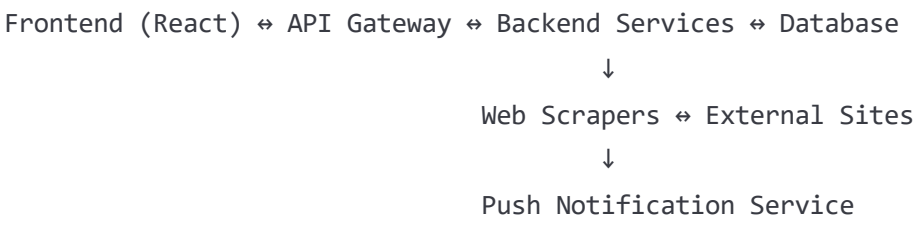
Backend Implementation Plan - Ledig Tid App

Arkitektur Oversikt



1. Tech Stack

Backend

- **Node.js + Express** eller **Python + FastAPI**
- **PostgreSQL** database
- **Redis** for caching og job queues
- **Docker** for containerization

Web Scraping

- **Puppeteer** (Node.js) eller **Selenium** (Python)
- **Cheerio** for HTML parsing
- **Proxy rotation** for rate limiting

Push Notifications

- **Firebase Cloud Messaging (FCM)** for web push
- **WebPush** library for browser notifications

2. Database Schema

sql

```sql
-- Baner/anlegg
CREATE TABLE venues (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address TEXT,
    club VARCHAR(255),
    source_url TEXT,
    venue_type VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW()
);

-- Tidslots for hver bane
CREATE TABLE time_slots (
    id SERIAL PRIMARY KEY,
    venue_id INTEGER REFERENCES venues(id),
    date DATE,
    start_time TIME,
    end_time TIME,
    status VARCHAR(50), -- 'ledig', 'opptatt'
    activity_description TEXT,
    last_updated TIMESTAMP DEFAULT NOW()
);

-- Brukere og favoritter
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    push_subscription JSONB, -- FCM subscription data
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE user_favorites (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    venue_id INTEGER REFERENCES venues(id),
    time_preferences JSONB, -- preferred times of day
    notification_enabled BOOLEAN DEFAULT true
);

-- Scraping log
CREATE TABLE scrape_logs (
    id SERIAL PRIMARY KEY,
    venue_id INTEGER REFERENCES venues(id),
    scrape_time TIMESTAMP DEFAULT NOW(),
    success BOOLEAN,
    error_message TEXT,
```

```
    data_hash VARCHAR(255) -- for detecting changes
);
```

3. Web Scraping Implementation

Gruner Fotball Scraper

javascript

```javascript
// scraper/gruner-scraper.js
class GrunerScraper {
    async scrapeDaelenenga() {
        const url = 'https://fotball.gruner.no/kamper-og-treninger-pa-daelenenga/';
        const browser = await puppeteer.launch();
        const page = await browser.newPage();

        try {
            await page.goto(url, { waitUntil: 'networkidle2' });

            // Extract schedule data
            const scheduleData = await page.evaluate(() => {
                // Parse HTML structure specific to Gruner's site
                const events = [];
                document.querySelectorAll('.schedule-item').forEach(item => {
                    // Extract time, activity, status
                });
                return events;
            });

            return this.processScheduleData(scheduleData, 'daelenenga');
        } finally {
            await browser.close();
        }
    }

    processScheduleData(rawData, venue) {
        // Convert to standardized format
        return rawData.map(item => ({
            venue,
            date: item.date,
            startTime: item.startTime,
            endTime: item.endTime,
            status: item.isBooked ? 'opptatt' : 'ledig',
            activity: item.activity || ''
        }));
    }
}
```

Arena.club Scraper

javascript

```javascript
// scraper/arena-scraper.js
class ArenaScraper {
    async scrapeMuselunden11er() {
        const url = 'https://arena.club.no/club/skeid/field/muselunden-11er';
        // Similar implementation, adapted for Arena.club's HTML structure
    }
}
```

4. API Endpoints

javascript

```javascript
// routes/api.js
const express = require('express');
const router = express.Router();

// Get all venues and their current availability
router.get('/venues', async (req, res) => {
    const { date, timeFilter } = req.query;
    const venues = await getVenuesWithAvailability(date, timeFilter);
    res.json(venues);
});

// Get specific venue details
router.get('/venues/:id', async (req, res) => {
    const venue = await getVenueById(req.params.id);
    res.json(venue);
});

// Subscribe to push notifications
router.post('/notifications/subscribe', async (req, res) => {
    const { subscription, preferences } = req.body;
    const userId = await createOrUpdateUser(subscription, preferences);
    res.json({ success: true, userId });
});

// Manual refresh trigger
router.post('/scrape/refresh', async (req, res) => {
    await triggerImmediateScrape();
    res.json({ message: 'Scrape triggered' });
});
```

