ソフトウェアアーキテクチャドキュメント

Trusted Web の実現に向けたユースケース実証事業

「共助アプリを横断したトラスト形成エコシステム」

大日本印刷株式会社

文書の改訂履歴

バージョン	著者	日付	詳細
1.0	アンドレイ・ムドリク	2023 年 6 月 1 日	新規作成
1.1	アンドレイ・ムドリク	2023 年 7 月 13 日	PoC フェーズのセクションを更新

内容

1 ドメインモデル

ドメインモデル（Domain Model）

ドメインモデル（Domain Model）とは、特定のドメインまたはサブジェクト領域内の主要なエンティティ、関係、および動作を概念的に表現したものである。ドメインモデルは、ドメインの概念、その属性、およびそれらの間の相互作用の構造化されたビューを提供する。ドメインモデルの目的は、問題ドメインの本質的な要素を捉え、ビジネスアナリスト、開発者、ドメイン専門家などの利害関係者間の共通言語として機能することである。

検証可能なクレデンシャル（Verifiable Credential）

あるエンティティに関する一連の主張を保持する、デジタル署名され改ざん防止されたクレデンシャル。発行者、サブジェクト（命名されるエンティティ）、クレーム、暗号証明などの情報を保持する。

サブジェクト（Subject）

検証可能なクレデンシャルに命名または代表されているエンティティ。個人、組織、またはクレデンシャルが発行されるその他のエンティティである。

クレーム（Claim）

対象者に関する特定の情報や属性を表す。たとえば、名前、生年月日、住所、資格、認定などである。クレームは検証可能なクレデンシャルに含まれ、通常は発行者によってデジタル署名される。

発行者（Issuer）

検証可能なクレデンシャルの発行に責任を負うエンティティまたは組織を示す。秘密鍵を使用して検証可能クレデンシャルを作成し、署名する。特定のドメインまたはコンテキスト内で信頼される権威となることができる。クレデンシャルを発行するための Hyperledger Indy フレームワークに準拠する。

ホルダー（Holder）

検証可能なクレデンシャルを保持する個人または団体。クレデンシャルの保管および提示を管理する。クレデンシャル内の特定の属性の開示を管理する。自己の ID に対する自己主権的な制御権を付与される。

検証者（Verifier）

検証可能なクレデンシャルの真正性および完全性を検証する責任を負うエンティティまたはシステム。クレデンシャルに対して暗号チェックを実行し、改ざんされていないことを確認する。事前に定義されたポリシーまたはルールに基づいてクレデンシャル内の主張を検証する。検証の判断には、より多くの情報またはコンテキストが必要な場合がある。

クレデンシャルスキーマ（Credential Schema）

特定のタイプの検証可能クレデンシャルの構造と形式を定義する。必要な属性およびそのデータタイプを指定する。クレデンシャルの発行および検証における相互運用性および一貫性を可能にする。

クレデンシャル定義（Credential Definition)

クレデンシャルスキーマの特定のインスタンスを表す。発行者の DID（分散識別子）やキー情報などの詳細情報を保持する。クレデンシャルの真正性を検証するために必要な情報を提供する。

提示要求（Presentation Request）

特定のクレデンシャルまたは属性の提示を検証者がホルダーに行う要求。特定のコンテキストに表示するクレデンシャルまたは属性を指定する。検証者は、検証に必要な情報のみを収集するために提示依頼を作成できる。

証明リクエスト（Proof Request）

提示要求と同様に、検証者がクレームや属性を検証するために必要な情報を定義する。検証に必要なクレーム、述語、条件を指定する。ゼロ知識証明など、より複雑な検証シナリオに使用できる。

プレゼンテーション

検証可能なクレデンシャルを検証者に提示する行為、または提示要求を満たす行為。ホルダーは、他の属性を保護しながら特定の属性を選択的に示すことができる。ホルダーに個人情報の開示を制御する権限を与える。

失効登録

以前に発行された検証可能なクレデンシャルを取り消すまたは無効にするメカニズム。取り消されたクレデンシャルの受理を防ぐために発行者が維持する。暗号化メカニズムを利用して、失効情報の完全性を保証する。

分散型識別子(DID)

一意の識別子は、アイデンティティと検証可能なクレデンシャルに対する分散制御を提供する。個人またはエンティティが自身の ID を管理できるようにすることで、自己主権的 ID を可能にする。クレデンシャルの発行、検証、および提示における相互運用性とプライバシーをサポートする。

分散型台帳（Hyperledger Indy ）

情報を記録および共有するための分散型耐改ざん性データベースとして機能する。クレデンシャルスキーマ、クレデンシャル定義、失効レジストリ、およびその他の成果物の 登録を格納および管理する。検証可能なクレデンシャル・エコシステムに透明性、不変性、および分散制御を提供する。

Aries プロトコル

分散型技術を使用して検証可能なクレデンシャル・エコシステムを構築するためのフレームワーク。発行者、ホルダー、検証者間の安全でプライベートなやり取りを可能にする。通信プロトコル、メッセージフォーマット、暗号操作を定義する。

エージェント

Aries プロトコルを実装するソフトウェア部分。エコシステム内のエンティティ（発行者、ホルダー、検証者）を表す。検証可能なクレデンシャルの通信、ネゴシエーション、交換を促進する。

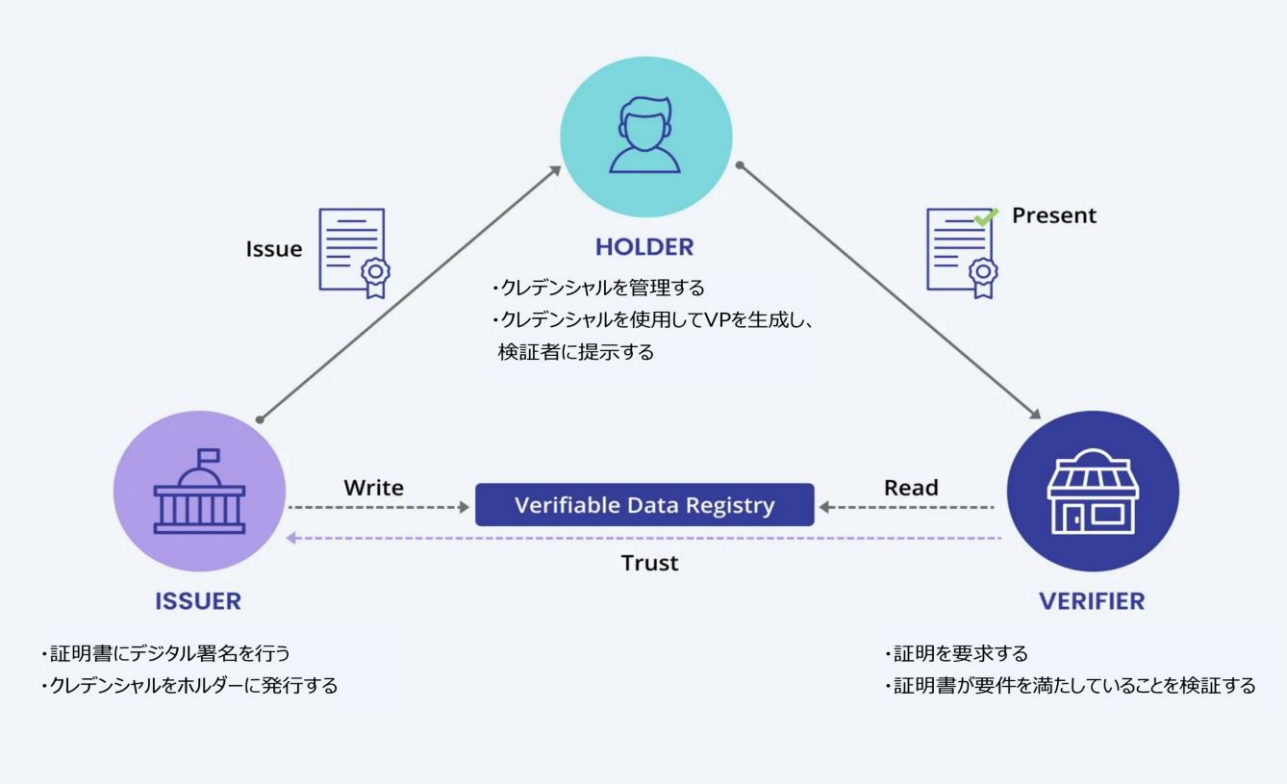

図 1 検証可能なクレデンシャル・ドメイン図

2 PoC フェーズ

ソリューション

Scene 1 と Scene 2 を設計し、ワークフローをデモするための単純な Wallet アプリケーションを実装することで、目標を達成する。

2.1 スコープ

概念実証（PoC）フェーズ で実装されたユーザストーリーのリストを以下に示す。

コード	特徴
US-01	***ログイン*** Wallet のユーザとして、Wallet にログインできるようにしたい。
US-02	***ログアウト*** Wallet のユーザとして、Wallet からログアウトできるようにしたい。
US-03	***証明書の一覧*** Wallet のユーザとして、自分の証明書の一覧を見たい。
US-04	***証明書の詳細*** Wallet ユーザとして、証明書の詳細を確認したい。
US-05	***証明書の削除*** Wallet のユーザとして、証明書を削除したい。
US-06	***証明書追加リクエストの作成・送信*** Wallet のユーザとして、サードパーティのアプリケーションから証明書の追加要求を受け取りたい。
US -07	***証明書追加リクエストの承認*** Wallet のユーザとして、Wallet に新しい証明書を追加することを承認したい。
US -08	***証明書追加リクエストの拒否*** Wallet のユーザとして、Wallet への新しい証明書の追加を拒否したい。
US -09	***証明書追加リクエストの一覧*** Wallet のユーザとして、すべての証明書追加リクエストとこれらのリクエストで実行できるアクションを確認したい。
US -10	***検証リクエストの作成・送信*** Wallet のユーザとして、検証者からデータを検証するリクエストを受け取りたい。
US-11	***検証リクエストの承認*** Wallet のユーザとして、検証リクエストを承認したい。
US-12	***検証リクエストの拒否*** Wallet のユーザとして、検証リクエストを拒否したい。

