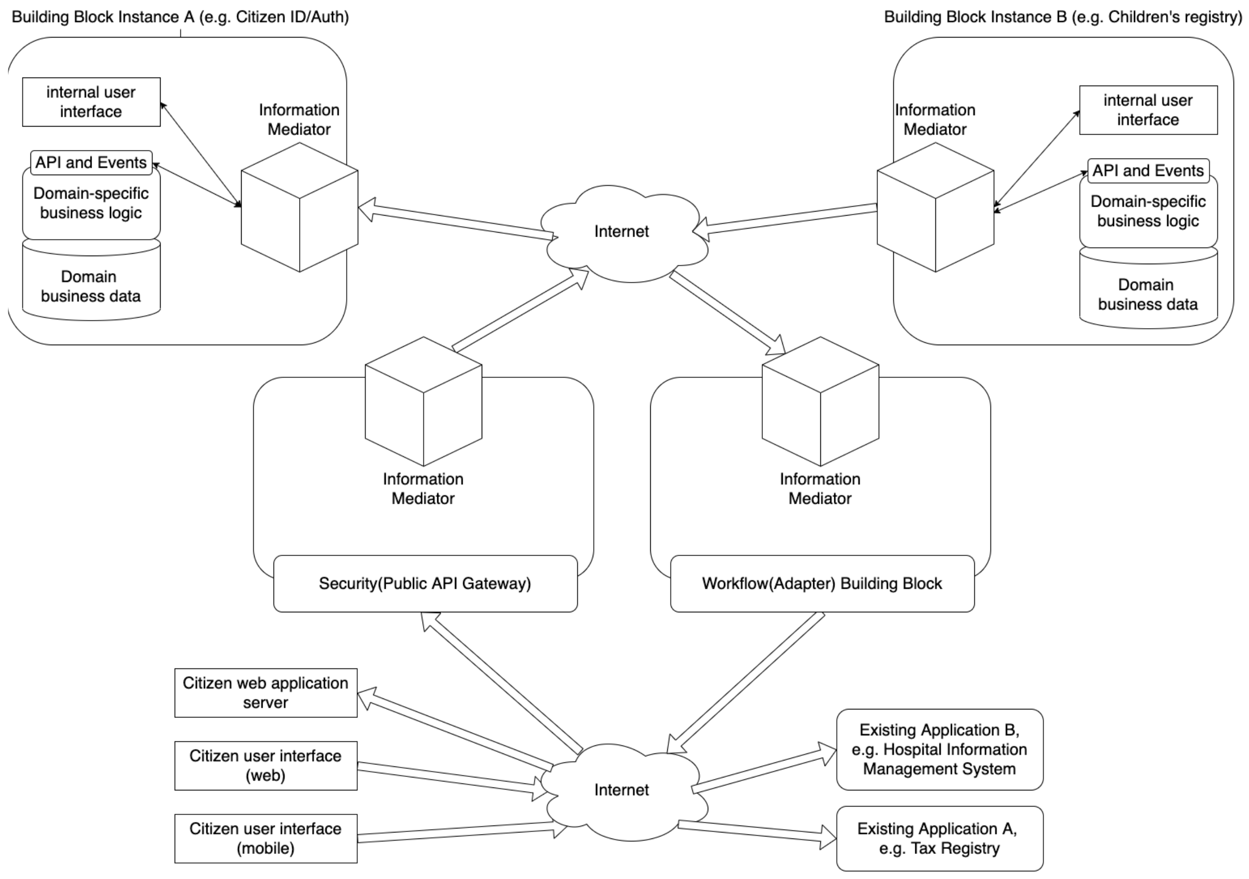
Editable version (github repo / image - link)

In this example, a hospital information management system can use the Citizen ID/Auth building block to authenticate a patient's foundational ID. Likewise, citizens access government services via external mobile or web applications calling through a shared public API gateway.

Any number of API gateways can be added to expose various GovStack services to users or external applications, each of which have different security requirements on the information mediator and public internet/API Gateway sides.

2.5.3 Complete GovStack Deployment

This shows a complete GovStack deployment with API gateways for citizen access via web or mobile and for existing applications to be able to call GovStack APIs on demand. The workflow building block is used as an adapter, exposing existing applications as GovStack resources via OpenAPI:


Building Block Instance A (e.g. Citizen ID/Auth)
internal user interface
API and Events
Domain-specific business logic
Domain business data
Information Mediator
Internet
Building Block Instance B (e.g. Children's registry)
Information Mediator
internal user interface
API and Events
Domain-specific business logic
Domain business data
Information Mediator
Security(Public API Gateway)
Information Mediator
Workflow(Adapter) Building Block
Citizen web application server
Citizen user interface (web)
Citizen user interface (mobile)
Internet
Existing Application B, e.g. Hospital Information Management System
Existing Application A, e.g. Tax Registry

Editable version (open in https://app.diagrams.net/)

Here, citizens and existing applications are provided API access for requests into GovStackvia a common API Gateway, while the workflow building block/adapter provides outgoing API access to existing applications from GovStack building blocks.

