uKinnect Backend — Technical Overview

Product: uKinnect (multi-role care marketplace: family, caregiver, admin)

Repository: Node.js (ES modules), Express, MongoDB (Mongoose)

API base path: `/api/v1` (see `app.js`)

This document summarizes **HTTP APIs**, **third-party integrations** (with emphasis on **payments**), and **high-level application flows**. It is derived from the codebase layout and route files; adjust for environment-specific URLs and secrets.

1. Core stack

| Layer | Technology |

|--------|----------------|

| Runtime | Node.js (`"type": "module"`) |

| HTTP | Express 4 |

| Database | MongoDB via Mongoose 8 |

| Auth tokens | `jose` (JWT encrypt/decrypt in middleware) |

| Validation | Joi |

| Real-time | Socket.io (`src/services/socketio`) |

| Background jobs | BullMQ + Redis (`src/services/queue/paymentQueue.js`, `config/bullmqConnection.js`) |

| Caching / queue connection | `ioredis` / `redis` |

| Scheduling | `node-cron` (`src/services/cron/cronScheduler.js`) |

| File uploads | Multer (local + optional S3 patterns in dependencies) |

| PDF | Puppeteer (e.g. receipts where implemented) |

2. Third-party and external services

2.1 Stripe (payments & Connect)

- **SDK:** `stripe` npm package; `STRIPE_SECRET` (and related URLs in env for onboarding redirects).
- **Typical usage in project:**
- **Customers & cards:** Stripe Customer on users; PaymentMethods for cards (`CardController`, booking payment flows).
- **PaymentIntents:** Booking creation / confirmation, authorization, capture, refunds (`src/services/stripes/Payments.js`, `StripeController`, `BookingController`, `TopUpController`).
- **Stripe Connect:** Caregiver connected accounts, onboarding links, Express dashboard link (`StripeController`, `ConnectAccount.js`, top-up / payout helpers).
- **Webhooks:** e.g. `POST /api/v1/users/stripe/webhook/paymentIntent` (`StripeController.paymentIntentWebhook`).
- **Payout path:** After a booking is marked completed, work can be queued on **BullMQ** (`paymentQueue`) to run **`releasePaymentToCaretaker`** (`paymentQueue.js`).

2.2 Firebase Cloud Messaging (push)

- **SDK:** `firebase-admin` (`src/services/Notification/index.js`).
- **Behavior:** Persist notification in MongoDB (`notifications` collection), then if `switch_notify` is true on the receiver user, send FCM to each registered **`Device`** (`deviceToken` / `deviceType`).
- **Service account:** JSON path referenced in notification service (configure per environment).

2.3 Email (transactional) — summary

- **Nodemailer** over **SMTP**; HTML templates on disk. Details: **Section 8 — SMTP service & email templates**.

2.4 AWS S3 (optional / configured paths)

- **`@aws-sdk/client-s3`**, **`multer-s3`**: available for media uploads where S3 storage is wired (see dependencies and multipart configuration).

2.5 Redis

- Used for **BullMQ** job backend and related Redis helpers (`src/services/Redis`).

2.6 HTTPS (production)

- When `APP_ENV === "production"`, HTTPS server uses `SSL_KEY_PATH`, `SSL_CERT_PATH`, `SSL_CA_PATH` (`app.js`).

3. API surface (by router)

All routes below are prefixed with **`/api/v1`** unless noted.

3.1 Public / user app routes (`UserRoutes.js`)

Auth (no user JWT on most):

- `POST /users/auth/register` — signup (optional `deviceToken` / `deviceType`).
- `POST /users/auth/login` — login; checks `is_deleted`, `is_active`; device linking.
- `POST /social-login` — OAuth-style login with provider token.
- `POST /users/auto-login` — token-based auto login.
- `POST /users/auth/forgot-password`, `reset-password`, `resend-otp`, `verify-otp`.

Public webforms:

- `POST /contact` — contact form.
- `POST /newsletter` — waitlist / newsletter.
- `POST /career` — career application (multipart).

Stripe (mixed public / callback):

- Stripe Connect return/refresh/success/complete URLs, webhook, express link (see routes under `/users/stripe/...` and `/users/get-express-link`).