コード	特徴
US -13	***検証リクエストと一致した証明書の一覧*** Wallet のユーザとして、検証データとして提示する証明書とその属性の一覧を見たい。
US-14	***提示する証明書とその属性の選択*** Wallet のユーザとして、検証データとして提示する証明書とその属性を選択したい。
US-15	***検証可能なプレゼンテーションの作成・送信*** Wallet ユーザとして、リクエストを送信した検証者が検証結果を受け取るようにしたい。
US-16	***検証状況の表示*** Wallet のユーザとして、Wallet 内のユーザデータの検証状況を確認したい。
US-17	***検証者リクエストの一覧*** Wallet のユーザとして、検証者からのすべてのリクエストと、これらのリクエストに対して実行できるアクションを確認したい。

2.2 シーン

2.2.1 シーン 1

シーン 1 では、Issuer コントローラを使用してクレデンシャルオファーを送信し、ID Wallet アプリのユーザがそれを受信するフローを考える。ユーザはオファーを受信すると、それを承認し、ID Wallet アプリのクレデンシャル画面で発行されたクレデンシャルを確認することができる。

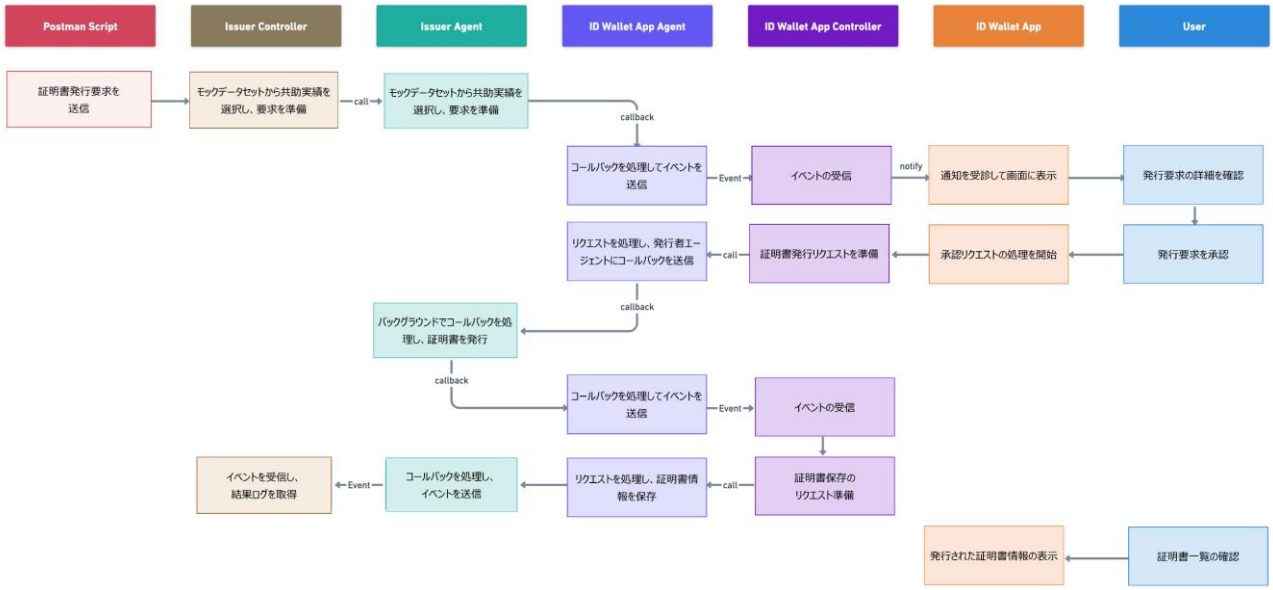

図 2 クレデンシャル発行の流れ

2.2.2 シーン 2

シーン 2 では、Verifier コントローラを使用して証明要求を送信し、ID Wallet アプリのユーザがそれを受信するフローを考える。証明要求を受信すると、ユーザは適切なクレデンシャルを選択し、検証可能なプレゼンテーション（VP）を準備することができる。VP が送信されると、Verifier はそれを検証し、検証プロセスの結果に関する通知を ID Wallet アプリを通じてユーザに提供する。