3 Terminology

3.1 Keyword Usage

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC2119 RFC8174 when, and only when, they appear in all capitals, as shown here.

3.2 Building Block

1. From this document: “Building blocks are software modules that can be deployed and combined in a standardized manner. Each building block is capable of working independently, but they can be combined to do much more.” Read more: above. (And please

read this!)

2. An application which provides re-usable interfaces:
    a. An **admin-only form builder** which facilitates building user interfaces (e.g., select questions to be displayed in a maternal-and-child-health registration process)

    b. **User interfaces** (i.e., forms) which can be used in lieu of individual end-user apps building their own forms (e.g. I'm building a *new* maternal and child health application; I'd like to use a registration screenflow that's been pre-built in the registration building block as part of a larger, *composed* application.)

    c. A **public API** which exposes the critical **back-end services** performed by this BB (adding a mom to a database; checking for a mom's enrollment status in a program) to be used (as a microservice) by existing or new applications with legacy/bespoke needs (e.g., i've already got a maternal and child health app that the CHWs are using, and I want to send a webhook to the registration BB after a CHW clicks "submit" on our custom form.)

3.3 Registration

Any approval/license/certificate issued by a public entity as a result of a request/declaration made by a user of the public service. The result of a "registration" is usually a number and/or a document (called certificate, license, permit, authorization, registration, clearance, approval, etc.)

3.4 Authentication

This is the technical process of establishing that the credentials (i.e. username, password, biometric etc.) provided by a party (user, system, other) is valid and that the party can be granted basic access to system resources with default access rights. Note that authorization also needs to be applied for a party to access protected resources.

3.5 Authorization

This is the technical process of establishing whether or not an authenticated party has rights to access a given protected resource. Access rights can typically be granted or revoked administratively on a read-only and/or read-write and/or execute basis through an administrative provisioning process. Permissions or rights defined for a party typically manifest in an access token that is granted at the time of authentication for the party. Hence the processes of authentication and authorization are intrinsically related.

3.6 Workflow Terminology

See more comprehensive descriptions of the workflow terminoloy in the Workflow and Business Process Automation Building Block specification.

- (Workflow) Activity - a single step in a workflow process.

- (Workflow) Process - a workflow process contains one or many activities.
- (Workflow) Instance - an instance of execution for a workflow process.

4 Building Block Design Principles

4.1 Citizen-Centric

- Right to be forgotten: everything must be deletable
- User-centered design
  The best tools evolve from empathizing, understanding and designing for the needs of end-users. Accordingly, we've identified a series of use cases and user journeys here: InfoMed - RFIMPL_Use Case Journeys.docx

  Each use case is composed of a collection of modules, or building blocks. As you can see, a relatively small set of these building blocks can be readily applied to a wide variety of applications in low-resource settings.

4.2 Open

- Based on open standards
- Based on Digital Development Principles, see https://digitalprinciples.org/ and https://digitalinvestmentprinciples.org
- Built on open-source software (where possible)
- Supports open development, see https://standard.publiccode.net/
- No vendor lock-in
- Cloud native (Docker/Docker Compose/OCI containers)
- Code is openly developed and available to anyone, via Github or Gitlab

4.3 Sustainable

- Stewardship is critical, see https://publiccode.net/codebase-stewardship/
- Continuous funding for maintenance, development and evolution
- Attractive to ICT industry and individual developers in deployment environment (incentives must be aligned)
- Lower cost than commercial solutions due to shared development costs
- Uses microservices-based architecture instead of monolithic.
  - This increases interoperability, development and deployment speed and reliability.
  - From Wikipedia: a variant of the service-oriented architecture (SOA) structural style – arranges an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight.