5. Automatisk Oppdatering

Cron Jobs

```javascript
// jobs/scheduler.js
const cron = require('node-cron');

// Scrape every 30 minutes during active hours (6-23)
cron.schedule('*/30 6-23 * * *', async () => {
    console.log('Starting scheduled scrape...');
    await runAllScrapers();
});

// Clean up old data daily at 2 AM
cron.schedule('0 2 * * *', async () => {
    await cleanupOldTimeSlots();
});

async function runAllScrapers() {
    const scrapers = [
        new GrunerScraper(),
        new ArenaScraper()
    ];

    for (const scraper of scrapers) {
        try {
            await scraper.scrapeAll();
            console.log(`${scraper.constructor.name} completed successfully`);
        } catch (error) {
            console.error(`${scraper.constructor.name} failed:`, error);
            // Log to monitoring system
        }
    }

    // Check for changes and trigger notifications
    await checkForAvailabilityChanges();
}
```

6. Push Notifications

javascript

```
// services/notification-service.js
const webpush = require('web-push');

class NotificationService {
    constructor() {
        webpush.setVapidDetails(
            'mailto:your-email@example.com',
            process.env.VAPID_PUBLIC_KEY,
            process.env.VAPID_PRIVATE_KEY
        );
    }

    async notifyAvailabilityChange(venueId, newAvailableSlots) {
        const interestedUsers = await getUsersWatchingVenue(venueId);

        for (const user of interestedUsers) {
            const matchingSlots = this.filterSlotsByUserPreferences(
                newAvailableSlots,
                user.preferences
            );

            if (matchingSlots.length > 0) {
                await this.sendPushNotification(user, venueId, matchingSlots);
            }
        }
    }

    async sendPushNotification(user, venueId, availableSlots) {
        const venue = await getVenueById(venueId);
        const payload = JSON.stringify({
            title: '🟢 Ny ledig tid!',
            body: `${venue.name} har ledig tid: ${availableSlots[0].time}`,
            icon: '/icons/icon-192x192.png',
            badge: '/icons/badge-72x72.png',
            url: `/?venue=${venueId}`,
            tag: `venue-${venueId}`
        });

        try {
            await webpush.sendNotification(user.pushSubscription, payload);
        } catch (error) {
            if (error.statusCode === 410) {
                // Subscription expired, remove from database
                await removeUserSubscription(user.id);
            }
        }
```

```
    }
}
```

7. Error Handling & Monitoring

javascript

```javascript
// middleware/error-handling.js
class ScrapingMonitor {
    async logScrapeAttempt(venueId, success, error = null) {
        await db.query(`
            INSERT INTO scrape_logs (venue_id, success, error_message)
            VALUES ($1, $2, $3)
        `, [venueId, success, error?.message]);
    }

    async checkScrapeHealth() {
        const failedScrapes = await db.query(`
            SELECT venue_id, COUNT(*) as failures
            FROM scrape_logs
            WHERE success = false
            AND scrape_time > NOW() - INTERVAL '2 hours'
            GROUP BY venue_id
            HAVING COUNT(*) >= 3
        `);

        if (failedScrapes.length > 0) {
            // Alert admin about failing scrapers
            await this.alertAdmin(failedScrapes);
        }
    }
}
```

8. Deployment & Infrastructure

Docker Setup

dockerfile

```dockerfile
# Dockerfile
FROM node:18-alpine

RUN apk add --no-cache chromium
ENV PUPPETEER_SKIP_CHROMIUM_DOWNLOAD=true
ENV CHROMIUM_PATH=/usr/bin/chromium-browser

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

Docker Compose

```yaml
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/ledigtid
      - REDIS_URL=redis://redis:6379
    depends_on:
      - db
      - redis

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: ledigtid
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine

volumes:
  postgres_data:
```

9. Implementering Timeline

Fase 1 (Uke 1-2): Core Backend

- [ ] Database setup og migrations
- [ ] Basic API endpoints
- [ ] Initial web scrapers for 4 baner

Fase 2 (Uke 3): Automatisering

- [ ] Cron jobs for automatisk scraping
- [ ] Error handling og logging
- [ ] Data validation og cleanup

Fase 3 (Uke 4): Push Notifications

- [ ] Web push implementation
- [ ] User preferences system
- [ ] Notification triggering logic

Fase 4 (Uke 5): Production

- [ ] Docker deployment
- [ ] Monitoring og alerting
- [ ] Performance optimization

10. Kostnader & Hosting

Månedlige kostnader (estimat):

- Digital Ocean Droplet (2GB): $12/mnd
- Database backup: $5/mnd
- Domain + SSL: $2/mnd
- **Total: ~$20/mnd**

Denne planen gir deg en robust backend som kan:

- Skrape data fra alle 4 nettsider automatisk
- Sende push-varsler når favorittbaner blir ledige
- Håndtere feil og holde data oppdatert
- Skalere til flere baner i fremtiden