flutter 앱 핵심 정리

👫 공통

1. 로그인

1) 흐름

(1) 아이디·비번·역할(P/D) 입력

(2) `AuthViewModel.loginUser()` 가 `POST /auth/login` 호출

(3) 응답의 `access_token` 을 FlutterSecureStorage에 저장

(4) `currentUser` 세팅 → `notifyListeners()`

(5) 역할에 따라 GoRouter로 이동: `D → /d_home` , 그 외 → `/home`

2) 연관 dart별 주요 사항

(1) LoginScreen: 입력/역할 토글, 버튼 클릭, 에러 SnackBar

(2) AuthViewModel: 네트워크 호출, 토큰 저장/삭제, `currentUser` 관리

(3) GoRouter: 초기 경로 `/login` , 성공 후 스택 정리(`context.go(...)`)

3) 요약 표

구역	핵심 포인트	사용 기법/패턴
입력/역할 선택 (LoginScreen)	아이디·비번 `TextEditingController` , 역할 `P/D` 토글	`StatefulWidget` + `GestureDetector` , `BoxDecoration` (선택 강조), `trim()`
로그인 호출	`AuthViewModel.loginUser(regi`	`Provider(context.read)` ,

	`sterId, password, role)` 실행	`async/await`, `http.post`(JSON)
토큰 저장	응답 `access_token` → 안전 저장	`FlutterSecureStorage.write(key: 'access_token', value: token)`
사용자 상태	`currentUser` 세팅 후 화면 갱신	`ChangeNotifier` + `notifyListeners()`
성공 라우팅	`role=='D'` → `/d_home`, 그 외 → `/home`(+ `extra: {userId}`)	**GoRouter** `context.go(...)`(스택 정리)
실패 처리	서버 `message` 또는 공통 문구 노출	`try/catch`, `SnackBar`, `errorMessage` 보관
서버 계약	요청: `{register_id, password, role}` / 성공: `{access_token, user{role, registerId…}}`	REST 컨트랙트 고정, DTO(`User.fromJson`)

보안 포인트	후속 요청에 `Authorization: Bearer <token>` / 로그아웃 시 토큰 삭제 / 운영은 HTTPS	헤더 주입, `secureStorage.delete`, TLS
웹/반응형	웹에서 카드 폭 제한(보기 좋게)	`kIsWeb` + `ConstrainedBox(maxWidth: 450)`
체크리스트	`baseUrl` /경로 정확, `trim()` ·공란 검사, 엔터 로그인, 에러 우선순위, `go` 사용	`TextInputAction.done`, 입력 검증 (필요 시 `Form` / `validator`)

2. 회원 가입

1) 흐름

(1) HTML 약관 2종 로드 → 체크 2개 모두 ON일 때만 “다음” 활성화 → ** `/register` **로 이동

(2) 역할(P/D) 선택 → 이름/성별/생년월일/전화/아이디/비번 입력

(3) **아이디 중복검사**: `AuthViewModel.checkUserIdDuplicate(id, role)`

(4) **유효성 검사**: 이름(한글만), 비번=확인, 생년월일(현재~최대 125세), 전화 하이픈 포맷

(5) **가입 요청**: `AuthViewModel.registerUser(userData)` → `POST /auth/register`

(6) 성공 시 스낵바 후 `context.go('/login')`

2) 연관 dart별 주요 사항

(1) agreement_screen : 약관 html 불러오기, 체크박스, 동의후 회원가입 페이지로 이동

(2) register_screen : 중복 검사, 생년월일 유효성 검사, 가입 요청 페이로드 등

(3) auth_veiwmodel.dart : 서버 요청 등

3) 요약 표

구역	핵심 포인트	실제 구현/기법
약관 화면(Agreement)	HTML 2종(이용약관/개인정보) 탭 렌더링, 체크 2개 모두 ON 시 “다음”	`flutter_html`, `rootBundle.loadString(...)`, `TabBar/TabBarView`, 체크 2개 상태 AND, `context.push('/register')`
역할 선택	환자(P)/의사(D) 카드 선택, 선택 시 아이디 중복 상태 초기화	카드 탭 → `_selectedRole` 변경, `_isIdChecked=false`, `_isDuplicate=true`, `_registerIdController.clear()`
아이디 중복검사	길이≥4, 서버 `exists==false` 여야 통과	`checkUserIdDuplicate(id, role)`, `duplicateCheck`