4.4 Secure

- Blocks are audited and certified before being made available
- Development processes and standards enforce quality and security
- Different certification levels reflect level of standards-compliance
- Regular security scanning and auditing
- Public ratings and reviews
- Comprehensive logging and exception handling

4.5 Accessible

- Meets users where they are: web, mobile, SMS and/or voice. UI supports accessibility technologies, e.g. screen readers.
- SSO allows for signing in once for multiple services
- Shared ownership of code
- Deployment and development processes and standards are open to contributors
- Community-driven development tools for documentation and support
- Blueprints, templates and documentation

4.6 Flexible

- Blocks can be reused in multiple contexts
- Each block is autonomous
  - Blocks are interoperable
  - Easy to set up
- Standardized configuration and communications protocols connecting blocks
- Blocks can be provided as a service (ICT opportunity)

4.7 Robust

- Operates in low-resource environments:
  - Occasional power
  - Low bandwidth
  - Low-reliability connectivity
- Easily scalable for high availability and reliability
- API-only based decoupling
- Asynchronous communications pattern decoupled through rooms is ideal
- Eventual consistency for data

5 Cross-Cutting Requirements

Building blocks are responsible for meeting all cross-cutting requirements or specifying why specific requirements do not apply. Govstack compliance and certification processes will validate these requirements.

5.1 APIs MUST Return a Response within 60 Seconds

5.2 MUST follow TM Forum Specification REST API Design Guidelines Part 1

See: https://drive.google.com/file/d/1lpl1IzLGzvbXALW4jqDxLAKqfwuJyeXy/view?usp=sharing

5.3 SHOULD follow TM Forum Specification REST API Design Guidelines Parts 2-7

See https://drive.google.com/drive/u/2/folders/1eS20wt_PP7CoJEOvzagYDTTMCfmv3y6l

5.4 EOL MUST be at Least 5 Years

Nothing SHALL be used in a block that has an EOL of less than 5 years.

5.5 MUST Only Use TIOBE Top 25 Languages

See https://www.tiobe.com/tiobe-index/

5.6 MUST Use API-only Based Decoupling

5.7 SHOULD Follow the Amazon API Mandate

See https://api-university.com/blog/the-api-mandate/

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

4. It doesn't matter what technology is used [**ed: internally**]. HTTP, Corba, Pubsub, custom protocols — doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired. Is this enforcable?

5.8 APIs MUST be Idempotent

Paraphrased from Wikipedia: APIs can be called multiple times without changing the result beyond the initial application.

5.9 Stateless APIs SHOULD be Used Wherever Possible to Enhance Scalability

5.10 Regular Security and Code Quality Audits MUST be Run

Thse should be run across the code base and dependencies, e.g. https://www.sonarqube.org/ and/or https://snyk.io/ .

5.11 MUST Include Integration Test Coverage

5.12 MUST Include Support for a Log Sink

5.13 Data Models MUST Include Version Numbers

5.14 APIs MUST be Versioned

As shown here: e.g. /api/v1 and /api/v2

5.15 API Calls MUST Include Transaction/Trace/Correlation IDs

5.16 MUST Use Semantic Versioning

See https://semver.org/

5.17 MUST Provide Configuration Only through the Environment

No configuration in code, use environment variables wherever possible

5.18 MUST Include Clearly-Defined Key Rotation Policies

Some blocks may require the use of security keys. Those that do must have clearly defined key rotation policies to enhance security.

5.19 Documentation MUST be Generated from Code and/or Checked in Alongside Source Code

5.20 MUST Not Use Shared Database/Filesystem/Memory

interconnection uses APIs only

5.21 Databases MUST Not Include Business Logic

This means no triggers/stored procedures shall be used.

5.22 MUST Enforce Transport Security

HTTPS with TLS 1.3 and insecure cyphers disabled.

5.23 MUST Only Use Unicode for Text

5.24 MUST Use ISO8601/UTC for Timestamps

5.25 MUST Follow REST Design Principles

MUST not include PII/session keys in URLs - use POST or other methods for this

MUST support caching/retries

- Resource identification in requests
  Individual resources are identified in requests, for example using URIs in RESTful Web services. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server could send data from its database as HTML, XML or as JSON—none of which are the server's internal representation.

- Resource manipulation through representations
  When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource's state.

- Self-descriptive messages
  Each message includes enough information to describe how to process the message. For example, which parser to invoke can be specified by a media type.[3]

See https://en.wikipedia.org/wiki/Representational_state_transfer for more.

5.26 MUST be Asynchronous First

- When entities change, they publish events, allowing loosely-coupled solutions to be assembled without changing existing APIs.
  Supports occasional connectivity/low bandwidth

5.27 MUST Support Both SAAS and Do-It-Yourself

- Many new governments prefer cloud-based deployments
- SAAS is preferred to make it easy for companies to provide new services
- Anyone should be able to test/deploy building blocks by themselves

5.28 MUST be Autonomous

Each block must be capable of running independently.

5.29 MUST be Open-source Only with No Vendor Lock-in

Each block must be composed of open-source software unless there are no alternatives.