Authenticated (`/users` + `UserMiddleware`):

- Profile, password, switch mode, questionnaire (`/question/...`), search (`/search/caretaker...`).
- **Bookings:** submit, confirm, pay, list by family/caretaker status, mark status, cancel, check-in/out, on-the-way confirmation, disputes, stats, leave.
- **Stripe / wallet:** bank transfer, onboarding, top-up, family transaction history, caretaker transfer.
- **Chat:** list, detail, send message (multipart), conversation, search.
- **Cards:** `POST /createCard`, `GET /getCards`.
- **Payment history** (`/payment-history...`).
- **Notifications** (in-app list, mark read).
- **Block user**, incident reports, user reports, support hub, settings pages, reviews (some review routes also under `/review` prefix — see below).

3.2 Admin routes (`AdminRoute.js`)

Public admin auth:

- `POST /admin/auth/register`, `login`, forgot/verify/resend OTP, reset password (shared validators with user where applicable).

Authenticated (`/admin` + `AdminMiddleware`):

- FAQs, CMS-style settings (family/caretaker help, about, privacy, terms), dashboard analytics, reports analytics.
- Care applications (CNA/RN list, detail, status), webform submissions (contacts, waitlist).
- Bookings: dashboard overview, recent, **section** list (status filters), detail, trigger reminders.
- Payments: dashboard stats, transactions list/export/detail/receipt PDF.
- Disputes and **user reports** (moderation).
- Caregivers / families CRUD-style lists, exports, detail; soft delete caregiver.
- **`PATCH /admin/users/:userId/is-active`** — toggle `is_active` for non-admin users.
- Notifications: list all, detail, delete, test push.
- Incident / user report **reason** management, reviews moderation, support content (multipart).

3.3 Website embed routes (`WebsiteRoute.js`)

- Prefix: **`/api/v1/website`** + **`BearerMiddleware`** (bearer token for embedded site).
- `POST /website/cna-application` — CNA form (multipart + validation).
- `POST /website/rn-application` — RN form (multipart + validation).

3.4 Review routes (`ReviewRoute.js`)

- Prefix: **`/api/v1/review`** (all behind `UserMiddleware`).
- Create/update/get/delete review endpoints (see `ReviewRoute.js`).

3.5 Task routes (`TasksRoute.js`)

- Prefix: `**/api/v1/tasks**`.
- Mix of **AdminMiddleware** (e.g. create task template) and **UserMiddleware** (user task progress, family tasks, etc.) — see `TasksRoute.js`.

3.6 Static / misc

- `GET /images/...` — static images from `src/services/uploads/images`.

4. Payment services — implemented concepts

1. **Stripe Customer** — Families (and app users) use a Stripe customer id for saved cards and PaymentIntents.
2. **PaymentIntents** — Used around booking pay flows: authorize/capture, status sync via webhooks where implemented.
3. **Booking economics** — Model stores amounts, platform fee, caretaker rate, breakdown, `paymentStatus`, `payment_id`, optional `paymentAtCreation`, `paymentDueAt` for legacy/unpaid paths.
4. **Stripe Connect** — Caregivers complete Connect onboarding; payouts / transfers use connected account ids (`stripe_account_id`, etc.).
5. **Refunds / reversals** — Booking cancel and dispute paths integrate Stripe refund/transfer logic (see `Payments.js`, dispute controller, booking cancel handlers).
6. **Transaction history** — `TransactionHistory` model records intents/invoices for admin and user history APIs.
7. **Queued payout** — **BullMQ** worker processes `releasePaymentToCaretaker` for completed bookings (`paymentQueue.js`).
8. **Top-ups / wallet-style flows** — `TopUpController` exposes deposits, connect account checks, family/caretaker distribution views (Stripe-backed).