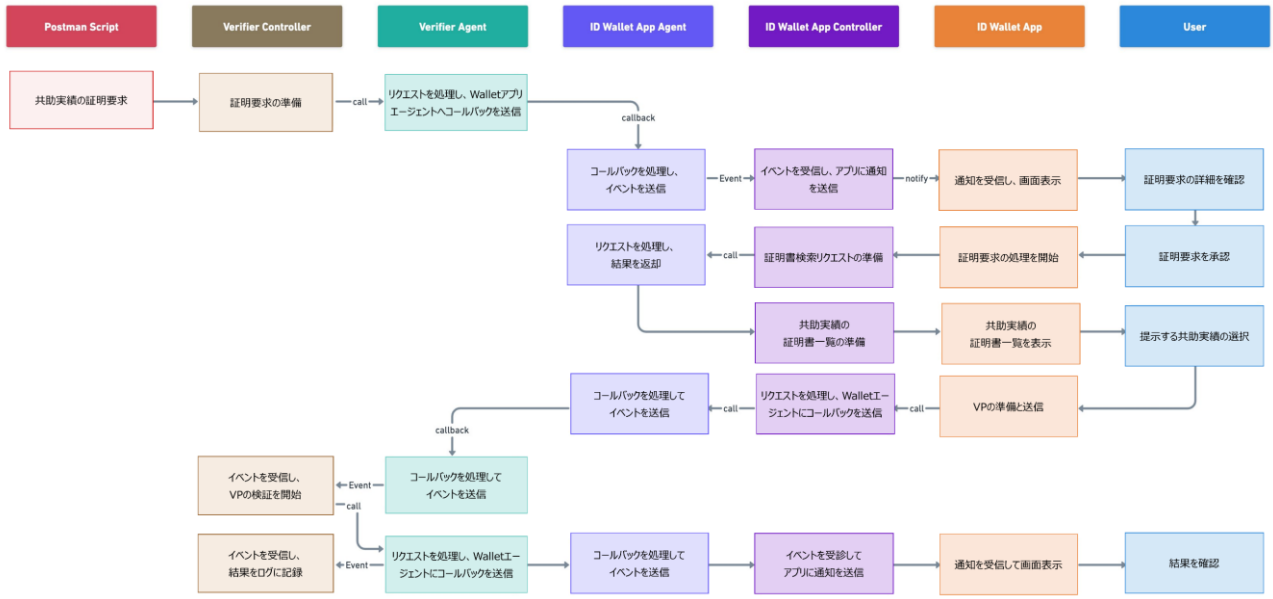

図3 検証の流れ

2.2.3 画面遷移図

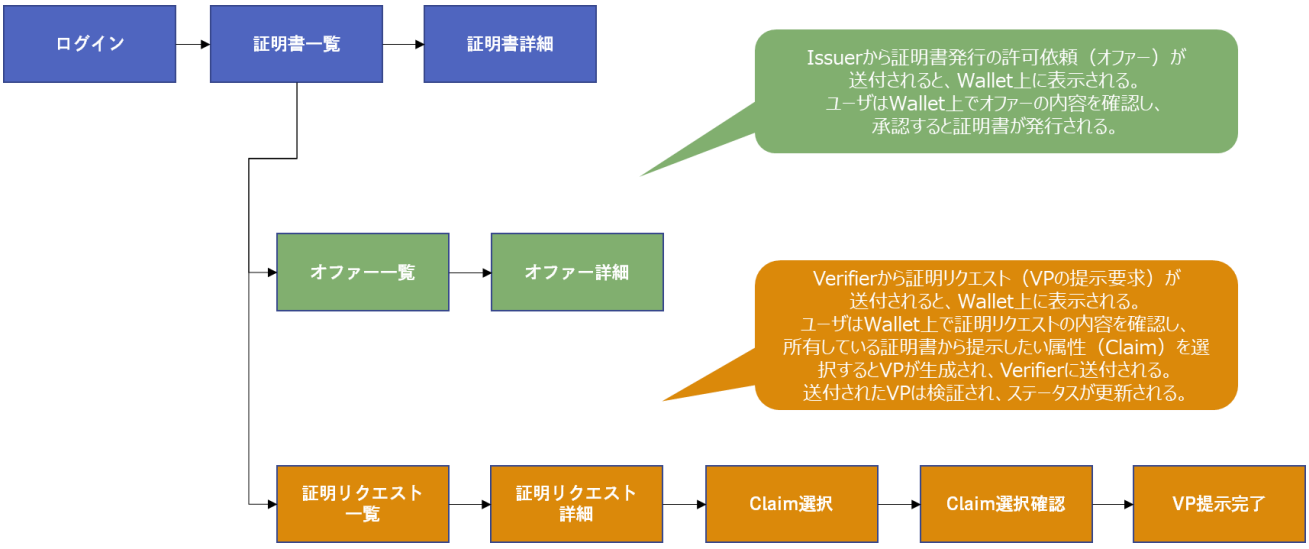

図4 画面遷移

2.2.4 画面構成

図 5 画面構成

2.3 仮説と成功指標

以下の表は、PoC フェーズで検証される仮説と、成功・失敗の指標をまとめたものである。

コード	仮説
PH-01	Wallet ユーザ は、Wallet アプリ内で検証可能クレデンシャル（証明書）を 受信し、閲覧することができる。 ***失敗：*** ● 新しい証明書が、Wallet のユーザ証明書リストに表示されない ***成功：*** ● Wallet ユーザが証明書追加確認要求を受領した ● 追加承認後、Wallet ユーザの証明書一覧に新しい証明書が表示された
PH-02	Wallet ユーザは、Wallet アプリを通じてデータ検証のリクエストを受け取り、それに応答することができる。 ***失敗：*** ユーザが Wallet アプリを通じて認証リクエストに応答できない ***成功：*** ● Wallet ユーザは、Wallet アプリを通じて検証要求を正常に受信する ● Wallet ユーザ は、Wallet アプリを通じて、利用可能な証明書およびその属性の一覧から、検証に必要な関連データを選択することができる ● 検証者は、生成された VP を Wallet アプリから正常に受信する ● 検証者は、Wallet アプリを通じて、検証状況をリアルタイムで Wallet ユーザに通知する

3 アーキテクチャビジョン

本セクションでは、技術的なアーキテクチャについて説明する。アーキテクチャとシステムコンポーネントのハイレベルな概要、基本的な統合、および実装情報が含まれる。

このビジョンのアーキテクチャを最適に記述し、理解しやすくするために、C4 モデルと呼ばれる文書化アーキテクチャ技術を使用した。このモデルは、4 つのダイアグラムから構成される階層的な抽象化の集合である：「コンテキスト」、「コンテナ」、「コンポーネント」、「コード」である。この文書では、最初の 3 つについて説明・紹介する。

3.1 制約条件

制約とは、顧客が提供する環境、統合する必要のある既存システム、コントロールできない、あるいはいかなる形でも変更することができないソリューション環境のその他の部分など、外部の力によってソリューションに課される制限のことである。

コード	説明
CON-01	このソリューションの実装には Hyperledger Aries フレームワークが利用されている。 詳細：事前フェーズでは、コア要件に対応するためにこれらのツールを使用する技術的実現可能性をテストした。
CON-02	事前フェーズからのインフラを活用し、PoC フェーズで再利用する。 詳細：PoC フェーズの準備に先立ち決定された。
CON-03	サポートは、Hyperledger Aries と互換性のある標準、プロトコル、仕様に限定される。 詳細：このソリューションは Hyperledger Aries フレームワークまたはその実装に基づいている。
CON-04	オフラインモードでは使用できない 詳細：このソリューションはリアルタイムのデータに依存しており、継続的なインターネット接続が必要となる。
CON-05	期間と予算 詳細：PoC フェーズには 8 週間の開発期間が割り当てられる。このタイムラインには、計画、設計から開発、テスト、最終評価までのすべての活動が含まれる。指定された 期間内に提案されたソリューションの実現可能性を実証するためには、タイムラインを遵守することが極めて重要である 。

3.2 アーキテクチャ上の決定事項

アーキテクチャ上の決定事項を以下に記載する。

コード	説明
ADR-01	**タイトル**：クラウド環境 **背景**：ソリューションが展開されるクラウド・プロバイダーを定義する必要がある。 **決定**：事前フェーズの既存インフラを活用・再利用するため、ソフトウェアは AWS のクラウド環境にデプロイする。 **結果**：環境が変わる場合、クラウド・プロバイダー間でインフラ、サービス、API に違いが生じる可能性がある。前の段階のインフラを再利用する場合、新しいクラウド・プロバイダーの環境との互換性を確保するために修正や適応が必要になることがある。 制約の参照 CON-02
ADR-02	**タイトル**：データベース技術 **背景**：スケーリング、パフォーマンス、合理的なデータベース・タイプを満たすためには、適切なデータベース・プロバイダーとタイプを定義しなければならない。 **決定**：Amazon Aurora PostgreSQL サーバーは、構造的に定義されたリレーショナルな方法で保存された大量のレコードのため、データベースサービスとして使用されている。Postgres SQL サーバーは、高可用性とレプリケーションもサポートしている。 **結果**：他のデータベースタイプを選択した場合、インフラを変更する必要がある。開発リソースは皆 SQL Server に精通しているため、ソリューションが拡張される際に技術的な知識がボトルネックになる可能性がなくなる。
ADR-03	**タイトル**：検証可能なクレデンシャル・エコシステムの基盤の選択 **背景**：検証可能なクレデンシャル（VC）エコシステムを構築するための基盤となるフレームワークを選択する必要がある。 **決定**：VC エコシステムの基盤フレームワークとして Aries Cloud Agent Python（ACA-Py）を利用することにした。 **結果**：この決定を変更すると、互換性の問題が生じ、インフラの再構築が必要となり、分散型 ID エコシステム内でのシームレスな統合が妨げられる。また、新しいフレームワークのための学習曲線が発生し、開発スケジュールとコミュニティのサポートを妨げる可能性がある。ACA-Py から変更しないことで、安全で相互運用性が高く、拡張可能な VC エコシステムが保証される。 制約の参照 CON-01
ADR-04	**タイトル**：サーバサイドコンポーネント実装のための言語とフレームワーク

	背景：システムのサーバー側コンポーネントを実装するためのプログラミング言語とフレームワークを決定する必要がある。 **決定：**コントローラを含むサーバー側のコンポーネントは、Python プログラミング言語と FastAPI フレームワークを使用して実装される。 **結果：**Python と FastAPI を選択することで、Python での ACA-Py の実装と一致し、均質な開発環境を保証する。この選択は、コードの保守性を促進し、統合の複雑さを軽減し、開発チーム間の知識共有を促進する。しかし、この決定を変更すると、新しい言語とフレームワークの学習曲線が発生し、開発スケジュールに影響を与える可能性があり、エコシステムと ACA-Py の統合に調整が必要になる。Python と FastAPI を使用することで、まとまりのある効率的な開発プロセスが保証される。
ADR-05	**タイトル：**モバイルアプリケーションのフレームワークと言語 **背景：**モバイルアプリケーションを実装するためのフレームワークとプログラミング言語を決定する必要がある。 **決定：**モバイルアプリケーションは、SwiftUI、UIKit、Swift プログラミング言語の組み合わせを使って実装される。 **結果：**SwiftUI、UIKit、Swift を選択することで、iOS プラットフォームのネイティブな開発機能を活用し、最適なパフォーマンスとプラットフォーム固有の機能へのアクセスを保証する。この選択はまた、幅広い UI（ユーザインターフェイス）コンポーネント、デザインツール、コミュニティのサポートを提供する。しかし、この決定を変更すると、異なるフレームワークとプログラミング言語の学習曲線が発生し、開発スケジュールに影響を与える可能性がある。また、UI アーキテクチャの調整や、他のコンポーネントとの統合も必要となる。
ADR-06	**タイトル：**インフラ配備ツール **背景：**プロジェクトのために AWS インフラをデプロイするツールを決定する必要がある。 **決定：**すべての AWS インフラのデプロイは Terraform を使って行う。 **結果：**インフラのデプロイに Terraform を選択することで、事前フェーズにおける Terraform の既存の使い方との一貫性を保つ。これにより、ツールへの習熟を保証し、チーム間の知識共有を促進する。また、Infrastructure-as-Code の実践、バージョン管理、スケーラビリティを可能にする。しかし、この決定を変更すると、別のインフラ配備ツールに移行する必要があり、学習曲線、既存のインフラコードの手直し、調整の課題が生じる可能性がある。Terraform にこだわることで、効率的なインフラ管理を保証し、デプロイプロセスにおける潜在的な混乱を減らすことができる。
ADR-07	**タイトル：**検証可能な証明書フォーマット **背景：**システムでクレデンシャルを発行するためのフォーマットを決定する必要がある。 **決定：**クレデンシャルの発行は、AnonCreds（匿名クレデンシャル）フォーマットを使用して行われる。 **結果：**クレデンシャル発行に AnonCreds フォーマットを選択することは、プロジェクトの時間と予算の制約を考慮し、事前フェーズで実施された技術的実現可能性テストに基づいている。しかし、この決定を変更すると、代替のクレデンシャル・フォーマットを調査して採用する必要が生じ、追加の開発およびテ