5.30 MUST be Cloud Native

- Each block MUST be ready to be deployed as independent Docker images, with complete source code and build instructions committed to GitHub.

- A block may be composed with Kubernetes or docker compose. All build files must be included alongside the source code.

5.31 MUST Closely Model the Organization/Roles Being Automated

See https://en.wikipedia.org/wiki/Conway%27s_law

5.32 MUST Use Standardized Configuration

- Configuration MUST only happen through environment variables or secrets
- Allows visual construction of blocks, perhaps with domain modeling tools

5.33 MUST Use Standardized Data Formats for Interchange

JSON is used for data models/services. See https://www.json.org/json-en.html

5.34 MUST Use Existing Standards for Data Interchange, Where Available

If an existing standard is available, it should be used, e.g. DICOM/Hl7/FHIR for healthcare. TMForum has a large library of standardized APIs and data models that can be used.

5.35 I/O Sanitization MUST be Used

- See an example https://json-schema.org/ for JSON data.
- Follows Postel's law: be liberal in what you consume but strict in what you emit

5.36 MUST Include Machine-Readable API descriptions

- Each block's service APIs are defined using a standardized machine-readable language. External APIs are described using the OpenAPI 3.0 specification for APIs: https://swagger.io/docs/specification/about/

5.37 MUST Provide a Compliance Test Kit

- Must provide a runnable https://www.postman.com/ to ensure compliance
- Part of building block design

5.38 MUST Use Web Hooks for Callbacks

5.39 MUST be Internationalizable

5.40 MUST Follow Best Practices for Public Code

See https://standard.publiccode.net/ and practices outlined here:

- Code in the Open
- Bundle Policy and Source Code
- Create Reusable and Portable Code
- Welcome Contributions
- Maintain Version Control
- Require Review of Contributions
- Document Your Objectives
- Document your code
- Use plain English
- Use open standards
- Use continuous integration
- Publish with an open license
- Use a coherent style
- Pay attention to codebase maturity

5.41 MUST Follow Strict Requirements Around NTP/RTC Synchronization

5.42 If an API Response will Take Longer than 5 Seconds, you SHOULD Return a Ticket with a Suggested Callback Time that is Resolved by Polling

5.43 MUST use STDOUT/ERR for Logging, to be Captured by the Docker/Container Environment

- Each block MUST be ready to be deployed as independent Docker images, with complete source code and build instructions committed to GitHub

- A block may be composed with Kubernetes or docker compose. All build files must be included alongside the source code.

5.44 SHOULD Include Kubernetes or Ansible Scripts

In addition to Docker Compose for more efficient and scalable deployments. This will make individual components of the building block independently scalable and make building blocks less monolithic and more efficient,

5.45 SHOULD Comply with GDPR Principles

Include the right to be forgotten account deletion), privacy requirements to protect the rights of individuals, etc. Note that these requirements may vary by region.

5.46 MUST Implement Existing Standards Where Possible

Building blocks and building block solutions MUST leverage existing standards, especially those listed in the Standards section below.

6 Standards

6.1 Unicode

Used for encoding text
https://en.wikipedia.org/wiki/Unicode

6.2 ISO8601/UTC

Used for dates and timestamps
https://en.wikipedia.org/wiki/ISO_8601#Coordinated_Universal_Time_(UTC)

6.3 JSON

Used for exchanging data
https://www.json.org/json-en.html

6.4 JSON Schema

Used for specifying data models. Note that OpenAPI 3.1 has full support for JSON Schema
http://json-schema.org/

6.5 REST

Used for implementing APIs
https://en.wikipedia.org/wiki/Representational_state_transfer

6.6 OpenAPI 3.1 (AKA Swagger)

Used for specifying and documenting APIs.
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.1.0.md
Note that OpenAPI 3.1 supports inline JSONSchema for model definitions

6.7 Docker/Docker Compose/OCI Containers

Used for packaging building block components for deployment
https://en.wikipedia.org/wiki/Docker_(software)
https://www.docker.com/resources/what-container

7 Key Decision Log

1. 2021-09-23: Added Key Decision Log, published version 1.0.1
2. 2021-12-10: Added GovStack System Architecture section, published version 1.0.2
3. 2022-03-09: Added links to workflow terminology in workflow BB, published version 1.1.0.

8 Future Considerations

- Include more information about the lifecycle of building blocks and connecting to existing solutions, e.g. start with adapters, compliance (including testing), certification and beyond based on sandbox/prototyping work

- Explore connecting to existing services

- Expand standards based on those used in wave 1 and wave 2 working group specifications