5. Application flow (conceptual)

5.1 End user (family / caregiver)

9. **Register / verify OTP / login** — JWT issued; optional **device** registration for push.
10. **Inactive account** — If `is_active === false`, login blocked with message including **`ADMIN_CONTACT_EMAIL`** (configurable).
11. **Profile & questionnaire** — Questions by `type` (family/caretaker); answers stored for matching.
12. **Search & discovery** — Caregiver search / profile by answers.
13. **Booking** — See **§9** for HTTP sequence; optional **`bookingLocation`** on submit; caregiver accept/reject; pay when required; status transitions.
14. **Visit lifecycle** — **On-the-way** confirmation, **check-in / check-out** (caregiver, geo + time rules), reminders via **cron**.
15. **Completion & payout** — Mark complete; queue/trigger caregiver payout; **reviews** (see **§10**).
16. **Disputes** — Family raises **booking dispute** (**§11**); admin resolves with Stripe refund or transfer; notifications.
17. **Messaging** — Socket + REST; push on new message when enabled.

5.2 Admin

18. **Admin login** — Same inactive rule if `is_active` false.
19. **Operations** — Monitor bookings, payments, disputes, reports, reviews, CMS content, care applications, webform leads.
20. **Notifications** — Admin API lists **all** notifications in DB (any receiver); push to admins for selected operational events when `notifyUser` targets admin users.

5.3 Website (public embed)

21. Bearer-protected **CNA/RN** applications submitted to backend; admins notified (in-app + push when configured).

6. Security & middleware (summary)

- **UserMiddleware** — Validates user JWT for `/users/*` protected routes.
- **AdminMiddleware** — Validates admin JWT for `/admin/*` protected routes.
- **BearerMiddleware** — Website embed bearer for `/website/*`.
- **Response middleware** — Normalized `res.success` / `res.error` envelopes.
- **CORS** — Currently permissive (`*`) in `app.js` (tighten for production as needed).

7. Environment & operations (checklist)

Configure at minimum (names may vary by deployment):

- MongoDB connection string
- `STRIPE_SECRET`, Stripe webhook signing secret (if used), public URLs for Stripe redirects
- Redis URL for BullMQ / Redis client
- Firebase Admin credentials path / JSON
- Email SMTP (`MAIL_HOST`, `MAIL_PORT`, `MAIL_USERNAME`, `MAIL_PASSWORD`, `MAIL_FROM_NAME`, `MAIL_ADMIN`, `SMTP_EMAIL_TO`, …)
- `PORT`, `APP_ENV`, SSL paths for production
- `ADMIN_CONTACT_EMAIL` (optional; defaults in `config/constant.js`)

8. SMTP service & email templates

8.1 Configuration (`config/mail.js` + environment)

| Variable (typical) | Purpose |

|--------------------|---------|

| `MAIL_HOST` | SMTP server hostname |

| `MAIL_PORT` | Defaults to **465** in config; `secure: true` (SSL) |

| `MAIL_USERNAME` / `MAIL_PASSWORD` | SMTP auth |

| `MAIL_FROM_NAME` | Display name on outbound mail |

| `MAIL_USERNAME` | Also used as `from.address` in `mailerService.js` |

| `MAIL_ADMIN` | Inbound recipient for contact / waitlist / some admin alerts |

| `SMTP_EMAIL_TO` | Career form recipient (see `sendcareerEmailTemplate`) |

| `BASE_URL` | Used when embedding asset URLs in HTML (e.g. CNA signatures) |

Transport: `nodemailer.createTransport(mailConfig)` in **`src/services/mailerService.js`**. **`sendEmails(to, subject, html, attachments, callback)`** sends **HTML** (`html` field), optional **attachments** (e.g. career uploads with `path` / `cid`).

8.2 Template pipeline

22. **`getFileContent(path)`** reads **`src/services/emails/*.html`** as UTF-8 text.
23. Helpers in **`src/services/helper/index.js`** replace placeholders (e.g. `{{otp}}`, `{{name}}`) with runtime values.
24. **`sendEmails`** delivers the rendered HTML.