	スト作業が必要になる可能性がある。また、他のコンポーネントやエコシステムとのシステムの相互運用性や互換性にも影 響を与える可能性がある。
ADR-08	**タイトル：**ID Wallet アプリエージェントの動作モード **背景：**ID Wallet アプリ Agent の動作モードを決定する必要がある。 **決定：**IDWallet アプリエージェントはマルチテナンシーモードで運用される。 **結果：**Wallet アプリエージェントにマルチテナンシーモードを選択すると、アプリのインフラとリソースを複数のテナントで共有できる。このモードでは、システムが複数のユーザまたは組織に同時にサービスを提供できるため、リソースの効率的な利用、コストの削減、拡張性が可能になる。しかし、マルチテナントから移行するには、アーキテクチャを大幅に変更する必要があり、リソースの割り当て、セキュリティ境界、およびシステム全体の複雑性に影響を与える可能性がある。
ADR-09	**タイトル：**エージェント間のコネクションの確立 **コンテキスト：**発行者エージェントと検証者エージェントとの接続を確立するプロセスを決定する必要がある。 **決定：**新しい Wallet ユーザの作成は、API 呼び出しを通じて実行され、発行者および検証エージェ ントとの接続が自動的に確立される。 **結果：**Wallet ユーザの作成と発行者および検証エージェントとの接続を自動化することで、オンボー ディングプロセスを合理化し、手作業によるエージェント接続の必要性を減らすことができる。このアプローチは、発行者および検証エージェントが実装の主要分野ではない、このフェーズの焦点と一致している。しかし、この決定から外れる場合、インタラクティブなエージェント接続を実装する必要があり、開発工数が増加し、システムが複雑化する可能性がある。
ADR-10	**タイトル：**証明リクエストの構造 **背景：**証明リクエストの構造と、Wallet アプリケーションに保存されている共助実績の構造との整合性を評価する必要がある。 **決定：**証明リクエストの構造は、Wallet アプリケーションに保存されている共済実績の構造と一致する。 **結果：**この整合性から逸脱すると、マッピングと変換の追加作業が必要となり、複雑さが生じ、検証プロ セスの効率が低下する可能性がある。
ADR-11	**タイトル：**リアルタイムのステータス更新 **背景：**証明書追加リクエスト、提示要求、および検証可能なプレゼンテーション（VP）の検証プロセスを受信するためのリアルタイムのステータス更新を提供するメカニズムを決定する必要がある。 **決定：**ソケットを使用して、証明リクエスト、提示要求、および VP 検証のプロセスを受信するためのリアルタイムのステータス更新を提供する。 **結果：**リアルタイムのステータス更新のためにソケットを実装することで、システムとユーザ間の即時通信が可能になり、アプリケーションの応答性と双方向性が強化される。これにより、シームレスでダイナミックなユーザ体験が提供され、信頼性、エンゲージメント、およびクレデンシャルと検証の管理効率

	が向上する。この決定から逸脱すると、更新が遅れたり非同期になり、ユーザの満足度やリアルタイムの意思決定に影響を与える可能性がある。
ADR-12	**タイトル**：ソフトウェアデザインパターン **背景**：システムのアーキテクチャに使用するソフトウェアデザインパターンを決定する必要がある。 **決定**：レイヤードアーキテクチャを導入する。 **結果**：レイヤードアーキテクチャを実装することで、いくつかの利点が得られる。システムをそれぞれ特定の責任を持つ明確なレイヤーに編成することによって、懸念事項の分離、モジュール性、保守性を促進する。これらの層には通常、プレゼンテーション層、アプリケーション層、データ層が含まれ、ユーザインターフェース、ビジネスロジック、データストレージの明確な分離を保証する。これにより、コードの再利用性、テスト容易性、柔軟性が促進され、システム全体に影響を与えることなく、個々のレイヤーの修正や拡張が容易になる。さらに、レイヤードアーキテクチャは、開発者が特定のレイヤーに集中し、指定された責任の範囲内で独立して作業できるため、チームのコラボレーションを促進する。レイヤードアーキテクチャを採用することで、構造化、拡張性、保守性の高いシステム設計を実現します。

3.3 ソリューション

次の図は、ソリューションが動作するコンテキスト、つまりソリューションが相互運用するすべての外部システムやアクターを示している。

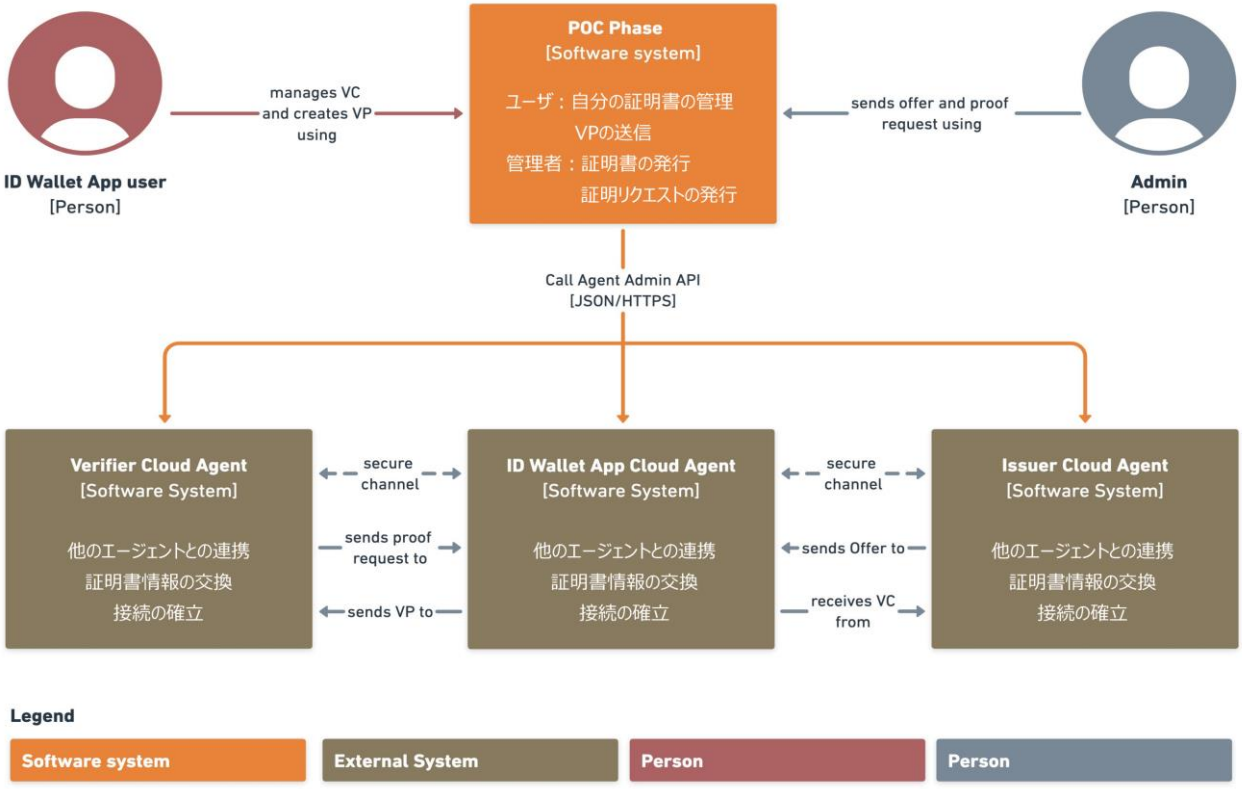

図 4 POC フェーズのシステム・コンテキスト図

アクター

- Wallet アプリのユーザ
- Admin (管理者)

外部システム

- **Verifier Cloud Agent** - 他のエージェントとの相互作用、検証可能なクレデンシャルの交換、接続の確立を可能にする。
- **Issuer Cloud Agent** - 他のエージェントとの相互作用、検証可能なクレデンシャルの交換、接続の確立を可能にする。
- **ID Wallet App Cloud Agent** - 他のエージェントとの相互作用、検証可能なクレデンシャルの交換、接続の確立を可能にする。

3.4 コンテナ図

コンテナ図は、POC フェーズのアーキテクチャの概要と、その中でどのように責任が分担されているかを示す。これは、主要な技術的選択を反映し、コンテナが互いにどのように通信するかを示している。