		`ErrorMessage` 처리, 로딩 시 버튼 비활성
입력 유효성	이름 한글만, 비번=확인, 생년월일(현재~125년), 전화	`TextFormField.validator`, 한글 정규식, **동적 일 수 계산**, 커스텀 포맷터
입력 포맷터	전화 자동 하이픈, 숫자만, 길이 제한 / 아이디 영숫자·길이 제한	`_PhoneNumberFormatter()`, `digitsOnly`, `LengthLimitingTextInputFormatter`
가입 요청	**payload 키**: `register_id`, `password`, `name`, `gender`, `birth`, `phone`, `role`	`registerUser(userData)` → 201 성공 시 `null` 반환, `context.go('/login')`
서버 계약	중복검사 GET /auth/check-username, 가입 POST /auth/register	200 `{exists}`, 201 성공 / 실패 시 `{message}` 파싱하여 스낵바 노출
UX 보호장치	중복검사 안 했거나 중복이면 제출 차단	`_isIdChecked`와 `_isDuplicate` 검사 후 진행

3 . 아이디/ 비밀 번호 찾기

1) 흐름

(1) 아이디 찾기 : /find_id → /find-id-result

- 이름 + 전화번호 입력(자동 하이픈 포맷)

- 찾기 버튼 → FindIdViewModel.findId(name, phone) 호출
- 성공시 `foundId` 를 **extra**로 넘겨 ** `/find-id-result` **로 이동
- 결과 화면에서 찾은 아이디 표시 → **로그인으로 이동**

(2) 비밀 번호 찾기 : /find_password → /find-password-result

- 이름 + 전화번호 입력(자동 하이픈 포맷)
- 찾기 버튼 → FindPasswordViewModel.findPassword(name, phone) 호출
- 성공 시 /find-password-result 이동
- 결과 화면에서 “비밀번호 재설정 링크가 이메일로 전송됨” 안내 → **로그인으로 이동**

2) 연관 dart별 주요 사항

(1) find_id_screen : 아이디 찾기 트리거

(2) find_password_screen : 비밀번호 찾기 트리거

3) 요약 표

구역	핵심 포인트	사용 기법/패턴
아이디 찾기 화면	이름 + 전화 입력 → 아이디 조회	`FindIdViewModel.findId` , 로딩 스피너, 에러 텍스트
아이디 결과 화면	찾은 아이디 보여주기	단순 네비게이션
비밀번호 찾기 화면	이름 + 전화 입력 → 재설정 메일 발송	`FindPasswordViewModel.findPassword` , 성공/실패 메시지 표시
비밀번호 결과 화면	“재설정 링크를 이메일로 전송” 안내	단순 네비게이션
전화번호 입력	한국형 자동 하이픈	`digitsOnly` + `KoreanPhoneNumberFormatter`

Provider 주입	아이디 찾기: **라우터에서** 주입 / 비번 찾기: **화면 내부**에서 생성	`ChangeNotifierProvider`
UX 디테일	웹 폭 제한, 통일된 카드/그라데이션 배경	`kIsWeb` + `BoxConstraints(maxWidth: 450)`, `Card`

🦽 환자

1. 메인 화면

1) 흐름

(1) 상단 AppBar : 좌측 마이페이지(/mypage), 우측 알림 버튼(+배지)

(2) 바디 : 그라데이션 배경 + 2x2 기능 타일

- AI 진단 → `/survey`
- 실시간 예측하기 → `/diagnosis/realtime` *(웹에선 비활성)*
- 진료 기록 → `/history`
- 주변 치과 찾기 → `/clinics`