8.3 HTML templates (`src/services/emails/`)

| Template file | Typical use |

|----------------|-------------|

| `otpVerification.html` | OTP for verify / forgot / login nudge (`sendOTPEmail`) |

| `contact.html` | Contact form submission to admin (`sendContactFormDetails`) |

| `waitlist.html` | Waitlist / newsletter signup to admin (`sendNewsLetterEmailTemplate`) |

| `career.html` | Career application to ops (`sendcareerEmailTemplate`, attachments) |

| `newsLetter.html` | Newsletter-style content (if referenced by flows) |

| `stripe-success.html` | Stripe-related success messaging (where wired) |

| `CNABackgroundCheck.html` | CNA background check email + PDF generation path |

| `CNAContractorAgreement.html` | CNA contractor agreement |

| `RegisteredNurseApplication.html` | RN application correspondence |

Note: In-app **push** notifications use **Firebase** (separate from SMTP); many user journeys use **both** (e.g. booking events + optional email).

9. Booking flow — APIs (family / caregiver)

Base path: **`/api/v1/users`** (all below require **`UserMiddleware`** unless stated).

9.1 Primary flow — family books caregiver (direct booking + payment at creation)

Typical sequence (exact branching depends on `paymentStatus`, `paymentAtCreation`, and legacy paths in `BookingController`):

25. **`POST /users/booking`** — **`submitBooking`**

- Body includes caregiver id, schedule, **`paymentMethodId`**, Stripe customer on user; optional **`latitude`**, **`longitude`**, **`address`** → stored as **`bookingLocation`** when any provided.
- Creates **`Book`**; runs **Stripe** payment intent path via **`createPaymentBooking`** (authorize / pay-at-creation); records **`TransactionHistory`** on success; **notifies** family + caregiver (`notifyUser`).

26. **`POST /users/booking/confirm`** — **`confirmBooking`**

- Confirms / syncs payment state for a booking (used where booking is created or held pending confirmation).

27. **`POST /users/booking/mark/status` — `markBookingStatus`**

- Caregiver **accept** (`upcoming`) / **reject**; family may **reject**; **completed** triggers payout queue hooks; notifications on transitions.

28. **`POST /users/pay-for-booking` — `payForBooking`**

- Used when payment is **after** accept or for unpaid legacy bookings (pays Stripe, updates booking).

29. **`GET /users/booking/family/:status`** / **`GET /users/booking/caretaker/:status`** — list bookings by **`status`** (`pending`, `upcoming`, `in_progress`, `completed`, `cancelled`, etc.).
30. **`GET /users/booking/:id`** — single booking detail (family or caregiver on booking).
31. **Visit lifecycle**

- **`POST /users/booking/:bookingId/on-the-way-confirmation`** — caregiver yes/no; notifies family + admins (push where configured).
- **`POST /users/booking-check-in-out`** — caregiver **check-in** / **check-out** (location + timezone in body); updates **`checkIns`** / **`checkOuts`**, buffer pricing on first check-in when applicable.

32. **`POST /users/booking/apply-leave`** — leave request on a booking (validation in controller).
33. **Cancellation — `POST /users/booking/:id/cancel/family`** or **`…/cancel/caretaker`** — refunds / notifications per role.
34. **`GET /users/caretaker-booking-stats`** — caregiver dashboard stats.

9.2 Alternate path — caregiver "booking request" (`BookingRequest` model)

- **`POST /users/booking-requesting/caretaker`** — caregiver ties a **`bookingId`** to a **`BookingRequest`**.
- **`POST /users/booking-requesting/mark/status`** — family (or permitted user) **accepts/rejects** request; on accept, **wallet / `PaymentDistribution`** and related payment rules run (see controller for full conditions).

9.3 Cards (Stripe customer)

- **`POST /users/createCard`** — attach / default payment method.
- **`GET /users/getCards`** — list saved cards (Stripe `paymentMethods`).

10. Reviews & ratings

10.1 Data model (`src/models/Review.js`)

- **Reviewer:** `reviewerUserId`, `reviewerRole` (`family` \| `caretaker`).
- **Target:** `targetUserId`, `targetRole` (`family` \| `caretaker`).
- **Content:** `rating` (1–5), `reviewText`, **`adminApprovalStatus`** (`pending` \| `approved` \| `rejected`), `isDeleted`.
- **Constraint:** One review per **(reviewerUserId, targetUserId)** pair (enforced in controller).