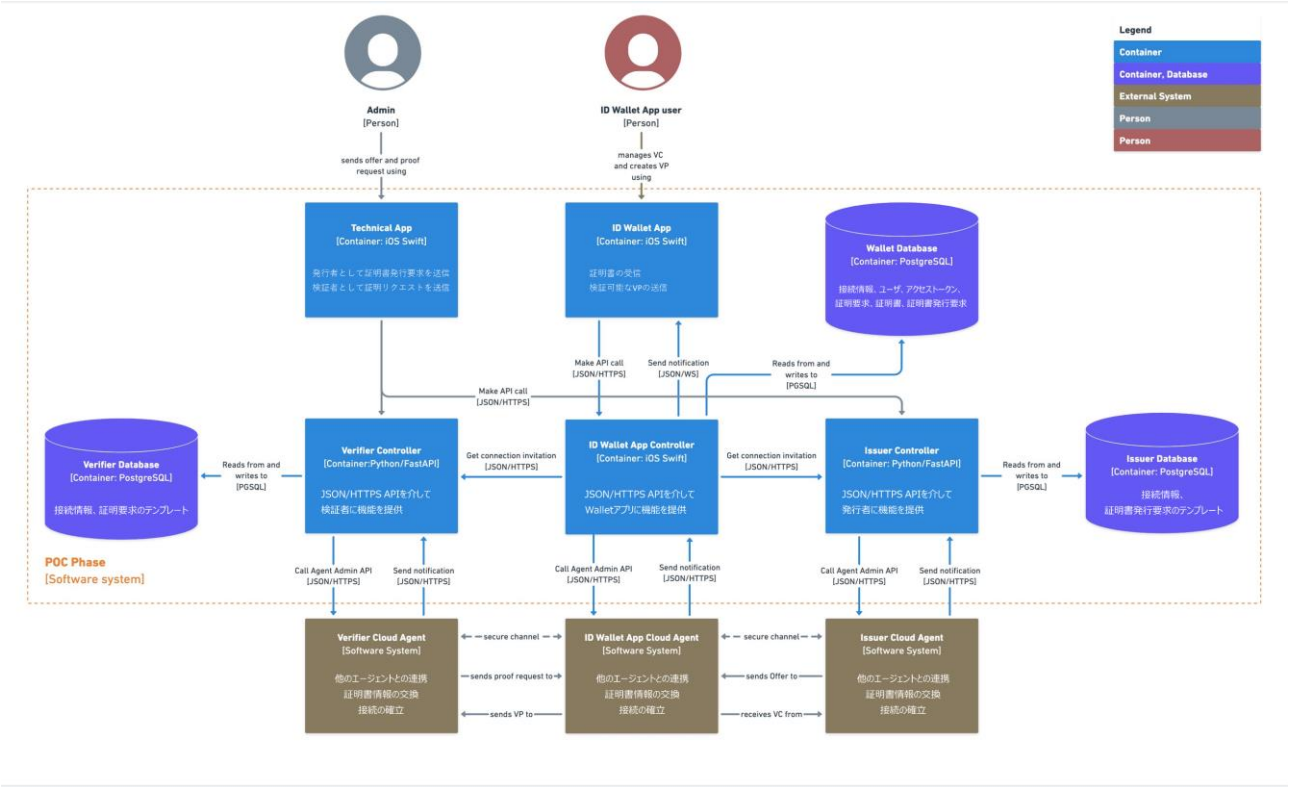

図 5 POC（概念実証）フェーズのコンテナ図

コンテナ

- **Technical App** - ユーザデバイスにインストールされ、管理者が ID Wallet アプリのユーザにクレデンシャルのオファー、準備、証明リクエストの送信を行うためのモバイルアプリケーション。
- **ID Wallet App** - ユーザデバイスにインストールされ、検証可能なクレデンシャルを受け取り、検証可能なプレゼンテーションを送信できるモバイルアプリケーション。
- **ID Wallet App Controller** - JSON/HTTPS API を介して Wallet アプリの機能を提供する。
- **Verifier Controller** - JSON/HTTPS API を介して検証者の機能を提供する。
- **Issuer Controller** - JSON/HTTPS API を介して発行者の機能を提供する。
- **Wallet Database** - データポイントを保存し、データポイントへのアクセスを提供する。
- **Issuer Database** - データポイントを保存し、データポイントへのアクセスを提供する。
- **Verifier Database** - データポイントを保存し、データポイントへのアクセスを提供する。