(3) 우측 상단 알림 팝업 토글/ 닫기

(4) 뒤로가기 → 앱 종료 확인 다이얼로그

(5) 하단 탭바(네비게이션바) : 챗봇/홈/마이페이지 \

2) 연관 dart별 주요 사항

(1) home_screen : 타일 → 라우팅, 알람 배치/팝업+토글,뒤로가기 종료 확인등

(2) main_scaffold : 하단 네비게이션 바 분기

(3) rouder : `/home` 진입 시 `router.dart` 에서 ** `state.extra` **로 넘어온 `userId` 를 `HomeScreen` 에 주입

3) 요약 표

구역	핵심 포인트	사용 기법/패턴
상단 AppBar	좌: 마이페이지, 우: 알림 토글 + 배지	`IconButton`, `Positioned` 배지, `setState`
배경/레이아웃	상→하 **그라데이션** + 스크롤 컨텐츠	`Stack` + `SafeArea` + `SingleChildScrollView`
기능 타일(2×2)	**AI 진단**	카드+InkWell, 동적 `iconSize/fontSize`
기능 타일(2×2)	**실시간 예측하기** *(웹 비활성)*	`kIsWeb ? null : push(...)`
기능 타일(2×2)	**진료 기록**	고정 라우팅
기능 타일(2×2)	**주변 치과 찾기**	고정 라우팅
알림 팝업	리스트 표시/닫기(밖 터치 시 닫힘)	`GestureDetector`로 외부 탭 감지, `Positioned` 팝업
뒤로가기	종료 확인 다이얼로그	`WillPopScope` + `AlertDialog`
하단 탭바	챗봇/홈/마이페이지 분기	**NavigationBar (M3)**, 경로 기반 인덱싱, 텍스트 스케일 클램프

2 . 챗봇

1) 흐름

(1) UI 표시: 상단 AppBar(대화 초기화, 알림 토글) → 메시지 리스트(말풍선+이미지) → 하단 입력창/전송

(2) **전송**: 입력 → `ChatbotViewModel.sendMessage()` 호출 → 서버 `POST /chatbot`

(3) **응답 처리**: 봇 텍스트 + (선택) 이미지 URL 표시

- `inference_result_id`가 오면 `/inference_results/{id}`로 **추가 이미지 URL** 보강

(4) **이미지 마스크 토글**: “충치/치아/위생”, “치석/보철물”, “치아번호” 중 하나 선택 → 해당 마스크 이미지 표시

(5) **상태관리/DI**: `AuthViewModel` 토큰/유저ID 주입(ProxyProvider). 유저 바뀌면 메시지 초기화+인사말.

2) 연관 dart별 주요 사항

(1) chatbot_screen : 입력 → 전송 트리거 , 마스크 토글 & 이미지 선택

(2) chatbot_viewmodel : 서버 호출 + 이미지 URL 보강

(3) chat_bubble : 말풍선 길이 제한 및 말풍선 디자인

3) 요약 표

구역	핵심 포인트	사용 기법/패턴
입력/전송	엔터/버튼 → 전송, 자동 스크롤	`TextField.onSubmitted`, `GestureDetector`, `ScrollController`

서버 통신	`POST /chatbot` (토큰 필요)	`http.post`, `Bearer` 토큰 (`FlutterSecureStorage`)
응답 확장	`inference_result_id` 있으면 전체 이미지 URL 재조회	후처리로 `image_urls` 병합
이미지 표시	마스크 토글 3종 중 택1 → 해당 모델 이미지 표시	스위치 토글, `setState`
말풍선	사용자/봇 말풍선, 마크다운 지원	`flutter_markdown`, 커스텀 Tail Painter
로딩/에러	전송 중 로딩 셀, 네트워크 실패/인증없음 안내	`isLoading` 상태, Snack/텍스트
초기화	유저 변경 시 대화 초기화+인사	`AuthViewModel` 리스너, 초기 인삿말 삽입
라우팅	사용자 셸 내부 `/chatbot`	**GoRouter**

3. 마이페이지/ 회원 정보 수정

1) 흐름

(1) **마이페이지**(`/mypage`) 진입 → 프로필/기록 등 표시(라우터 상 경로 확인)