10.2 User APIs (prefix **`/api/v1/review`**, **`UserMiddleware`**)

| Method | Route | Purpose |

|--------|--------|---------|

| POST | `/review/create-review` | Create review (`targetUserId`, `targetRole`, `rating`, `reviewText`) |

| POST | `/review/update-review` | Update own review (`id` query, body fields) |

| GET | `/review/get-reviews?userId=` | List **approved**, non-deleted reviews for a profile |

| GET | `/review/review-by-id?_id=` | Fetch one review (populated) |

| POST | `/review/delete-review` | Soft-delete (`isDeleted`) own review |

Note: `createReview` in code sets **`adminApprovalStatus: "approved"`** on save while the success message still refers to admin approval — align product copy with code or change code to `pending` if moderation is required before visibility.

10.3 Admin moderation (`/api/v1/admin` + **`AdminMiddleware`**)

- List / stats / approve / reject / delete reviews under **`/admin/reviews...`** (`AdminReviewController`).
- **`getReviewsForUser`** only returns **`approved`** reviews for public-style listing; booking list endpoints may combine **approved** aggregates with **"my pending review"** fields where implemented.

11. Booking disputes — scenario & APIs

11.1 Who can file

- **Family only** — `createDispute` checks `booking.family` matches authenticated user.

11.2 Create dispute (user)

| Method | Route | Body / upload |

|--------|--------|----------------|

| POST | `/users/booking/dispute` | **`bookingId`**, **`description`**; optional **file** (`multipart`, field used as document) |

Effects:

- Inserts **`BookingDispute`** (links `bookingId`, `filedBy`, `description`, document path if uploaded).
- Sets booking **`status`** to **`dispute_raised`**.
- **`notifyUser`** → **caregiver** only (type **`dispute_created`**, redirect **`dispute_details`**). *(Admin push for new disputes is not in this handler; admin uses dashboard/APIs.)*

11.3 List & read (user)

- **`GET /users/booking/disputes`** — disputes for bookings where user is **family or caretaker**.
- **`GET /users/booking/dispute/:id`** — dispute detail.

11.4 Resolve dispute (admin)

| Method | Route | Body |

|--------|--------|------|

| PATCH | `/users/booking/dispute/:id/status` | **`{ "status": "accepted" \| "rejected" }`** — requires **`adminDisputeMiddleware`** (admin token on user-route pattern used here) |

Payment outcome (`BookingDisputeController.updateDisputeStatus`):

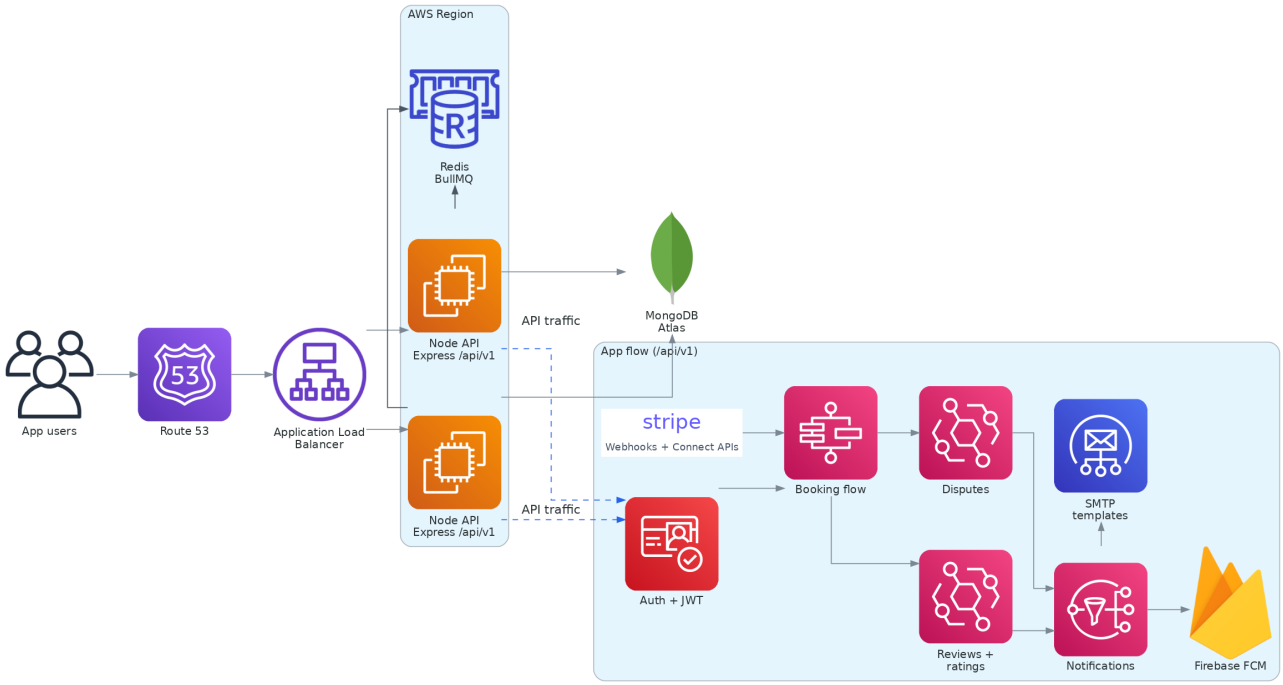
- **`accepted`** (family wins dispute): if booking was **`paid`**, run **`processDisputeRefund`** → booking **`paymentStatus`** → **`refunded`**.
- **`rejected`** (dispute denied): if **`paid`**, run **`processDisputeTransfer`** to caregiver → **`paymentStatus`** → **`transferred`**.
- **Notifications:** `notifyUser` to **family** and **caretaker** with dispute resolution copy.