3.5 コンポーネント図

コンポーネント図は、コンテナがどのように「コンポーネント」で構成されているか、各コンポーネントが何であるか、それぞれの責任、そして技術／実装の詳細を示している。

3.5.1 コンポーネント図 - 発行者コントローラ

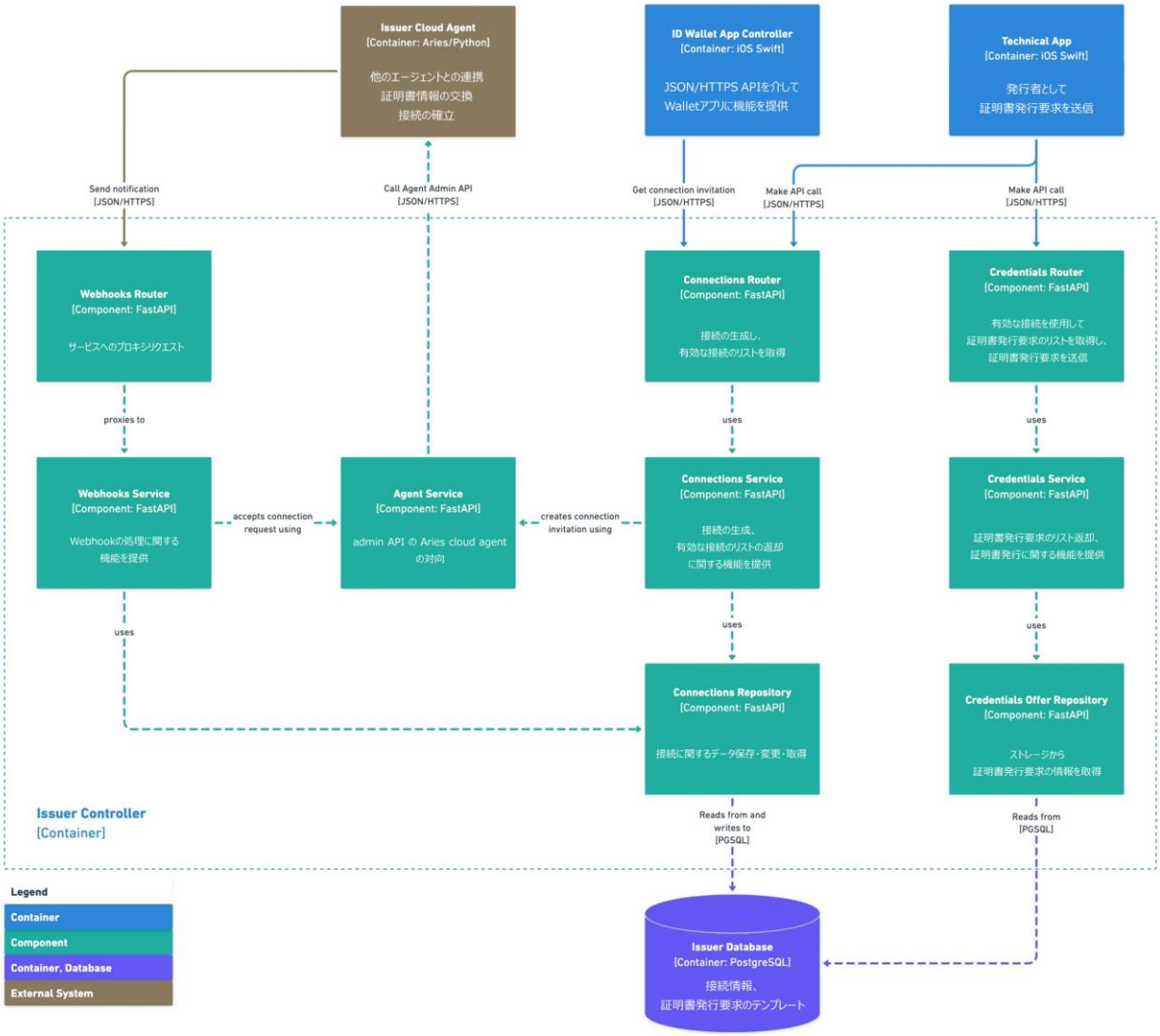

図 6 POC フェーズ- 発行者コントローラのコンポーネント図

3.5.2 コンポーネント図 - 検証者コントローラ

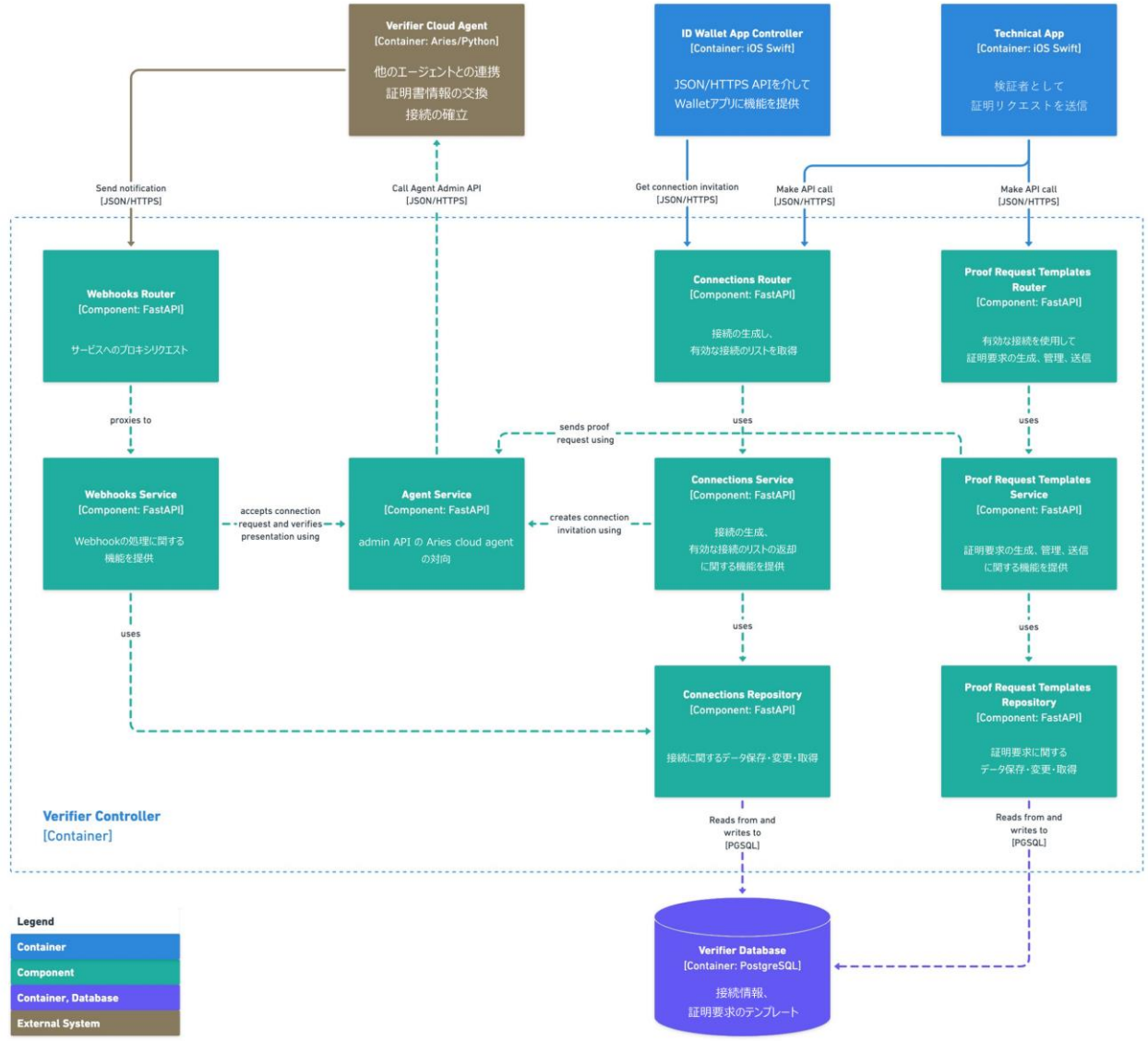

図7 POC フェーズ - 検証者コントローラのコンポーネント図

3.5.3 コンポーネント図 - Wallet アプリコントローラ

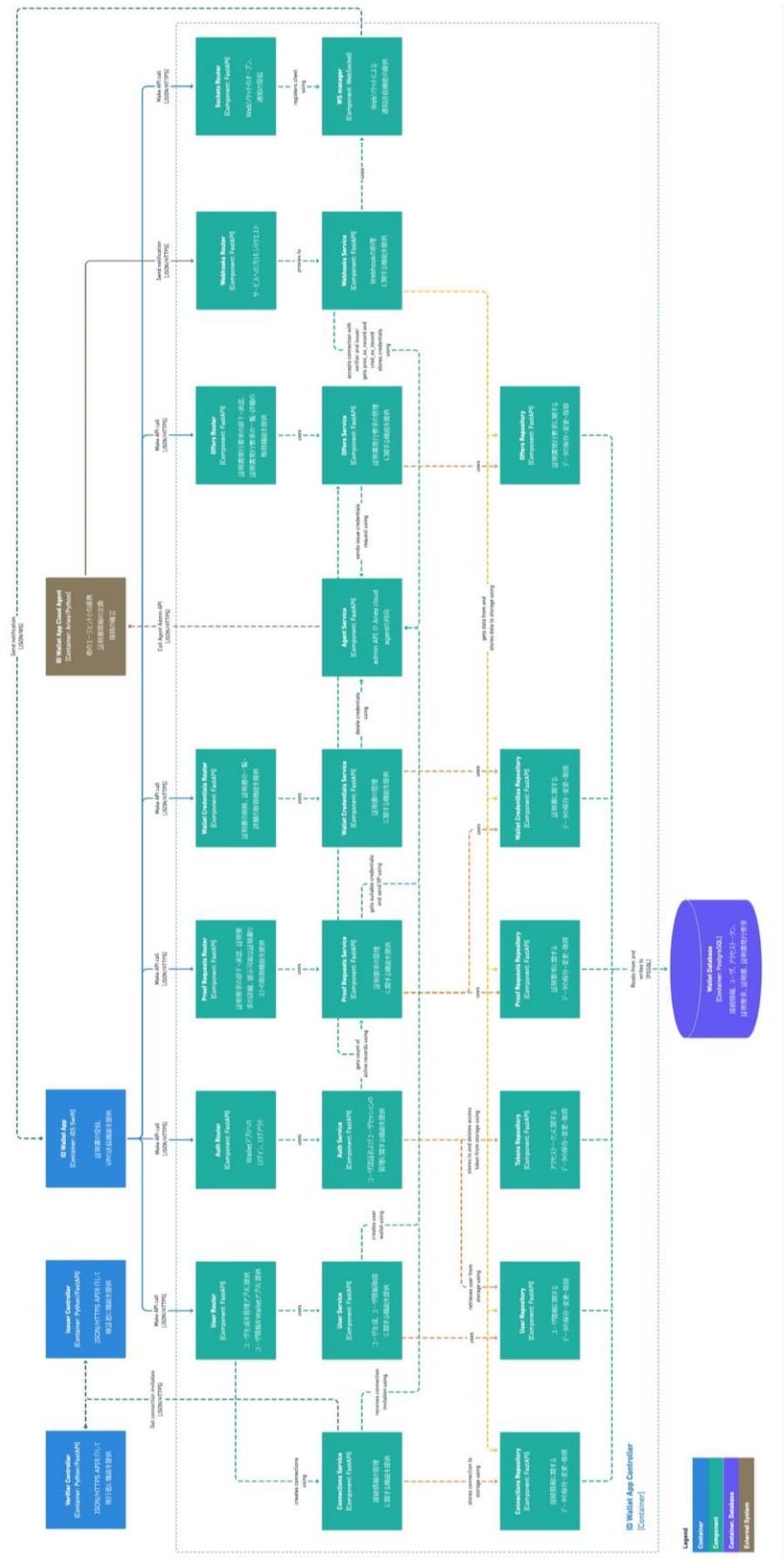

図 8 POC フェーズ - Wallet アプリコントローラのコンポーネント図

3.6 ATAM

ATAM（アーキテクチャ・トレードオフ分析法）は、アーキテクチャに関する意思決定を評価・分析するために適用できる。これは、特徴的な設計の選択に関連するトレードオフと結果を理解するのに役立つ。

コード	タイトル	トレードオフ	リスク
ADR-07	検証可能な証明書フォーマット	TD-01	
ADR-09	エージェント間のコネクションの確立	TD-02	
ADR-10	証明リクエストの構造	TD-03	RS-01
ADR-11	リアルタイムのステータス更新		RS-02

3.7 トレードオフ

コード	説明
TD-01	クレデンシャル発行に AnonCreds フォーマットを選択するという決定は、事前に実施された技術的実現可能性テストによって裏付けられている。このトレードオフは、システムの実証済みの互換性と AnonCreds との統合を強調するものである。しかし、特定の要件または業界標準によりよく合致する可能性のある代替クレデンシャル形式を模索し採用する能力が制限され、その面での柔軟性が制限される。
TD-02	Wallet ユーザの作成を自動化し、発行者および検証エージェントとの接続を確立することで、オンボーディングを合理化し、手作業によるエージェント接続への依存を減らす。このトレードオフは、自動化と簡素化の利点を強調する。しかし、このアプローチは、エージェント間のダイナミックな接続の確立に限られた労力しか割かれていない現在のフェーズに特化したものであることに注意する必要がある。
TD-03	IDWallet アプリに格納される「証明リクエスト」と「共助実績」の構造を一致させることで、スムーズな検証を実現することを目的としている。この決定は、検証フローの効率向上につながる一貫性の維持に重点を置いている。互換性のある構造を持つことで、大規模なマッピングや変換作業を行うことなく、データを容易に利用することができる。