(2) 정보 수정 누르면 → 재인증 화면(/reauth)에서 비밀번호 확인

(3) 재인증 성공 시 → 프로필 수정(/edit-profile)

(4) 수정 제출 → `PUT /auth/update-profile` → 결과 화면 (`/edit_profile_result`)에서 성공/실패 안내 → 성공 시 마이페이지로 복귀

2) 연관 dart별 주요 사항

(1) reauth_screen : 재인증 → 편집화면 진입

(2) edit_profile_screen : 초기값 주입 & 제출

(3) auth_viewmodel : 실제 api 호출(재인증, 프로필 수정)

(4) edit_profile_result_screen : 결과 화면 → 복귀

3) 요약 표

구역	핵심 포인트	사용 기법/패턴
마이페이지	개인 정보/기능 허브(경로만 확인됨)	라우터에서 `baseUrl` 전달
재인증	비번 확인 후만 프로필 수정 진입	`AuthViewModel.reauthenticate()`
프로필 수정	현재 값 프리필 → 검증 → 전송	`Form` + `TextFormField`, `Provider`, `setState`
검증/포맷	이름(한글), 비번(선택·6+), 생년월일(YYYY-MM-DD·110년 제한·미래 금지), 전화(자동 하이픈)	`DobValidator`, `_PhoneNumberFormatter`
전송 페이로드	`register_id, name, gender, birth, phone(숫자만), password?, role`	`http.put` (JSON)

상태 갱신	성공 시 로컬 `currentUser` 갱신 → 화면 반영	`notifyListeners()`
결과 화면	성공/실패 카드 + 버튼	`EditProfileResultScreen`

4 . AI진단

1) 흐름

(1) 문진 응답 수집 : 각 문항 타입에 맞게 값 모아 `Map<String,dynamic>` 생성

(2) 업로드 화면 이동 : `/upload` 로 이동하며 `extra` 로 `survey` 전달

(3) 이미지 선택 : 일반/엑스레이 타입 선택 후 파일(모바일: path / 웹: bytes) 고르기

(4) 멀티파트 업로드 : `POST /upload` (헤더: Bearer 토큰 / 필드: `user_id` , `survey` , `image_type` , 파일: `file`)

(5) 결과 분기 : 응답의 `image_type` 기준

- `normal` → `/upload_result_detail`
- `xray` → `/upload_xray_result_detail`

2) 연관 dart별 주요 사항

(1) dental_survey_screen : 문진 업로드로 응답 전달

(2) upload_screen : 업로드 요청, 분기 네비게이션

3) 요약 본

구역	핵심 포인트	사용 기법/패턴
문진 수집	`q.id → 값` 맵핑(단일/다중/텍스트/슬라이더/예·아니오)	타입별 스위치, `trim()` , null 제외

문진→업로드 전달	`/upload` 로 이동 시 `survey` 를 extra로 전달	GoRouter
이미지 타입	일반 ↔ 엑스레이 선택	xray`
업로드 전송	멀티파트 + 토큰	`MultipartRequest`, `Authorization: Bearer`
동봉 데이터	사용자/문진 같이 전송	`jsonEncode(widget.survey)`
정적 URL	`/api` 제거하여 이미지 URL 조립	문자열 치환
결과 분기	타입별 상세 화면 이동	`context.push(..., extra: {...})`

5 . 진료 기록

1) 흐름

(1) 결과 상세 화면에서 원본/오버레이 표시, **AI 소견** 요청(`multimodal_gemini` / `multimodal_gemini_xray`), **이미지 저장** 가능

(2) /history 목록에서 항목 선택 시, 저장된 결과를 /history_result_detail(일반)//history_xray_result_detail(X-ray)로 열람