11.5 Admin console (separate admin router)

- **`GET /admin/disputes`**, **`GET /admin/disputes/:disputeId`**, **`PATCH /admin/disputes/:disputeId/status`** — admin dispute list/detail/status (`AdminDisputeController`).

End-to-End Technical Architecture

Topology diagram: users, Route 53, load balancer, Node API, Redis/BullMQ, MongoDB Atlas, app flows (auth, Stripe, booking, notifications, FCM, SMTP). Source asset: docs/diagrams/ukinnect-01-overall-topology.png.



uKinnect Flutter App — Technical Architecture

This section documents the mobile client (Flutter) architecture: navigation, state, networking, and integrations. Numbered headings match the backend overview above.

1. Project overview

uKinnect is a Flutter application connecting families with caregivers. It supports authentication, dual roles (family and caretaker), booking lifecycle, payments (Stripe), chat, tasks, disputes, notifications, and maps integration.

2. Architecture analysis

Feature-based monolithic Flutter structure. GetX is used for state management, dependency injection, and navigation. Controllers manage business logic; repositories handle API abstraction.

3. State management

GetX is used with global and feature-level controllers. Reactive programming via Rx and Obx. GlobalVariable is used for session-level data.

4. Navigation

- Entry: GetMaterialApp with splash entry.
- Routing: Get.to, Get.off, Get.offAll.
- Flows: PageView for multi-step flows.
- Deep links: app_links.

5. API and data layer

- Primary HTTP client: http via NetworkApiServices.
- Dio used in limited cases.
- Repositories structured per domain.
- Models follow JSON serialization.

6. Backend communication

- REST APIs with Bearer token authentication; base URLs via Appurls.
- Socket.IO for real-time events (messages, booking updates).

7. Local storage

- SharedPreferences for tokens, user data, and flags.

8. Dependency management

- Get.put globally and locally within features.

9. Key modules

- Auth, role selection, family and caretaker modules, booking, payments, chat, tasks, forms, notifications, disputes, support, maps, media, deep links.

10. Third-party integrations

- Firebase, Stripe, Google Maps, Places API, Socket.IO, WebView, UI libraries.

11. Security practices

- Bearer token authentication.
- Token refresh handling.
- FCM token management.

12. Build and environment

- Configuration via Appurls constants.
- Portrait orientation.
- Theming via GetMaterialApp.
- Responsive layout with ScreenUtil.

13. Folder structure

Path	Purpose
lib/main.dart	App bootstrap
lib/FamilyFolder/	Family module
lib/caretakerfolder/	Caretaker module
lib/Repositories/	API layer
lib/Network/	Network services
lib/models/	Data models
lib/utils/	Utilities
assets/	Media resources