3.8 リスク

コード	説明
RS-01	証明リクエストの構造を共助実績と一致させることから逸脱すると、複雑さが生じ、マッピングと変換の作業負荷が増大する。統合と互換性を確保するために、追加的な開発とテストが必要になるかもしれない。 **インパクト：**中 **確率：**低 **緩和策：**アプリケーションレベルで多様なクレデンシャル・スキーマのサポートを実装する。これには、大規模な修正または変換を行わずにさまざまなクレデンシャル・フォーマットに対応できるようにシステムを設計することが必要である。
RS-02	分散システムでウェブソケットを拡張すると、複雑な接続管理のリスクが生じる。タイムアウトの処理、健全性の監視、障害の管理はより難しくなり、システムのスケーラビリティとパフォーマンスに影響を与える可能性がある。 **インパクト：**中 **確率**：中 **緩和策**：今後のフェーズで、現在の実装をウェブソケットから Amazon SNS（Simple Notification Service）に変更することを検討する。SNS を利用することで、AWS（Amazon Web Services）がスケーリングと信頼性の高いメッセージ配信の確保に責任を持つ。この移行により、スケーラビリティが容易になり、メッセージ配信機能が強化される一方で、ウェブソケットと関連する複雑なインフラの管理負担が軽減される。

3.9 技術スタック

名称	バージョン	ライセンス	説明
Aries Cloud Agent python (ACA-Py)	0.8.1	Apache 2.0	Aries Cloud Agent Python（ACA-Py）は、Python による Aries Agent Framework の実装。Aries は、個人、組織、モノの間で安全で、プライベートで、相互運用可能なインタラクションを可能にする分散型 ID プロトコル。
Python	3.11	PSF	Python は多機能で読みやすいプログラミング言語であり、以下のようなコア機能を備えている： • シンプルさ：Python のすっきりした構文とインデントベースのブロック構造により、コードを書きやすく理解しやすい。 • 汎用性：Python は、ウェブ開発、データ分析、機械学習、自動化など、さまざまな目的に使用できる。 • 大規模なコミュニティとライブラリ：Python には膨大な開発者コミュニティとサードパーティライブラリの豊富なエコシステムがあり、あらかじめ構築されたツールやソリューションにアクセスすることができる。 • クロスプラットフォーム：Python は主要なオペレーティングシステムと互換性があり、異なるプラットフォーム間でシームレスにコードを実行できる。 • オブジェクト指向プログラミング（OOP）：Python はオブジェクト指向プログラミングをサポートしており、コーディングにモジュラーで再利用可能なアプローチを提供する。 • 統合：Python は他の言語と簡単に統合できるため、相互運用性を必要とするプロジェクトに柔軟に対応できる。 • ドキュメントとサポート Python は、学習とトラブルシューティングのための包括的なドキュメントと強力なオンラインコミュニティを提供している。
Aurora PostgreSQL	12.13	PostgreSQL ライセンス	Aurora PostgreSQL は、Amazon Web Services（AWS）が提供するフルマネージドのリレーショナルデータベースサービス。PostgreSQL のワーク

			ロードに高いパフォーマンス、スケーラビリティ、信頼性を提供するように設計されている。Aurora PostgreSQL は PostgreSQL と互換性があり、さまざまな PostgreSQL 拡張、ツール、ライブラリとの互換性を提供する。 Aurora PostgreSQL はオープンソースの PostgreSQL データベースをベースにしており、PostgreSQL ライセンスでリリースされている。PostgreSQL ライセンスは寛容なオープンソースライセンスであり、ユーザは自由にソフトウェアを使用、変更、配布することができる。このライセンスは、ユーザがライセンス料や制限なしに商用目的で Aurora PostgreSQL を使用する自由を提供する。 しかし、Aurora PostgreSQL 自体が Amazon Web Services (AWS)によって提供されるマネージドサービスとして、独自の価格モデルとサービス条件を持っていることに注意することが重要。基礎となる PostgreSQL ソフトウェアはオープンソースだが、Aurora PostgreSQL の AWS の管理とインフラストラクチャはプロプライエタリであり、AWS のサービス条件と価格設定に従う。AWS 上の Aurora PostgreSQL のユーザは、AWS のサービス規約を遵守する必要があり、サービスの利用に応じてコストが発生する可能性がある。
FAST API	0.95.2	MIT ライセンス	FastAPI は、標準的な Python 型ヒントをベースに API を構築するための、モダンで高速（高性能）な Web フレームワーク。 主な特徴は以下の通り： **速い**：驚くべき高性能。 **コーディングが速い**：機能開発のスピードを約 200%から 300%向上。 **短い**：コードの重複を最小限に抑える。各パラメータ宣言から複数の機能が得られる。バグが少ない。 **堅牢**：プロダクションですぐに使えるコードを取得。自動対話型ドキュメント付き。 **標準ベース**：API のオープンスタンダードに基づく（そして完全に互換性がある）：OpenAPI（以前は Swagger として知られていた）と JSON スキーマ。
SQLAlchemy	1.4.48	MIT ライセンス	SQLAlchemy は Python SQL ツールキットであり、アプリケーション開発者に SQL の完全なパワーと柔軟性を提供する Object Relational Mapper です。これは、効率的で高性能なデータベースアクセスのために設計された、よく知られたエンタープライズ

			レベルの永続化パターン一式を、シンプルで Pythonic なドメイン言語に適合させたものです。
websocket	11.0.3	BSD-3-Clause ライセンス	websockets は、Python で WebSocket サーバーとクライアントを構築するためのライブラリで、正しさ、シンプルさ、堅牢性、パフォーマンスに重点を置いている。 Python の標準的な非同期 I/O フレームワークである asyncio の上に構築され、デフォルトの実装はエレガントなコルーチンベースの API を提供する。
PyNaCl	1.5.0	Apache 2.0	PyNaCl は libsodium の Python バインディングです。これらのライブラリは、ユーザビリティ、セキュリティ、スピードを向上させることを目標としている。
Swift	5.8	Apache 2.0	Swift は、iOS、Mac、Apple TV、Apple Watch 用のアプリを作るためにアップルが作ったプログラミング言語。
Swift UI	4.0	MIT ライセンス	SwiftUI は、iOS、tvOS、macOS、watchOS 用のユーザインターフェースを構築するための Apple のフレームワーク。
Combine	-	-	Combine フレームワークは、時間をかけて値を処理するための宣言的な Swift API を提供する。これらの値は多くの種類の非同期イベントを表すことができる。Combine は時間と共に変化する値を公開するパブリッシャーと、パブリッシャーからそれらの値を受け取るサブスクライバーを宣言する。
Starscream	4.0.4	Apache 2.0	Starscream は Swift に準拠した WebSocket ライブラリ。
SwiftUIFlowLayout	1.0.5	MIT ライセンス	フローレイアウトは、画面の幅に応じて新しい「行」に分割され、ビューを順番に並べるコンテナである。左寄せのテキストブロックに置き換えることができ、すべての単語がビューになる。

4 ソリューション・インフラ

ソリューションインフラストラクチャは、ソリューションを構築するために必要なインフラストラクチャコンポーネントを記述・定義する。以下の図は、AWS 環境の上位/中位レベルのインフラコンポーネントを示している。