(3) 히스토리 목록은 `GET /inference_results?role=P&user_id=...` 로 로드

2) 연관 dart별 주요 사항

(1) upload_result_detail_screen : 이미지 로드/AI 소견/이미지 저장

(2) upload_xray_result_detail_screen : X-ray 결과 상세

(3) history_viewmodel : 히스토리 목록 로드

3) 요약 표

구역	핵심 포인트	사용기법 패턴
결과 상세(일반)	원본+오버레이(3종) 표시, AI 소견, 이미지 저장	이미지 GET 시 `Authorization: Bearer`
결과 상세(X-ray)	원본+오버레이(2종), AI 소견	모델/예측 수를 본문으로 전달
히스토리 목록	사용자 기록 로드	로딩/에러 상태 관리
히스토리 상세	저장 결과 열람(일반/X-ray)	`isRequested` / `isReplied` 표시
라우팅 연결	화면 간 데이터 전달	`extra` 로 URL·모델정보·상태 전달
전역 주입	History VM 등록	`HistoryViewModel(baseUrl)`

6 . 카메라 진단

1) 흐름

(1) 모델링 & 실시간 추론

- 앱 자산(`assets/models/*.tflite`)을 문서 디렉토리로 복사 후 경로 YOLOView에 전달\
- `onResult` 에서 YOLO 결과 수신 → 직렬화 저장 → FPS 표시 *(camera_inference_screen.dart)*

(2) 캡처& 저장&업로드

- ‘YOLOViewController.captureRawFrame()’으로 원본 프레임 추출
- Android에서 사진 권한 요청 → 갤러리에 저장

- 'HttpService.uploadImageWithToken(userId, imageData, filename, yoloResultsJson)'로 서버 업로드

(4) UI제어

- 카메라 전환, 줌(0.65/1/3X 토글),Threshold슬라이더(NumItems/Confidence/IoU)

2) 연관 dart별 주요 사항

(1) camera_inference_screen

- YOLO 결과 직렬화 & 콜백
- 모델 로딩(자산 → 앱폴더) & 모델 선택
- 캡처 → 갤러리 저장 → 서버 업로드
- 카메라/줌/슬라이더 제어

3) 요약 표

구역	핵심 포인트	사용 기법과 패턴
진입 라우트	`/diagnosis/realtime` (로그인 사용자 ID 주입)· `/camera` 로 진입	**GoRouter** `state.extra` / `Provider(AuthViewModel)` 로 `baseUrl/userId` 전달
모델 로딩	자산 tflite를 앱 문서 폴더로 복사 후 경로 지정	`rootBundle.load` → `path_provider.getApplicationDocumentsDirectory()` → 파일 저장 → `_modelPath` 세팅
실시간 추론	카메라 프리뷰에서 YOLO 추론·FPS 표시·결	**YOLOView/YOLOViewController**,

	과 콜백	`onResult` (바운딩 박스 직렬화), `onPerformanceMetrics` (FPS)
캡처	실시간 프레임을 원본 이미지로 캡처	`YOLOViewController.captureRawFrame()` , 캡처 전 `setVisibility(false)` 로 안정화, 재시도 루프
저장	캡처 이미지를 갤러리에 저장	**permission_handler**(Android 13+ `photos`), **image_gallery_saver.saveImage**
업로드	이미지+YOLO 결과를 서버로 전송	**HttpService.uploadImageWithToken**(멀티파트), `Authorization: Bearer` 토큰, `userId/filename/yoloResultsJson` 동봉
제어 UI	전·후면 전환, 0.5×/1×/3× 줌, 임계값 조절	`switchCamera()` , `setZoomLevel()` , `setNumItemsThreshold/` `setConfidenceThreshold/`

		`setIoUThreshold`
안정화/UX	로딩 상태/메시지·스낵바·오류 처리	캡처 중 로딩 문구, 예외 try/catch, 실패 시 사용자 피드백(SnackBar)
보안	인증 필요 리소스 접근	토큰은 **AuthViewModel.getAccessToken()**에서 획득, 모든 업로드/이미지 요청에 **Bearer** 헤더 적용
성능	불필요 연산 최소화·프레임 드랍 방지	모델 파일 1회 로딩 캐시, 캡처 시에만 가시성 토글, FPS 모니터링으로 설정(임계값/해상도) 튜닝