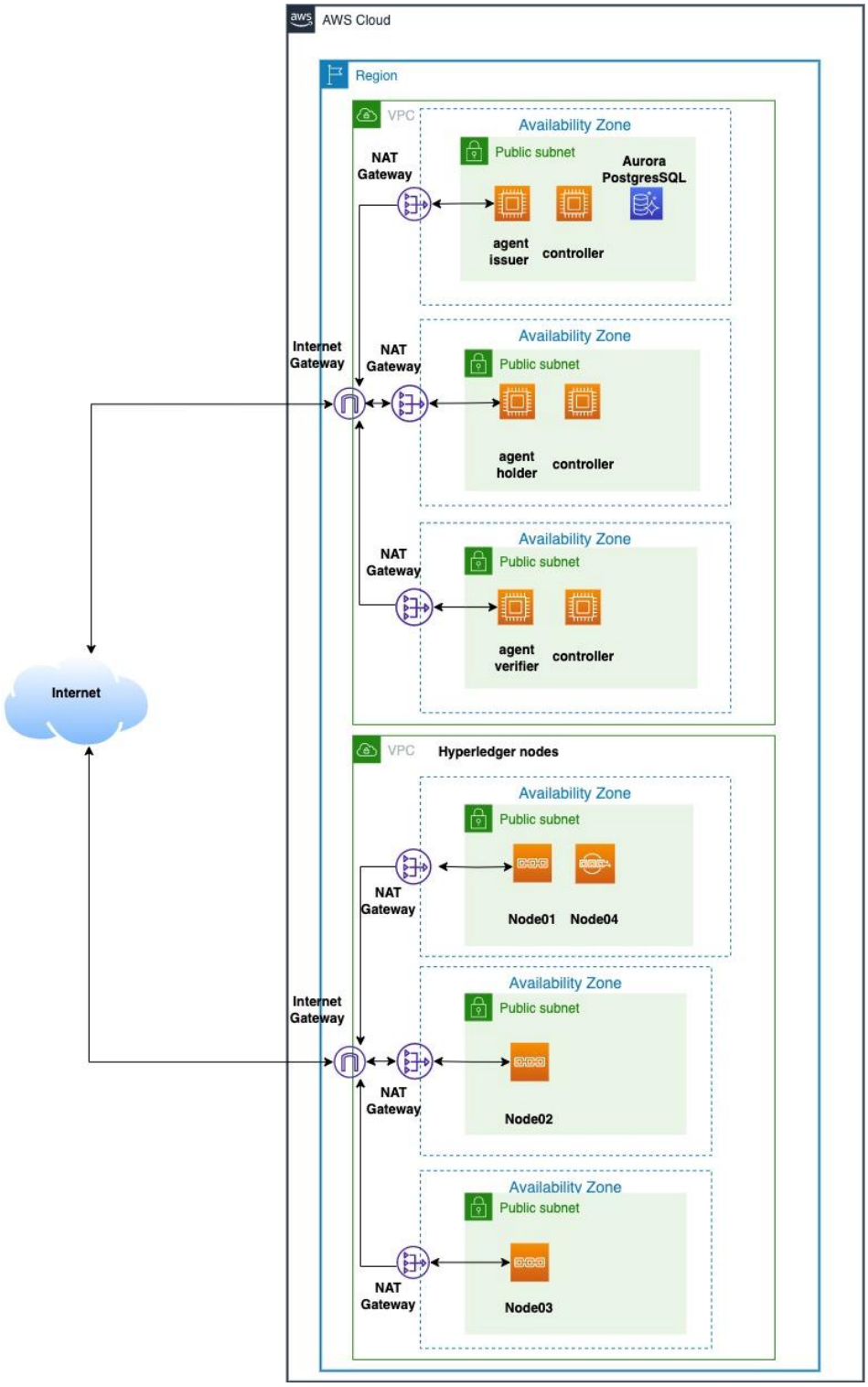

図 9 POC（概念実証）フェーズ - Wallet アプリコントローラのコンポーネント図

4.1 Dockerizing／コンテナ化

AWS における Dockerizing とは、Docker コンテナを使用して AWS クラウド環境にアプリケーションをパッケージ化し、デプロイするプロセスを指す。Dockerizing をよりよく理解するために、より簡単な用語に分解して説明します。

コンテナ：アプリケーション（マイクロサービス）の実行に必要なすべてを含む、軽量のスタンドアロンパッケージである。これにはコード、実行環境、ライブラリ、依存関係が含まれる。自己完結型のユニットとして機能し、Docker をサポートするマシンであれば簡単に移動して実行できる。

Docker：開発者がコンテナを作成・管理できる一般的なプラットフォーム。アプリケーション（マイクロサービス）をコンテナとして構築、パッケージ化、配布するためのツールとリソースを提供する。

Dockerizing：アプリケーション（マイクロサービス）を Docker コンテナに作成するプロセス。アプリケーションの依存関係を定義し、実行環境を設定し、すべてをコンテナ・イメージにパッケージ化する。

AWS：アマゾンが提供するクラウド・コンピューティング・プラットフォーム。クラウド上でアプリケーションを展開・管理するための多くのサービスやリソースを提供している。

AWS で Dockerizing を使用したい場合、それは Docker コンテナをデプロイして実行するために AWS の機能を活用することを意味します。AWS は、AWS Elastic Container Service (ECS)、Amazon ECR (Elastic Container Registry)、AWS Elastic Kubernetes Service (EKS)など、コンテナのオーケストレーションと管理のために特別に設計された様々なサービスとツールを提供している。

4.2 CI / CD

CI/CD システムを導入することで、開発ワークフローを強化し、手作業の介入を減らし、Amazon EC2 インスタンスへの効率的で信頼性の高いデプロイを確実にすることを目指している。

CI/CD パイプラインは、開発者が CodeCommit リポジトリに変更をプッシュするたびに開始される。AWS CodeBuild はサービスの Docker イメージを作成し、AWS ECR (Elastic Container Registry)に保存される。最後に、AWS CodeDeploy は、適切な Amazon EC2 インスタンスへのこれらの変更のデプロイを処理する。

このパイプラインをサポートするために、AWS CodeCommit、AWS CodeBuild、AWS CodeDeploy、AWS CodePipeline を含む様々な AWS サービスが利用される。さらに、AWS Artifact Bucket は、関連する成果物を保存し、全体的なデプロイプロセスを合理化するために採用されている。

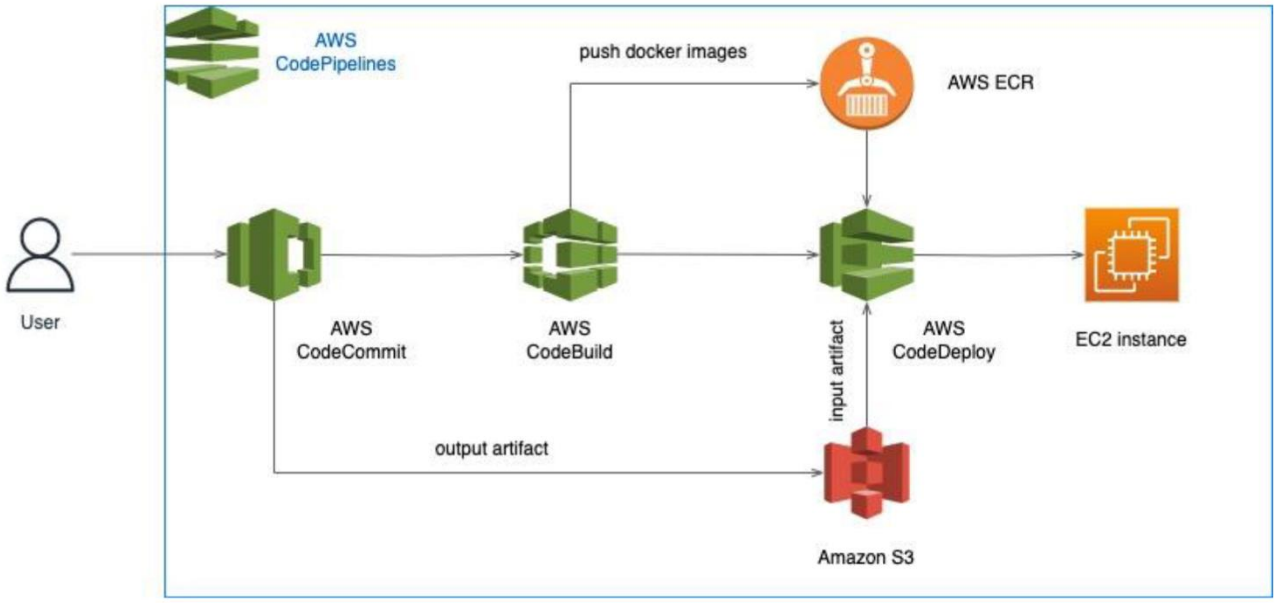

図 10 CI（継続的インテグレーション）／CD（継続的デリバリー）システム

付録 1

レイヤードアーキテクチャは、システムのコンポーネントを、それぞれが責任と依存関係を持つ明確なレイヤに編成するソフトウェア設計パターンである。レイヤードアーキテクチャーパターンは、レイヤー間の通信ルールを厳格にすることで、懸念事項の分離、モジュール性、保守性を促進する。

レイヤードアーキテクチャーでは、システムは通常 3 つの主要レイヤーに分けられる

プレゼンテーション層（ユーザインターフェース層とも呼ばれる）：

ユーザとのやり取りを処理し、ユーザに情報を提供する。
ユーザインターフェース、入力検証、ユーザ入力処理などのコンポーネントを含む。
データを取得または更新するために他のレイヤーと通信する。

ビジネスロジック層（アプリケーション層またはサービス層とも呼ばれる）：

システムの中核となるビジネスロジックとルールを含む。
ユースケースとビジネスオペレーションを実装する。
データの検証、ビジネスルールの実施、ワークフローの調整を管理する。
基礎となるデータレイヤーおよびプレゼンテーションレイヤーと通信する。

データレイヤー（パーシステンスレイヤーまたはインフラレイヤーとも呼ばれる）：

データの保存、検索、管理を行う。
データベースやファイルシステムなどのコンポーネントを含む。
ビジネスロジック層がデータにアクセスして操作するためのインターフェースを提供する。

レイヤードアーキテクチャパターンの主な特徴と利点は以下の通り

懸念事項の分離：各レイヤーはそれぞれ特定の責任を持ち、モジュール設計とメンテナンスの容易さを促進する。
抽象化とカプセル化：各レイヤーは他のレイヤーから実装の詳細を隠し、明確なインターフェイスを提供し、依存関係を減らす。
拡張性と柔軟性：レイヤーは独立して拡張できるため、特定のレイヤー内のコンポーネントの追加や変更が容易になる。
テスト容易性：レイヤー構造により、ユニットテストと各レイヤー内のコンポーネントの分離が可能になる。
再利用性：レイヤー内のコンポーネントは、システムのさまざまな部分や、同じアーキテクチャに従う他のシステムで再利用できる。