👨‍⚕️ 의사

1. 대시 보드

1) 흐름

(1) 화면 진입 시 **지표/차트/이미지 데이터 일괄 로드**

(2) 날짜를 달력에서 선택하면 해당 일자 기준으로 시간대 통계*사진*영상타입 비율 재로드

(3) 상단 카드/차트/이미지/날씨 카드로 **한 화면에서 상태 파악**

(4) 상단 카드 클릭 등으로 **세부 화면**(원격진료 신청 등) 이동

2) 연관 dart별 주요 사항

(1) d_real_home_screen(의사 대시보드 화면)

- 대시보드 진입: 초기 로딩 묶음
- 날짜 선택 시 재로드
- 시간대 차트·이미지 카드 UI(압축)

(2) d_dashboard_viewmodel

- 오늘 지표 로드
- 최근 7일 라인차트 데이터
- 시간대별 건수(24개 보장)
- 성별·연령 분포
- 날짜별 이미지(원본/오버레이 포함)
- 영상 타입 비율(도넛 차트)

(3) national_weather_card / main : 날씨 카드(상단 미니 HUD)

3) 요약 표

구역	핵심 포인트	사용 기법과 패턴
상단 지표 카드	**오늘의 진료/진단 대기/진단 완료** 숫자 표시·클릭 이동	`GET /consult/stats?date=YYYYMMDD` → `requestsToday/answeredToday/unreadNotifications`, **GoRouter**로 세부화면 이동
최근 7일 추이	일별 신청 건수 라인 차트	`GET /consult/recent-7-days` → `FlSpot` 생성, **fl_chart(LineChart)**
시간대별 건수	0–23시 24슬롯 보장, 값/라벨 길이 보정	`GET /consult/hourly-stats?date=YYYYMMDD`, 길이 미스매치 보정, **FlChart**
성별·연령 분포	성별 카운트, 연령대 맵	`GET /consult/demographics`, `male/female`,

		`ageDistributionDat a`
영상 타입 비율	**X-ray vs 구강이미지** 비율 도넛 차트	`GET /consult/video-type-ratio? date=YYYY-MM-DD`, **PieChart**
이미지(원본/오버레이)	자동 레이어 순환, 전체보기, 상태 배지	`GET /consult/images? date=YYYY-MM-DD&limit=&offset=`, `DashboardImageItem{ imageType, overlays}`, **AnimatedSwitcher + Timer**
달력(일자 선택)	선택된 날짜 기준으로 데이터 새로 로드	**TableCalendar** `onDaySelected` → `loadHourlyStats/loa dImagesByDate/loadV ideoTypeRatio`
날씨 미니 카드	도시 로테이트·현재날씨·시계	**NationalWeatherService** Provider, `NationalWeatherCar d` (애니메이션 전환)
상태 관리	로딩/에러·뷰모델 단일 소스	**Provider(ChangeNotifier)**, `notifyListeners()`, `isLoadingImages`

2 . 결과 상세(일반/Xray)&의사 답변

1) 흐름

(1) 상태 조회

(2) 결과/이미지 로딩

(3) 오버레이 제어

(4) AI 소견

(5) 의사 답변 제출

2) 연관 dart별 주요 사항

(1) d_result_detail_screen

- 일반(구강) 결과 상세 UI. 상태 조회
- 인퍼런스/오버레이 로딩
- 오버레이 토글
- AI 소견 생성
- 의사 코멘트 제출

(2) d_xray_result_detail_screen : X_ray 전용(임플란트 제조사 포함)

- X-ray 결과 상세 UI. 일반 흐름
- **임플란트 제조사 분류** 호출/표시. 이미지는 **토큰 GET**으로 바이트 로딩

3) 요약 표

구역	핵심 포인트	사용 기법과 패턴
상태 조회	응답 여부/기존 코멘트 프리필	`GET /consult/status` + **Bearer**, `TextEditingController`
결과/이미지	모델 결과·오버레이 로딩	`GET /inference_results` (role=D), **상대→정적**

		URL(`baseUrl` - `/api`), **토큰 이미지 GET**
오버레이 UI	레이어 토글/스택	`Stack` + `Switch/Toggle` + `AnimatedSwitcher`
AI 소견	모델 요약값 기반 생성	`POST /multimodal_gemini(_xray)` + **flutter_markdown**
X-ray 제조사	상대경로로 분류 요청	`POST /xray_implant_classify` 또는 `/implants/detect-manufacturer`
의사 답변	코멘트 저장 & 상태 반영	`POST /consult/reply`, 성공 후 알림/다이얼로그

3. 비대면 신청(신청 목록/썸네일)

1) 흐름

(1) **좌측 필터/상태칩 + 우측 리스트/썸네일**(웹은 좌측 고정, 모바일은 드로어)

(2) **썸네일 로딩**: 각 항목의 `url`을 **Bearer 토큰**으로 GET → 메모리 이미지

(3) **페이지네이션/검색**: 상태칩/검색어/페이지 변경에 따른 리스트 갱신

2) 연관 dart별 주요 사항

(1) d_telemedicine_application_screen

- 비대면 신청 리스트/필터/썸네일 UI
- 토큰 인증 썸네일 로딩, 페이지네이션, 상세 라우팅

(2) DoctorDrawer : 모바일

- 모바일에서 사용하는 **사이드 드로어**
- 대시보드/목록 등 네비 항목과 필터 트리거 제공

3) 요약 표

구역	핵심 포인트	사용 기법과 패턴
레이아웃	사이드 필터 + 리스트	웹 고정 사이드, 모바일 Drawer(`DoctorDrawer`)
썸네일	토큰 인증 이미지	`http.get(url, Bearer) → Image.memory`
상태칩/검색	필터링/페이지 이동	`setState` 기반 상태칩·검색·페이지네이션
반응형	웹/모바일 최소 폭·툴팁	`kIsWeb` 분기, hover/press 애니

4 . 환자 목록/상세

1) 핵심 흐름

(1) 로그인한 **의사 ID**로 환자 목록 **조회**

(2) **검색/정렬**로 화면 내 필터링

(3) **환자 추가** 다이얼로그 입력 → `addPatient` 성공 시 **스낵바** + 목록 **재조회**

(4) 환자 탭하면 **상세 화면**으로 이동(기본 인적 사항 카드, 진단 탭으로 유도)

2) 연관 dart별 주요 사항

(1) d_patient_list_screen(실제 환자 목록 화면)

- 의사 로그인 검증 후 목록 불러오기
- **추가/검색** 포함

(2) d_patient_detail_screen(환자 기본 정보 카드 중심 상세 화면)

- 진단/결과 탭으로 이동 유도(해당 API는 별도 뷰모델)

3) 요약 표

구역	핵심 포인트	사용 기법과 패턴
목록 로드	의사 계정 기준으로 조회	`AuthViewModel` 에서 doctorId → `fetchPatients(doctorId)`
검색/정렬	화면 내 실시간 필터	입력 컨트롤 + `setState`
환자 추가	다이얼로그 → 생성 → 재조회	`addPatient(...)` 후 스낵바 + 갱신
상세 보기	기본 정보 / 진단 탭 유도	카드 UI, **GoRouter** 이동

5 . 기록/연동(의사용 히스토리 VM)

1) 흐름

(1) 날짜/조건으로 기록 목록 로드

(2) 원본/오버레이 상대경로 → 정적 URL 변환

(3) 로딩/에러/빈 상태를 뷰모델 상태로 관리 → 구독 화면에 반영

(4) 상세로 넘길 extra(userid,original/overlays,inferenceResultId...) 키 일관 유지

2) 연관 dart별 주요 사항

(1) d_history_viewmodel(의사용 **히스토리 전담 뷰모델**)

- 기록 목록/이미지 URL/상태값을 중앙에서 관리하고, 구독 화면(상세/목록)에 제공

3) 요약 표

구역	핵심 포인트	사용 기법과 패턴

기록 조회	날짜/조건 기반	`ChangeNotifier` VM, `notifyListeners()`
URL 정규화	상대→정적 URL	`baseUrl.replaceFirst('/api','')`
상태 관리	로딩/에러/빈	불변 리스트 교체, 에러 문자열
데이터 전달	상세 파라미터 통일	`extra` 키 표준화